Department of Computer Science
The University of Auckland
New Zealand

Characterizing Drifts for Proactive Drift Detection in Data Streams

Kylie Chen

May 2016

Supervisors:

Dr. Yun Sing Koh

Dr. Patricia Riddle

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Abstract

The evolution of data such as changes in the underlying model known as concept drift present many challenges for data stream research. Currently most drift detection methods are able to locate the point of change, but are unable to provide meaningful information on the characteristics of change or utilize historical trends. In this thesis, we investigate two streams of research: (1) the magnitude of change which we refer to as drift severity, and (2) the rate of change which we refer to as the stream volatility [7].

In the first part, we propose a drift detector, MAGSEED, for tracking the drift severity of a stream. Monitoring drift severity provides crucial information to users allowing them to formulate a more adaptive response. We show that our technique is capable of tracking drift severity with a high rate of true positives and a low rate of false positives and compare it to state-of-the art drift detectors ADWIN2 and DDM.

In the second part, we explore ways to learn historical drift rate trends, and develop a proactive drift detection system. The main motivation for our work comes from the observation of volatility trends resulting from the application of current drift detection methods to real data streams. We observe that these patterns of change vary across different data streams. We use the term "volatility pattern" to describe change rates with a distinct distribution. We propose a novel drift prediction method, DPM, to predict the location of future drift points based on historical drift trends which we model as transitions between stream volatility patterns. Our method uses a probabilistic network to learn drift trends and is independent of the drift detection technique. We demonstrate that our method is able to learn and predict drift trends in streams with reoccurring volatility patterns. This allows the anticipation of future changes which enables users and detection methods to be more proactive. We then apply our drift prediction algorithm by incorporating the drift estimates into a drift detector, PROSEED, to improve its performance by decreasing the false positive rate.

Acknowledgements

I would like to thank my supervisors Dr. Yun Sing Koh and Dr. Patricia Riddle for their immense support, guidance and endless patience throughout my Masters degree. They have given me an appreciation for the scientific method and are my role models.

In addition, I would like to thank the reviewers that greatly improved the quality of our paper, and the participants of the 2015 Australasian Joint Conference on Artificial Intelligence for sharing their knowledge with me.

I would also like to thank my family for allowing me to pursue my research interests and my friends for their words of encouragement.

List of Publications

Parts of Chapter 3 have been published in:

- Kylie Chen, Yun Sing Koh, and Patricia Riddle. Tracking Drift Severity in Data Streams. In AI 2015: Advances in Artificial Intelligence, pages 96-108, 2015.

Parts of Chapter 4 have been accepted for publication in:

- Kylie Chen, Yun Sing Koh, and Patricia Riddle. Proactive Drift Detection: Predicting Concept Drifts in Data Streams using Probabilistic Networks. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2016.

Contents

List of Figures

List of Tables

1 Introduction

Data stream mining has been an increasingly popular area of research in recent years. Mining refers to the act of finding useful information such as relationships or trends from data. For example, identifying the types of patient symptoms linked to various cancer treatments over time. Data streams consists of unbounded data that typically arrive at a fast rate. We believe the main driver of this is the rapid generation and availability of large amounts of data. For example video feeds, GPS sensors, social networks, and biological sequencing techniques all produce vast amounts of data. However the nature of big data also poses many challenges to the acquisition of knowledge from these sources. There are five key components to big data [20] [26].

- **Volume**: The volume describes the size of the data. In this era, we are producing more data than ever before. IBM estimates that 90% of the world's data was produced in the last two years [18]. The high volume of data creates a need for processing techniques that are memory efficient.
- **Velocity**: This aspect refers to the high speed at which data arrives, which creates a need for memory and time efficient techniques that can cope with high speeds of data arrival.
- **Variety**: The variety describes the different types of the data available. For example structured data such as profiles in databases, or unstructured data like free form text, images and video recordings. The richness of data types available allow us

to integrate data from a variety of sources. The advantage of this is that different types of data can provide different views of the situation.

- **Veracity**: Veracity refers to data quality, and the belief we have in our data. The quality of data is an important issue that needs to be addressed in order to attain trustworthy insights from data.

- **Variability**: Variability refers to the change in meaning or context over time. For example, a word can have different meanings depending on its context.

Streaming data is characterized by large volumes of data arriving at high velocities, and the possibility of concept drift. Concept drift is a term used to describe changes in the underlying processes governing the generation of data such as a change in the behaviour of a system or population. Behavioural changes are very interesting as they can exhibit certain trends or patterns that reoccur over time.

Examples of this can be found in customer purchasing habits. The benefits of mining customer behaviour include increased revenue by targeting the appropriate set of customers at the optimal time. Another example is the tracking of the movement of rodents when exploring a new area. The potential benefits of modelling the exploratory behaviour could be used for predicting predator movement for conservation purposes. There is a great need for the development of techniques to efficiently process big data to allow the identification of interesting and useful patterns for knowledge discovery. This can be likened to finding a needle in a haystack.

1.1 Motivation

Much of scientific research involves the generation and testing of hypotheses that can facilitate the development of accurate models for a system. In machine learning the automated building of accurate models is desired. However traditional machine learning often assumes that the underlying models are static and unchanging over time. In reality there are many applications where the underlying model or system changes over time. This may be caused by changes in the conditions of the system, or a fundamental change in how the system behaves such as the development of antibiotic resistance in pathogens. This creates a need for systems that are able to detect and adapt to changes in the underlying model.

One limitation of current research is that few studies explore ways to characterize properties of changes in data streams. Most systems are only able to locate the point of change and are reactive in nature. This allows systems to respond to change, but are unable to anticipate changes. Methods for capturing the magnitude or size of changes have

been previously proposed by Kosina et al. [24] but their approach is mainly applicable to streams with abrupt changes in concepts. These are changes that occur suddenly such as the reversal of the magnetic poles. In contrast gradual changes occur over a period of time, for example, the spread of an innovation or a virus through a community. Being able to the monitor the magnitude of change can give us an early indication of how severe we expect a change to be. For example, suppose we monitor a population for the outbreak of flu. The magnitude could give us an indication of how many people are infected. Thus, we believe detecting the magnitudes of gradual changes is an important area to explore.

Recently the work of Huang et al. [17] introduced a characteristic called stream volatility that describes the drift rate of a stream. They develop a proactive method that is able to reduce the rate of false alarms based on predicting the next change location using the average drift rate of the stream. This shows promise in the use of the characteristics of changes for the development of an accurate proactive system, but the authors do not utilize information from the drift rate trends [16]. This does not account for the variability in drift rates over time. Consider the analogy where the drift rate is analogous to the rate of electricity consumption of a customer. The rate of electricity usage varies depending on the season and the time of day. In this case the average usage rate may not be an accurate estimate of future electricity usage. Therefore we are motivated to work towards a proactive system that can fully utilize the meta-knowledge from the drift rate trends over time.

1.2 Problem Statement

We are interested in the characterization of changes in streaming data and ways to utilize these features for change detection systems. The main research questions we explore are:

1. How can we accurately capture the magnitude of changes in a stream?

2. How can we characterize changes in data streams to provide interesting and valuable information?

3. How can we use the characteristics and trends in historical changes to enhance current change detection systems?

1.3 Objectives

Our research targets two facets of change in streaming data: drift magnitude, and drift rate. The first characteristic describes the size of changes, whereas the second describes the rate at which changes occur. We are interested in developing techniques to monitor

and utilize these characteristics for accurate detection and prediction of future changes. The aims of our research are:

1. To develop an algorithm that is capable of accurately tracking the magnitude of changes in streaming data
2. To develop an algorithm for making inferences on the characteristics of future changes based on historical drift rate trends
3. To develop an algorithm for the proactive detection of changes that is accurate for streams with reoccurring drift rate trends

1.4 Contributions

The main contributions of our work are:

- A drift detection algorithm, MAGSEED (MAGnitude SEED), that accurately tracks the magnitude of gradual changes in streaming data. To the best of our knowledge this is the first algorithm that focuses on monitoring magnitudes of gradual changes. Previous work in this area such as the DDM approach [24] work well for streams with abrupt changes but perform poorly on streams with gradual changes. We believe this is important as it allows the magnitude of changes in a larger variety of streams to be explored.
- A forecasting algorithm, DPM (Drift Prediction Method), that monitors the rate of changes and accurately predicts the location of future change points for streams with reoccurring drift rate trends. By drift rate trend, we are referring to the general movement of the drift rate over time. Previous work has focused on the identification of changes in the drift rate [17] but has not explored the use or detection of drift rate trends.
- A drift detection algorithm, PROSEED (PROactive SEED), that uses drift rate information and our forecasting algorithm to decrease the likelihood of triggering false alarms. Many drift detectors can be highly accurate at detecting the location of true changes achieving up to 100% true detection rate, but leave much to be desired in terms of the rate of false alarms. Detectors such as ADWIN2 [4] allow the rate of false alarms to be controlled by a confidence parameter that affects the sensitivity of the detector. However, due to the volume and velocity of streaming data even low false alarm rates such as 1% can be problematic. For example, lets assume we receive 1000 data instances per day, a 1% false alarm rate would trigger about 10 false alarms every day.

1.5 Overview of Research

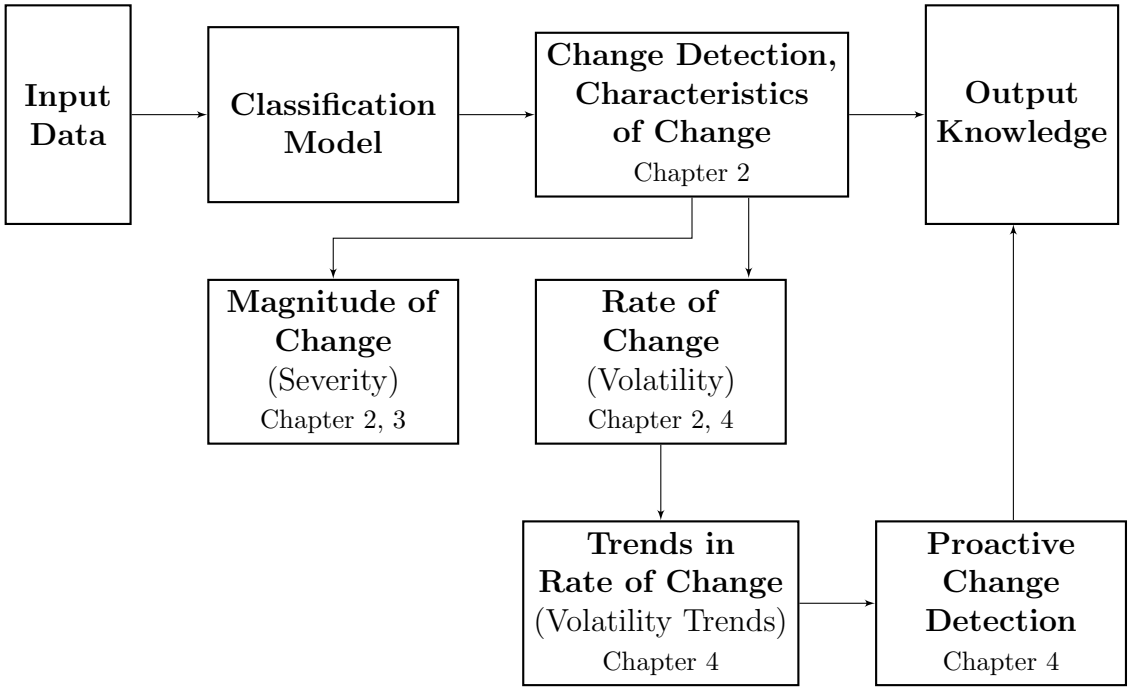


Figure 1.1: An overview of the topics covered in this thesis

Figure 1.1 shows the topics covered in this thesis, and the relationships between each topic. Our research is focused on characterizing and utilizing the nature of changes in data streams. This is a layer on top of current change detection systems that are used to detect changes in models such as classification models. We note that the application of change detection methods are not limited to classification models and can be applied to regression models or numeric data. Change detectors monitor characteristics of a model to identify when the underlying model which generates the data is likely to have changed. We look at the nature of these changes such as the magnitude and rate of change. Based on this, we can identify trends in the behaviour of changes which enable us to gain valuable insights into future change events and develop methods that are more proactive.

1.6 Structure of Thesis

This thesis is structured into the following chapters.

- Chapter 2: Background and Related Work
  We introduce the concept drift problem and review key concepts relevant to our research area. We discuss different drift detection methods that have been proposed and work related to the characterization of drifts such as the magnitude of change

(severity) and rate of change (volatility). In this chapter we also describe some datasets that are commonly used as benchmarks in the literature.

- Chapter 3: Tracking Drift Severity
  We highlight the challenges in detecting drift severity which was introduced in Chapter 2, and propose a novel drift detector, MAGSEED, for detecting the severity of concept change. We compare our detector to state-of-the-art drift detection methods and show that our detector outperforms current approaches for detecting the severity of streams with gradual change.

- Chapter 4: Proactive Drift Detection
  In Chapter 2 we described the concept of stream volatility which is the drift rate of a stream. In this chapter we explore ways to characterize and use volatility trends. We propose a forecasting algorithm, DPM, for predicting the location of future changes based on observed volatility trends. We show that our algorithm is able to accurately learn these trends for streams with reoccurring volatility trends. We then propose a proactive drift detector, PROSEED, that utilizes this additional information to accurately locate future changes.

- Chapter 5: Conclusions
  We conclude this thesis by discussing the limitations of our work and possible future directions.

2

Background and Related Work

In this chapter we review the background and current work in the area of drift detection in data streams. In Section 2.1 we introduce the area and formally define the concept drift problem. In Section 2.2 we present an overview of current drift detection techniques for addressing this. Section 2.3 outlines drift detectors aimed at detecting reoccurring models. Section 2.4 discusses work related to the characterization of drifts in streaming data and Section 2.5 describes some datasets that are commonly used in our research area.

2.1 Concept Drift

Concept drift is a phenomena in data streams where there is a change in the data distribution over time. It is usually studied in the supervised machine learning context, but there have been some attempts to branch into semi-supervised learning.

In supervised machine learning the aim is to accurately predict the target variable $y \in R^1$ of unseen instances given its corresponding set of input features $X \in R^P$ by using labelled training data. When the target variable y takes the form of discrete class labels it is known as the classification problem, otherwise in the case where y is continuous it is referred to as the regression problem. A typical approach for supervised classification is to train the learner (a classification algorithm) on a batch of labelled instances (training data) so a mapping from input features to the target variable can be used to model the

target function $p(y|X)$. Each labelled instance is a pair (X, y) of input feature values $X \in R^P$ and its class label $y \in R^1$. The training data is used to build a model to predict new unlabelled instances where the true class label y is unknown at the time of prediction. In traditional machine learning and data mining it is assumed that the process generating the data is stationary. That is the data comes from a single source. This differs from the streaming environment which is dynamic and poses many challenges to the memory, time and flexibility requirements of the algorithms. We will focus on the classification problem applied to streaming data.

Streaming data is defined as an unbounded ordered sequence of instances that arrive at a fast rate. The arrival rate may vary throughout the stream, but most studies assume instances arrive at a constant rate. There are a number of characteristics that make data streams different from non streaming data. Firstly the unbounded nature of the stream and fast arrival rate make it impossible to store all of the data into memory and imposes a memory constraint. The high speed of the data also restricts algorithms to requiring a single pass through the data. Instances need to be processed in real time, so that a prediction can be made at any point in time. The data may evolve over time - that is the distribution generating the data may change. This is referred to as concept drift. It is important to note that not all streams exhibit concept drift as the nature of the stream depends on the application domain and the underlying hidden processes that generate the data.

Typically the labels of the data are assumed to be correct and available shortly after prediction, though some work has been done on mining unlabelled or noisy streaming data. A formal definition of the classification problem and concept drift follows below.

The classification problem can be described using Bayes rule as

$$p(y|X) = \frac{p(y)p(X|y)}{p(X)}, \tag{2.1}$$

where there are c classes, and $p(X) = \sum_{y=1}^{c} p(y)p(X|y)$ [12]. Here $p(y)$ represents the prior probabilities of the classes. $p(X|y)$ represents the class conditional probabilities for all classes $y = 1, 2, \ldots, c$ [12]. $p(X)$ is the distribution of the input data, and $p(y|X)$ is the target function or posterior which represents the relationship between the input features and target variable we are trying to learn.

For example, suppose we have a stream of emails, and are interested in classifying emails into two categories: spam, and non spam. The input distribution $p(X)$ could represent the distribution of features like email size, keywords, or the domain of the sender's address. $p(y)$ represents the proportion of emails in each category - spam, and non spam. $p(X|y)$ would represent the distribution of input features conditioned on the class label y spam or non spam. $p(y|X)$ is the probability of an email being spam (or non

spam) given its input features.

We refer to a concept as the target function of interest generated by a source S with distribution D_S. Two concepts are identical if the sources are the same. Formally concept drift refers to a change in the target function $p(y|X)$ between some time points t_0 and t_1 such that the joint distribution $pt_0(X|y)$ between X and Y at time t_0 is different to the joint distribution at t_1. That is $pt_0(X|y) \neq pt_1(X|y)$ for some time points t_0 and t_1 [12]. Changes may occur in the class priors $p(y)$, class conditional probabilities $p(X|y)$ or the input distribution $p(X)$ which may or may not affect the classification $p(y|X)$. There may also be an arrival of new information such as new classes or concepts.

Gama et al. distinguish between two different types of concept drifts: real concept drift, and virtual drift [12]. Real concept drift refers to changes in the target function $p(y|X)$ and a change in the decision boundary between classes. Whereas virtual drift (also known as feature drift) refers to changes in the input data distribution $p(X)$ that do not affect the target function $p(y|X)$ [12].

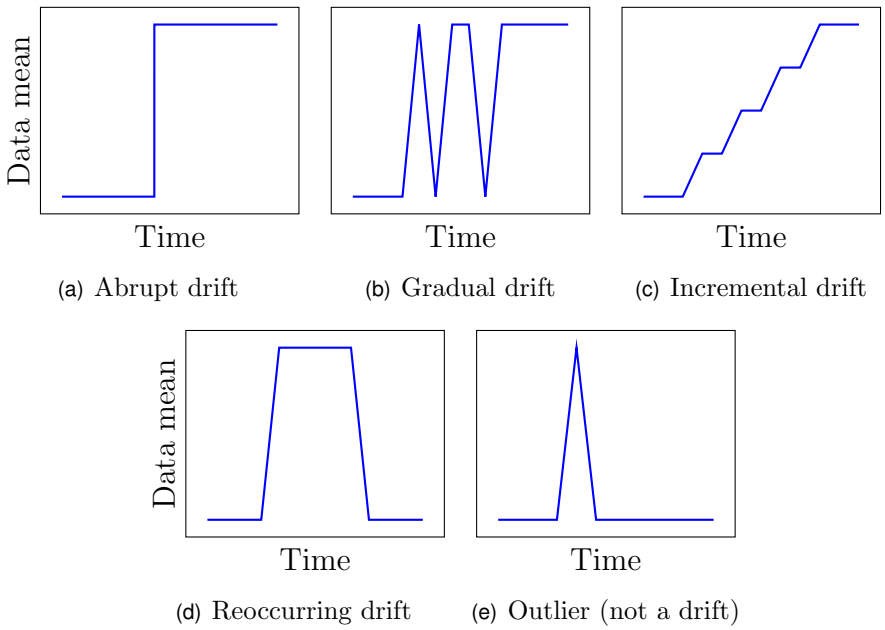
Using the same spam detection example above, there may be a feature drift if a larger proportion of emails become longer. This does not necessarily affect the decision rules that decide if an email is spam because although there is a change in the distribution of email lengths the behaviour of spammers may be unchanged. If this is the case, the learned model should remain valid and the learner's accuracy should not be affected. A real concept drift occurs if there is a change in what is considered spam and non spam. For example spammers may actively change their behaviour to bypass filters so there is a change in the relationship between features and the class label associated with spam mail.

As virtual concept drifts do not alter the decision boundary, this review will focus on work related to real concept drifts. In addition to this, there are four main types of drifts identified in the current literature: abrupt, gradual, incremental, and reoccurring concepts [12].

- Abrupt drift: A concept drift is said to be abrupt if there is an instantaneous change in the data generation process from one concept to a different concept at time point t. The new concept should persist for a period of time.

- Gradual drift: A gradual drift describes a slower speed of change where the replacement of a concept occurs over a period of time. There is a period of fluctuation between concepts from different sources where the proportion of examples from one concept is slowly overtaken by another.

- Incremental drift: This form of drift is where there are many intermediate concepts that change in a step-wise manner. The change between two neighbouring concepts may be small, but the difference between two concepts with time stamps that are



(a) Abrupt drift (b) Gradual drift (c) Incremental drift

(d) Reoccurring drift (e) Outlier (not a drift)

Figure 2.1: Patterns of change in streaming data showing four types of concept drift.

further apart may be more significant. This is also referred to as gradual drift by some researchers.

- Reoccurring concepts: Drifts with reoccurring concepts refer to changes where a previously seen concept reappears at some later point in time.

Figure 2.1 shows a visual depiction of four drift types and an outlier on one dimensional data assuming the source can be characterised by the mean of the data.

Most studies categorize concept drifts based on the speed of change, or the occurrence of novel or previously seen concepts and focus on the efficient detection of drifts. However little work has been done on identifying other characteristics of drifts such as the magnitude [17] and the shape of drifts. Work in this direction may benefit data exploration scenarios and provide insight into the underlying data generation process.

Here we will also make a distinction between reoccurring patterns which have been well studied in the community, and reoccurring drifts in the streaming context. Reoccurring patterns research in data streams is focused on identifying patterns or concepts that have previously occurred. This is often used to learn reoccurring concepts more quickly to improve the learning process. For reoccurring drifts we are interested in how the concepts change, which is different to reoccurring patterns that use the concept model as a pattern. This shares some similarities with motif matching in time series where patterns of change in the variable of interest are mapped to other similar change patterns. Some key differences between time series work and reoccurring patterns is that the former is a regression task whilst the latter is a classification task which has a more complex

hypothesis space, and may be characterised in more ways. For example in addition to magnitude, speed and rate of change, there may be ways to describe how the source distribution has changed.

Other different but related areas of research include time series prediction, pattern recognition in time series, and sequence classification. The main difference between work in time series and classification in data streams is that the variable of interest in time series is continuous (i.e. the regression problem) rather than discrete as in classification [42]. In sequence classification the examples are sequences rather than instances, and require all the training data to be available during the learning process [42].

Many types of approaches have been proposed to address concept drift, the main types include (1) blind adaptation such as gradual or abrupt forgetting mechanisms, (2) change detection, (3) adapting the learner [8], and (4) ensemble methods [38] [5].

2.2 Drift Detectors

Drift detection is an informed adaptive approach to the concept drift problem as they rely on the explicit detection of changes. Drift detection algorithms are often coupled with learners such as neural networks, naive Bayes, decision trees and K-nearest neighbours (KNN) as a method of dealing with concept drift. The detectors work by detecting the location of drifts which allows the learner to be readjusted. Drift detectors can be reactive - able to alert when a drift occurs, and predictive - able to predict the time of future drift occurrences. In contrast to methods that continuously adapt the learner and blind adaptation methods such as fixed window abrupt or gradual forgetting mechanisms, drift detectors can more accurately pinpoint the time of change and provide valuable information on the underlying process that generates the data [12]. For example in credit card fraud and intrusion detection it would be useful to be able to identify when and how the malicious behaviour has changed [31]. However drift detection methods are often prone to false positives, and do not cope well with noisy data.

Some desirable properties of a good drift detector are

1. Robustness to noise but with sensitivity to real changes
2. Fast detection of drifts
3. Be able to identify and respond to reoccurring concepts
4. Use limited memory and processing time
5. Require few user specified parameters

6. Provide statistical guarantees on performance

Parameters give us finer control on the drift detection process and often can not be avoided during the abstraction process. Thus it is important to provide guidelines on how to set parameters for different types of streams. Gama's survey [12] groups drift detection algorithms into four categories: sequential analysis [30], statistical process control [10] [2] [31], monitoring of two distributions [4] [34] [1] [29] [32] [36] [22], and contextual meta learning approaches. In our comparison of different drift detectors, the memory used by the base classifier is ignored and only the additional memory used by the detector is considered. The runtime of algorithms are given in terms of processing time per instance. We will highlight the main contributions of previous work in drift detection below.

2.2.1 CUSUM

The CUMulative SUM (CUSUM) method is a sequential analysis technique that detects changes by using the residue of the learner as input, and testing if there is a significant departure from zero in the mean of the input data [30]. It uses the variable g_t to store the mean of the input at time t. When the mean is greater than a user defined threshold h it raises an alarm indicating that a drift has occurred. It uses the following update rules and condition for drift detection

$$g_0 = 0, \quad g_t = max(0, g_{t-1} + x_t - v) \tag{2.2}$$

$$\text{if } g_t > h \text{ then raise alarm, and set } g_t = 0, \tag{2.3}$$

where x_t represents the current observed value at time t, v is a parameter that determines how much change is allowed, and h is the alarm threshold parameter.

The CUSUM approach is memoryless, but it uses an asymmetric test and assumes changes occur only in one direction. As noted by Bifet et al. [6] this means it is only able to detect increases in the monitored statistic (e.g. positive deviations from zero). The accuracy of this method depends on the choice of parameters h and v. Increasing the h parameter decreases the sensitivity of the detector requiring greater deviations for a change to be detected but may be less susceptible to false alarms (smaller deviations may be from noise). Lowering the v parameter allows faster detection of possible drifts, but increases the false alarm rate. There is a trade-off between faster detection of changes and allowing more false alarms.

2.2.2 PHT

The Page-Hinkley Test (PHT) is a variation on the CUSUM method that is used to detect sudden shifts in Gaussian signals [30]. An important application of this method is in signal processing. It records a minimum cumulative value (G_t) in addition to the current cumulative value (g_t). For an increasing signal, the conditions for flagging an alarm and updating are

$$g_0 = 0, \quad g_t = g_{t-1} + (x_t - v), \quad G_t = min(g_t, G_{t-1}) \tag{2.4}$$

$$\text{if } g_t - G_t > h \text{ then raise alarm, and set } g_t = 0 \tag{2.5}$$

For a decreasing signal, the equations become

$$g_0 = 0, \quad g_t = g_{t-1} + (x_t - v), \quad G_t = max(g_t, G_{t-1}) \tag{2.6}$$

$$\text{if } G_t - g_t > h \text{ then raise alarm, and set } g_t = 0 \tag{2.7}$$

The accuracy of this method also depends on the parameters v and h [6]. An advantage of using the Page-Hinkley method is that it is memoryless and has low computational overhead with O(1) time complexity at time point t. In a comparative study by [33] using synthetic data streams generated from Bernoulli distributions with a single drift point, PHT has been shown to have low delay times which are comparable to ADWIN and DDM, but has a high rate of false alarms.

2.2.3 GMA

The Geometric Moving Average (GMA) method is a sequential analysis method based on the CUSUM test. It provides a way to weight the importance of previous data using a forgetting factor (λ).

$$g_0 = 0, \quad g_t = \lambda g_{t-1} + (1 - \lambda)x_t \tag{2.8}$$

$$\text{if } g_t > h \text{ then raise alarm, and set } g_t = 0 \tag{2.9}$$

Setting a higher λ value will give more importance to old data, whilst a lower value will give more importance to the current observation x_t. The threshold parameter h is important for controlling the trade-off between faster drift detection and the number of false positives. Similar to other sequential analysis methods (CUSUM, PHT), it is a

memoryless approach and only needs O(1) processing time and memory at time point t. An advantage of this approach is that it allows the contribution of previous observations to be controlled so more relevance is given to recent data. This is in agreement with the common assumption in data stream mining that recent data has more relevance to future data.

2.2.4 FLORA

The FLORA family of algorithms [39] uses the overall accuracy and coverage of the learner as a measure of change, where a large decrease in accuracy suggests the occurrence of a concept drift. It monitors the accuracy and coverage of the model to allow the window size to be adjusted. A simple representation language of attribute-value logic without negation is used to represent the data [39]. They use three sets to describe a concept: accepted descriptors (ADES) containing positive examples, negative descriptors (NDES) with negative examples, and potential descriptors (PDES) containing examples that are too general (these may be positive or negative examples) [39].

Later implementations such as FLORA3 include capabilities for dealing with reoccurring concepts, and FLORA4 improves on its ability to handle noise [39]. The algorithms presented showed robust behaviour for synthetic datasets with abrupt drift, but the simple representation language restricts the types of data that it is applicable to, and is only appropriate for data of small size [40].

2.2.5 DDM

The Drift Detection Method (DDM) proposed by Gama et al. [10] uses the error rate of the classifier for drift detection. They assume that when a concept change occurs there would be an increase in the classification error as newer instances generated from a different distribution may be misclassified by the current model [10]. It records the current mean error rate p_i, and error standard deviation s_i as well as the minimum values p_{min}, s_{min} as each new instance is seen [10]. A normal distribution is used to set the thresholds as the number of errors approximates a Bernoulli distribution for samples sizes greater than 30. This means that there is at least a delay of 30 errors until a drift is detected [21]. This method uses two thresholds: a warning threshold, and an alarm threshold (which indicates there is a drift). The conditions for the raising the warning, and alarm signals are defined as

$$p_i + s_i \geq p_{min} + \alpha \,.\, s_{min} \quad \text{(warning)} \tag{2.10}$$

$$p_i + s_i \geq p_{min} + \beta \,.\, s_{min} \quad \text{(alarm)} \tag{2.11}$$

The authors suggest setting α to 2 for a 95% confidence of warning, and β to 3 for 99% confidence of drift. Using the proposed modifications, the warning threshold becomes $p_{min} + 2 \,.\, s_{min}$ and is updated with the arrival of new instances. A warning is raised when the current value of pi + si is greater or equal to the warning threshold and subsequent instances are stored in a warning window. If the error rate surpasses the drift threshold of $p_{min} + 3 \,.\, s_{min}$, an alarm is raised and the learner is retrained on instances stored in the warning window and the minimum values p_{min} and s_{min} are reset.

DDM has been shown to work well for detecting abrupt drifts, but may have trouble identifying gradual drifts as gradual changes may be undetected if the error does not surpass the threshold. An analysis done by Gonçalves et al. [21] on real and synthetic datasets shows that DDM performed favourably on datasets affected by gradual drifts based on accuracies and the average ranking. However in a study by Sebastiao et al. [33], DDM has been shown to perform poorly in comparison to ADWIN, PHT and FCWM detectors. In [33], DDM showed a high missed detection rate for data generated from Bernoulli distributions, and high detection delays for abrupt drifts using the SEA Concepts benchmark. This highlights the main limitations of the drift detection method.

2.2.6 EDDM

The Early Drift Detection Method (EDDM) is a variation on the drift detection method proposed by Gama et al. [10] and was designed to handle gradual concept drifts [2]. It differs in that it uses the distance of the errors rather than the error rate as a measure of change. Let p'_i denote the mean distance between two consecutive errors, and s'_i represent its standard deviation. The conditions for triggering a warning or alarm are

$$\frac{p'_i + 2 \,.\, s'_i}{p'_{max} + 2 \,.\, s'_{max}} < \alpha \quad \text{(warning)} \tag{2.12}$$

$$\frac{p'_i + 3 \,.\, s'_i}{p'_{max} + 3 \,.\, s'_{max}} < \beta \quad \text{(alarm)} \tag{2.13}$$

The main strength of this algorithm is the ability to detect slow gradual changes [21]. On the other hand it may be less resistant to noise, and is less effective for detecting sudden changes. Hence it may have a high false alarm and miss detection rate for these datasets as shown in the experiments of Gonçalves et al. [21] on datasets with abrupt drifts. Similar to DDM, this method may have a delay of 30 instances in the best case until a drift is detected due to the assumptions of the normal distribution approximation.

2.2.7 ADWIN

The ADaptive WINdowing approach (ADWIN) is a popular change detection method based on comparing data distributions from two different windows via the use of an adaptive sliding window [4]. The algorithm relies on the intuition that when two "large enough" windows of data have "distinct enough" averages then the expected values and hence source distributions will also be different [4]. They make three key assumptions about the data: (1) each data point x_i is a real value and can be scaled to [0,1] (the range is known), (2) each x_t is independent (does not depend on x_{t-1}), and (3) each x_t is generated according to some distribution D_t. The input for the algorithm is a sequence of real numbers $x_1, x_2, x_3, \ldots$ and a user defined confidence bound $\delta \in (0, 1)$ which indicates our confidence in the output. The authors use a binary sequence representing the classification error of the learner as the input data to their algorithm [4].

The detection method works by finding two sub-windows of the window W that have significantly different means. When two significantly different sub-windows are found the algorithm indicates a concept drift has occurred, and the window size is decreased by dropping the older sub-window. Otherwise the window size is enlarged when no drift is detected. As the window grows when no change is detected, there is no upper limit to the window size during long periods of stability - this leads to an unbounded memory requirement (this problem is remedied in a later version of the algorithm ADWIN2 by incorporating an upper bound on the window size). To test for distinct averages between two sub-windows W_0 and W_1 with observed means $\hat{\mu}W_0$ and $\hat{\mu}W_1$ they use

$$|\hat{\mu}W_0 - \hat{\mu}W_1| \geq \epsilon_{cut} \tag{2.14}$$

The value of ϵ_{cut} for partitioning window W is based on the Hoeffding bound, although the authors note that other statistic measures could be used [4]. In their implementation the ϵ_{cut} for partitioning the window W into two sub-windows W_0 and W_1 of lengths n_0 and n_1 respectively is defined as

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} . ln\frac{4}{\delta'}} \tag{2.15}$$

$$\delta' = \frac{\delta}{n}, \quad \text{where } \delta \in (0, 1), \text{ and } n = n_0 + n_1 \tag{2.16}$$

$$m = \frac{1}{1/n_0 + 1/n_1} \tag{2.17}$$

However, the authors observed that the definition of ϵ_{cut} in Equation 2.15 is too conservative and overestimates the probability of large deviations in distributions with small

variance. They suggest using a less rigorous bound with Bonferroni correction in practice (given below).

$$\epsilon_{cut} = \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot ln\frac{2}{\delta'}} + \frac{2}{3m} ln\frac{2}{\delta'}, \quad \text{where } \sigma_W^2 \text{ is the observed variance in } W \tag{2.18}$$

The checking of sub-window cut points is the most computationally expensive part of the algorithm. In ADWIN all large enough sub-windows are exhaustively checked. An improved version ADWIN2 uses a more efficient way to find the cut point by using a variation of exponential histograms as a data structure to provide exact counts of 1 bits for O(logW) sub-windows [4]. The amount of memory required for sub-window counts is comparable to the memory needed to keep counts for a single window W.

One limitation of the ADWIN2 approach is that it may require more memory than sequential or statistical approaches as it has a time and memory complexity of O(logW) compared to O(1) for the other approaches [12]. In comparison to statistical process control methods, it may give more precise information about the location of the concept drift [12], but requires multiple passes due to the generation of potential cut points with the arrival of new instances. An advantage of this approach is that it requires no parameters in addition to the confidence bound δ so does not require extensive parameter tuning. It has strict guarantees on the false positive and negative rates, and has been shown to outperform statistical process control methods and PHT in terms of detection delay and false positive rates [33]. The average delay time when handling data with gradual drifts is higher than that of abrupt drifts but is comparable to the delay times achieved by DDM and PHT [33]. Let μ_t represent the true mean at time step t, given that μ_t is constant in window W, the probability that a change is detected is at most δ [4]. Therefore the false positive rate is bounded by δ. A similar argument follows for the false negative rate: suppose some partition W_0W_1 has true means that are significantly distinct, the probability of change detection (shrinking the window to W_1 or shorter) is $1 - \delta$ [4]. So the probability of a false negative (missed detection) is bounded by δ.

Whilst this approach has many strengths, in the best case it has a delay of W instances before a drift is detected, as well as a higher time complexity and memory cost than statistical process control and sequential analysis methods. However experiments by Gonçalves et al. [21], Bifet and Gavaldà [4] show that the runtime is lower than all other algorithms included in the studies.

2.2.8 Re-Pro

In [40] and [41] the authors develop an algorithm (Re-Pro) that is both proactive and reactive. For drift detection they adopt a sliding window strategy with two parameters -

a window size, and an error threshold for drift detection, where the first instance in the window is always a misclassified instance. If the window is full and the error rate exceeds the error threshold, then the detector suggests a concept drift has occurred and the first instance is flagged as the trigger. Otherwise the window slides to the next misclassified instance. The main contribution of their paper is the use of a proactive mode that is used to predict the new concept when a change occurs. In proactive mode, the concept history is stored as a Markov Chain and a transition matrix is built by updating the probability of an alternate concept following a candidate concept [40]. This can be used to find the most probable future state given the previous stable concept. The agreement of class labels between two concepts on current data D is used as a measure of conceptual equivalence to prevent redundant concepts from being stored. The reactive mode works by retrieving the historical concept with the highest accuracy on instances in the trigger window. After a drift is detected, if both the proactive and reactive modes give models with low accuracy the system concludes the new concept is very different to the historical concepts and can train a new classifier using the data in the trigger window.

Experiments from [40] showed that the Re-Pro system has low error rates and is able to adapt quickly to concept change for gradual and abrupt drifts. For synthetic datasets, the Re-Pro approach outperforms ensemble methods (WCE and DWCE) and the concept adapting very fast decision tree (CVFDT) [40]. It has also been shown to adapt well to one real world dataset (the Network Intrusion dataset) and is comparable to CVFDT in terms of the error rate [40].

2.2.9 STEPD

Nishida et al. propose a drift detection method (Statistical Test of Equal Proportions) based on comparing the accuracy of the most recent W examples with the overall accuracy of the classifier from the beginning of the learning process [29]. They assume a concept is stable if the accuracy of the most recent W examples is equal to the overall accuracy, and a significant decrease in the accuracy of the recent examples indicates a concept drift [29]. To test if there is a significant decrease between the two accuracies a Chi-square test with Yates's continuity correction is performed by computing

$$T(r_o, r_r, n_o, n_r) = \frac{|r_o/n_o - r_r/n_r| - 0.5(1/n_o + 1/n_r)}{\sqrt{\hat{p}(1-\hat{p})(1/n_o + 1/n_r)}} \tag{2.19}$$

where r_o is the number of correct classifications in the overall n_o examples excluding the recent W examples. n_r is the number of recent examples ($n_r = W$), r_r is the number of correct classifications in the recent W examples, and $\hat{p} = (r_o + r_r)/(n_o + n_r)$. The value of T is then compared to the percentile of the normal distribution to obtain a P-value P which represents the observed significance level [29]. The null hypothesis is that there is

no difference between the overall and most recent accuracy ($H_0 : r_o/n_o = r_r/n_r$). The alternate hypothesis is that there is a decrease in accuracy when comparing the overall and most recent examples ($H_1 : r_o/n_o > r_r/n_r$). The detector is only able to start detecting drifts after at least $2W$ examples are seen (when the condition $n_o + n_r \geq 2W$ is satisfied) [29]. If the P-value P falls below a significance level then the null hypothesis is rejected and the alternate hypothesis is accepted suggesting a concept drift. Similar to DDM and EDDM it uses two thresholds: a warning threshold (α_w), and an alarm threshold (α_d). When the warning threshold is reached $P < \alpha_w$, examples are stored in a short term memory in anticipation of a drift. If $P < \alpha_d$ the detector indicates a drift has occurred, the classifier is retrained on the stored examples and all variables are reset. Otherwise if $P \geq \alpha_w$ the stored examples are cleared from memory.

Results from their experiments [29] suggest that the STEPD drift detection method works best on abrupt drifts in terms of number of true positives (correct identification of the change point), false negatives (miss detections), and error rate. For gradual drifts they showed that the performance of STEPD is comparable to EDDM in terms of true positives but with less sensitivity to noise as STEPD has much fewer false negatives than EDDM in all synthetic streams except the hyperplane stream with very gradual drift using Naive Bayes as the classifier [29]. In [21] this method was shown to have low miss detection rates, and low false alarm rates which is consistent with the original paper [29]. They also showed it is very effective at locating abrupt drifts, and almost always detects drift within the same time step [21].

2.2.10 PL

The Paired Learners (PL) [1] method uses two learners - a stable and a reactive learner and uses the approach of monitoring distributions from different time windows. The stable learner S is a classifier that makes predictions based on all of its experience, and the reactive learner R_W makes predictions using the most recent window of W examples [1]. It uses training data as the input to the algorithm and has two parameters: W - the window size of the reactive learner, and θ - the threshold for creating a new stable learner. The idea behind the algorithm is that when the reactive learner is able to make better predictions than the stable learner, the model learned by the stable learner is outdated and should be replaced. To achieve this they use a circular list C of size W, and process each instance in the following way:

Suppose an instance arrives at time point t. Let $\hat{y}_S$ be the prediction from the stable classifier S, let $\hat{y}_R$ be the prediction result from the reactive learner R_W, and y_t be the correct label available at a later time. If the stable learner incorrectly classifies the instance that the reactive learner correctly classifies $\hat{y}_R = y_t$ AND $\hat{y}_S \neq y_t$, then the t^{th} bit in the list C is set to one. Otherwise the t^{th} bit is reset to zero. The condition for detecting

drift is

$$Set(C) > \theta, \tag{2.20}$$

where $Set(C)$ is the proportion of 1 bits in list C. When the proportion of bits exceeds the threshold θ a drift is detected. Following the detection of a drift, S is replaced with a new learner, all bits in C are reset, and the concept description is set to R_W's description.

For the stable learner an online implementation of Naive Bayes was used. However as the reactive classifier learns from the most recent examples it needs to be continuously updated, the authors propose two ways to achieve this: (1) by rebuilding the learner's model with the most recent W examples, and (2) using a retractable learner to unlearn the oldest instance in the window and incorporate the newest instance into the model [1]. The first method is more general, but requires more time which increases with W [1]. The latter method can be easily implemented using the Naive Bayes classifier, but does not work for all learners. Smaller window sizes allow more rapid detection of abrupt drifts at the expense of an increased false positive rate.

The processing time required at time t is O(W) if the reactive learner is completely retrained at every time step, otherwise the processing time is O(1) if the reactive learner is retractable. The memory required is O(W). The detection delay in the best case is $\theta \cdot W$ instances and depends on both W and θ. In the original paper [1] this method was compared to a range of ensemble approaches (DWM, SEA, AWE), and was shown to achieve comparable performance using less memory. In [21], the paired learners method was shown to have the lowest accuracy for most synthetic streams with gradual drifts compared to seven other detectors (DDM, EDDM, PHT, STEPD, DOF, ADWIN, and ECDD). However, this method shows promise in the detection of abrupt drifts as it is able to identify drift points close to the location of the real concept change for datasets with abrupt drift [21]. Although parameter tuning was performed in an attempt to find the best settings [21], it is difficult to control the false alarm rate and achieve fast detection at the same time. Further work could be done to improve the false alarm rate of this detector.

2.2.11 FCWM

The Fixed Cumulative Windows Model (FCWM) algorithm [34] monitors distributions from two different windows: a reference window representing the distribution from past observations, and a current window which represents the most recent data. To detect drift the distributions in the two windows are compared using the Kullback-Leibler divergence (KLD) to evaluate the distance between the two distributions. They use histograms as an estimator of the current most relevant distribution through the use of the Partition

Incremental Discretization algorithm [11] designed for high speed data streams.

First the data from the two windows are summarized using the Partition Incremental Discretization algorithm. Let p represent the probability distribution of the reference window, and q represent the distribution of the current window. The condition for detecting drift is

$$|KLD(p||q) - KLD(q||p)| > \lambda, \quad \text{where } \lambda \text{ is the drift threshold} \tag{2.21}$$

The Kullback-Leibler divergence is computed by

$$KLD(p||q) = \sum_i p(i) log_2 p(i)/q(i), \tag{2.22}$$

where $p(i)$ and $q(i)$ are the empirical probabilities. The Kullback-Leibler divergence is asymmetric ($KLD(p||q) \neq KLD(q||p)$), and is equal to zero when $p(i) = q(i)$. A small difference between $KLD(p||q)$ and $KLD(q||p)$ indicates that the distributions p and q are similar. The authors suggest using a drift threshold λ of 1% for the 99th percentile, or 5% for the 95th percentile [34]. When no change occurs the reference distribution is updated using the new data points, otherwise when a change is detected an alarm is raised and the reference data is cleared.

This method has been shown to have high detection delays compared to DDM, ADWIN, and PHT when tested on synthetic datasets with abrupt drift as well as gradual drift [33]. Compared to sequential analysis, and statistical process control techniques it requires more memory and time for processing data, but it offers more detail about the distribution of the concept which may be better for visualization. Furthermore the granularity of the histogram summaries can also be controlled by a parameter that acts as a constraint on the relative error (ϵ - the upper bound for the relative error) [34].

2.2.12 EWMA

The Exponentially Weighted Moving Average algorithm by [31] is a statistical process control method that uses an exponentially weighted moving average chart to monitor the classification error rate. An exponentially moving average chart is a control chart for determining when a process is in control. It is more suitable for detecting small changes in the process [23] and the authors aim to use this for detecting abrupt drifts. Suppose the mean is μ_0 before the change point and μ_1 afterwards. Let μ_t be the mean at time t. This is estimated by down-weighting older data

$$Z_0 = \mu_0, \quad Z_t = (1 - \lambda) Z_{t-1} + \lambda e_t, \quad t > 0, \tag{2.23}$$

where e_t is the error at the current example. The mean of Z_t is μz_t and the standard deviation is σz_t. The trigger condition is

$$Z_t > \mu_0 + L\sigma z_t, \tag{2.24}$$

where L is the control limit. This advantages of this approach are that it only requires a single pass through the data, has low computational overhead $O(1)$ per instance, and allows the rate of false positives to be controlled [31]. Experiments on synthetic data in the original paper [31] show its performance is comparable to the Paired Learners approach for streams with abrupt drifts in terms of accuracy, but is outperformed by the Paired Learners algorithm for streams with gradual drifts.

2.2.13 OnePassSampler

In [32], Sakthithasan et al. develop an efficient one pass algorithm for change detection. They use the Bernstein inequality as the bound for their test statistic. The main advantages of their algorithm is that it is single pass, has low false positive rates that are comparable to ADWIN2, and has lower computational overhead compared to other algorithms that use windowing approaches [32]. However it was shown to have much higher detection delay times than ADWIN2 [32]. This difference in delay times appears to be more prominent for data with more gradual changes.

2.2.14 SEED

In [17] Huang et al. propose a drift detection algorithm, SEED, which builds on the ideas used by ADWIN2 [4], but uses blocking and data compression techniques to improve the runtime and memory usage. Compared to ADWIN2, this detector is comparable in true positive rates but has lower computational overhead and false positive rates.

SEED reads data in as blocks of size b, and uses the block boundaries as potential drift points. It keeps a sliding window of t blocks $W = (B_1, B_2, \ldots, B_t)$, where B_i represents a block. Like ADWIN2, it attempts to find two sub-windows $W_L W_R$ of W that have distinct averages. They assume this corresponds to the two sub-windows having different expected values suggesting a concept drift. When a drift is detected, the left sub-window is dropped from the sliding window. Let μ_1 be the population mean of the left sub-window W_L, and μ_2 be the population mean of the right sub-window W_R. The test statistic ϵ_{cut} used for hypothesis testing ($H_1 : \mu_1 \neq \mu_2$) is computed using the Hoeffding inequality with Bonferroni correction 2.18 from [4].

A block compression scheme is used to compress homogeneous blocks to reduce the number of potential drift points and the amount of memory used [17]. This test is per-

formed by analysing the difference in means between the two boundary blocks μB_t and μB_{t+1}. If $|\mu B_t - \mu B_{t+1}| < \epsilon'$ the two blocks are said to have the same underlying distribution and can be merged into a single block. The value of ϵ' is computed using a linear function $\epsilon'_t = \hat{\epsilon} \,.\, \alpha \,.\, t$, where t is the relative arrival time, $\hat{\epsilon}$ is the base value (typically a small number) and $\alpha \in (0, 1)$ is a growth parameter controlling the magnitude of linear increments.

Results from experiments using synthetic data generated from Bernoulli distributions with varying levels of gradual drift and no drift showed that the SEED detector achieves comparable performance to ADWIN2 in terms of false positives and detection delay. The algorithm always has faster execution times than ADWIN2, and requires less memory than ADWIN2 in most cases [17].

Detectors based on monitoring two distributions may be more precise at locating the point of change, but have an increased memory and processing time cost compared to statistical process control or sequential analysis methods. In addition to this they are often not single pass algorithms or suffer from an increase in detection delay. Methods based on control charts are memoryless and efficiently process instances with O(1) overhead but either have trouble dealing with gradual drifts or controlling the false positive rate. Sequential analysis approaches are also memoryless and fast with low O(1) processing overhead but the main limitation is the difficulty in controlling the false positive rate whilst achieving low detection delay. Most detectors are able to identify abrupt drifts, but few detectors cope well with gradual concept drifts. Thus slow changing concepts present a challenging problem for change detectors.

Many drift detectors [10] [4] rely on changes in the classification error rate as an indication of concept drift and can not directly handle multidimensional data. Though Kuncheva points out that these univariate change detection methods are often statistically sound [25]. A large proportion of current detectors are purely reactive in nature, and do not have inbuilt capabilities for future drift prediction. The work of Yang et al. [40] was the first to introduce the idea of predicting future concepts. This is an important direction for future studies as it may facilitate better decision making and prevention strategies in some applications. The performance of detection techniques depends on a number of factors: (1) the choice of parameters, (2) the structure of the algorithm, (3) the characteristics of the data, and (4) the assumptions about the future including expected patterns of change. A comparative study by Gonçalves et al. showed that different detectors work well in different situations [21].

Gao et al. [13] criticise current work for having assumptions that are often too strong. They challenge the shared distribution assumption which states that the testing and training data comes from the same distribution. Although detectors have been developed

to react to change, the core assumption that the most recent data is closest to future data still holds so the delay is inevitable. Many techniques assume that there is no temporal dependence in the data. While this may be true in some situations, it is likely that there are some dependencies on previous examples or events in streaming data [42]. There are few studies that take temporal dependence into account. Current drift detection methods are focused on the two class classification problem, and it is unclear how they would perform under the multi-class setting.

2.3 Reoccurring Concepts

Reoccurring concepts relate to models that reoccur over time. For example, suppose we are interested in modelling seasons. The models would represent the different seasons, such as spring, summer, autumn and winter which reoccur annually. Research on reoccurring concepts aims to build a history of previously seen concept models, and reuse these historical models when similar concepts reoccur in the future. This is different to the characteristic of changes which we will discuss in the next section.

Many proposed algorithms such as [39] and [40] have capabilities for handling reoccurring concepts. Usually ensembles or meta-learning strategies are applied. FLORA3 [39] is the first adaptive learner to address this. In [15] Gonçalves and Barros present a framework (RCD) for detecting reoccurring concepts by creating a collection of classifiers. Each classifier has an associated buffer with examples that were used during training. The current concept is monitored using a classifier and drift detector with a warning and alarm state. When the detector enters the warning state a new classifier with a buffer is created and updated. If the error rate of the new classifier surpasses the warning threshold a statistical test (e.g. KNN in [15]) is performed to compare data in the new buffer and all stored buffers to determine whether the concept is an old or new concept. This approach can deal with both categorical and numerical data and showed improvements in accuracy compared to single classifier approaches [15]. In [14] the Cramer test is proposed as the test statistic for the RCD framework and showed it is comparable to using the KNN test. Gama and Kosina [9] propose a two layer meta-learning approach that uses a base classifier and referee classifier. The base learner makes predictions as soon as examples arrive, and the referee learns the feature space that the base learner accurately predicts once class labels are available [9]. It chooses the model from a pool of models with referees by using the referee's prediction about the base learner expressed as a confidence score, given the score is above a user defined threshold. Otherwise the current model is updated until drift is detected [9]. The main advantage of [9] is that it is able to select similar concepts from its history regardless if true class labels are available in the examples. Work by [37] uses the Discrete Fourier Transform to compress and capture decision tree classifiers

for identifying reoccurring concepts. It has been shown to be more robust to noise and outperforms [9]'s approach in terms of accuracy [37].

2.4 Characterization of Concept Drift

In contrast to reoccurring concepts research which relates to models that reoccur, research on the characterization of concept drift aims to study the nature of the changes. There are currently two streams of research on the characteristics of concept change. The first relates to the magnitude of change which is referred to as the drift severity. The second relates to the rate of concept change which is referred to as drift volatility. In this section we will review the formal definitions of these characteristics and discuss how these have been applied to drift detection methods.

2.4.1 Drift Severity

Kosina et al. [24] define two measures which they call drift rate, and drift severity that can be used to describe how much change a concept undergoes after a drift. The main contributions of their work are that they provided a simple metric capable of describing the amount of change between two consecutive concepts during concept drift that is compatible with any detector that uses two thresholds for control (e.g. warning and alarm levels). They use the definitions of drift rate from [3] and drift severity from [27].

Drift Rate: The drift rate is the probability that two consecutive concepts give different labels to a random example. Let f_t be the target function of the concept at time t, and assigns a label to the example $x_t \in X : f_t = X \rightarrow \{0, 1\}$. In this notation, the drift rate can be written as $Prob(f_t \neq f_{t+1})$.

Drift Severity: The severity of drift is the proportion of examples in the input space that have their labels changed after a concept drift occurs. Suppose C_i and C_{i+1} are two consecutive concepts, and are from the same input space S. The severity of drift is the percentage of S that has its class label changed after concept drift.

They designed their metric to make use of detectors with a warning and alarm state. A detector enters a warning state when the warning threshold is surpassed which suggests there may be a drift in the future. Once the alarm threshold is reached, the detector enters the alarm state signalling that a drift has occurred. As the warning state suggests that a change is occurring they keep counters of the number of examples processed and the number of misclassified examples during the warning period. When the alarm state is reached the severity metric is computed as

$$\frac{M_w}{T_w}, \tag{2.25}$$

where M_w is the number of misclassified examples during the warning period, and T_w is the total number of examples observed in the warning period. If the detector does not reach the alarm state after being in the warning state, then the metric is not computed and it is treated as a false alarm [24].

Application of Drift Severity: The drift severity metric was applied to two different detectors DDM [10] which has two thresholds, and a modified version of PHT. To compute the drift severity metric the PHT detector [30] was modified to include two thresholds λ_w for warning, and λ_d for alarm. PHT was coupled with a Naive Bayes learner, and DDM was combined with two different learners Naive Bayes, and Hoeffding Tree. Nine synthetic datasets (including SEA, LED, STAGGER) from related literature were used. Both gradual and abrupt drifts were examined, and the synthetic data was constructed such that the true drift points were known. To show the validity of their severity metric it was compared with a different metric that measures the difference between concepts. We will refer to the severity metric as SM the alternate metric as SM'. Their results showed a high correlation between the alternate metric SM' and their proposed metric SM [24]. This is in agreement with their expectations as both metrics measure difference between concepts. However this has not been evaluated on real datasets so it would be useful to include this in future studies to show the relevance to real world data.

Monitoring drift severity could reveal important information about the size of changes in streaming data [24], but has received relatively little attention in concept drift research. The authors also suggest that their proposed metric could also be used for comparing the behaviour of algorithms on different datasets [24].

2.4.2 Drift Volatility

Huang et al. [17] propose a metric called drift volatility which describes the rate of concept change. They develop a volatility detector for detecting changes in drift rate that is fast and accurate [17]. The volatility detector is an extra layer on top of drift detection that uses the drift detector's output for finding changes in volatility. Definitions for the drift volatility metric, and volatility shift from [17] follow below.

Drift Volatility: The volatility of a stream describes the rate of concept change. This is defined as the length between consecutive change points which are identified by a drift detector [17].

Volatility Shift: A volatility shift is where there is a change in the volatility of the stream. Let $C_1 = (c_1, \ldots, c_k)$ and $C_2 = (c_{k+1}, \ldots, c_t)$ be a sample of drift points from the stream. Let p_i represent the interval between consecutive drift points c_i and c_{i+1}. Suppose two volatility windows $P_1 = (p_1, \ldots, p_k)$ and $P_{k+1} = (p_{k+1}, \ldots, p_{t-1})$ have sample variances of σ_1 and σ_2 respectively. We say there is a shift in volatility when $\frac{\sigma_2}{\sigma_1} \lessgtr 1.0 \pm \beta$, where β is a user defined threshold [17].

The volatility detector takes a sequence of real numbers $p_1, p_2, \ldots, p_t$ representing the interval lengths between drift points found by the drift detector as input. It uses a buffer and a reservoir. The buffer is a sliding window that keeps the most recent intervals, and the reservoir stores samples that represent the overall stream. As the window slides along the oldest interval is removed from the buffer and replaces a random sample in the reservoir. To detect for shifts in volatility they compare the variance in the buffer and reservoir using the relative variance measure $\frac{\sigma_B}{\sigma_R}$, where σ_B is the variance of the buffer and σ_R is the variance in the reservoir. Let $\beta \in [0, 1]$ be a user defined tolerance threshold. If $\frac{\sigma_B}{\sigma_R} \lessgtr 1.0 \pm \beta$, then the volatility detector concludes there is a shift in volatility. Setting a low β value will allow small changes in volatility to be detected, whereas a high β value will be less sensitive and more suitable for detecting large changes in volatility [17].

Analysis of true and false positives were performed using synthetic Bernoulli streams where the volatility levels and shifts were controlled. They demonstrated that combining the volatility detector with SEED has low false positive rates and gives lower delays in volatility detection than in the case where it is combined with ADWIN2 [17]. Applying their technique to real datasets (Sensor Stream, Forest Covertype, and Poker Hand) they were able to show that volatility shifts are present in real world data streams [17]. From the plots in [17] it can be observed that the drift volatility and its patterns of change differs from stream to stream. They propose that information about the volatility of the stream can be useful in predicting future change points and may enable users to be proactive rather than reactive [17]. Furthermore this may also facilitate the development of a faster reacting drift detector [17].

Application of Drift Volatility: To show the significance of drift volatility, the authors apply their idea of drift volatility to improve the performance of drift detectors. In [16], Huang et al. apply this to ADWIN2 by adaptively adjust the drift bound according to the probability of drift. They compute the probability of drift using the average volatility of the stream. The novelty of their approach is the use of the historical drift rate to adaptively adjust the drift bound.

2.5 Datasets

In this section we describe some synthetic and real datasets that are commonly used as benchmarks for drift detection and adaptive learning algorithms.

2.5.1 Synthetic Datasets

Synthetic datasets allow the characteristics of change such as the speed of drift and severity of drift to be controlled. This enables researchers to analyze the performance of

their algorithms on different types of concept change. We describe three types of synthetic datasets: Bernoulli [33], SEA Concepts [24], and CIRCLES [10].

Bernoulli

A Bernoulli generator [33] generates binary streams that are used to simulate the error rate of a classifier. Each instance represents the classification outcome where a 1 represents a classification error, and a 0 represents a correct classification. It assumes that a classifier generates errors with a mean probability of P, and correct classifications with a probability $Q = 1 - P$. This is modelled as a Bernoulli trial with mean P, where the probability of success corresponds to the classification error rate. The probability mass function for a Bernoulli distribution is

$$P(X = x) = \begin{cases} Q & \text{if } x = 0 \\ P & \text{if } x = 1 \end{cases} \tag{2.26}$$

Abrupt drifts can be simulated by changing the mean, P, of the Bernoulli generator. Gradual drifts can be simulated by gradually increasing the mean P by a slope of $s \in (0, 1)$ in the last T time steps. Usually T is set to 1000 time steps and the slope value is chosen between $0.0001 - 0.0004$ [33].

SEA Concepts

SEA Concepts [24] are data streams with instances that have attribute and value pairs, and are often used to represent a binary classification problem. Each instance has three numeric attributes f_1, f_2, and f_3 with values between 0 and 10, and a class label $y \in \{0, 1\}$. Only the last two attributes are relevant to the class of the instance. An instance has label class 1 if $f_2 + f_3 \leq \theta$, and class 0 if $f_2 + f_3 > \theta$, where θ is a threshold parameter. This divides the class of the instances linearly into two sections. Concept drifts are simulated by changing the threshold parameter θ.

CIRCLES

CIRCLES streams [10] have concepts that are represented by a circle. Each instance has a class label and two numeric attributes x and y that can take values between 0 and 1 which represents its position on a two dimensional space. The concept is defined by a circle centered at (c_x, c_y) with a radius of r. Instances that fall within the area inside of the circle are labelled as class 0, and instances outside of the circle are labelled as class 1. This can be used to represent a binary classification problem that is more difficult to learn than SEA Concept streams. Concept drifts are simulated by changing the attributes associated with the circle concept such as the center position (c_x, c_y) or the radius r.

2.5.2 Real Datasets

Real datasets are often more complex than synthetic datasets, and can be used to explore the applicability of algorithms in real scenarios.

Forest Covertype

The Forest Covertype dataset [6] has 581,012 instances, 54 attributes and a class label that represents the forest cover type of a 30 x 30m area. The data was obtained from the US Forest Service (USFS) Region 2 Resource Information System (RIS). Each instance contains information such as the elevation, slope and soil type of the area. The classification task is to predict the type of forest cover given cartographic information.

Pokerhand

The Pokerhand dataset [6] has 1,000,000 instances, 11 attributes and a class label which represents the "Poker hand". Each instance represents a hand containing 5 cards and has information such as the rank and suit of the cards. The classification task is to predict the type of hand a player has.

Airlines

The Airlines dataset [19] has 539,383 instances, 7 attributes and a class label that represents whether a flight is delayed. Each instance represents a single flight with information such as the day, month and destination. The data was collected from arrival and departure flights in the US between October 1987 and April 2008. The classification task is to predict whether a flight will be delayed.

3

Tracking Drift Severity

In the previous chapter we reviewed the main concepts and discussed related work in the area. In this chapter we introduce a method for detecting the magnitude of concept changes. This chapter is organised as follows. Section 3.1 presents the motivation of our work. Section 3.2 reviews some terminology relevant to this chapter, and highlights the design objectives. We introduce our algorithm in Section 3.3 and discuss the details of our technique in Section 3.4. In Section 3.5 we describe possible alternative approaches. Section 3.6 presents our experimental results, and Section 3.7 concludes this chapter.

3.1 Introduction

Mining unbounded high speed data streams is very challenging for a number of reasons. Firstly the high speed and volume of data makes it infeasible to store all the data in memory which poses significant memory and time constraints on the algorithms. The speed of data arrival limits the number of passes through the input data, and makes traditional multipass algorithms unsuitable for the streaming environment. Secondly stream mining algorithms should be anytime algorithms where predictions can be made at any point in time.

One unique property of data streams is the possibility of changes in the underlying model over time. This is referred to as concept drift. In particular, we are interested in characterising how large these changes are which we refer to as drift severity. The main

advantage of detecting drift severity is it gives an indication of the size of concept change which allows responses to be better adapted to the current situation.

For example, change mining techniques may be applied to data from a range of sources to identify the onset of viral outbreaks in a population. Such monitoring could be beneficial to public health by providing an indication of when episodes occur and how severe an outbreak is likely to be. This allows preventative and responsive measures to be taken. For example, stocking medicine or vaccinating high risk groups. Thus it is important to be able to detect and react to these changes in a timely manner.

Most existing work in drift detection focuses on accurately and efficiently finding true drift points whilst minimising the delay time of detection. However they are unable to detect drift severity. There has been relatively little work that addresses the problem of drift severity, and how to accurately measure the magnitude of concept drift for streams with different characteristics. One of the challenges in measuring drift severity is that it is difficult to define a measure to directly compare classifiers before and after change. Recently [24] introduced a metric for measuring drift severity and presented a technique for detectors that use statistical process control. It has been shown to be effective at tracking changes of large magnitudes, and at high speed but performs poorly for more gradual and less severe changes [24]. We will attempt to address these issues by developing a drift detector with the capability for severity detection and a greater sensitivity to smaller changes. Our main focus will be on streams with gradual change. In this chapter, we discuss our proposal of a new drift detector, MAGSEED that is able to accurately capture drift severity.

3.2 Preliminaries

Let us revisit some definitions from Section 2. Recall that concept drift refers to a change in the concept function that generates the data. Suppose we are interested in classifying the type of flu a patient has based on their symptoms. For simplicity lets assume that our data has two attributes that represent symptoms and a class label that represents the strand of flu. We show an example of this data in Table 3.1, where each row represents the diagnosis of a patient. The attributes “Fever” and “Headache” can take two values “Yes” which indicates the presence of the symptom, and “No” which indicates the absence of the symptom in a patient.

Definition 1 (Concept Drift) *A concept drift is a change in the concept function responsible for the generation of data, where a concept function maps input to class labels.*

Let Figure 3.1 represent a concept function that maps patient symptoms to their flu type. The nodes on the left represent the possible symptoms a patient could have in the

Table 3.1: Example of patient data

	Fever	Headache	Type of Flu
Patient 1	No	Yes	Flu B
Patient 2	Yes	Yes	Flu B
Patient 3	Yes	No	Flu A
Patient 4	Yes	No	Flu A
Patient 5	No	No	Non Flu

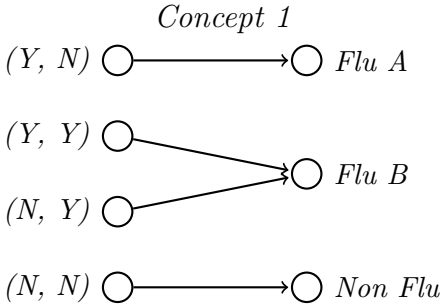


Figure 3.1: Example of a concept function

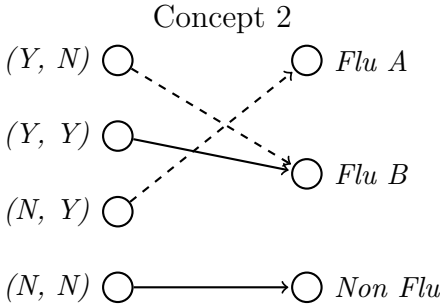


Figure 3.2: Example of a different concept

form of a vector. This is the input space. The first element of the vector corresponds to the "Fever" symptom, and the second corresponds to the "Headache" symptom. For example the vector *(Y, N)* indicates the presence of a fever and the absence of a headache. The nodes on the right are the possible flu class labels, *Flu A*, *Flu B* and *Non Flu*. The arrows represent a mapping from the attribute values to the class labels. We refer to Figure 3.1 as a concept. In this concept, a patient with only a fever will be diagnosed with *Flu A*.

A concept drift is a change in concepts, for example from *Concept 1* in Figure 3.1 to *Concept 2* in Figure 3.2. The difference between the two concepts is illustrated by the dashed arrows. In *Concept 1* a patient with only a fever would be classified as having *Flu A*, but in *Concept 2* this patient would be classified as having *Flu B*. The drift severity or magnitude is the size of the concept change. In this example, the drift severity is the percentage of the possible symptoms that have a different flu type after the change which is $2/4 = 50\%$. We give the formal definition of drift severity in Definition 2.

Definition 2 (Drift Severity) *The drift severity or drift magnitude is the size of a concept change. Given the concept function changes from f_t to f_{t+1}, the drift severity is the percentage of the input space that is mapped to different class labels in f_t and f_{t+1}.*

Drift detectors use drift bounds for finding out when a concept change has occurred. These drift bounds are often formalized as statistical tests which test for significant increases in the error rate of a model. This can be illustrated by the dashed line in Figure 3.3. When this bound is surpassed the detector signals a drift indicating the possibility of a real change. We refer to the first drift detected after a real change as a true positive. In

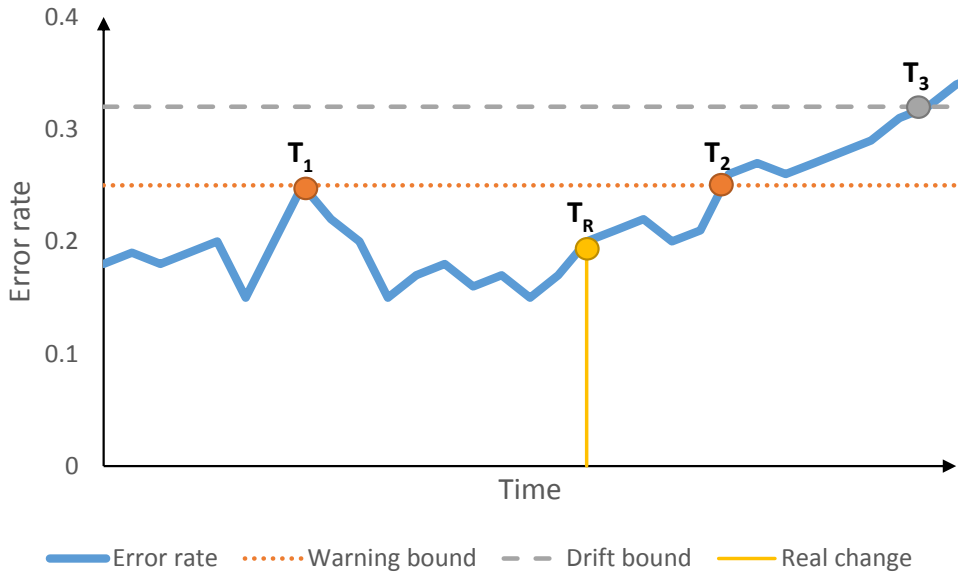


Figure 3.3: Example of warning detection and drift detection

our example, a true positive is detected at time T_3. A drift that is detected after T_3 or before the true change at T_R is referred to as a false positive because there is no change occurring at these points. The detection delay is the number of time steps between the true change and the detection of a true positive drift. In Figure 3.3, the delay is $T_3 - T_R$, where T_R is the time of the true change.

Definition 3 (True positive) *A true positive is the first drift detected by the detector that occurs after the true drift point.*

Definition 4 (False positive) *A false positive is the detection of a drift before the true drift point, or after a true positive when there is no concept change.*

Definition 5 (Detection Delay) *The detection delay is the number of time steps after the true drift point that a drift was detected.*

Only using a drift bound does not give us enough information to compute the severity of change because it only tells us that a change is likely to have occurred. We could use the error rate to estimate the drift severity, but in order to do this we need to define a time period to compute this over. Using a time period that is too large may include the non changing period which would have a dampening effect, whereas using a period that is too small could decrease the accuracy of our metric because of the small sample size. For example, suppose we observe the error rate of our model to be

$$0001 \quad 1010 \quad 0011 \quad 0100 \quad 1110,$$

where a 1 represents a classification error, and 0 represents no error. We have grouped the error rate values for readability. Suppose we know the true change occurs after making ten observations and the real drift severity is 60%. If we use all the observations to compute the drift severity, we get a computed severity of $9/20 = 45\%$. If we use a smaller period, such as the last six observations the computed severity becomes $3/6 = 50\%$.

If we are able to identify a time period where the concept is changing, we can use the percentage of mistakes made by our current model in this period to estimate the drift severity [24]. This can be achieved by introducing a warning bound that aims to identify the start of a change period. A warning bound should be more generous than a drift bound. In Figure 3.3 we show an example of a drift bound and a warning bound. The first warning detected at time T_1 occurred too early, before the true change so we refer to this as a false warning. The second warning at time T_2 is detected after the real change so we refer to this as a true warning. If we did not detect a warning prior to a drift, this means we have missed a warning.

Definition 6 (True warning) *A true warning is a warning that was triggered between the true drift point and true detection by the detector.*

Definition 7 (False warning) *A false warning is a warning that was triggered when there is no concept change.*

Definition 8 (Missed warning) *A missed warning is the scenario where no warning was triggered prior to the detection of a drift.*

Design Objectives

A crucial aspect of our algorithm design is the choice of the warning and drift bounds. This affects the algorithm's ability to detect real changes as well as its robustness to noise. Our aim is to develop a detector with the following properties:

- A high true positive drift detection rate
- A low false positive drift detection rate
- A high true warning detection rate
- A low false warning rate
- A low missed warning rate

3.3 Overview

In this section we present an overview of our proposed drift detector, MAGSEED. Our detector extends the SEED algorithm [17] which uses classification error as input data. The novelty of our approach is that it allows the detection of drift severity. Our algorithm has two parts: drift detection, and magnitude tracking. It has three possible states: (1) no change - indicating there is no concept drift, (2) warning - in anticipation of a drift, and (3) drift - signalling a concept drift has occurred.

Example: Suppose we observe a binary stream of classification errors, where 0 represents correct prediction, and 1 represents error:

$$01101010011101110101001001111111$$

Our detector groups data into blocks of size b. Using a block size of 4, the data would be grouped into blocks of size 4. Let the block boundaries be represented as spaces, and t_i represent time step i, the stream becomes:

$$\begin{array}{cccccccc} 0110 & 1010 & 0111 & 0111 & 0101 & 0010 & 0111 & 1111 \\ t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \end{array}$$

Initially the detector is in the "No change" state, and arriving homogeneous blocks are compressed. At time t_2 the first two observed blocks are compressed, and the stream becomes: 01101010. As a new instance arrives it is grouped into a new block. Suppose at time t_3, the small increase in the error rate causes a warning to be triggered but does not trigger a drift. At this point, the detector would enter the "Warning" state. While the detector is in the "Warning" state, homogeneous data blocks are not compressed. At time t_4 the stream becomes:

$$01101010 \quad 0111 \quad 0111$$

Suppose after t_5, the error rate decreases significantly, the detector would flag the previous warning as a false warning, and enter the "No change" state. Homogeneous blocks are then merged to reduce the number of block boundaries. At time t_5, the stream becomes:

$$01101010 \quad 01110111 \quad 0101,$$

and after time step t_6 this becomes:

$$01101010 \quad 01110111 \quad 0101 \quad 0010$$

Suppose the detector enters the "Warning" state after t_6, and the "Drift" state after t_8. The severity can be computed as the percentage of errors between the warning and the

drift, this is $7/8 = 87.5\%$. The significant increase in error at t_8 causes our detector to signal a drift due to a significant difference in the means between the block boundary indicated by a vertical bar. The detector would remove all blocks to the left of the boundary, and enter the "Drift" state.

01101010 01110111 0101 0010 | 0111 1111

3.4 Drift Magnitude Detection

In this Section we discuss the details of our algorithm. Our method uses binary classification error data as input, and has two thresholds for detecting change: (1) a warning threshold which is signalled in anticipation of concept drift, and (2) a drift threshold which indicates the occurrence of a concept drift. We will describe these components separately.

3.4.1 Drift Detection

First we describe the drift detection part of our algorithm. As input data arrives, it is partitioned into blocks of size b where the block boundaries represent potential drift points. Drift detection is performed by testing for a significant difference in means between data on the left W_L and right W_R sides of the block boundaries based on the Hoeffding bound with Bonferroni correction. The condition for triggering a drift is

$$|\hat{\mu}_{W_L} - \hat{\mu}_{W_R}| > \epsilon_d, \tag{3.1}$$

$$\epsilon_d = \sqrt{\frac{2}{m}.\sigma^2_W.ln\frac{2}{\delta'}} + \frac{2}{3m}ln\frac{2}{\delta'}, \quad \delta' = \frac{\delta_d}{n} \tag{3.2}$$

where $\hat{\mu}_{W_L}$ represents the mean error rate of data W_L to the left of the block boundary, and $\hat{\mu_{W_R}}$ represents the mean error rate of the data W_R to the right of the block boundary. ϵ_d is the Hoeffding bound with Bonferroni correction using a confidence parameter $\delta_d \in (0, 1)$, m is the harmonic mean of the lengths of W_L and W_R, and n is the length of W where $W = W_L + W_R$. When the drift threshold is surpassed, the detector enters a drift state.

When the detector is not in a warning or drift state, block compression is enabled to improve efficiency by removing potential drift points that have low probability of becoming actual drift points. We use the block compression algorithm detailed in [17] which uses the bound $\epsilon' = \hat{\epsilon} \cdot \alpha \cdot t$, where the parameters $\hat{\epsilon}$ is a base value, α is the linear growth term, and t is the relative arrival position.

3.4.2 Warning Detection

To allow the quantification of the magnitude of change, as well as the anticipation of concept drift we introduced a more relaxed threshold for detecting warnings. Our first warning threshold is similar to the drift detection trigger but uses the bound ϵ_w, where $\epsilon_w < \epsilon_d$. We use the condition below for warning detection

$$|\hat{\mu}_{W_L} - \hat{\mu}_{W_R}| > \epsilon_w, \tag{3.3}$$

where ϵ_w is the Hoeffding bound with Bonferroni correction using a confidence parameter $\delta_w \in (0, 1)$, where $\delta_w > \delta_d$. By using only the first warning condition, it may cause warnings to be triggered too early. For example if the error rate increases and passes the ϵ_w threshold, followed by a decrease that falls below the ϵ_w threshold. This introduces a false warning. To address this, we included a second warning threshold. We monitor the current mean p and standard deviation s of the classification error over a sliding window of 100 instances, as well as the minimum p and s values denoted by p_{min} and s_{min}. Our second warning threshold is defined as

$$p + s > p_{min} + c_w * s_{min}, \tag{3.4}$$

$$Rate = \frac{\text{number of misclassifications in warning}}{\text{total instances seen in warning}} \tag{3.5}$$

In Equation 3.4 above, $c_w \in (1, 3)$ is a user defined parameter which defines the confidence level for warning detection. When either Equation 3.3 or Equation 3.4 is satisfied the detector enters a warning state. We also use them to control the rate of false warnings. When either Equation 3.4 or Equation 3.3 is not satisfied for all block boundaries then the warning is flagged as a false warning, and the detector enters the no change state. For computing drift severity we use the metric defined by [24] which we denote as *Rate*, that computes the error rate in the warning window.

Magnitude Detection Algorithm: Algorithm 1 shows the pseudocode for the MAGSEED algorithm. Lines 1-3 initializes the change detector. The algorithm (lines 4-7) processes the input as each instance arrives by passing the classification output $x_t \in \{1, 0\}$ at time t to the *SetInput* function, and returns the state of the detector indicating whether a drift or warning has occurred at each time step t (line 6). The *SetInput* function (lines 8-32) contains the main algorithm used for change detection. Line 9 adds the classification output k to the window W of data blocks by the *AddElement* function. Line 10 updates the current p and s values by computing a rolling average of the classification error, and standard deviation of the error respectively, and the minimums

Algorithm 1 MAGSEED Algorithm

Initialize window W as blocks $\{B_0, ..., B_t\}$ each of size n
Initialize $state \leftarrow no\ change$
Initialize $p_{min \leftarrow \infty}$, $s_{min \leftarrow \infty}$

for $t > 0$ **do** where t is time
    SetInput(x_t, W) where x_t is classification error at t
    return *state*
end for

function SetInput(item k, List W)
    AddElement(k, W) adds element into tail block
    update p, s, p_{min}, s_{min}
    $warningFound \leftarrow false$
    if $p + s > p_{min} + c_w * s_{min}$ **then**
        $state \leftarrow warning$
    else $state \leftarrow no\ change$
    end if
    for every split of W into W = W_L, W_R **do**
        if $|\mu_{W_L} - \mu_{W_R}| > \epsilon_d$ **then**
            $state \leftarrow drift$
            $p_{min \leftarrow \infty}$
            $s_{min \leftarrow \infty}$
            remove all blocks in W_L
        else if $|\mu_{W_L} - \mu_{W_R}| > \epsilon_w$ **then**
            $state \leftarrow warning$
            $warningFound \leftarrow true$
        end if
    end for
    if $warningFound = false$ **then** $state \leftarrow no\ change$
    end if
    if $state = no\ change$ **then**
        CompressionCheck(W) compresses blocks in W
    end if
end function

function AddElement(item k, List W)
    if Tail Block of W is full **then** create Block B with k
        $W \leftarrow W \cup \{B\}$
    else add k to tail block of W
    end if
end function

function CompressionCheck(List W)
    compressCount++
    if compressCount = compressionInterval **then**
        for each two consecutive blocks B_t, B_{t+1} **do**
            if $|\mu B_t - \mu B_{t+1}| < \epsilon'$ **then** merge B_t, B_{t+1}
            end if
        end for
    end if
end function

p_{min}, s_{min} are updated when $p + s < p_{min} + s_{min}$. If the warning threshold (Equation 3.4) is surpassed (line 12), the detector enters the warning state (line 13). Line 16 checks every block boundary in W against the drift threshold (line 17), and the warning threshold (Equation 3.3) in line 22 which also acts as a false warning threshold. Lastly, if the detector is in the *no change* state after the data is processed, the blocks may be compressed by the *CompressionCheck* function (line 30). The *AddElement* function (lines 33-38) adds an element k that represents the classification error to the window W by appending k to the last block in W if the last block is not full (line 36), or by adding a new block with element k to the end of the window W (line 35). The *CompressionCheck* function (lines 39-47) merges consecutive homogeneous blocks in the window at set intervals using the ϵ' bound which is detailed above (line 43).

3.5 Alternative approaches

Although we only present one approach for computing drift magnitude in this chapter we recognize that there are many possible approaches. For example, we can use different statistical bounds for drift or warning detection such as the Kolmogorov-Smirnov test or Anderson-Darling test. As these statistical tests have different properties, applying these tests may cause warnings and drifts to be found at different locations.

Currently we have used the drift severity metric proposed by [24] for magnitude computation. The drift severity metric assumes that all errors that occur in the warning period contribute to the change in concepts. However this ignores the errors introduced by noisy data, overfitting and limitations of the model. A more direct approach would be to compare the errors that are produced by the new model with the old model after the new model has stabilized by computing the difference in error rates of the two models over a set of instances.

3.6 Experiments

Our evaluation has three parts. First we evaluate the effectiveness of drift detection. Second we evaluate the effectiveness of the warning technique, and its capability for tracking severity. Third we evaluate the prediction accuracy of our algorithm on real data streams using a Hoeffding tree learner. All experiments were run on machines with an Intel Core i7-3770S CPU at 3.10 GHz, 16GB RAM running Windows 7 OS.

3.6.1 Synthetic Streams

We evaluate the algorithms on synthetic streams with a single concept drift over 100 trials. To test the robustness of the algorithms we use three different noise levels, 0%, 5% and 10%. We use three different types of synthetic streams: (1) Bernoulli [33], (2) SEA Concepts [24], and (3) CIRCLES [10] as discussed in Section 2.5.1.

Bernoulli Streams: We follow the approach used by [33] to generate error rate streams with gradual drift. We generate 1,000,000 instances per stream. During the first 999,000 instances data is generated according to a stationary Bernoulli distribution with a mean of $P = 0.2$. In the last 1,000 instances, gradual drift was simulated by increasing the mean error rate P by a slope value $s \in (0.0001, 0.0002, 0.0003, 0.0004)$.

SEA Concepts Streams: We follow the approach used by [24] to generate streams with abrupt drift. We generate 90,000 instances per stream with two balanced classes. We introduce a single point of abrupt drift by changing the threshold θ that controls the concept function at the midpoint of the stream. A larger difference in the thresholds between the two concepts corresponds to a higher level of drift severity. We present these threshold values in Table 3.2.

Table 3.2: Theta values for SEA Concepts streams

θ_1	6.0	9.0	8.5	8.5	8.0	6.5	7.0	9.5	9.5
θ_2	6.5	8.5	9.5	7.0	9.5	8.5	9.0	6.5	6.0
Severity	0.03	0.05	0.09	0.12	0.13	0.15	0.16	0.24	0.27

CIRCLES Streams: Our generator is based on the CIRCLES generator by [28]. We generate 1,000,000 instances per stream with two balanced classes. Our concept is defined by a circle centered at (0.5, 0.5) with a radius of 0.2. We introduce gradual drift by gradually increasing the probability of generating instances from the new concept in the last 1000 time steps. Our new concept is defined by a circle centered at (0.5, 0.5) with a radius $r \in (0.3, 0.4, 0.5)$. A larger difference in the radius of the circle corresponds to a higher level of drift severity.

3.6.2 Parameter Selection

We evaluate the algorithms over a range of parameters shown in Table 3.3, and present the best and worst performance for each detector in our results. For synthetic streams we select the best and worst settings based on the number of true positive drifts and the ratio of true and false positives. For real data streams we use the best settings obtained from the synthetic experiments.

Table 3.3: Parameter range for evaluation on synthetic data

Detector	Parameter	Values			Detector	Parameter	Values		
MAGSEED	δ_d	0.05	0.1	0.3	SEED	δ_d	0.05	0.1	0.3
	$\hat{\epsilon}$	0.0025	0.00625	0.01		$\hat{\epsilon}$	0.0025	0.00625	0.01
	α	0.2	0.5	0.8		α	0.2	0.5	0.8
	τ	50	75	100		τ	50	75	100
	δ_w	0.1	0.2	0.3					
	c_w	1	2						
DDM	α	2	1	1.5	ADWIN2	δ	0.05	0.1	0.3
	β	3	2	1					

3.6.3 Evaluation Metrics

We evaluate the algorithms using the rate of true positive drifts (RD), rate of false positive drifts (FP), detection delay, memory (in bytes), time (in milliseconds) used by the detector, computed severity (Rate), rate of true warnings (TW), and correlation between computed and actual severity. A high correlation would suggest that the computed severity reflects actual severity. A true warning is a warning that was triggered between the true point of drift and true detection by the detector as described in Section 3.2.

3.6.4 Drift Detection

In these experiments, we test the drift detection capabilities of our algorithm compared to current drift detection techniques: ADWIN2 [4], and DDM [10]. We compare our drift detector to ADWIN2 as it is a state-of-the-art detector that uses the same drift detection bound (the Hoeffding bound), and has been shown to be effective at detecting gradual drifts [33], which are the type of streams we will be focusing on. We also compare our detector with DDM, which is a technique that is effective at detecting abrupt drifts. DDM also has a warning threshold that would allow us to directly compare its performance in capturing the warning state against our technique. Thus it makes sense that we chose these two different detectors as our benchmark comparisons.

Tables 3.4 and 3.5 show the rate of true drift detection for Bernoulli streams with gradual concept drift. MAGSEED is comparable to ADWIN2 in terms of delay and true positive drift rate given the same confidence level δ. Both MAGSEED and ADWIN2 outperform DDM on gradual Bernoulli streams in terms of true drift detection rate, but require more memory as they monitor the error distribution. In all cases, the rate of false positive drifts in MAGSEED and ADWIN2 are below the theoretical upper bound for false positives δ. In terms of memory ADWIN2 outperforms MAGSEED as the latter requires additional memory to monitor drift warnings.

Table 3.4: Drift detection: Rate of true and false positives (Best case)

			Best				
Noise	Detector	Slope	RD	FP	Delay(SD)	Memory(SD)	Time(SD)
0%	MAGSEED	0.0001	**84**	0.001	759.00 ±(165.61)	2134.86 ±(388.56)	146.58 ±(5.77)
		0.0002	**100**	0.001	543.32 ±(118.79)	2197.92 ±(316.59)	147.68 ±(5.64)
		0.0003	**100**	0.001	433.24 ±(97.32)	1986.72 ±(328.89)	147.09 ±(5.09)
		0.0004	**100**	0.001	361.56 ±(77.94)	1848.96 ±(282.66)	148.32 ±(5.24)
	ADWIN2	0.0001	83	0.001	782.71 ±(162.45)	1770.31 ±(198.92)	353.65 ±(11.60)
		0.0002	100	0.001	555.80 ±(113.10)	1565.92 ±(52.42)	353.08 ±(12.51)
		0.0003	100	0.001	439.96 ±(94.04)	1520.56 ±(79.39)	352.63 ±(11.78)
		0.0004	100	0.001	370.20 ±(74.79)	1478.56 ±(89.92)	352.00 ±(11.41)
	DDM	0.0001	1	< 0.001	785.00 ±(0)	248.00 ±(0)	23.22 ±(15.10)
		0.0002	1	< 0.001	560.00 ±(0)	248.00 ±(0)	23.77 ±(14.31)
		0.0003	1	< 0.001	456.00 ±(0)	248.00 ±(0)	22.53 ±(14.43)
		0.0004	1	< 0.001	379.00 ±(0)	248.00 ±(0)	22.24 ±(14.78)
5%	MAGSEED	0.0001	**71**	0.001	796.63 ±(162.79)	2178.93 ±(406.77)	148.92 ±(5.96)
		0.0002	**100**	0.001	590.04 ±(124.94)	2225.76 ±(300.16)	147.21 ±(7.17)
		0.0003	**100**	0.001	465.24 ±(95.72)	2076.00 ±(303.99)	149.18 ±(6.45)
		0.0004	**100**	0.001	391.96 ±(86.35)	1902.24 ±(276.85)	148.74 ±(6.60)
	ADWIN2	0.0001	68	0.001	809.35 ±(157.96)	1847.76 ±(249.74)	355.12 ±(10.07)
		0.0002	100	0.001	604.12 ±(124.08)	1587.76 ±(72.64)	354.20 ±(10.08)
		0.0003	100	0.001	474.52 ±(97.75)	1550.80 ±(60.29)	356.15 ±(10.26)
		0.0004	100	0.001	401.56 ±(81.24)	1503.76 ±(86.94)	353.49 ±(11.65)
	DDM	0.0001	0	< 0.001	-	248.00 ±(0.00)	19.51 ±(14.23)
		0.0002	0	< 0.001	-	248.00 ±(0.00)	22.60 ±(15.70)
		0.0003	0	< 0.001	-	248.00 ±(0.00)	21.22 ±(12.53)
		0.0004	0	< 0.001	-	248.00 ±(0.00)	21.03 ±(15.71)
10%	MAGSEED	0.0001	**54**	0.001	852.63 ±(148.49)	2218.67 ±(431.11)	149.00 ±(5.78)
		0.0002	**100**	0.001	658.84 ±(133.12)	2237.28 ±(330.23)	150.14 ±(6.29)
		0.0003	**100**	0.001	517.72 ±(107.82)	2160.48 ±(284.63)	148.39 ±(6.15)
		0.0004	**100**	0.001	432.28 ±(87.17)	2044.80 ±(321.96)	150.85 ±(7.12)
	ADWIN2	0.0001	49	0.001	859.90 ±(148.31)	1918.86 ±(265.54)	353.80 ±(12.86)
		0.0002	100	0.001	669.40 ±(136.56)	1641.52 ±(101.01)	354.37 ±(11.16)
		0.0003	100	0.001	527.32 ±(109.60)	1559.20 ±(50.65)	355.76 ±(10.96)
		0.0004	100	0.001	437.72 ±(91.47)	1532.32 ±(74.06)	354.11 ±(12.80)
	DDM	0.0001	0	< 0.001	-	248.00 ±(0.00)	21.58 ±(15.66)
		0.0002	0	< 0.001	-	248.00 ±(0.00)	22.17 ±(16.17)
		0.0003	0	< 0.001	-	248.00 ±(0.00)	23.21 ±(15.35)
		0.0004	0	< 0.001	-	248.00 ±(0.00)	20.87 ±(15.25)

Best Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.1$, ADWIN2 $\delta = 0.05$, DDM $\alpha = 2$ $\beta = 3$

3.6.5 Warning Detection and Severity Measure

In these experiments, we use drift detectors combined with a Hoeffding tree learner to test the accuracy of warning detection, and our proposed severity measure. We compare our method to the DDM method presented by Kosina et al. [24], as their technique has warning detection and can also capture drift severity. We did not compare our method with the PHT method [24] as it was shown to perform worse in terms of severity tracking. We did not include ADWIN2 in our comparison as the nature of the exponential histogram data structure used in the detector makes it difficult to implement a warning threshold for computing severity.

Table 3.6 shows the rate of true warning detection on Bernoulli streams with the

Table 3.5: Drift detection: Rate of true and false positives (Worst case)

			Worst				
Noise	Detector	Slope	RD	FP	Delay(SD)	Memory(SD)	Time(SD)
0%	MAGSEED	0.0001	84	0.002	754.43 ±(165.77)	2220.00 ±(654.54)	862.82 ±(52.31)
		0.0002	100	0.002	539.48 ±(122.93)	2192.64 ±(315.08)	864.47 ±(50.79)
		0.0003	100	0.002	431.96 ±(100.28)	1977.12 ±(318.59)	864.57 ±(52.90)
		0.0004	100	0.002	359.96 ±(78.05)	1857.60 ±(285.12)	862.89 ±(52.66)
	ADWIN2	0.0001	**92**	0.011	671.70 ±(196.98)	1627.13 ±(133.52)	302.51 ±(8.58)
		0.0002	**100**	0.011	476.44 ±(140.49)	1512.16 ±(88.64)	303.96 ±(6.96)
		0.0003	**100**	0.011	372.12 ±(104.52)	1461.76 ±(98.11)	303.57 ±(6.16)
		0.0004	**100**	0.011	314.20 ±(82.87)	1414.72 ±(78.91)	301.33 ±(7.66)
	DDM	0.0001	1	< 0.001	785.00 ±(0)	248.00 ±(0)	23.41 ±(16.04)
		0.0002	1	< 0.001	560.00 ±(0)	248.00 ±(0)	23.79 ±(13.57)
		0.0003	1	< 0.001	456.00 ±(0)	248.00 ±(0)	23.70 ±(14.58)
		0.0004	1	< 0.001	379.00 ±(0)	248.00 ±(0)	24.37 ±(14.47)
5%	MAGSEED	0.0001	71	0.002	797.54 ±(164.19)	2242.48 ±(591.57)	901.61 ±(52.14)
		0.0002	100	0.002	586.52 ±(123.97)	2262.24 ±(367.95)	908.45 ±(58.19)
		0.0003	100	0.002	464.92 ±(96.13)	2081.28 ±(309.09)	908.51 ±(55.70)
		0.0004	100	0.002	391.00 ±(85.45)	1913.76 ±(276.51)	907.25 ±(57.86)
	ADWIN2	0.0001	**87**	0.011	684.89 ±(220.38)	1672.55 ±(141.80)	302.77 ±(7.42)
		0.0002	**100**	0.011	483.16 ±(151.55)	1539.04 ±(81.26)	301.54 ±(8.49)
		0.0003	**100**	0.011	395.48 ±(119.65)	1475.20 ±(92.48)	301.16 ±(7.28)
		0.0004	**100**	0.011	332.76 ±(98.43)	1424.80 ±(77.38)	302.88 ±(7.15)
	DDM	0.0001	0	< 0.001	-	248.00 ±(0.00)	23.92 ±(14.63)
		0.0002	0	< 0.001	-	248.00 ±(0.00)	22.61 ±(15.20)
		0.0003	0	< 0.001	-	248.00 ±(0.00)	22.43 ±(16.90)
		0.0004	0	< 0.001	-	248.00 ±(0.00)	19.80 ±(15.51)
10%	MAGSEED	0.0001	53	0.002	842.62 ±(158.10)	2311.25 ±(604.04)	949.91 ±(76.14)
		0.0002	99	0.002	652.49 ±(138.28)	2266.18 ±(406.15)	944.83 ±(78.22)
		0.0003	100	0.002	515.48 ±(113.33)	2162.88 ±(300.25)	945.61 ±(75.83)
		0.0004	100	0.002	432.28 ±(92.35)	2021.76 ±(316.66)	940.68 ±(77.88)
	ADWIN2	0.0001	**74**	0.011	714.03 ±(231.13)	1719.03 ±(168.42)	300.72 ±(7.90)
		0.0002	**100**	0.011	559.00 ±(181.27)	1564.24 ±(76.47)	300.85 ±(7.39)
		0.0003	**100**	0.011	425.56 ±(134.13)	1517.20 ±(84.00)	301.11 ±(7.44)
		0.0004	**100**	0.011	363.80 ±(115.99)	1456.72 ±(87.07)	302.68 ±(7.78)
	DDM	0.0001	0	< 0.001	-	248.00 ±(0.00)	23.60 ±(16.96)
		0.0002	0	< 0.001	-	248.00 ±(0.00)	20.57 ±(15.16)
		0.0003	0	< 0.001	-	248.00 ±(0.00)	19.64 ±(13.40)
		0.0004	0	< 0.001	-	248.00 ±(0.00)	20.96 ±(16.63)

Worst Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.2$, ADWIN2 $\delta = 0.3$, DDM $\alpha = 1$ $\beta = 3$

correlation between average computed severity and slope of change highlighted in bold. MAGSEED has a high rate of true warning detection (98-100% given a true drift is detected) and is able to detect more true drifts than DDM. Both detectors have high correlations for noise free data which shows that they are capable of capturing the speed of concept drift. For noisy data with 5% or 10% noise, MAGSEED shows a clear advantage as it is able to detector more true drifts, true warnings and the measure correlates well with speed of change. This suggests that our detector is capable of tracking the speed of concept drift.

Table 3.7 shows results for CIRCLES streams with gradual concept drift that have low levels of severity. The third column (Actual Severity) is the difference in area between the new and old circle concepts. The MAGSEED detector shows high correlation between

the computed and theoretical severity of the stream, and has higher true drift and true warning detection rates without compromising the rate of false positive drifts which is below 0.34%.

Table 3.8 shows results for SEA Concepts streams with abrupt concept drift simulated with a range of severity levels. The third column in the table (Actual Severity) is the theoretical severity of the streams computed as the area difference between two concepts. In these experiments, the severity measure of MAGSEED shows high correlation with actual severity and consistently performed better than DDM in terms of true warning rate and true drift rate. For the noise free streams as the severity increases there is also a decrease in delay which decreases the area for correct warning detection. Most of these incorrect warnings are detected too early and the increasing error rate prevents the warning from shifting forward in time. In contrast the noisy streams have greater fluctuations in the error rate and are more apt to recover from local maxima

Table 3.6: Warning detection: Bernoulli streams

			Best			Worst		
Noise	Detector	Slope	RD	TW	Rate(SD)	RD	TW	Rate(SD)
0%	MAGSEED	0.0001	84	84	0.36 ±(0.06)	84	84	0.34 ±(0.05)
		0.0002	100	100	0.37 ±(0.06)	100	99	0.36 ±(0.06)
		0.0003	100	100	0.38 ±(0.06)	100	99	0.37 ±(0.06)
		0.0004	100	100	0.39 ±(0.06)	100	99	0.38 ±(0.06)
Correlation (actual and computed severity)					**0.9973**			**0.9802**
	DDM	0.0001	1	1	0.26 ±(0.00)	1	0	0.23 ±(0.00)
		0.0002	1	1	0.30 ±(0.00)	1	0	0.24 ±(0.00)
		0.0003	1	1	0.34 ±(0.00)	1	0	0.24 ±(0.00)
		0.0004	1	1	0.39 ±(0.00)	1	0	0.24 ±(0.00)
Correlation (actual and computed severity)					**0.9963**			**0.9876**
5%	MAGSEED	0.0001	71	71	0.38 ±(0.06)	71	71	0.37 ±(0.05)
		0.0002	100	100	0.41 ±(0.06)	100	99	0.39 ±(0.06)
		0.0003	100	100	0.42 ±(0.06)	100	99	0.41 ±(0.06)
		0.0004	100	100	0.43 ±(0.07)	100	99	0.42 ±(0.07)
Correlation (actual and computed severity)					**0.9465**			**0.9583**
	DDM	0.0001	0	0	0	0	0	0
		0.0002	0	0	0	0	0	0
		0.0003	0	0	0	0	0	0
		0.0004	0	0	0	0	0	0
Correlation (actual and computed severity)					**0.0000**			**0.0000**
10%	MAGSEED	0.0001	54	54	0.41 ±(0.05)	53	52	0.40 ±(0.05)
		0.0002	100	100	0.42 ±(0.05)	99	98	0.41 ±(0.06)
		0.0003	100	100	0.46 ±(0.06)	100	99	0.44 ±(0.06)
		0.0004	100	100	0.46 ±(0.07)	100	99	0.44 ±(0.06)
Correlation (actual and computed severity)					**0.9365**			**0.9844**
	DDM	0.0001	0	0	0	0	0	0
		0.0002	0	0	0	0	0	0
		0.0003	0	0	0	0	0	0
		0.0004	0	0	0	0	0	0
Correlation (actual and computed severity)					**0.0000**			**0.0000**

Best Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.1$, ADWIN2 $\delta = 0.05$, DDM $\alpha = 2$ $\beta = 3$
Worst Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.2$, ADWIN2 $\delta = 0.3$, DDM $\alpha = 1$ $\beta = 3$

3.6.6 Accuracy on Real Data Streams

We also examine the performance of our algorithm on real world data, and compare it to ADWIN2 and DDM using a Hoeffding tree learner which is retrained using examples from the warning period for detectors with warnings.

Table 3.9 shows the performance of MAGSEED, DDM and ADWIN2 on three real world datasets: Forest Covertype [6], Pokerhand [6], and Airlines [19] as discussed in Section 2.5.2 . All the detectors are comparable in terms of overall prediction accuracy. However it is difficult to access the performance of the severity measure as we do not know the location or magnitude of the true drifts in these real world datasets. For the MAGSEED detector there is a large number of drifts that did not trigger a warning, this may be caused by changes of large magnitudes which do not trigger the warning threshold prior to the drift threshold.

Table 3.7: Warning detection: CIRCLES streams

		Actual	Best			Worst		
Noise	Detector	Severity	RD	TW	Rate(SD)	RD	TW	Rate(SD)
0%	MAGSEED	0.07	100	100	0.08 ±(0.03)	100	98	0.07 ±(0.03)
		0.16	100	100	0.09 ±(0.03)	100	99	0.07 ±(0.03)
		0.26	100	100	0.09 ±(0.03)	100	99	0.08 ±(0.03)
		0.38	100	100	0.09 ±(0.03)	100	99	0.08 ±(0.03)
Correlation (actual and computed severity)					**0.8482**			**0.9782**
	DDM	0.07	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.16	8	8	0.25 ±(0.02)	8	8	0.15 ±(0.01)
		0.26	97	97	0.26 ±(0.02)	97	62	0.15 ±(0.03)
		0.38	100	100	0.28 ±(0.02)	100	65	0.16 ±(0.03)
Correlation (actual and computed severity)					**0.7892**			**0.7514**
5%	MagSEED	0.07	100	100	0.17 ±(0.04)	100	98	0.15 ±(0.05)
		0.16	100	100	0.18 ±(0.04)	100	98	0.17 ±(0.04)
		0.26	100	100	0.19 ±(0.05)	100	98	0.17 ±(0.05)
		0.38	100	100	0.19 ±(0.04)	100	98	0.18 ±(0.05)
Correlation (actual and computed severity)					**0.9522**			**0.9278**
	DDM	0.07	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.16	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.26	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.38	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
Correlation (actual and computed severity)					**0.0000**			**0.0000**
10%	MAGSEED	0.07	100	100	0.24 ±(0.05)	100	98	0.23 ±(0.05)
		0.16	100	100	0.25 ±(0.05)	100	98	0.24 ±(0.05)
		0.26	100	100	0.26 ±(0.06)	100	97	0.25 ±(0.06)
		0.38	100	100	0.26 ±(0.06)	100	97	0.25 ±(0.06)
Correlation (actual and computed severity)					**0.9654**			**0.9374**
	DDM	0.07	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.16	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.26	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.38	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
Correlation (actual and computed severity)					**0.0000**			**0.0000**

Best Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.1$, DDM $\alpha = 2$ $\beta = 3$
Worst Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.3$, DDM $\alpha = 1$ $\beta = 3$

Table 3.8: Warning detection: SEA Concepts streams

		Actual	Best			Worst		
Noise	Detector	Severity	RD	TW	Rate(SD)	RD	TW	Rate(SD)
0%	MAGSEED	0.03	100	100	0.08 ±(0.03)	100	99	0.09 ±(0.03)
		0.05	100	100	0.09 ±(0.04)	100	98	0.08 ±(0.03)
		0.09	100	100	0.13 ±(0.04)	100	100	0.12 ±(0.04)
		0.12	100	100	0.12 ±(0.04)	100	99	0.12 ±(0.04)
		0.13	100	99	0.16 ±(0.05)	100	98	0.16 ±(0.05)
		0.15	100	88	0.20 ±(0.06)	100	76	0.20 ±(0.05)
		0.16	100	92	0.21 ±(0.06)	100	85	0.20 ±(0.06)
		0.24	100	96	0.19 ±(0.06)	100	90	0.19 ±(0.06)
		0.27	100	91	0.21 ±(0.06)	100	84	0.21 ±(0.07)
Correlation (actual and computed severity)					**0.8670**			**0.8760**
	DDM	0.03	90	90	0.04 ±(0.01)	90	55	0.04 ±(0.01)
		0.05	100	100	0.04 ±(0.01)	100	47	0.04 ±(0.00)
		0.09	100	100	0.10 ±(0.02)	100	51	0.09 ±(0.01)
		0.12	100	100	0.08 ±(0.01)	100	49	0.08 ±(0.01)
		0.13	100	100	0.14 ±(0.02)	100	43	0.12 ±(0.02)
		0.15	100	100	0.20 ±(0.03)	100	54	0.18 ±(0.03)
		0.16	100	100	0.19 ±(0.02)	100	52	0.17 ±(0.03)
		0.24	100	100	0.16 ±(0.02)	100	40	0.14 ±(0.03)
		0.27	100	100	0.17 ±(0.03)	100	40	0.15 ±(0.03)
Correlation (actual and computed severity)					**0.7980**			**0.7710**
5%	MagSEED	0.03	39	39	0.15 ±(0.08)	81	80	0.12 ±(0.07)
		0.05	80	80	0.15 ±(0.06)	99	98	0.16 ±(0.06)
		0.09	100	100	0.19 ±(0.04)	100	100	0.19 ±(0.05)
		0.12	100	100	0.19 ±(0.04)	100	100	0.19 ±(0.05)
		0.13	100	99	0.22 ±(0.05)	100	96	0.21 ±(0.05)
		0.15	100	97	0.25 ±(0.06)	100	94	0.24 ±(0.06)
		0.16	100	98	0.25 ±(0.06)	100	93	0.24 ±(0.06)
		0.24	100	98	0.24 ±(0.06)	100	94	0.24 ±(0.06)
		0.27	100	98	0.25 ±(0.06)	100	97	0.25 ±(0.07)
Correlation (actual and computed severity)					**0.8747**			**0.8853**
	DDM	0.03	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.05	31	31	0.09 ±(0.01)	31	9	0.09 ±(0.01)
		0.09	100	100	0.13 ±(0.01)	100	24	0.12 ±(0.01)
		0.12	100	100	0.13 ±(0.01)	100	26	0.12 ±(0.01)
		0.13	99	99	0.17 ±(0.01)	99	18	0.15 ±(0.02)
		0.15	100	100	0.21 ±(0.02)	100	20	0.19 ±(0.03)
		0.16	100	100	0.20 ±(0.02)	100	23	0.19 ±(0.03)
		0.24	100	100	0.19 ±(0.02)	100	15	0.16 ±(0.03)
		0.27	100	100	0.20 ±(0.02)	100	16	0.17 ±(0.04)
Correlation (actual and computed severity)					**0.8200**			**0.7494**
10%	MAGSEED	0.03	29	28	0.14 ±(0.09)	70	69	0.16 ±(0.13)
		0.05	38	36	0.21 ±(0.09)	86	85	0.19 ±(0.08)
		0.09	100	100	0.24 ±(0.05)	100	99	0.24 ±(0.05)
		0.12	100	100	0.25 ±(0.06)	100	100	0.23 ±(0.05)
		0.13	100	99	0.26 ±(0.05)	100	98	0.25 ±(0.05)
		0.15	100	100	0.29 ±(0.06)	100	99	0.29 ±(0.06)
		0.16	100	100	0.30 ±(0.06)	100	99	0.29 ±(0.06)
		0.24	100	100	0.29 ±(0.06)	100	99	0.28 ±(0.06)
		0.27	100	100	0.30 ±(0.05)	100	99	0.30 ±(0.06)
Correlation (actual and computed severity)					**0.8216**			**0.8662**
	DDM	0.03	0	0	0.00 ±(0.00)	0	0	0.00 ±(0.00)
		0.05	6	6	0.14 ±(0.01)	6	2	0.14 ±(0.01)
		0.09	99	99	0.17 ±(0.01)	99	17	0.16 ±(0.01)
		0.12	99	99	0.17 ±(0.01)	99	18	0.16 ±(0.01)
		0.13	100	100	0.20 ±(0.01)	100	15	0.19 ±(0.02)
		0.15	98	98	0.24 ±(0.02)	98	23	0.23 ±(0.02)
		0.16	100	100	0.24 ±(0.02)	100	16	0.23 ±(0.02)
		0.24	98	97	0.22 ±(0.02)	98	10	0.19 ±(0.03)
		0.27	98	97	0.23 ±(0.02)	98	8	0.20 ±(0.04)
Correlation (actual and computed severity)					**0.7608**			**0.6732**

Best Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.1$, DDM $\alpha = 2$ $\beta = 3$
Worst Parameters: MAGSEED $\delta_d = 0.05$ $\delta_w = 0.2$, DDM $\alpha = 1$ $\beta = 3$

Table 3.9: Performance on real data streams

Stream	Detector	Accuracy	Drifts	Warnings	Rate (SD)
Forest Covertype	MAGSEED	0.84	2380	1395	0.49 ±(0.23)
	DDM	0.83	1942	1475	0.83 ±(0.27)
	ADWIN2	**0.85**	2492	0	-
Pokerhand	MAGSEED	**0.75**	2032	1248	0.54 ±(0.20)
	DDM	0.73	1046	1007	0.63 ±(0.25)
	ADWIN2	**0.75**	2130	0	-
Airlines	MAGSEED	**0.66**	173	131	0.40 ±(0.15)
	DDM	0.65	14	14	0.38 ±(0.07)
	ADWIN2	0.65	384	0	-

Forest Covertype: On the Forest cover dataset, MAGSEED detected 1395 warnings out of 2380 drifts detected with an average severity of 0.49. DDM detected 1475 warnings out of 1942 drifts with an average severity of 0.83.

Pokerhand: On the Pokerhand dataset MAGSEED detected 1248 warnings out of 2032 drifts detected with an average severity of 0.54. DDM detected 1007 warnings out of 1046 drifts with an average severity of 0.63.

Airlines: On the Airlines dataset MAGSEED detected 131 warnings out of 173 drifts with an average severity of 0.40. DDM detected 14 warnings out of 14 drifts with an average severity of 0.38.

3.7 Conclusions

In this chapter, we presented a drift detector that is capable of detecting drift severity. We experimentally showed that there is a strong correlation between the computed and real magnitudes in our synthetic datasets which suggests that the computed and real magnitudes are similar. One limitation of our detector is that it may be unable to capture the drift severity for changes of high magnitudes where there are drastic changes in concepts. This is often due to the small delay time between the true change and detected drift point which provides less opportunity for correct warning detection. In the future, we would like to extend MAGSEED to adaptively determine the window size of our second warning threshold. We would also like to conduct experiments on a wider range of data streams and use drift severity for analysing characteristics of real data streams. In the next chapter we explore the use of stream volatility for the proactive detection of drifts.

4

Proactive Drift Detection

In the previous chapter we explored ways to capture the magnitude of concept drift in streaming data. In this chapter we will focus on characterizing the rate of these changes and explore how we can use this information to predict the behaviour of the stream for proactive detection of future drifts. We aim to use meta-knowledge of stream characteristics to improve the accuracy of current drift detection techniques. We believe this could facilitate the development of accurate models that can adapt to dynamic systems and non stationary environments.

This chapter is organized as follows. In Section 4.1 we motivate and highlight the difference between our work with previous research. In Section 4.2 we highlight the importance of mapping drift trends and define the key terminology used in this chapter. Section 4.3 presents an overview of our methods. Sections 4.4 and 4.5 discuss the inner workings of our technique. Our experiments are presented and discussed in Section 4.6. Finally we conclude this chapter in Section 4.7.

4.1 Introduction

Many drift detection methods have been developed for the detection of changes in the streaming environment [30] [10] [4] [16]. Traditional drift detection methods focus on detecting changes in the underlying model based on the error rate of a classifier. First we will review the traditional drift detection system. In Figure 4.1 we show a depiction of the

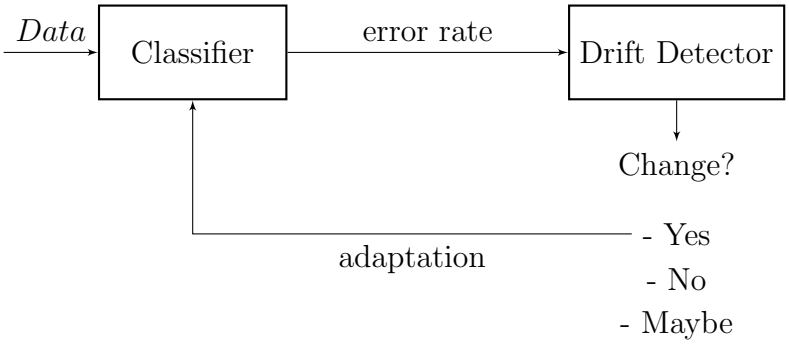


Figure 4.1: Work flow of traditional drift detection systems

work flow of a typical drift detection system. First the data is passed into a classifier which incrementally builds a model for classification based on the input data. The accuracy of the classifier is monitored using a binary error rate stream to detect concept changes signalled by significant increases in the error rate. The error rate stream is then passed into a drift detector that detects change using statistical tests. The drift detector can produce three signals. We present an analogy of this in Figure 4.1. The first is the drift signal which suggests that there is a high probability that a change has occurred. The second is the no change signal. This suggests the concept is stable and that it is unlikely a change has occurred. The third signal is the warning signal. It suggests the concept is likely to be changing and is used to anticipate real changes. Being able to find these changes allows a classifier to be adapted to changes in the data. However this adaptation process is not well studied. In this chapter we focus on the change detection phase.

Currently most methods do not consider additional information such as historical drift trends that could allow the anticipation of future change points. Huang et al. introduced a measure called stream volatility that represents the rate of change of the stream [17]. In their paper, they propose a volatility detector to detect changes in stream volatility and show that there are stream volatility trends for real data streams [17]. Unfortunately they did not use the information beyond finding the volatility trend. By incorporating volatility into traditional drift detectors we can use the additional information to proactively determine future changes. This is revolutionary as we are no longer monitoring historical events but anticipating future changes in the data stream. If we are able to map the frequency or interval of changes that occur to a particular pattern we can use this information to predict a future time range that has a high probability of change occurrence.

In a recent paper, Huang et al. extend their work to use historical drift points for improving drift detection by introducing a predictive approach using the mean stream volatility to estimate the location of the next drift [16]. However their method assumes that the overall drift rate of the stream is representative of the current drift rate and

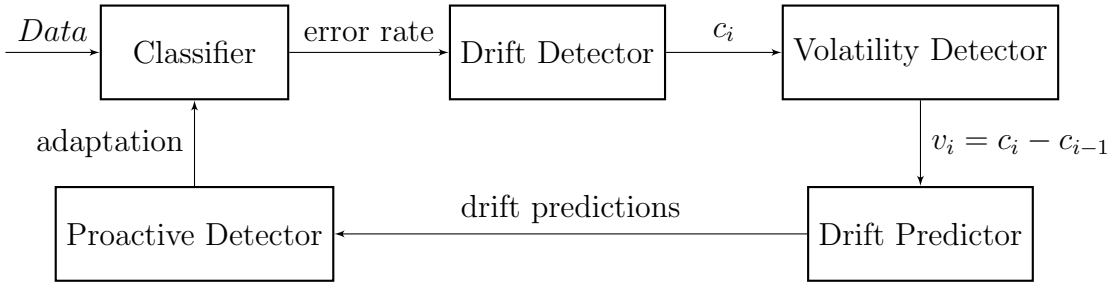


Figure 4.2: Work flow of our proactive drift detection system

does not utilize additional information such as trends of temporal changes in drift rate which may provide more accurate predictions for streams that show reoccurring patterns of volatility change. We believe there are many real life applications that have inherent trends in the rate of changes. For example the rate of electricity usage has both seasonal patterns, and trends that are specific to the personal habits of a customer. Therefore being able to model these trends may provide us with insights into the behaviour of the entity or concept of interest.

We propose a drift predictor for learning stream volatility trends and a proactive drift detector, PROSEED, that uses drift predictions to proactively search for drifts. Our methodology can be summarized by the diagram in Figure 4.2. We modify the traditional drift detection system to incorporate three additional layers, a volatility detector, a drift predictor and a proactive drift detector. The base detector is used to monitor the error rate of the classifier to find change points c_i. Based on this, the volatility detector is used to find changes in the drift rate v_i which represents the time between consecutive drifts. For example, if a change occurred every 15 days, the drift rate would be 15. We use a drift predictor to learn these drift rate trends and predict the possible locations of the next drift. These predictions are then used to modify the behaviour of PROSEED to reduce search effort during periods where drifts are less likely to occur.

The main contributions of our work are: (1) A drift prediction algorithm that can accurately learn drift trends of a stream and (2) a drift detector which incorporates historical drift rate information that is accurate for streams with reoccurring volatility trends. We analyze our drift prediction technique by comparing it to ground truth in synthetic data streams and show that it can accurately capture trends for streams with reoccurring volatility patterns. We evaluated the performance of our drift detector by comparing it against detectors ADWIN2 [4], SEED [17] and DDM [10] on synthetic and real data streams. We show that our technique is able to lower the rate of false positives for synthetic streams with these trends.

Relation to other research: Our research differs from research in drift detection with recurring patterns [41] as their methods are aimed at detecting models that reoccur whereas our method aims to learn the characteristics of drift rate trends. For example,

suppose we are trying to learn the concept of seasons, research in reoccurring patterns focuses on the order that concepts reoccur such as: spring, summer, autumn, winter, spring. Our research aims to look at the rate of concept change, that is the time period between season changes. Unlike research in temporal forecasting for seasonal patterns, we do not assume there is any seasonal effect to the changes, and the trends do not necessarily occur periodically.

4.2 Modelling Stream Volatility

The volatility of a stream describes the rate of change. High volatility indicates frequent changes and low volatility indicates that changes occur less frequently and the stream is more stable. Monitoring volatility can help us understand the nature of changes in a stream and give us important information about where changes may occur in the future.

For example, suppose we are interested in modelling traffic flow for GPS systems. Drifts can be analogous to changes in the speed of traffic such as a change in average car speeds from 40km per hour to 20km per hour. Drift volatility can be analogous to the rate of changes in traffic flow. This will tell us how frequent changes in traffic speed are occurring. A high volatility will indicate that changes in traffic speeds are occurring frequently, whereas low volatility suggests the traffic speeds are relatively consistent. During peak times such as rush hour, the speed of traffic would be more consistent so changes may occur less frequently.

For example, we may observe decreases in traffic speed between 7am-8am, reaching a minimum around 8.30am when the roads are heavily congested. This could be followed by increases in traffic speed after 10am. As there are more changes before and after 8.30am, the volatility would be higher before and after this time period. Therefore modelling these volatility trends may help us estimate when the next change in traffic speed may occur. For example, if we know that a road is currently congested, being able to estimate the future changes may help us predict when there is less traffic. This could help us decide whether we should take an alternate route which could save us time. If we are able to accurately pinpoint when the next change should occur, it may help us develop GPS routing methods that are more adaptive.

Definition 9 (Volatility Shift) *A volatility shift is a change in the drift rate. A shift in volatility is said to occur when there is a significant difference in the variance of two volatility windows. Let* $V_1 = (v_j, v_{j+1}..., v_k)$ *and* $V_2 = (v_{k+1}, v_{k+2}..., v_{t-1})$ *represent two volatility windows with sample variances* σ_1, σ_2 *respectively. A volatility shift is detected when* $\frac{\sigma_1}{\sigma_2} \lessgtr 1.0 + \beta$ *where* β *is a user defined threshold.*

We follow the volatility shift definition proposed by Huang et al. [17] that formalizes

this as a change in the variance between two volatility windows. Recall that a volatility window consists of intervals between consecutive drifts. We refer to these intervals as drift intervals, drift rates or volatility. We are interested in modelling the changes in volatility, so we need to keep a history of the volatility changes that we have previously observed. To model the volatility of a stream, we introduce a term called volatility pattern that represents a snapshot of the stream's volatility at a particular time. We assume that this has a certain distribution. We give a formal definition of volatility pattern in Definition 10.

Definition 10 (Volatility Pattern) *A volatility pattern* $p = \{v_m, v_{m+1}, ..., v_n\}$ *consists of a set of volatility values and a pattern length* l_p. *The set of drift intervals* $\{v_m, v_{m+1}, ..., v_n\}$ *is a subset of the intervals in the volatility windows* $V_1' \cup V_2$ *where the window* $V_1' = (v_i, v_{i+1}, ..., v_k)$ *and* $i \leq j < k$. *This represents a snapshot of the stream volatility at time* t *that has a distribution* D *with a mean of* μ *and variance of* σ^2. *The pattern length* l_p *is the average number of time steps spent in pattern* p *prior to a volatility shift at* t .

For example, given we detected a volatility shift at time t and observed two volatility windows $V_1' = (100, 150, 100)$ and $V_2 = (100, 150, 500)$. The pattern can be estimated to be $p_1 = \{100, 150, 100, 100, 150\}$ which has a distribution with a mean of $\mu_{\{100,150,100,100,150\}} = 120$ and a variance of $\sigma^2_{\{100,150,100,100,150\}} = 750$. Pattern p_1 represents a snapshot of the stable portion of the stream volatility. This then allows us to characterize the volatility changes as transitions between volatility patterns.

Definition 11 (Pattern Transition) *A pattern transition* $p_1 \rightarrow p_2$ *at time* t *is a volatility shift at time* t, *where there is a change in the distribution generating the drift rate from distribution* D_1 *corresponding to the volatility pattern* p_1 *to a distribution* D_2 *which corresponds to the volatility pattern* p_2.

Following from the previous example. Suppose we first observed pattern p_1, then at a later point in time t we detect a volatility shift and observe the pattern p_2. Let $p_1 = \{100, 150, 100, 100, 150\}$ and $p_2 = \{1000, 1500, 1000, 1000, 1500\}$. A transition $p_1 \rightarrow p_2$ at time step t means that there is a change in volatility at t from observing a change every 100 time steps to observing a change every 1000 time steps.

We generalize streams with volatility changes into two categories: (1) streams with rapid volatility change, and (2) streams with progressive volatility change. Rapid volatility changes represent periods of stable volatility punctuated by large sudden changes in the volatility distribution. Progressive volatility changes represent small incremental changes in the volatility distribution. In Figures 4.3 and 4.4 we show the volatility trends in blue, where each data point represents a drift interval. We highlight the local maxima and minima in yellow. Ideally this is where we would like to detect a volatility shift.

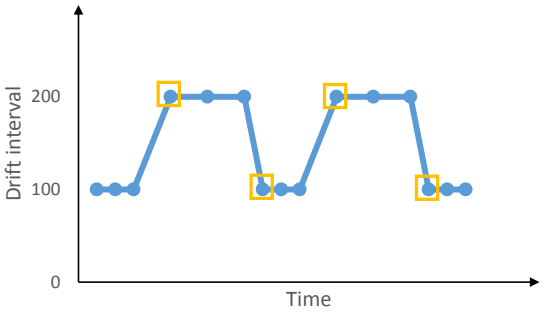


Figure 4.3: Example of rapid volatility change

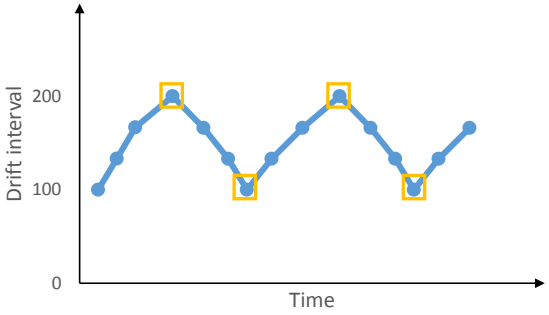


Figure 4.4: Example of progressive volatility change

For example, suppose the volatility of the stream is $100, 100, 100, 200, 200, 200, 100, ...$ as shown in Figure 4.3. This represents a stream with rapid volatility change because there is a sudden change in drift rate from 100 to 300, preceded by periods of stability.

For example, suppose the volatility of the stream is $100, 130, 160, 200, 160, 130, 100, ...$ as shown in Figure 4.4. This would be considered a progressive volatility change because the volatility changes are incremental in nature.

If we are able to capture the volatility shifts at the correct locations, and accurately estimate the patterns this would allow us to predict the future volatility. This may be advantageous for streams with reoccurring volatility trends because predicting the next drift location allows us to anticipate this change and adapt our behaviour based on the gained information. For example, suppose we are using a GPS for routing and we predict that there would be heavy traffic on Great South Road during the next two hours, having this extra information allows us to avoid routes that are potentially congested.

4.3 Overview

In this section we provide a general overview of our drift prediction method, DPM, that estimates the next drift point based on historical drift trends and our proposed algorithm, PROSEED, a proactive drift detector that uses the estimates from the drift predictor to guide the search for changes.

4.3.1 Drift Prediction Method

Our drift prediction method has three layers - a drift detector that provides the drift points and drift intervals, a volatility detector that is able to locate local volatility change points using the drift intervals, and a drift prediction algorithm that uses the location of volatility shifts to estimate the next drift. We use a pattern reservoir and probabilistic network to learn the volatility trends of a stream.

A pattern reservoir is a pool of size P that stores volatility patterns from the stream. Volatility patterns capture a snapshot representing a period of the stream. We use these patterns to build an overall picture of stream volatility and assume that each pattern has some underlying distribution. Each pattern has a sample of drift intervals which we use to approximate the underlying distribution that generates the intervals, and a pattern length that denotes the number of time steps the pattern persists before transitioning to another pattern.

We use a probabilistic network in the form of a transition matrix to learn the trends of these changes. In each row of our transition matrix we store the probability of transitioning to the next pattern given the current pattern.

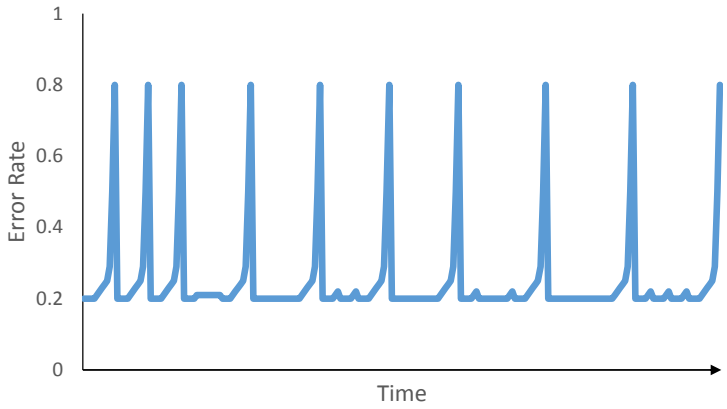


Figure 4.5: Error rate of the stream

For example, suppose the error rate of the stream is shown in Figure 4.5. Let the spikes represent increases in error rate caused by changes in concepts that are identified as cut points followed by an adaptation period where the classifier relearns. Let the time steps 100, 200, 300, 600, 900, 1200, 1500, 1900, 2300, 2700 represent the change points detected by the drift detector. The corresponding drift intervals are 100, 100, 100, 300, 300, 300, 300, 400, 400, 400. The trend of the drift intervals are shown in Figure 4.6.

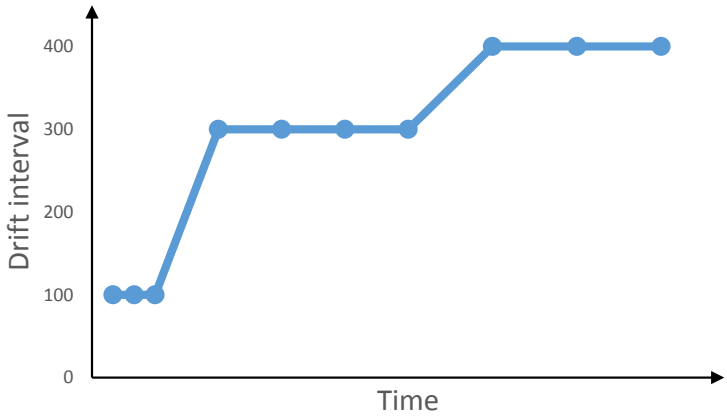


Figure 4.6: Drift intervals of the stream

Suppose we are able to identify the first volatility shift using the volatility detector at time step 600 after observing the drift interval of 300. When there is a volatility shift, a pattern will be stored into our pattern reservoir. This pattern stores a sample of recent data close to the volatility change which represents a snapshot of the stable drift intervals preceding the volatility change.

Let the volatility windows be $V_1' = (100, 100)$ and $V_2 = (100, 300)$. We construct a pattern p by removing outliers from $V_1' \cup V_2$ to get an approximation of the intervals in the stable period prior to the change. The first pattern $p_1 = \{100, 100, 100\}$ is added to the empty pattern reservoir. As this is the first volatility change we observed, the pattern length of p_1 is set to $l_{p_1} = 600$ which is the number of time steps between the volatility change point and the start of the stream.

When the next volatility shift is detected at time step 1900 and the volatility windows are $V_1' = (300, 300)$ and $V_2 = (300, 400)$, removing the outliers the pattern becomes $p = \{300, 300, 300\}$. Since the time between consecutive volatility shifts is $1900 - 600 = 1300$, the pattern length for p is 1300. Then we attempt to add a new pattern $p = \{300, 300, 300\}$ into the pattern reservoir. First we perform pattern matching between the potential pattern p and the pattern reservoir by testing for equivalence using the Kolmogorov-Smirnov test detailed in Section 4.4.2. If an equivalent pattern is found, we update the data samples in the pattern using the samples in p. Otherwise we add p as a new pattern p_2, and update the network by adding the transition $p_1 \rightarrow p_2$ to the probabilistic transition matrix. This transition can be illustrated by the network shown in Figure 4.7.

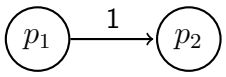


Figure 4.7: Network of pattern transitions

Once we have learned a network, we make predictions by doing a linear projection of the possible pattern means from the mean of the latest pattern that we have detected. The details of this process is given in Section 4.4 .

To control the size of our probabilistic network, we merge similar transitions based on the similarity of the slopes between consecutive pattern means. For example, assume we now have three patterns and our network has the transitions $p_1 \rightarrow p_2 \rightarrow p_3$. Let the slope of $\mu_{p_1} \rightarrow \mu_{p_2}$ be $s_1 = 0.4$, and the slope of $\mu_{p_2} \rightarrow \mu_{p_3}$ be $s_2 = 0.45$. If the difference between the slopes are less than a user-defined slope parameter r, then the transition will be compressed. Suppose we use $r = 0.1$, since $|s_1 - s_2| = 0.05 < r$, we compress the transition $p_1 \rightarrow p_2 \rightarrow p_3$ into $p_1 \rightarrow p_3$.

4.3.2 Proactive Drift Detection

Our drift detection method uses a two phase approach. In the first phase we train our prediction algorithm. In the second phase we use the predictions to adjust the behaviour of our drift detector, PROSEED. First, we use the drift intervals from the SEED detector [17] to train our drift prediction algorithm to allow the exploration of the drift trends without any prior information. Then we use the network generated from our drift predictor to guide PROSEED by controlling data compression. PROSEED groups the error rate data into blocks of size b. For example, suppose the error stream is:

$$00011000100110110111$$

Using blocks of size four $b = 4$, the data becomes:

$$\begin{array}{ccccccccc} 0001 & | & 1000 & | & 1001 & | & 1011 & | & 0111 \\ c_1 & & c_2 & & c_3 & & c_4 & & \end{array}$$

Each vertical bar represents a block boundary and is a potential drift point c_1, c_2, c_3, c_4. We use the drift estimates from the drift predictor to find time steps where the next drift is likely to occur. For example, let $t_1, t_2, ..., t_{20}$ represent the time stamps associated with each classification error. If the drift predictor estimates the next likely drift to be at t_6 and t_{18}, the second and fifth blocks will not be compressed as they contain the estimated time steps. Blocks between the estimates will be compressed as drifts are less likely to occur during those periods. All blocks following the highest estimated drift time t_{18} will not be compressed. This would compress the third and fourth blocks, removing the third potential drift point from being checked.

$$\begin{array}{ccccccc} 0001 & | & 1000 & | & 10011011 & | & 0111 \\ c_1 & & c_2 & & c_3 & & \end{array}$$

Then the drift detector will search for a drift by testing for a difference in means at the block boundaries using the Hoeffding bound detailed in Section 3.4.1. The novelty of our method is that it uses historical drift trends to adjust the compression of blocks in contrast to SEED which compresses based on a linear block similarity measure. This could be advantageous for data streams with reoccurring drift trends because it could allow us to accurately forecast the location of the next drift and adjust our drift detection method based on our belief of drift.

4.4 Drift Prediction Method

In this section we provide the details of our drift prediction algorithm. We use a drift detector to find points of change and the volatility detector proposed by Huang et al. [17]

to locate the local peaks and troughs of volatility change. We will discuss each component of our technique separately.

4.4.1 Pattern Construction

When we construct a pattern we store a pool of pattern data with mean μ_p to represent the distribution of the drift intervals, and the average pattern length l_p to represent the average time spent in the pattern before a volatility change was detected. We use the mean of the pattern μ_p to decide what the next drift interval may be, and l_p to determine when the change will occur. The average pattern length l_p is updated using the time between consecutive volatility changes each time we detect the pattern p.

We populate the pool of a pattern by sampling from a recent volatility window after a volatility change was detected. Given two volatility windows $V_1' = (v_i, v_{i+1}, ..., v_k)$ and $V_2 = (v_{k+1}, v_{k+2}, ..., v_{t-1})$ where $i < k < t-1$. Here V_2 represents the buffer that triggered the volatility change. First we compute the interquartile ranges of $V_1' \cup V_2$ and V_2, and choose the set of intervals that has the smaller interquartile range as the pool to sample our pattern data from. We assume that there are trends to the location of the true drifts, and false alarms are deviations from these trends. To filter out the possible false alarms and account for the period of volatility change we remove the outliers from the set of drift intervals. We assume the outliers contribute to the volatility change, and follow Tukey's definition for outliers

$$Y < (Q_1 - 1.5 \cdot IQR) \text{ or } Y > (Q_3 + 1.5 \cdot IQR), \tag{4.1}$$

where Y is an outlier, Q_1 is the first quartile, Q_3 is the third quartile and IQR is the interquartile range.

4.4.2 Pattern Matching

When we add a potential pattern p' to the pattern reservoir of size P, we first perform pattern matching to ensure that the pattern is not already present in the reservoir. This is done by testing for pattern equality using the two sample Kolmogorov-Smirnov test [35]

$$D_{n,n'} > c(\alpha) \cdot \frac{n + n'}{nn'}, \tag{4.2}$$

where the D Statistic denoted by $D_{n,n'}$ is the maximum difference between the cumulative distributions, $c(\alpha)$ is a function of the confidence parameter $\alpha \in (0, 1)$ and n, n' are the sample sizes. If Equation 4.2 holds the patterns are deemed to be different. Otherwise we update the samples stored by the pattern by randomly replacing old data points with the new data samples from p'. This allows patterns to evolve over time, and can reduce the

number of redundant patterns in the reservoir. We recommend selecting a small value for the alpha parameter such as $\alpha = 0.05$ for 95% confidence.

4.4.3 Transition Compression

We use a transition compression scheme to compress transitions that are similar. An example of this method is presented in Section 4.3.1. The aim is to achieve a more compact representation of the network and also to reduce the number of pattern transitions to itself which has a dampening effect on the true transitions in the network.

4.4.4 Predicting the Next Drift

Given that the latest pattern we have seen is pattern p_x we can predict the next patterns that are likely to occur $p_{y1}, p_{y2}, ..., p_{yk}$ using the network, where k is a parameter for the number of predictions to use. We recommend setting k as the size of the pattern reservoir, $k = P$, when we have no prior knowledge. This will allow all possibilities that have been previously observed to be mapped and give us information on what the drift interval is likely to change to. The average length of each predicted pattern $l_{y1}, l_{y2}, ..., l_{yk}$ will allow us to determine how long we expect the predicted pattern to persist.

For streams with rapid volatility change, that have stable periods punctuated by sudden changes we use the pattern means $\mu_{y1}, \mu_{y2}, ..., \mu_{yk}$ as the predicted drifts. For streams with progressive volatility change, that have incremental changes we calculate the next drift for each predicted pattern $p_{y_1}, p_{y_2}, ..., p_{yk}$ by

$$\frac{\mu_{yi} - \mu_x}{l_{yi}} \cdot t, \tag{4.3}$$

where μ_{yi} is the mean of pattern p_{yi}, μ_x is the mean of the latest pattern p_x, l_{yi} is the average length of pattern p_{yi} and t is the current time step. Our current algorithm assumes the nature of the volatility changes of the stream are known in advance, however in reality the nature of the stream is often unknown. We outline a method to address this in Section 4.4.5.

Drift Prediction Algorithm: The pseudocode for our prediction method is shown in Algorithm 2. In our pseudocode the buffer B corresponds to V_2 and the large buffer L corresponds to $V_1' \cup V_2$. Lines 1-5 initializes our algorithm, and lines 8-17 outline the work flow of our prediction system. We use a drift detector to detect drifts in lines 8-9 and construct drift intervals in lines 15-16. These intervals are passed into a volatility detector in line 11. The *AddPattern* function called in line 13 is used to learn drift interval trends by updating the pattern network and reservoir. Lines 20-25 perform pattern construction

detailed in Section 4.4.1. Lines 33-40 performs pattern matching as shown in Section 4.4.2. At line 31 we update the network and perform transition compression from Section 4.4.3. Lines 41-45 outlines the pattern replacement scheme of our pattern reservoir. Lines 46-58 performs drift prediction as described in Section 4.4.4 using linear projections.

4.4.5 Characterizing Volatility Change

In real streams the characteristics of the stream are often unknown. This leads to challenges in modelling stream behaviour and parameter selection. We present a method to address the first issue that allows us to determine the nature of volatility changes given only the output from the drift detector.

Given drifts are detected at time steps $c_1, c_2, ..., c_t$, and $i_1, i_2, ...i_{t-1}$ are the corresponding drift intervals between consecutive drifts. The drift intervals can be grouped into blocks $B_1, B_2, B_3...$ of size $b \geq 32$. We can compute the difference in means between consecutive blocks as $|\mu B_i - \mu B_{i+1}|$. The volatility of the stream can be monitored using a histogram that is updated incrementally. This gives us a view of the overall volatility of the stream that can allow us to determine the volatility nature by matching the characteristics of the histogram with the characteristics of streams with rapid or progressive volatility. Streams with rapid volatility changes are defined as having stable periods with sudden changes in volatility so the distribution of means between consecutive blocks should be more concentrated at lower values due to the periods of stability, with a right hand tail that represents the sudden changes. In contrast streams with progressive volatility changes should have a relatively even distribution. However the shape of this distribution also depends on the variance of the drift intervals.

4.5 Proactive Drift Detection

In this section we provide the details of our drift detection method. PROSEED extends the SEED detector [17] to use drift trends to adapt its behaviour through data compression as shown in Section 4.3.2. This affects where and how often the drift bound is checked.

Drift Bound: Our detector uses the Hoeffding bound with Bonferroni correction from Section 3.4.1 to determine whether a drift has occurred. This bound is used to test for significant differences in the means of two sub-windows. We use this bound as it has been shown to be more sensitive to small changes and gives theoretical guarantees on the false positive rate.

Block Compression: Here we will formally define our block compression scheme. Given k predictions represented by the vector $Y = (y_1, y_2, ..., y_k)$, where Y_{max} is the maximum predicted value from Y. Let the time interval (t_{start}, t_{end}) be the relative time

Algorithm 2 Drift Prediction Algorithm

Input: $X_t \in \{0, 1\}$ classification result at time t
Output: estimated location of the next concept drift

Initialize driftDetector D
Initialize volatilityDetector V, with buffer B of size b
Initialize $driftInterval \leftarrow 0$
Initialize a pattern reservoir P, and network N of size n
Initialize a large buffer L of size $\geq 2b$

for $t > 0$ **do**
    $estimatedInterval \leftarrow PredictNextDrift()$
    pass X_t to D
    if D detects a drift **then**
        update L with $driftInterval$
        pass $driftInterval$ to V
        if V detects a volatility shift **then**
            $AddPattern(L, B$ from $V)$
        end if
        $driftInterval \leftarrow 0$
    else $driftInterval \leftarrow driftInterval + 1$
    end if
end for

function AddPattern(Large Buffer L, Buffer B)
    $data \leftarrow B$
    $l_r \leftarrow$ interquartile range of L
    $b_r \leftarrow$ interquartile range of B
    if $l_r \leq b_r$ **then** $data \leftarrow L$
    end if
    remove outliers in $data$ by Tukey's method
    if $PatternFound(data)$ **then**
        update pattern in P
    else
        $AddToReservoir(data)$
    end if
    update network N
end function

function PatternFound(data d)
    $found \leftarrow false$
    for each element e in P **do**
        if d equals e by the Kolmogorov-Smirnov test **then** $found \leftarrow true$
        end if
    end for
    return $found$
end function

function AddToReservoir(data d)
    if P is not full **then** add d to P
    else replace rarest element in P with d
    end if
end function

```
46: function PredictNextDrift
47:     E ← list of estimates
48:     F ← list of top k transitions from current pattern p_x
49:     for pattern f in F do
50:         if nature of stream is progressive then
51:             estimate ← ((μ_f − μ_px) / l_f) · t
52:         else if nature of stream is rapid then
53:             estimate ← μ_f
54:         end if
55:         E ← E ∪ estimate
56:     end for
57:     return E
58: end function
```

period associated with block B, where t_{start} is the time stamp from the first instance in block B, and t_{end} is the time stamp from the last instance in block B.

$$t \in (t_{start}, t_{end}), \text{ and } t \in (y_1, y_2, ..., y_k) \tag{4.4}$$

$$t_{start} > Y_{max} \tag{4.5}$$

Blocks with time steps that satisfy either Equations 4.4 or 4.5 will not be compressed. The former equation prevents likely drift locations from being overlooked whereas the latter allows the anticipation of future drifts when we are less certain of where the next drift will occur. Through compression we are able to remove block boundaries that are less likely to be drift points.

Learning Period: The accuracy of our future predictions relies on the accuracy of our network and the predictability of the drifts in the stream. In our system we learn the network incrementally as new drift rate trends are discovered. Similar to other learning techniques, our network model may take some time to stabilize so we introduce the notion of a learning period where we construct the network using volatility changes in the drifts found by a base drift detected. We define the length of our learning period using the number of detected volatility changes which we refer to as X.

For example, suppose we set X to be 20. This means that we will not make any predictions for the first 20 volatility changes, and the compression of the drift detector will not be adapted until we have observed at least 20 volatility changes.

4.6 Experiments

We divide our experiments into two phases. In the first phase we evaluate the accuracy of our drift prediction algorithm. In the second phase we evaluate our drift detector by comparing it with SEED [17], ADWIN2 [4] and DDM [10]. We will briefly discuss our

synthetic data generation process below.

4.6.1 Synthetic Data Streams

We evaluate our technique on streams with rapid and progressive volatility changes as described in Section 4.2. We use these terms to describe how the rate of drifts change. This is a different dimensionality to the speed of concept drift which is often described as abrupt or gradual in previous work. We generate data with these trends by first defining patterns and a transition network. We define patterns with distinct means $\mu_H = \{100, 200, 300, ...\}$ each with variance $\sigma_H^2 = 100$ representing streams with high volatility, and $\mu_L = \{1000, 2000, 3000, ...\}$, $\sigma_L^2 = 1000$ for streams with low volatility. Based on these patterns we define cyclic networks which we use to generate cyclic drift rates. Consider

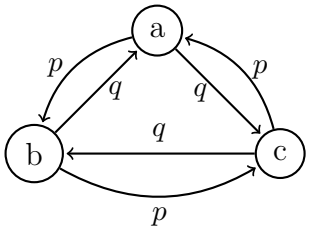


Figure 4.8: Example of a cyclic network with three patterns

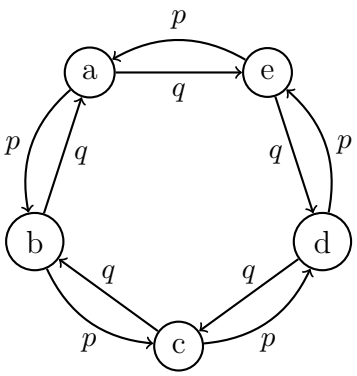


Figure 4.9: Example of a cyclic network with five patterns

the simple example with three patterns a, b, c, and the cyclic network $a \rightarrow b \rightarrow c \rightarrow a$. This network is illustrated in Figure 4.8. The transition probabilities are labelled as p and $q = 1 - p$. We present the results for $p = 0.75$ in Sections 4.6.3 and 4.6.4. First we generate drift intervals using a cyclic network to determine the location of change points, then we simulate drifts after each drift interval by changing the error rate or concept function for the stream.

We use three types of noise free streams: (1) Bernoulli streams [4] which simulates the error rate of a learner, (2) SEA Concepts streams [24] with abrupt concept change, and (3) CIRCLES streams [10] with gradual concept change. As the data from the Bernoulli streams represent the error rate of a learner this can be used directly by the drift detectors. SEA and CIRCLES generate instances with feature and value pairs that need to be passed through a classifier to produce the error rate stream used by the drift detectors. We use a Hoeffding Tree as our classifier because it learns incrementally which makes it suitable for data streams. For Bernoulli streams we alternate the mean error rate P between 0.2 and 0.8 after each change point to simulate abrupt changes. For SEA streams we change the

concept function threshold. We select thresholds of 7 and 9.5 to simulate abrupt changes of low magnitudes. For CIRCLES streams we alternate the radius of the circle concept between 0.2 and 0.3, and simulate gradual changes by gradually increasing the probability of generating instances from the new concept using a slope of 0.002 in the last 500 time steps of each drift interval. We present the average and standard deviation values over 100 runs.

4.6.2 Parameter Selection

To ensure the input to our algorithms are sufficiently accurate we tune the volatility detector to capture local changes in the drift rate. For synthetic streams, we use a buffer size of 32 with confidence $\beta = \{0.2, 0.5\}$. For detectors we use recommended settings of $\delta = 0.05$ for ADWIN2, SEED, PROSEED and $\alpha = 2, \beta = 3$ for DDM.

Accuracy of Input: As our drift prediction method uses a volatility detector and a drift detector, we need to ensure these inputs are relatively accurate. The accuracy of these input algorithms form an upper limit on the true positive rate of our technique. We assess the accuracies of the input algorithms and show that we are able to obtain a high true positive rate on synthetic Bernoulli streams with cyclic trends.

Drift Detection: We demonstrate that the SEED detector used to train our network is accurate in terms of the number of true positive and false positive drifts detected in Section 4.6.4. This allows our method to learn an accurate network in Section 4.6.3.

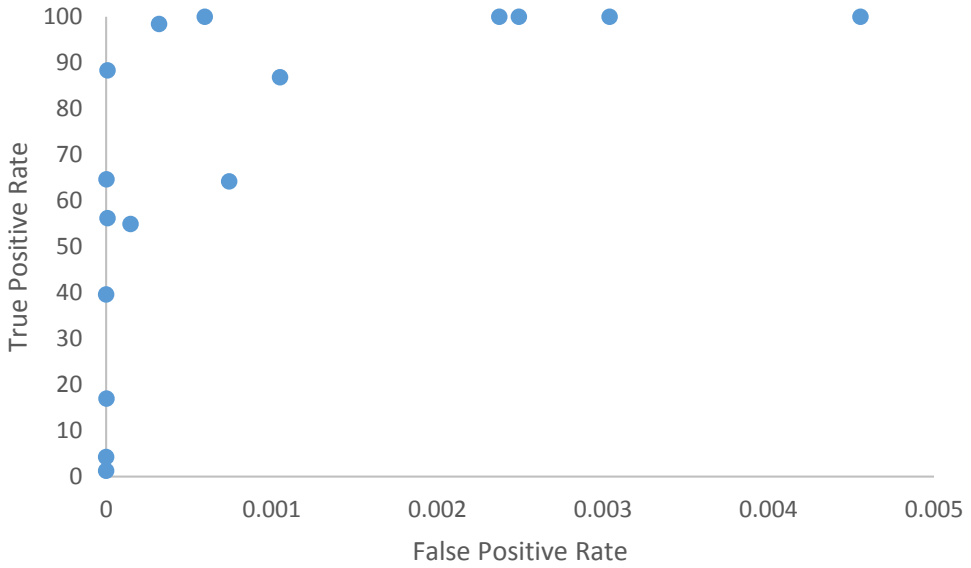


Figure 4.10: Accuracy of volatility detector under various parameter settings

Volatility Detection: We present the accuracy of the volatility detector under a range of parameter settings, buffer size $\in \{32, 50, 75, 100\}$, and $\beta \in \{0.2, 0.5, 0.75, 0.9\}$. We compare the location of the detected volatility shifts with the location of the true

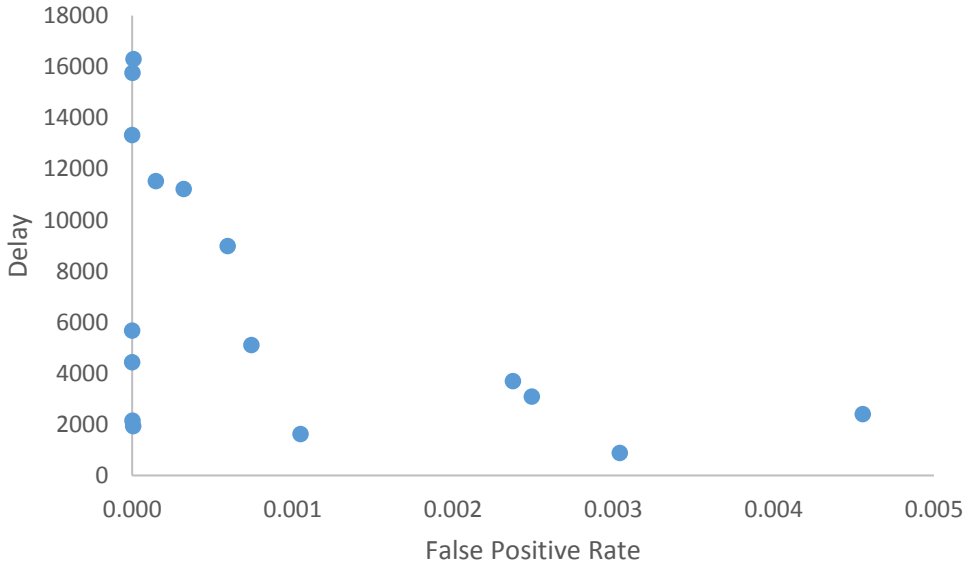


Figure 4.11: Delay of volatility detector under various parameter settings

volatility shifts where pattern transitions occur. We assume the first change detected following a true change is a true positive, and all other detections between pattern transitions are false positives. We monitor the delay in terms of the number of instances since the true volatility change. In Figures 4.10 and 4.11 each data point corresponds to a volatility detector with a particular parameter setting.

4.6.3 Drift Prediction

First we evaluate our drift prediction algorithm on Bernoulli streams and compare the patterns and network produced by our predictor to the ground truth. For this part we only present the comparison results for streams with rapid volatility changes. We do not present the results for progressive streams as the existence of many equivalent networks makes comparisons difficult.

Evaluation metrics: We use two metrics for measuring network similarity: The number of true transitions out of 500 (TT) which shows how many transitions from the stream is captured by the drift predictor's network, and the number of additional transitions (AT) which measures how many additional transitions are present in the drift predictor's network that are not in the stream's network. For comparing different sized networks, we compress the larger network from the drift predictor so that it is the same size as the network from the stream. Network compression is performed by merging the most similar patterns according to the lowest D statistic value from the Kolmogorov-Smirnov test until both networks have the same size.

In these experiments we present the results on streams with rapid volatility change. For the experiments presented in Tables 4.1, 4.3, 4.4 we generate synthetic streams with

Table 4.1: Network comparison: Bernoulli streams with different network sizes

Metric	N / P	10	30	100
TT (SD)	3	497.99 ± (0.90)	498.26 ± (0.66)	498.26 ± (0.66)
	5	496.94 ± (2.33)	498.29 ± (0.68)	498.29 ± (0.68)
	10	267.04 ± (61.59)	481.88 ± (20.83)	481.88 ± (20.83)
AT (SD)	3	603.59 ± (9.92)	603.89 ± (9.92)	603.89 ± (9.92)
	5	601.06 ± (10.37)	601.99 ± (10.32)	601.99 ± (10.32)
	10	482.50 ± (36.38)	624.43 ± (20.50)	624.43 ± (20.50)
Memory (SD)	3	10498.80 ± (950.36)	17610.24 ± (1262.56)	92090.24 ± (1262.56)
	5	10832.88 ± (630.05)	18493.68 ± (1420.28)	92973.68 ± (1420.28)
	10	11016.40 ± (78.48)	29319.20 ± (1777.46)	103799.20 ± (1777.46)

Table 4.2: Network comparison: Bernoulli streams with different volatility levels

V	TT (SD)	AT (SD)	Time (SD)
100	498.26 ± (0.66)	603.89 ± (9.92)	991.54 ± (50.64)
200	498.63 ± (0.49)	899.34 ± (20.13)	1345.64 ± (87.02)
500	498.91 ± (0.29)	1747.37 ± (44.31)	2368.44 ± (150.66)

500 transitions between patterns with high volatility μ_H. We use a volatility interval of 100 which is the number of drift points between volatility changes.

In Table 4.1 we show the effect of the drift predictor's pattern reservoir size on networks of various sizes. Here P is the pattern reservoir size, and N is the size of the synthetic network. An example of a three pattern network is presented in Figure 4.8, and a five pattern network is presented in Figure 4.9. The number of true transitions is close to optimal (500) for networks of size 3 and 5, this indicates that the predictor has a high rate of true positives. For networks of size 10, this is near optimal for $P \geq 30$, as the number of patterns detected centers around 30. In this case setting $P < 30$ restricts the number of transitions that can be captured. A larger network can give a finer granularity of the trends whereas a smaller network may be better at capturing frequent patterns of the stream. We note that the additional transitions are equivalent to self transitions (e.g. pattern $a \rightarrow a$) and are caused by false positive input from the volatility detector. This has a dampening effect on the transition probabilities. We reduce the effect of this on our drift prediction accuracy by selecting the top P transitions in the network which maps all probable volatility changes.

For the later experiments in Tables 4.2, 4.3, 4.4 we keep the stream's network size fixed at $N = 3$, and vary other parameters of the stream. In Table 4.2 we examine the effect of the stability of the drift rate on the accuracy of our computed network. Here V denotes the volatility interval which is the number of cut points between changes in the drift rate, a higher value represents a more stable stream. The number of true transitions captured is relatively robust to the stability of the trends, but the number of additional transitions detected increases with respect to the size of the volatility interval.

Table 4.3 shows the effect of adding Gaussian noise with a standard deviation of σ'

Table 4.3: Network comparison: Bernoulli streams with Gaussian pattern noise

Pattern Noise (σ')	TT (SD)	AT (SD)
0	498.26 ± (0.66)	603.89 ± (9.92)
25	457.89 ± (51.06)	507.32 ± (52.77)
50	230.70 ± (57.88)	210.88 ± (58.71)

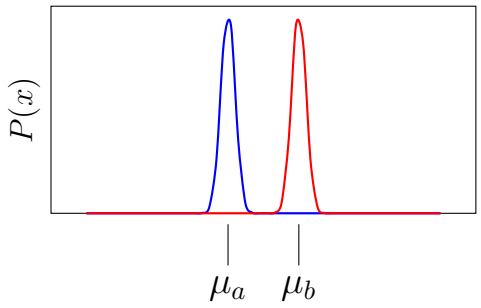


Figure 4.12: Pattern probability density functions without noise

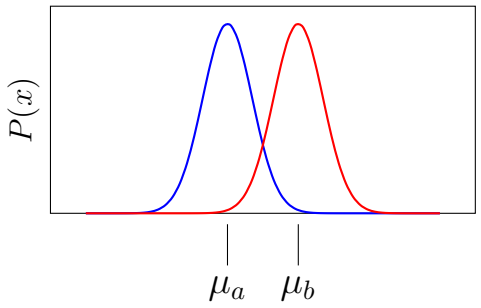


Figure 4.13: Pattern probability density functions with Gaussian noise ($\sigma' = 25$)

to the patterns. This increases the spread and amount of overlap between between tails of the pattern distributions. For example, suppose we have two noise free patterns that are normally distributed $a = (\mu_a = 100, \sigma_a = 10)$ and $b = (\mu_b = 200, \sigma_b = 10)$. Adding Gaussian noise of $\sigma' = 25$ changes the distributions to $a = (\mu_a = 100, \sigma_a = 35), b = (\mu_b = 200, \sigma_b = 35)$. The effect of this is illustrated in Figures 4.12 and 4.13. We observe that the accuracy of the network degrades as the percentage of overlap between the patterns increases.

Table 4.4 compares the accuracy of the patterns detected. μ_p and σ_p^2 are the mean and variance of the true pattern prior to the addition of noise. $E(\mu) = \mu_p$ is the expected mean of the pattern, and $E(\sigma^2)$ is the expected variance which is calculated as $E(\sigma^2) = (\sigma' + \sigma_p)^2$, where $\sigma_p = \sqrt{\sigma_p^2}$. We present the best, worst and median observations compared to the expected means and variances. Our estimates of the distribution means are very close to the expected values in the best and average cases. The estimation of variance is also accurate in the best case, but is less reliable in the worst case. We observe that in the best case the computed standard deviation is $0.98 - 1.04$ times the value of the expected standard deviation. As this value is close to 1, it indicates that the spread of our estimated distribution is close to the expected distribution of the pattern. On average, the standard deviation of the patterns is $0.75 - 2.71$ times the value of the expected standard deviation. In the worst case this is $1.54 - 7.37$ times the expected value. This occurs when the drift prediction algorithm is unable to eliminate all the false positive drifts in a window or incorrectly separates two patterns following a volatility shift. We show that our estimates of the distribution means are relatively accurate. In the future we would like to work towards improving the accuracy of our variance estimates.

In Table 4.5 we examine the effect of adding noise to the network by corrupting the

Table 4.4: Pattern comparison: Bernoulli streams with Gaussian pattern noise

Pattern Noise (σ')	μ_p	σ_p^2	$E(\mu)$	$E(\sigma^2)$	μ_{Best}	σ^2_{Best}	μ_{Worst}	σ^2_{Worst}	μ_{Median}	σ^2_{Median}
0	100	100	100	100	99.84	105.40	120.96	3383.96	99.84	438.30
0	200	100	200	100	200.00	99.71	219.84	2448.81	195.52	300.74
0	300	100	300	100	299.20	108.71	263.36	5437.65	289.76	733.76
25	100	100	100	1225	99.84	1199.84	145.60	6699.54	108.64	1828.25
25	200	100	200	1225	200.00	1230.87	254.72	5239.05	194.08	1709.72
25	300	100	300	1225	299.20	1275.97	250.88	6915.21	282.88	2444.31
50	100	100	100	3600	100.16	3454.29	176.64	8532.92	117.60	2033.26
50	200	100	200	3600	200.32	3599.52	128.00	10382.64	197.92	4127.65
50	300	100	300	3600	299.84	3625.48	217.60	8578.74	288.96	4114.36

Table 4.5: Network comparison: Bernoulli streams with network noise

Metric	T / N	3	5	10
TT (SD)	0	498.26 ± (0.66)	498.29 ± (0.68)	481.88 ± (20.83)
	0.01	495.08 ± (1.66)	493.55 ± (2.02)	468.63 ± (38.19)
	0.05	486.67 ± (3.73)	476.03 ± (24.49)	455.75 ± (29.75)
	0.1	473.57 ± (6.35)	459.55 ± (5.71)	434.35 ± (30.74)
AT (SD)	0	603.89 ± (9.92)	601.99 ± (10.32)	624.43 ± (20.50)
	0.01	608.29 ± (9.60)	612.8 ± (13.40)	642.74 ± (42.64)
	0.05	616.76 ± (10.53)	646.35 ± (27.44)	668.88 ± (32.50)
	0.1	630.50 ± (11.92)	670.30 ± (14.81)	707.89 ± (36.30)

S_1 : $a, \boldsymbol{b}, c, a, \boldsymbol{c}$

S_2 : $a, \boldsymbol{c}, c, a, \boldsymbol{b}$

Figure 4.14: Sequence of patterns

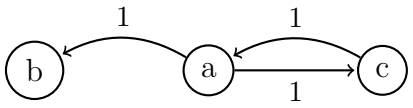


Figure 4.15: Network constructed by sequences

transitions. For example, let a, b represent patterns, a transition $a \rightarrow b$ is corrupted to become $a \rightarrow x$ such that $x \neq b$ with probability $T \in [0, 1]$. We refer to T as the network noise. In our experiments we tested on networks of size $N \in \{3, 5, 10\}$ with various levels of noise $T \in \{0.01, 0.05, 0.1\}$. Our aim is to investigate the effect of corruption on our ability to recover the underlying noise free network. We compare our detected network to the underlying network prior to corruption. For small networks $N \leq 5$ we achieve true transition rates that are close to the expected value of $1 - T$.

Introducing noise to small networks may result in some of the noise being hidden due to the occurrence of transitions that compensate each other. For example suppose the sequence of patterns from our underlying network is represented by S_1, and the corrupted network is S_2. In Figure 4.14 each letter in the sequence corresponds to a pattern, letter a represents pattern a, letter b represents pattern b and letter c represents pattern c. The order of the sequence represents the order of pattern transitions over time. Here the sequence $S_1 : a, b, c, a, c$ indicates there are four transitions, (1) $a \rightarrow b$, (2) $b \rightarrow c$, (3) $c \rightarrow a$ and (4) $a \rightarrow c$.

The first transition $a \rightarrow b$ in S_1 is corrupted to become $a \rightarrow c$ in S_2 and the last transition $a \rightarrow c$ in $S_1$is corrupted to become $a \rightarrow b$ in S_2. The effect of this on the pattern sequence is highlighted in bold. In this example, the two corruption events compensate each other so the network produced by S_1 and S_2 shown in Figure 4.15 are indistinguishable. In larger networks, this is less likely to occur as there are more patterns to select from when corrupting a transition. For large networks of size $N = 10$, we are able to obtain a true transition rate of $\geq 86\%$ with up to 10% noise.

4.6.4 Drift Detection

Next, we evaluate the accuracy of drift detection. We present results on streams with rapid volatility changes in Tables 4.6, 4.8 and 4.10 and progressive volatility changes in Tables 4.7, 4.9 and 4.11. In the following experiments we use patterns of low volatility μ_L, with 10000 drift points in each stream. We use patterns with low volatility so that each concept persists long enough for the accuracy of the classifier to stabilize. For the following experiments we keep the volatility interval fixed at 100, but would like to examine the effect of varying this such that changes in the drift rate do not occur after exactly 100 cut points in the future. For the experiments presented in Tables 4.6 - 4.11 and 4.13 - 4.14 we do not use a learning period for PROSEED.

Evaluation metrics: We use standard evaluation metrics - the number of true positives out of 10,000 (TP), number of false positives (FP), detection delay, memory (in bytes) and time (in milliseconds) for each detector. We present the memory and time used by PROSEED excluding the drift prediction phase as the results have been shown previously in [17] and Section 4.6.3.

In terms of time, PROSEED is faster than both ADWIN2 and SEED as we are able to reduce the number of potential cut points that are checked. However the overall time required for our method including the prediction phase exceeds the time used by a single

Table 4.6: Drift detection: Bernoulli streams with rapid volatility change

Detector	TP (SD)	FP (SD)	Delay (SD)	Memory (SD)	Time (SD)
PROSEED	9998.89 ± 0.31	33.10 ± 5.93	34.64 ± 10.09	683.84 ± 157.90	598.78 ± 14.57
SEED	9999.00 ± 0.00	213.34 ± 16.90	34.51 ± 10.15	2207.04 ± 974.11	1539.03 ± 41.15
ADWIN2	9999.00 ± 0.00	12689.68 ± 78.04	34.01 ± 10.14	1747.36 ± 137.13	5262.68 ± 108.51
DDM	5000.01 ± 0.22	97.41 ± 10.25	201.51 ± 52.57	128.00 ± 0.00	322.62 ± 15.52

Table 4.7: Drift detection: Bernoulli streams with progressive volatility change

Detector	TP (SD)	FP (SD)	Delay (SD)	Memory (SD)	Time (SD)
PROSEED	9998.92 ± 0.27	44.32 ± 6.43	34.53 ± 10.10	852.56 ± 349.62	558.01 ± 20.51
SEED	9998.99 ± 0.10	210.50 ± 15.21	34.46 ± 10.14	1968.48 ± 871.98	1479.92 ± 45.38
ADWIN2	9999.00 ± 0.00	12698.08 ± 68.53	33.98 ± 10.13	1740.64 ± 132.91	5254.83 ± 116.28
DDM	5000.04 ± 0.20	100.98 ± 10.03	203.29 ± 38.28	128.00 ± 0.00	299.54 ± 15.22

detector. In Tables 4.6 - 4.11 we observe that SEED and ADWIN2 are comparable in terms of the number of true drifts found. However ADWIN2 is more sensitive to false positives. This may be due to the way the data is stored which facilitates the detection of small changes. We show that PROSEED is able to accurately detect many true drifts, and is comparable to SEED and ADWIN2 in terms of true positives for Bernoulli streams. Although PROSEED finds fewer true positives than SEED, it is able to effectively lower the number of false positives by $78 - 93\%$. There is a natural trade off between true positives and false positives, so it is inevitable that decreases in false positives may also coincide with decreases in true positive rates. However, as we are able to effectively reduce the false positive rate whist attaining a high true positive rate $> 90\%$ in our synthetic datasets, this gives promise to the use of drift trends for achieving a more accurate drift detector.

We investigate the effects of using a learning period where we initially learn the network of the drift predictor without making any predictions as described in Section 4.5. We divide our data stream into a learning set with 1000 drift points, and a test set with 10,000 drift points. We select a learning set of size 1000 with 10 transitions because it would provide sufficient information to map 1-3 cycles from the network. For this experiment

Table 4.8: Drift detection: SEA Concepts streams with rapid volatility change

Detector	TP (SD)	FP (SD)	Delay (SD)	Memory (SD)	Time (SD)
PROSEED	9895.42 ± 47.34	37.02 ± 11.47	135.59 ± 92.23	641.60 ± 138.40	624.50 ± 16.47
SEED	9997.14 ± 1.83	165.69 ± 11.45	117.15 ± 65.97	1845.12 ± 818.37	1578.80 ± 38.42
ADWIN2	9997.00 ± 1.98	24920.11 ± 173.14	127.01 ± 74.95	1750.72 ± 139.07	8576.84 ± 171.26
DDM	2453.68 ± 2507.21	945.90 ± 966.72	436.25 ± 231.54	128.00 ± 0.00	357.25 ± 47.99

Table 4.9: Drift detection: SEA Concepts streams with progressive volatility change

Detector	TP (SD)	FP (SD)	Delay (SD)	Memory (SD)	Time (SD)
PROSEED	9820.32 ± 22.33	12.70 ± 3.63	217.24 ± 246.87	874.16 ± 315.96	599.08 ± 19.62
SEED	9997.18 ± 1.94	169.41 ± 12.37	116.85 ± 65.93	1876.32 ± 946.58	1536.42 ± 46.24
ADWIN2	9996.92 ± 2.04	25391.84 ± 183.48	126.12 ± 76.20	1740.64 ± 132.91	8547.74 ± 192.83
DDM	2301.24 ± 2501.38	906.03 ± 985.18	440.90 ± 221.87	128.00 ± 0.00	342.29 ± 42.40

Table 4.10: Drift detection: CIRCLES streams with rapid volatility change

Detector	TP (SD)	FP (SD)	Delay (SD)	Memory (SD)	Time (SD)
PROSEED	9714.49 ± 63.38	271.44 ± 56.95	547.69 ± 170.19	585.20 ± 91.94	798.29 ± 33.08
SEED	9993.52 ± 2.79	481.77 ± 24.96	482.33 ± 130.12	733.44 ± 333.92	1645.84 ± 40.85
ADWIN2	9984.44 ± 6.42	21956.39 ± 347.80	490.25 ± 179.02	1567.60 ± 107.78	5393.76 ± 100.39
DDM	522.84 ± 1252.72	306.94 ± 741.54	588.25 ± 227.85	128.00 ± 0.00	315.53 ± 18.52

Table 4.11: Drift detection: CIRCLES streams with progressive volatility change

Detector	TP (SD)	FP (SD)	Delay (SD)	Memory (SD)	Time (SD)
PROSEED	9273.94 ± 45.11	10.05 ± 4.68	594.95 ± 292.69	744.80 ± 290.12	710.96 ± 23.27
SEED	9997.93 ± 2.08	531.62 ± 25.30	474.28 ± 128.28	812.16 ± 458.08	1630.73 ± 48.39
ADWIN2	9995.55 ± 2.84	23943.53 ± 299.48	487.95 ± 159.98	1582.72 ± 98.22	5370.69 ± 123.56
DDM	608.69 ± 1643.70	380.32 ± 1030.09	456.82 ± 152.19	128.00 ± 0.00	308.80 ± 21.80

Table 4.12: Drift detection: PROSEED on synthetic data streams

Metric	Stream	Learning Period	No Learning Period
TP (SD)	Bernoulli R.	**9998.92 ± 0.27**	9998.89 ± 0.31
	Bernoulli P.	9998.89 ± 0.35	**9998.92 ± 0.27**
	SEA R.	**9931.68 ± 40.33**	9895.42 ± 47.34
	SEA P.	**9833.04 ± 20.59**	9820.32 ± 22.33
	CIRCLES R.	**9740.63 ± 62.32**	9714.49 ± 63.38
	CIRCLES P.	9271.09 ± 46.39	**9273.94 ± 45.11**
FP (SD)	Bernoulli R.	**28.93 ± 5.98**	33.10 ± 5.93
	Bernoulli P.	**40.17 ± 6.84**	44.32 ± 6.43
	SEA R.	**25.25 ± 12.02**	37.02 ± 11.47
	SEA P.	**4.39 ± 2.20**	12.70 ± 3.63
	CIRCLES R.	**259.77 ± 66.68**	271.44 ± 56.95
	CIRCLES P.	**7.10 ± 3.51**	10.05 ± 4.68
Delay (SD)	Bernoulli R.	**34.63 ± 10.09**	34.64 ± 10.09
	Bernoulli P.	**34.54 ± 10.10**	34.53 ± 10.10
	SEA R.	**128.73 ± 73.13**	135.59 ± 92.23
	SEA P.	**214.47 ± 245.59**	217.24 ± 246.8
	CIRCLES R.	**543.36 ± 365.23**	547.69 ± 170.19
	CIRCLES P.	**591.32 ± 292.37**	594.95 ± 292.69

Table 4.13: Drift detection: Forest Covertype

Detector	Drifts Detected	Memory	Time
PROSEED	2033.00	1064.00	177.78
SEED	2376.00	1080.00	47.98
ADWIN2	2492.00	1576.00	194.20
DDM	3953.00	128.00	22.81

Table 4.14: Drift detection: Pokerhand

Detector	Drifts Detected	Memory	Time
PROSEED	1651.00	464.00	66.53
SEED	1940.00	984.00	58.95
ADWIN2	2130.00	1408.00	241.81
DDM	857.00	128.00	28.73

we present the number of true positives, false positives and delay for instances in the test set. In Table 4.12 we present the results for six synthetic streams with rapid (R.) or progressive (P.) volatility change. We show that we are able to further reduce the number of false positives PROSEED detected by $4.30 - 65.43\%$. The number of true positives is comparable regardless of whether a learning period is used. In practice we can emulate a learning period by setting the use of predictions adaptively based on how many times we have seen the current pattern (e.g. use predictions only when we have observed a pattern X times), or based on the stability of the drift rate. When the drift rate of a stream has stable periods we can be more confident of our predictions, but when the drift rate has many fluctuations we are less confident of our predictions. In these cases we could adjust our compression method to search for drifts at regular intervals or use a stochastic compression method.

In the final experiment, we analyze the effect of our proactive drift detector on a complete drift detection system by using our detector for the adaptation of Hoeffding Trees. We adopt a simple adaptation approach where we reset the learner after a drift is detected. We evaluate our drift detector on two real datasets Forest Covertype [17] and Pokerhand [6] as discussed in Section 2.5.2. For these streams we do not know the location of true drifts so we can not assess the true positive and false positive rates. We show the number of drifts detected by PROSEED is lower than the number of drifts detected by SEED and ADWIN2 in Tables 4.13 and 4.14.

4.7 Conclusions

In this chapter we proposed a drift prediction algorithm that can accurately learn drift trends of a stream using a probabilistic network. We then proposed a new drift detector which incorporates historical drift rate information that is accurate for streams with reoccurring volatility trends. The highlight of our new technique is that it allows us to proactively determine when we would see future drift points. This in itself changes the landscape of how drift detectors are currently developed and used. To show the accuracy and feasibility of our technique, we analyze our drift prediction technique by comparing it to ground truth in synthetic data streams and show that it can accurately capture trends for streams with reoccurring volatility patterns. In our experiments we compared the performance of our drift detector against other benchmark detectors such as ADWIN2 [4], SEED [17] and DDM [10]. We show that we are able to lower the rate of false positives on synthetic streams with cyclic drift rate trends. We also evaluated the use of drift detectors for the adaptation of Hoeffding Trees on real data streams and show that our proactive approach detects fewer drifts than ADWIN2 and SEED.

One current limitation of our technique is that it only uses drift interval information for predicting future drift locations by matching the drift rate patterns to the pattern network. In the future we would like to explore the use of additional information such as concept models to enhance the drift detection capability of current drift detection methods. This would provide us with more knowledge about future predictions and enhance the drift detection capability by further reducing the false positives and hence increase the accuracy of our system.

5 Conclusions

In the final chapter we highlight the major achievements of our work, identify the current limitations and possible future directions.

5.1 Achievements

The following list highlights the major achievements of our research:

Chapter 3

- We proposed a novel drift detector (MAGSEED) to track the magnitude of changes in data streams with gradual changes.

Chapter 4

- We proposed a drift prediction method (DPM) for predicting the location of future drift points based on historical drift rate trends. We showed that our technique can accurately track reoccurring drift rate trends.
- We proposed a proactive drift detector (PROSEED) that incorporates historical drift rate trends in the form of predicted locations of future drift. We showed that by using a proactive approach we are able to effectively lower the rate of false positive drifts detected in streams with reoccurring cyclic drift rate trends.

5.2 Limitations

The first limitation relates to parameter selection for our magnitude detector in Chapter 3. Our detector uses a fixed size window for monitoring the error rate that contributes to the second warning threshold. We selected a size of 100 instances because it can be seen as a statistically significant sample size and confirmed that this setting works well on synthetic streams. We note that it may be more intuitive to set this according to the stability of the stream such that a larger size is selected for stable streams and a smaller size is selected for streams that are less stable.

The second limitation also relates to Chapter 3. The detector we presented is aimed at computing the magnitudes of change for gradual streams. We showed that our detector is able to estimate the magnitude for low to medium levels of magnitude change, but is unable to do so for abrupt changes with high magnitudes. High magnitudes of change are where we have large changes in concepts such as a complete reversal of the concept function.

The third limitation relates to the accuracy of the input to the proactive system presented in Chapter 4. There are two input components to our system. A drift detector for detecting changes which we refer to as a base drift detector, and a volatility detector for locating changes in the drift rate of the base drift detector. The output of our system is a proactive drift detector that is different to the base drift detector. The quality of our proactive system is limited by the accuracy of the input components. If the input is sufficiently accurate we may be able to utilize the drift rate trends to gain meaningful insights and produce an accurate proactive detector. However if the input is very noisy the new knowledge gained would also contain noise and may not improve the drift detection accuracy.

The fourth limitation is the assumption of a stable period in Chapter 4. We were interested in the overall volatility trends of a stream, and how the drift rate distributions changed over time so it seemed sensible to characterize the stream volatility as having periods of stability. However in the scenario that the stream is very volatility and does not exhibit these characteristics our modelling technique would not be sufficient to accurately capture the characteristics of the stream's volatility.

5.3 Future Work

We present four main directions for future work below.

Drift Magnitude

In Chapter 3 we introduced a method for detecting the drift magnitude based on the drift severity metric. This metric can only be applied to detectors with a warning phase, but as most detectors do not have a warning phase this limits its applicability. In addition to this, the drift severity metric assumes all errors in the warning window contribute to the change, but does not account for other factors that affect the error rate such as noise and limitations of the model. We would like to look at other ways to formalize metrics for the computation of drift magnitude and apply this to a range of drift detectors.

The method we presented relies on the correct setting of many parameters. One of which is the window size used to compute the moving average of the error rate. We have chosen to use a fixed size window of 100 instances which is statistically significant. However it would be more ideal to set this parameter adaptively based on the stability of the stream to solve the first limitation.

Another future research direction is to use the magnitudes of drifts for forecasting future changes. In our work, we only attempted to monitor the magnitude of drifts, but did not utilize these values. This could be applied to streams that have trends in the magnitude of changes to provide more information for users beyond the location of drifts.

Stream Volatility

In Chapter 4 we introduced a proactive drift detection method that uses historical trends in stream volatility modelled as volatility patterns and pattern transitions. Our framework assumes there are periods where the drift rate is stable, so it may not work well for volatile streams that have frequent fluctuations in the drift rate. We would like to look at ways to characterize streams with more frequent changes in the drift rate to address limitation four. In addition to this we would like to develop methods that are more flexible in terms of the way the drift predictions are used, and ways to monitor the confidence of our predictions. We believe this could allow us to decrease false positives for streams with less predictable trends.

We would also like to look at ways to provide better abstractions of volatility patterns so the projection of volatility trends can be scaled depending on the current drift rate of the stream.

Model Changes and Adaptation

One interesting area for future work is the study of changes in the model and change adaptation strategies. Currently most change detection systems employ a simple adaptation approach by completely relearning the model after a change. It would be useful to be able to determine which part of the model is changing, and adapt the model to

changes accordingly. This could be beneficial in scenarios where we have changes of lower magnitudes, or gradual changes because adopting a different adaptation strategy could allow models to be updated more quickly by decreasing the loss of relevant information.

Evaluation Benchmarks

The current datasets commonly used for algorithm evaluation only represent a small portion of the input space. Only using these datasets could cause the performance of algorithms to be misrepresented. Therefore we believe there is a need for more variety and standardization of benchmark datasets for the testing of machine learning algorithms. This requires the identification of key aspects of the input data that may be relevant to our research problem and from this generate test sets from different regions of the input space. The purpose of this is to increase the variety of datasets available.

Another area to explore is the generation of pseudo-synthetic datasets that have properties that are similar to real datasets. We believe this could increase the reliability of our evaluation methods.

Bibliography

[1] Stephen H Bach and M Maloof. Paired learners for concept drift. In *Eighth IEEE International Conference on Data Mining, 2008. ICDM'08.*, pages 23–32. IEEE, 2008.

[2] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86, 2006.

[3] Peter L Bartlett, Shai Ben-David, and Sanjeev R Kulkarni. Learning changing concepts by exploiting the structure of change. *Machine Learning*, 41(2):153–174, 2000.

[4] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*, pages 443–448, 2007.

[5] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148. ACM, 2009.

[6] Albert Bifet, Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Indrė Žliobaitė. CD-MOA: Change detection framework for massive online analysis. In *Advances in Intelligent Data Analysis XII*, volume 8207 of *Lecture Notes in Computer Science*, pages 92–103. Springer Berlin Heidelberg, 2013.

[7] Kylie Chen, Yun Sing Koh, and Patricia Riddle. Tracking drift severity in data streams. In *AI 2015: Advances in Artificial Intelligence - 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 - December 4, 2015, Proceedings*, pages 96–108, 2015.

[8] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[9] João Gama and Petr Kosina. Tracking recurring concepts with meta-learners. In *Progress in Artificial Intelligence*, pages 423–434. Springer, 2009.

[10] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence - SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer Berlin Heidelberg, 2004.

[11] João Gama and Carlos Pinto. Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 662–667. ACM, 2006.

[12] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

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

[14] Paulo M Gonçalves Jr and Roberto SM Barros. A comparison on how statistical tests deal with concept drifts. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[15] Paulo M Gonçalves Jr and Roberto SM Barros. RCD: A recurring concept drift framework. *Pattern Recognition Letters*, 34(9):1018 – 1025, 2013.

[16] David Tse Jung Huang, Yun Sing Koh, Gillian Dobbie, and Albert Bifet. Drift detection using stream volatility. In *Machine Learning and Knowledge Discovery in Databases*, pages 417–432. Springer, 2015.

[17] David Tse Jung Huang, Yun Sing Koh, Gillian Dobbie, and Russel Pears. Detecting volatility shift in data streams. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 863–868, 2014.

[18] IBM. Big data. http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[19] Dino Ienco, Albert Bifet, Bernhard Pfahringer, and Pascal Poncelet. Change detection in categorical evolving data streams. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 792–797, 2014.

[20] Yichao Jin and Barbara Hammer. Computational intelligence in big data. *Computational Intelligence Magazine, IEEE*, 9(3):12–13, 2014.

[21] Paulo M. Gonçalves Jr., Silas G.T. de Carvalho Santos, Roberto S.M. Barros, and Davi C.L. Vieira. A comparative study on concept drift detectors. *Expert Systems with Applications*, 41(18):8144 – 8156, 2014.

[22] Imen Khamassi and Moamar Sayed-Mouchaweh. Drift detection and monitoring in non-stationary environments. In *2014 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–6. IEEE, 2014.

[23] X Z Kong, Y X Bi, and D H Glass. A geometric moving average martingale method for detecting changes in data streams. In *Research and Development in Intelligent Systems*, pages 79–92, 2012.

[24] Petr Kosina, João Gama, and Raquel Sebastião. Drift severity metric. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 1119–1120, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[25] Ludmila I Kuncheva. Change detection in streaming multivariate data using likelihood detectors. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1175–1180, 2013.

[26] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity and Variety, 2001. http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[27] Fernanda L. Minku and Xin Yao. Using diversity to handle concept drift in online learning. In *Proceedings of the 2009 International Joint Conference on Neural Networks*, IJCNN'09, pages 2968–2975. IEEE Press, 2009.

[28] Leandro L Minku, Allan P White, and Xin Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, 2010.

[29] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *Discovery Science*, pages 264–269. Springer, 2007.

[30] ES Page. Continuous inspection schemes. *Biometrika*, pages 100–115, 1954.

[31] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191 – 198, 2012.

[32] Sripirakas Sakthithasan, Russel Pears, and Yun Sing Koh. One pass concept change detection for data streams. In *Advances in Knowledge Discovery and Data Mining*, volume 7819 of *Lecture Notes in Computer Science*, pages 461–472. Springer Berlin Heidelberg, 2013.

[33] Raquel Sebastião and João Gama. A study on change detection methods. In *4th Portuguese Conference on Artificial Intelligence, Lisbon*, 2009.

[34] Raquel Sebastião, João Gama, Pedro Pereira Rodrigues, and João Bernardes. Monitoring incremental histogram distribution for change detection in data streams. In *Knowledge Discovery from Sensor Data*, pages 25–42. Springer, 2010.

[35] Nikolai V Smirnov. Estimate of deviation between empirical distribution functions in two independent samples. *Bulletin Moscow University*, 2(2):3–16, 1939.

[36] Parinaz Sobhani and Hamid Beigy. *New drift detection method for data streams.* Springer, 2011.

[37] Sakthithasan Sripirakas and Russel Pears. Mining recurrent concepts in data streams using the discrete fourier transform. In *Data Warehousing and Knowledge Discovery*, pages 439–451. Springer, 2014.

[38] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

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

[40] Ying Yang, Xindong Wu, and Xingquan Zhu. Combining proactive and reactive predictions for data streams. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 710–715. ACM, 2005.

[41] Ying Yang, Xindong Wu, and Xingquan Zhu. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data Mining and Knowledge Discovery*, 13(3):261–289, 2006.

[42] Indrė Žliobaitė, Albert Bifet, Jesse Read, Bernhard Pfahringer, and Geoff Holmes. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, 98(3):455–482, 2015.