Capturing Recurring Concepts in High Speed Data Streams

Sakthithasan Sripirakas

A thesis submitted to Auckland University of Technology
in fulfilment of the requirement for
the degree of Doctor of Philosophy

Supervisors:
Assoc. Prof. Russel Pears
Dr. Kate Lee

prepared at The School of Computing and Mathematical Sciences, SCMS

submitted on 27.11.2014

School of Computing and Mathematical Sciences

Abstract: This research addresses two key issues in high speed data stream mining that are related to each other. One fundamental issue is the detection of concept change that is an inherent feature of data streams in general in order to make timely and accurate structural changes to classification or prediction models. The shortcomings in the past research were addressed in two versions of a change detector that were produced during this research. The second major issue is the detection of recurring patterns in a supervised learning context to gain significant efficiency and accuracy advantages over systems that have severe time constraints on response time to change due to safety and time critical requirements. Capturing recurrent patterns requires the detection of concept change with minimal false positives. This research addresses this latter problem as a pre-requisite to formulating a novel mechanism for recognizing recurrences in a dynamic data stream environment.

The first approach to change detection, termed SeqDrift1 that relies on a detection threshold derived using the Bernstein bound and sequential hypothesis strategy ensured much lower false positive rates and processing time than the most widely used change detector, ADWIN.

The second version of the change detector, SeqDrift2, achieved significant improvement on detection sensitivity over SeqDrift1. This was achieved through two separate strategies. The first was the use of reservoir sampling to retain a larger proportion of older instances thus providing for better contrast with newer arriving instances belonging to a changed concept. The second strategy was to trade off false positive rate for detection delay in an optimization procedure. The net result was that SeqDrift2 achieved much lower detection delay than SeqDrift1 but sacrificed some of its false positive rate

when compared to SeqDrift1, while still retaining its superiority with respect to this measure vis-à-vis ADWIN and other change detectors.

Having proposed a robust and efficient mechanism for change detection two different meta- learning schemes for recurrent concept capture were proposed. A novel framework using the two schemes consists of concept change detectors to locate concept boundaries, a Hoeffding tree compressor to exploit the application of Discrete Fourier Transform on Decision Trees to produce compact Fourier Spectra, a forest of Hoeffding Trees to actively learn and a pool of Fourier spectra to be reused on similar recurring concepts.

In the first scheme, termed Fourier Concept Trees (FCT), each Fourier spectrum is separately stored and reused on similar concepts. Accuracy and memory advantages have been empirically shown over an existing method called, MetaCT. In the second scheme, instead of storing each spectrum on its own, an ensemble approach, Ensemble Pool (EP), was adopted whereby several spectra were aggregated into single composite spectrum. The major advantage of this strategy over the first was the reduction in storage overhead as redundancies in separate spectra are eliminated by merging into one single entity. In addition, Fourier spectrum generation is optimized with theoretical guarantees to suit high speed environments. Extensive experimentation that demonstrated the benefits including accuracy stabilization, memory gain, reusability of existing models etc., has been done with a number of synthetic and real world datasets. This includes a case study on a Flight simulator system which is one of the target applications of this research.

Keywords: Recurrent Concepts, Concept Drift Detection, Data Stream Mining,Discrete Fourier Transform, Bernstein Bound, Sequential Hypothesis Testing, Reservoir Sampling

Contents

List of Figures

List of Tables

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

.....

Signature

Chapter 1

Introduction

1.1 High Speed Data Mining and its Research Challenges

Data is an invaluable resource in this information technology era. Even before the start of this era data was recorded and stored but the importance of it was hardly realized to the extent that is reflected in contemporary times. Data records often describe the past and embedded in them are trends that describe the behavior of the underlying data generation mechanism. Prior to the information era, discovery of trends was very limited due to enormous degree of manual work involved. Furthermore, methods to capture hidden trends did not exist or were simply not pragmatic.

Only a few organizations such as governments and large manufacturing plants recorded data about people, employees, production, products etc. The main purpose of collection of data was simply to query in order to retrieve more details attached to a record. Introduction of computerized database management systems minimized the manual work involved with the evolution of computers. However, the requirement of organizations started to change from *just queries* into *analysis* of the past to prediction of the future. This is when data mining emerged as a field of research and application. Data mining can be categorized into: Classification and Prediction, Clustering, Association rule mining, Anomaly detection, Regression and Summarization [Fayyad 1996].

Methods that support each of these categories can be used to reveal hidden information about the data. The retrieved information is then converted into knowledge that is directly useful to the end users, for example a business organization. Traditional data mining techniques were designed to be applied on static data chunks. Therefore, several algorithms used in traditional data mining utilize statistics or the properties of the entire dataset. With the advancement of communication technologies, real time data transfers have been increasingly becoming popular. Banks, large retailers, astronomical observatories, weather forecasting centers, financial markets and Internet applications are a few industries that generate and use the data obtained from real time data transfers. The availability of data flowing in real time was the trigger for real-time data analysis. As a result of this new requirement of real time data processing, data stream mining techniques emerged. Several algorithms and methodologies have been proposed aiming at each of the above data mining categories. In addition, stream mining imposes additional challenges. The below is a list of such challenges [Gaber 2005] [Krempl 2014].

- Continuous learning as new data stream instances arrive and incremental update to classifiers
- Uncertain future instances question the applicability of current models in use
- Detecting changes in concepts in underlying data stream to know when and where to perform modifications to current models to cope with new concepts
- Memory and computational complexity problems of the algorithms and data structures due to infinite nature of data streams
- Theoretical bounds on performance of algorithms

- Optimal use of models constructed and performance stabilization over changing data streams
- Interactivity and visualization of mining outcomes and models
- Preprocessing of data stream to remove noise, to reduce number of features etc. to improve mining activity and model and memory complexity
- Real time performance evaluation even in a very high speed data stream
- Prediction of future status of the data stream using the historical statistics gathered
- Other requirements of real time applications such as hardware software requirements and limitations etc.

Addressing all the above challenges in a single system is a too ambitious task. Prioritizing a few of the above challenges is more feasible. This research focuses on designing a system that is not susceptible to changing concepts in a data stream. The research primarily addresses the challenge to stabilize model performance through optimal use of past models. Due to strong dependence of the solution to this challenge on change detection, substantial focus is laid on constructing a change detector that could positively support the primary objective of performance stabilization. Moreover, models are compressed and aggregated to save memory consumption and processing overhead to favor high speed data streams.

The solution proposed by this research has a number of attractive properties and benefits in application. As mentioned earlier, it is a necessity to have consistently good performance in terms of accuracy and processing speed in time critical applications such as auto-pilot systems, military applications, stock markets, disaster monitoring systems, patient monitoring systems and driving systems found in driver-less trains and cars. Incorrect predictions and

classifications even for a very short period of time causes serious damages to end users and equipment. In all of the above examples, there is a high probability that a previously seen situation reappears in future. An autopilot system faces similar weather patterns, similar flight segments such as take off, cruise and landing and similar flying paths over a number of different flights. Similarly, stock markets show similar trends and scenarios over a period of time. Relearning under similar situations is not only a waste of effort but has the possibility of long delay to recover from the impact of a concept change. The simple, practical and effective scheme would be to reuse a model constructed during the previous occurrence of the current concept in order to prevent relearning and thus minimize delay. An attempt to reuse knowledge from the past whenever concepts recur in a data stream is the problem of capturing recurring concepts in machine learning research.

In past research, recurrence concept capture has received little attention. There are a number of reasons for this: Data Stream Mining was at its initial stage of development, applications of Data Stream Mining hardly existed, the modules to design a Recurrence Capturing Framework such as Concept Change Detector were poor in performance. Nowadays, recurrent concept capture has started to attract the attention of an increasing number of researchers.

Previous research laid its focus on this problem based on a few different approaches. Among such approaches are: the construction and reuse of concept representations by caching representative data instances or classifiers [Alippi 2013]; classifier adaptation mechanisms [Lazarescu 2005] and ensemble based methods [Ramamurthy 2007], [Katakis 2008] are a few of which have proved to be promising. Some solutions are dependent on concept change detectors to recognize significant changes in the underlying data stream and to trigger specific tasks to store or reuse an existing concept representation. Models presented in [Gama 2011] and [Gomes 2010] are examples that are

dependent on Concept Change Detectors. These models rely on the performance of Concept Change Detectors. False positive change detections and long delay in detection negatively impact recurrence capture. Additionally, required parameters of change detectors are hard to set and adjust on data streams until ADWIN [Bifet 2007], which is essentially a parameter free method [Bifet 2007] was introduced. ADWIN has been widely used in many applications and praised for its sensitivity even on slowly varying data streams. The major drawback of ADWIN is its high false positive rate and noise intolerance, as shown in [Pears 2014]. Reducing the maximum allowable false positive rate parameter has a negative impact of decreasing sensitivity to changes. The other change detectors like Page-Hinckley Test [Mouss 2004] [Page 1954], Gama's method [Gama 2010], EDDM [Baena-García 2006] and EWMA [Ross 2012] charts have poor performance compared to ADWIN and are ineffective in practice in coping with various degree of changes in underlying data streams. There is a high correlation between a concept change and the performance of current model. Concept change often causes a decrease in accuracy of the current model. All of the above change detectors are designed to monitor the error rate of a model to flag a concept change. The other strategy is to recognize changes in concepts directly from data stream instances. This approach is rarely used due to its high computational complexity and several limitations on its existing implementations.

This research addresses a number of shortcomings in previous approaches in addition to proposing enhancements for capturing recurrent concepts and reuse of classifiers. Two change detectors, namely SeqDrift1 and SeqDrift2 that monitor classifier accuracy to recognize statistically significantly decreases in accuracy in order to flag a concept change have been proposed. The SeqDrift algorithms have been designed with an improved test statistic, data structures, and algorithmic optimizations to minimize false positive rate and to process data stream instances faster than ADWIN, while achieving com-

parable detection delay. A detailed experimentation has also been done comparing their performances against a number of other change detectors.

The next section presents the key objectives of this research.

1.2 Objectives

This research is aimed at a solution that reuses past models when similar concepts reappear in a high speed data stream. At a granular level, the purpose of this research is to assess the effectiveness of Discrete Fourier Transform of Decision Trees in a recurring concept environment. The Discrete Fourier Transform has long being recognized as a mechanism for capturing recurrences and it would be interesting to test its effectiveness in capturing recurrences of concepts represented by decision trees built from data in a data stream environment.

The novel framework that is introduced in this work depends on a concept change detector to identify when to update the current classifier model to suit new concepts in a data stream.

False signals generated by a change detector have the potential to introduce instability in classifier performance. Moreover, a change detector should have the capability to process data instances at the speed that they arrive. Though there are a number of change detectors proposed in the literature as mentioned earlier, each of these suffers from one or more of key performance problems such as high detection delay, high rate of false signals, high processing speed, complex parameter optimization procedures or high memory consumption. Therefore, further objectives of this research are set at implementing a change detector that has all the above performance metrics at an acceptable level in a concept recurring data stream. In addition to empirical evidence, this research is aimed at providing theoretical guarantees on the performance of the proposed models, wherever applicable. These objectives are explored through

the use of the research questions listed in the next section.

1.3 Research Questions

This section presents the set of research questions to achieve the objectives of this research.

- How do we reduce the false positive rate while ensuring that the other key performance measures such as processing time, detection delay, true positive rate, noise tolerance and memory consumption are competitive with respect to the current state of art change detectors?
- How does the Discrete Fourier Transform (DFT) perform in relation to the storage and computational overheads when compared to standard methods of recurrent concept capture?
- Are DFT encoded concepts capable of generalizing to new forms of concepts that have appeared in the past? A better generalization ability will improve the recurrence capture rate and lead to better classification accuracy.
- What are the alternative schemes for encoding concepts using the DFT? How does the aggregation of DFT encoded concepts effect standard performance metrics when compared to a non-aggregation strategy?

The next section defines the scope of this study.

1.4 Scope

The scope of this research is limited to the application of the Discrete Fourier Transformation (DFT) on Hoeffding Decision Trees [Hoeglinger 2007]. with the assumption that this provides a good representation to capture recurring

concepts in highly compressed form. Hoeffding Decision Trees represent the modified version of the standard decision tree algorithm that has been adapted to suit incremental learning in a data stream environment. There are a number of variations of Hoeffding Decision Trees such as the Hoeffding Adaptive Tree, the Hoeffding Option Tree [Bifet 2010a],[Hulten 2001] that have been proposed in literature.

As mentioned earlier, the application of the DFT on Decision Trees have been shown to produce compact schema which are highly compressed versions of the underlying decision tree without compromising on classification accuracy [Kargupta 2004], [Kargupta 2006]. However Kargupta et al's research was in the distributed data arena and not aimed at capturing recurring concepts in data streams. Thus the application of the DFT will enable the main hypothesis to be tested in this research which is that *the DFT will enable recurring concepts to be captured in a memory efficient manner while preserving or enhancing the classification accuracy achieved by standard incremental decision tree classifier such as the Hoeffding Tree.* The framework and methods developed used in this research can easily be used on variants of the Hoeffding Tree.

In addition, the scope of the work on change detection is defined to construct a change detector that is versatile enough to operate in any type of data stream environment and recognize either gradual concept change or abrupt change with high sensitivity.

The experimentation in this research tracks performance metrics such as accuracy, processing speed, noise tolerance, memory consumption, model stability, ability to recognize recurring concepts and comparative delay in recovery due to a concept change. As the base model, an implementation of a Hoeffding Tree forest (CBDT) [Hoeglinger 2009] forest is used.

As experimental testbeds, the benchmark synthetic data generators as well as real world datasets, including a case study that exemplifies the problem

addressed by this research that is the recurring concepts, are used.

1.5 Overview of Research Strategy

The overall research strategy is presented briefly in this section. Two change detectors are developed with different objectives to suit the recurrence capturing context. This is because recurrence capture depends on a reliable change detector that is good at locating only the actual change points. In other words, false alarm rate should be minimal to get an undistorted representation of concepts. The first change detector named SeqDrift1 is proposed with the key aim of minimal false alarm rate and high processing speed. The second change detector namely SeqDrift2 has then been presented to optimize detection delay while maintaining the benefits of SeqDrift1. Both change detectors are modeled using statistical techniques such as sequential hypothesis testing and Bernstein Bound. A performance comparison has been made with a number of widely used change detectors namely, Page-Hinckley Test, Gama's methods, EWMA charts and ADWIN. This demonstrates the strengths and limitations of each of the change detectors and helps with deciding on the choice to be used for the recurrence capture model that is the ultimate aim of this research.

As mentioned earlier, to produce a highly compressed model that extracts the essence of a concept that occurs in a data stream, the Discrete Fourier Transform has been selected to be applied on the Hoeffding Decision Tree. Decision trees were preferred over the other classifiers mainly due to its self-interpretability of knowledge captured. The Discrete Fourier Transform produces a set of coefficients that fully captures the classification power of the underlying decision tree. As mentioned earlier, the CBDT Descision Tree forest is the choice of the base classifier model to implement the recurrence capture model. At each concept change point, the winner tree that has the

best classification accuracy over the current concept among all trees in a forest is converted into Fourier form (Fourier Tree) and stored in a repository for future use. At each concept change point, an assessment is also made on all Fourier Trees for a possibility of reuse on the newly emerging concept that has been signaled by the concept change detector. If no existing Fourier Tree is found to be better in terms of accuracy than all of the Hoeffding Trees in the CBDT forest, the system is set to learn the newly emerging concept using the decision trees in the forest. This overall strategy is named as Fourier Concept Trees (FCT) and is evaluated against MetaCT and the base classifier CBDT.

An improvement to FCT is made by exploiting a key property of Fourier spectra that support aggregation of different spectra into one integrated spectral unit. A Fourier spectrum, in essence being a mathematical function, lends itself easily to aggregation via a relatively simple algebraic process which is described in detail in a subsequent chapter of this thesis. This in effect creates a simple way of aggregating two or more decision trees together. In a recurrence capturing context, aggregation of classifiers can have a positive effect by generalizing concept representations to extend the ability to recognize similar concepts that occur in the future. This hypothesis is tested with a model named Ensemble Pool, EP as an extension to FCT. Structurally similar trees are aggregated to produce better representation and to reduce memory and computational overheads. In addition, a number of computational optimizations also has been proposed. Performance is primarily compared against FCT to study the impact of aggregation in concept recurring environment.

In addition, the contribution of a change detector especially with respect to its false alarm rate is also assessed by evaluating FCT and EP with ADWIN and EP change detectors.

The following sections presents the contributions made by this research.

1.5.1 Theoretical Contribution of This Research

This research introduces a strategy to capture recurring concepts using change detectors and a pool of classifiers. Therefore, the key theoretical contribution is the assessment on how well an explicit use of change detectors monitoring classifier error rate and storing previously best fitted classifiers for future use perform in a data stream that has concept recurrences. Moreover, this research is the first work that evaluates whether classifier compression and aggregation is a promising method that yields better performance measures compared to the methods that do not apply the above in concept recurring environment.

The above theoretical contribution is made with proposing a number of new and modified methods in granular level with a number of low level sub-contributions.

Such sub-contributions made in this research are summarized below:

- A new detection threshold for change detection based on the Bernstein Bound that minimizes the false positive rate, along with theoretical guarantees on key performance metrics [Sripirakas 2013], [Pears 2014]
- Novel application of reservoir sampling algorithm in the change detection context to draw a representative sample efficiently within the memory constraints inherent in a very high speed data stream [Pears 2014]
- A novel scheme to compensate for accumulated false positive error due to multiple sequential hypothesis testing in place of the conservative Bonferroni correction [Pears 2014]
- Two novel algorithms, namely SeqDrift1 and SeqDrift2, with significantly better false positive rates than existing change detectors have been proposed. In addition to the above, processing time and noise tolerance advantages over the widely used ADWIN change detector were also obtained [Sripirakas 2013], [Pears 2014].

- Novel application of Discrete Fourier Transform on decision trees constructed in a data stream environment and exploitation of its key properties such as aggregation of decision trees to capture similar recurring concepts [Sripirakas 2014a], [Sripirakas 2014b]

- Optimizations in the derivation of Fourier coefficient generation along with theoretical guarantees [Sripirakas 2014b] on the computational overheads of the inner product operation required to derive Fourier coefficients.

- Two novel algorithms, named Fourier Concept Trees and Ensemble Pool to capture recurring concepts and to reuse stored models when similar concepts reappear in a data stream [Sripirakas 2014a], [Sripirakas 2014b].

- A novel scheme for energy thresholding on the Fourier spectrum that results in a more efficient method for extracting a core subset of significant Fourier coefficients that together preserve a guaranteed amount of energy in the resulting spectrum. This ensures that the resulting spectrum faithfully captures the classification power inherent in the original decision tree while ensuring the resulting spectrum is as compact as possible [Sripirakas 2014b].

- An improved set of experimental strategies to assess a concept change detector and a recurrence concept capturing algorithm [Sripirakas 2014b]

1.6 Publications

- Pears, Sakthithasan Sripirakas and Yun Sing Koh. Detecting concept change in dynamic data streams. Machine Learning,vol. 97(3), pages 259-293, 2014.

- Sakthithasan Sripirakas, 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.

- Sakthithasan Sripirakas and Russel Pears. Mining Recurrent Concepts in Data Streams Using the Discrete Fourier Transform.In Ladjel Bellatreche and Mukesh K. Mohania, editors, Data Warehousing and Knowledge Discovery, volume 8646 of Lecture Notes in Computer Science, pages 439-451. Springer International Publishing.

- Sakthithasan Sripirakas and Russel Pears. Use of Ensembles of Fourier Spectra in Capturing Recurring Concepts in Data Streams. International Joint Conference on Neural Networks. 2015, Ireland.

The algorithms, SeqDrift1 and SeqDrift2 presented in Chapter 4 and 5 are available in Massive Online Analysis (MOA) [Bifet 2010b] software [1].

1.7 Thesis Structure

The next Chapter outlines the basics of data mining and data stream mining identifying the key challenges including recurrence capture and concept change detection. In addition, it proposes a general framework to recurring concept capturing algorithm. Chapter 3 introduces the problem of concept change detection in a data stream environment and presents the state of art of the research. Chapter 4 and 5 propose the two change detection algorithms, SeqDrift1 and SeqDrift2 respectively proposed in the research as an enhancement to the current state of change detectors. Then, recurrence capture in data stream environment is described in Chapter 6 with recent literature and

[1] https://code.google.com/p/moa/

an algorithm called, FCT that applies Discrete Fourier Transform (DFT) on decision trees to capture recurring concepts. Chapter 7 presents an extended version of FCT, termed EP that exploits DFT by aggregating Fourier spectra of similar concepts with two computational optimizations on Discrete Fourier Transform. Chapter 8 evaluates the performance of FCT and EP when coupled with SeqDrift2 change detector on a real world dataset that approximates a data stream which is the target application of this research. The research achievements, limitations, future directions and work and open questions are discussed in Chapter 9.

Chapter 2

A General Framework for Capturing Recurring Concepts

This chapter summarizes the evolution of data mining research along with the associated problems and challenges faced in capturing recurring concepts in a data stream environment. Beginning with a traditional broad definition of data mining, the focus is laid on specialized environments such as data streams and recurring concepts. Finally, it proposes a general framework for integrating all components together in order to capture recurring concepts.

2.1 Data Mining and its Components

Data Analysis is the process of analyzing data with a view to extracting useful information from it. The information extracted is significantly useful in virtually any type of application including Business, Crime analysis, Health Informatics, Scientific Data Analysis etc. Beginning as a data collection process, Data Analysis then evolved into Data Management, followed by Data Warehousing and Decision Support through the use of On Line Analytical Processing (OLAP) technology. In the next phase of evolution Data Mining emerged. With numerous real world applications producing data streams, a new category of Data Mining has emerged over the last decade or so, known as "Data Stream Mining".

A typical Data Mining system consists of a number of components or sub-

systems. Sub-systems are required to store and access data; a set of algorithms to perform classification, categorization and prediction; a number of methods to represent and visualize data and knowledge; and a number of evaluation methods including data generators and evaluation metrics. Decision trees, Artificial Neural Networks, Genetic algorithms, k-nearest neighbor and rule extraction algorithms are but a few examples of algorithms used in Data Mining. In the classification context, k-fold cross validation, holdout method, random sub sampling, leave one out method, bootstrap, confusion matrix and receiver operating curves are a few of the many evaluation methods used in Data Mining.

Traditional data mining algorithms are specialized to operate on a given training data set in off-line mode rather than on an incoming stream of data arriving in real time. In off-line, models could be learned and tuned on the entire training data set to capture the knowledge hidden in the data. In addition, availability of the full training data set enables static model construction which can then be applied either on historical hold out test data or incoming data not used in the training process. Though there are challenges in learning from a dataset in capturing significant patterns/information, incremental updates to current models constructed are not addressed in traditional data mining contexts. In contrast, data stream mining introduces additional complexities and challenges compared to data mining. The next section briefs what data stream mining is and its components.

2.2 Data Stream Mining and its Properties

Dynamic incoming data are found in many real world applications. Videos, ongoing chats and transactions, real time web monitoring systems, stock exchange applications, incoming sensor data from ground or space based sensors, cellular records and social media data are some examples of data streams

which often require analysis to effectively prepare for future trends in the data.

Data Stream Mining can be viewed as a method that does repetitive and incremental application of data mining techniques. When underlying data changes are compared to the past, the models built need to be updated to reflect new trends. Therefore model learning needs to be carried out on an on-going basis. At the same time, current model needs to be refined incrementally as new data instances arrive. This implies that the evaluation criteria/metrics used in traditional data mining are applicable for data stream mining methods as well. In addition, the properties of data streams such as speed of incoming instances, and delay in receiving true class labels impose new metrics in assessing performance of data stream mining algorithms.

The next section focuses on the new challenges faced in data stream mining approaches. The below is a list of such challenges [Gaber 2005] [Krempl 2014].

- *Continuous learning as new data stream instances arrive.* This involves challenges in developing models from summary statistics to predict future trends in the stream. Furthermore, continuous learning forces models to incrementally learn. Therefore, traditional models designed for a data mining environment should be redesigned to support incremental learning. In addition, computer hardware and software also need to be improved to process high speed streams.

- *Uncertain future instances.* The time dimension of a data stream adds a further complication on the status of current trends or concepts. Thus, in a classification setting, the degree of correlation between the target variable and predictor variables changes over time. This precipitates current models to be outdated. Therefore, a data stream mining algorithm needs to handle such changes and update models accordingly.

- *Detecting changes in concepts.* Many real world data streams carry changing concepts over time. For example, data stream produced from

Sales may embed different concepts due to seasonal effects. The unknown nature of future data instances thwarts model construction that is suitable for every possible variation in a data stream. The only pragmatic solution would be to use different models on dissimilar segments of data chunks or to modify the current model to suit different concepts whenever those appear in the data stream. In order to perform any of the above mentioned actions, changes in stream elements need to be recognized as those occur. A very challenging requirement of such change detection is that it should be independent of domain knowledge or of any particular concept that may occur in any given data stream in order for it to be both generalizable and robust.

- *Memory and computational complexity problems.* Incoming data should be processed in real time, otherwise data is lost. In a high speed high dimensional data stream, it would be impractical to store data from the very beginning of the stream due to limited available memory. Processing such a large dataset would also be a bottleneck in a real world application. In order to satisfy memory and computational complexity requirements, there should be methods that summarize, sample or index data instances. Incorporating such methods should not overload existing models.

- *Theoretical bounds on performance.* Though this is not a unique requirement imposed by data stream mining per se, this introduces extra challenges in a stream mining environment. The uncertainty of data distribution and unbounded nature of data streams are root causes that complicate the specification of bounds on performance on models performing classification, change detection etc.

- *Optimal use of models constructed and performance stabilization.* This is

an important requirement especially in safety critical systems. Changes in data stream elements could make current models obsolete. Construction of new models or modifications to existing models will be made in response to a detected change in data stream. Due to the delay in adapting an existing model into a new one, performance will deteriorate in this learning period. To overcome significant performance degradation, one solution would be to minimize delay by optimizing learning process. The other would be to reuse any existing models that can cope up with the new concept. The first solution would be impractical as sufficient samples of a new concept should be gathered for learning and there is always an inherent upper limit in the ability of any learning process. The second would be a plausible method to regain or maintain the performance of stream mining models by simply switching to an existing model or to an aggregated structure of existing models to face new data instances.

- *Interactivity and visualization of mining outcomes and models.* From an end user's perspective, this requirement may be essential. Models should be capable of describing the patterns captured rather than being just a black box of mathematical values. The Decision Tree is an example of such a descriptive model that can be presented to end users. In addition to detecting concept changes, it would be desirable to describe a concept and its change over time. This will benefit end users to understand and interpret the data generation process and data stream elements.

- *Preprocessing of data stream.* Preprocessing, in theory, could reduce memory and complexity overheads of a high speed high dimensional data stream. However, the presence of concept changes over time makes a consistent preprocessing inapplicable in a data stream environment. Moreover, in an unsupervised preprocessing setting, uncertainty of the

composition of future data instances would be a major barrier for a machine learner to decide on a suitable preprocessing technique.

- *Real time performance evaluation.* This is an interesting problem that needs to be addressed when evaluating data stream mining models. As opposed to traditional, static data mining, the time frames of evaluation metrics should also change as concepts changes in a data stream. Otherwise, performance metrics would show an erroneous real time performance value that does not represent the currency of the data stream mining models. Dependent on concept change detection, evaluation metrics need to be reset or weighted to reflect the importance of current concept rather than being historic.

- *Prediction of future status of the data stream.* Assuming that there is a relationship between the past and the future, meta mining techniques could be used to predict how data stream could behave in the future. Solutions to this problem provides valuable information to prepare models to face the future changes promptly. However, strict theoretical guarantees on such prediction can hardly be given.

- *Minimum delay in learning.* In a highly dynamic high velocity streaming environment, models need to be learned with minimum delay consequent to changes taking place in data stream elements.

- *Other requirements of real time applications.* This includes the requirements such as distributed data streams and processing, communication over networks using compressed formats, specific software and hardware requirements of various devices including mobile phones, merging of multiple data streams of different formats and different dimensions, noisy and missing valued attributes and expectations on predictions such as for how long and how far ahead. Optimizing a data stream mining model

to address any of these problems is very challenging.

The challenge due to high speed is often addressed with classifiers that process instances with high processing speed by means of statistical models that forecast future trends based on data observed in the past. Moreover, data summarization techniques such as sampling used to forecast future trends need to have rigorous guarantees within acceptable probabilistic confidence limits. In addition, to cope with the speed of stream elements, models should fit in fast access memory such as RAM with no reference to secondary storage.

When a data stream is dynamic, models which are kept in memory should be updated to suit new data instances. If such changes occur with high frequency, the length of the stream segment that carries such knowledge will be correspondingly small. As such, this requires classifiers that can operate on small training sets and yet produce acceptable performance. This challenge is hard to resolve as the construction of a mature classifier exhibiting high accuracy needs a statistically large sample. The data stream version of the well-known Decision Tree algorithm called Hoeffding Tree is one such example of a classifier that requires sufficiently large sample sizes to achieve high accuracy. Due to lack of sample elements, the learning process may become unstable with a low accuracy profile in highly dynamic environments.

Identification of change position creates a boundary between outdated and current stream elements thus providing a well-defined reference point for either modification of the existing classifier or storage of the existing version of the classifier in a repository for future use when concepts recur. A delay in recognizing the change point may restrict model updates thus resulting in degraded performance. Thus, alongside with a classifier, a change detection algorithm operating in parallel also should process in real time by coping with the speed of data stream elements.

Some classifiers depend on statistics over the entire data segment avail-

able to construct their model representation. These are not suitable in a data stream environment as only partial/incomplete statistics are available at a time. Therefore, classifiers should only use partial/incomplete statistics to construct the initial model and then incrementally refine its model representation as more instances are seen. As opposed to a standard Decision Tree algorithm such as C4.5 that depends on a full data set, a specialized version known as the Hoeffding Tree [Bifet 2010a] is designed to exploit available statistics.

As changes occur in data, models are invalidated and the current model is modified or replaced by a new model. This leads to loss of information that was gathered in the past. Moreover, learning is duplicated (relearning) if the models are not saved but past trends repeat. As a solution to this problem past models could be archived and reused when similar trends repeat. The focus of this research is on such environments and the unique challenges faced in capturing recurring concepts are presented in the following section.

2.3 Challenges in Recurring Concepts Environments

As this is a specialized data stream environment where concepts reoccur, all issues presented in the previous section should be resolved in addition to the following:

- Remembering and compressing past models to store in a memory-effective manner
- Choosing the right model that suits the current concept in the data stream, from the archived classifier set
- Memory management of archived Classifier set to retain the most reusable/highly

accurate classifiers

- Capturing *exactly recurring concepts* versus *similar concepts*

The first challenge is to decide which models to remember and in what form. It is necessary to store a classifier in a way that it can be reused when a concept recurs in the future. All classifiers constructed need not necessarily be archived. Archival criteria could include the significance of a classifier and its dissimilarity to existing archived models. It is obvious that preference be given to the ones with high performance and those that are distinct from the existing set. At the same time, if classifiers are too large, memory will be flooded soon. To reduce the severity of memory flooding issue, a compression mechanism if applicable is an appropriate strategy to reduce memory consumption and complexity of the model archived.

The next question is that how the classifier that best suits the current concept could be selected from the archived set. A number of strategies may be used. A sample taken from current concept can be used to evaluate the performances of all archived models. Then the best one based on a criterion like accuracy can be used to select the best available model. If class labels are available for the instances of current concept, *current accuracy on the current concept* may be compared to choose the best model rather than drawing a sample of elements. Otherwise, meta learning may be used to identify the best classifier. Whatever the method employed, the method needs to be precise. An incorrect selection of classifiers may lead to artificial instability in performance.

The other issue is the management of memory, especially in memory constrained environments. In data streams that are highly volatile and embed a large number of distinct concepts, a large number of classifiers need to be archived. In cases where memory is insufficient to store classifiers representing all recurring concepts, a strategy to maintain the most significant set of

classifiers should be employed. This strategy should be aimed at minimizing relearning by exploiting the benefits of available archived set.

In many real world environments, concepts rarely recur in the exact form. In other words, concepts usually recur in a similar form. The most significant patterns seen in a concept survive but there will almost inevitably be some degree of difference between the current and its most similar counterpart seen earlier. It would be an advantage if classifiers are refined to only hold the most significant patterns in such environments. Otherwise, a small dissimilarity may be sufficient to disqualify a similar existing archived model from being reused.

In this research, each one of the above mentioned problems is addressed and solutions have been proposed. The next section identifies the components of a framework to capture recurring concepts and shows the interconnections among those.

2.4 A General Framework for Data Stream Mining with Recurrent Concept Capturing

As shown in Chapter 6 there are two major strategies to tackle recurrent concepts in data streams. One method is to create a conceptual representation using stream instances (conceptual maps [Katakis 2008]) whereas the second method is to archive classifiers that performed well on previously seen concepts so that they can be reused in the future. The conceptual maps created could be exploited to construct a new classifier or to adapt the current classifier to suit the current concept. A number of previous works prefer the second strategy due to its simplicity and precision over the first which lacks the ability to create conceptual maps accurately. Also, the second strategy has the added advantage of not requiring additional computation to construct the conceptual

maps. This research also favors the second strategy and enhances it further by optimizing the representation and reuse of archived classifiers.

A general framework for a data stream with a recurrent concept capture capability requires the use of a number of components. Each component is designed to resolve each sub-problem. *A collection of classifiers that actively learn the current concepts, a concept change detector and an archival pool consisting of classifiers* that captured past concepts are the main components of this framework. Figure 2.1 shows how each component interacts with each other in conceptual terms.

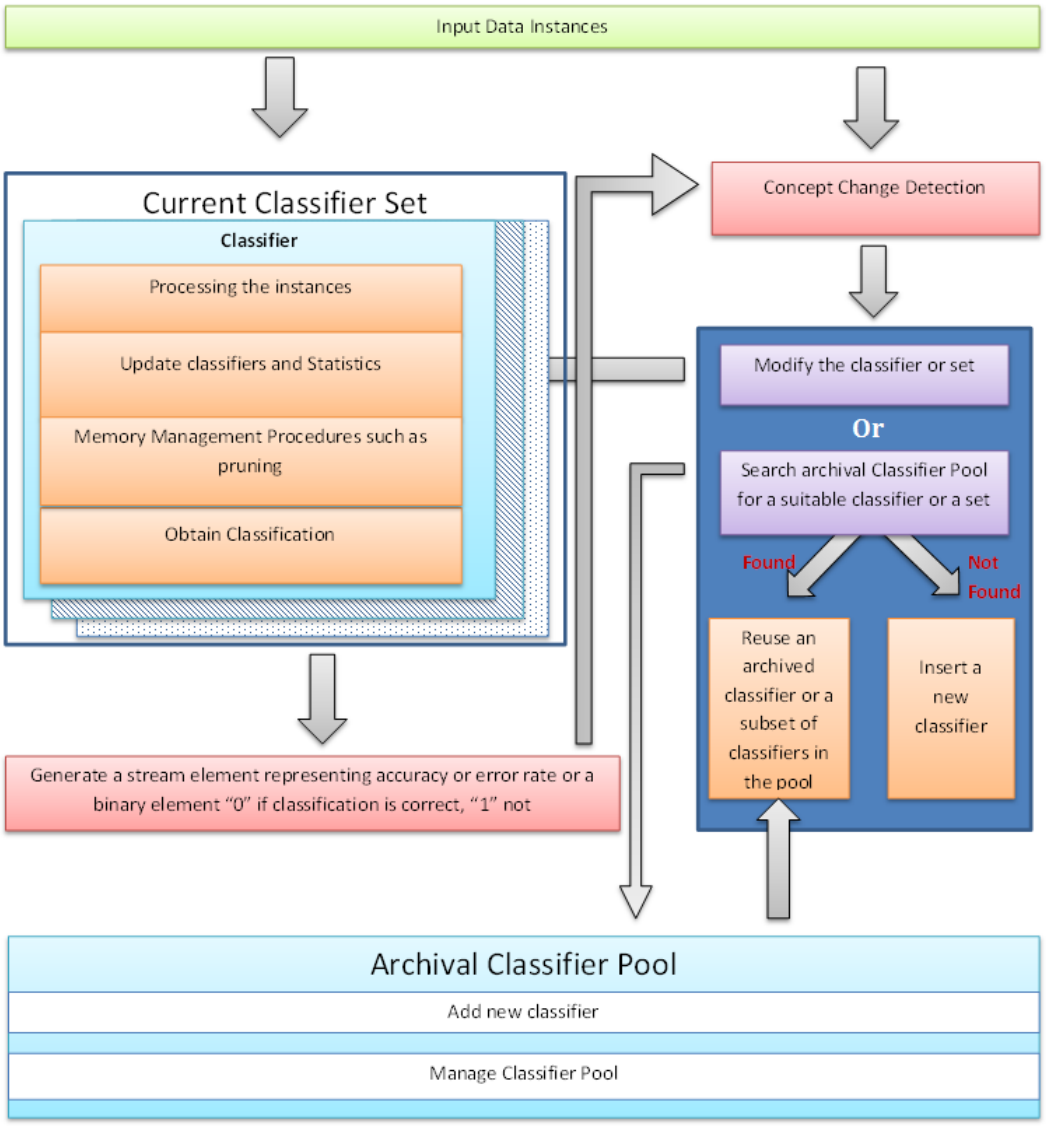


Figure 2.1: A recurrent concept capturing framework for data streams

The classifier component performs classification or prediction by directly processing the incoming data instances. The concept change detector component may either interact directly with data stream elements or indirectly via the classifier. If concept change detection is carried out by directly by processing data stream instances, concept change is signaled by detecting a statistically significant difference in values of the attributes of data instances

between consecutive stream segments [Ho 2005]. Otherwise, classifier output for each data instance in conjunction with the true class label can be used to generate a binary stream of digits, with binary value "1" if the classification for the given data instance is correct or "0" otherwise. This value has to be fed to the concept change detector that monitors the classifier performance.

The next component pair that closely interacts with each other is the current classifier pool and the archival classifier pool. At concept change points, if criteria are met to archive the current classifier, it is transferred to the archival pool. Subsequently, the archival classifier pool is searched to find any reusable classifier that suits the current concept. Classifier update functionality is wholly contained within the classifier component. The test to check eligibility criteria for archival and pool management functionality is embedded in the archival classifier pool component.

Required functionality for change detection such as a statistical test to recognize when statistically significant changes occur in the stream are embedded in the concept change detector component.

This component model allows for parallelization to a greater degree. If multiple classifiers are active to process a single instance, then each classifier could be assigned to a dedicated processing unit in a parallel processing environment. Similarly, while checking for archival criteria, the selection of reusable classifiers could be performed in parallel. If there are multiple concept change detectors monitoring different classifiers or parts of a classifier (sub trees in a decision tree) then each concept change detector can be assigned to different processors for simultaneous processing. This will drastically reduce processing time of the overall system.

Execution of this system involves timely activation of functionality of each component. The next paragraph elaborates the algorithm that implements the framework presented in Figure 2.1.

The algorithm is briefly depicted in Figure 2.2. Incoming data instances

are processed by the current classifier.The Current classifier may be standalone or an ensemble. Once true class labels are available, the accuracy of the classifier could be computed. Changes in underlying concepts often leads to a decrease in accuracy of current classifier. A significant decrease in accuracy indicates a noticeable change in underlying concept. Therefore, accuracy can be monitored by a concept change detector to identify concept changes. At this point, a decision can be made on adapting the system to cope with the current concept. The options are to modify classifiers using the stream instance or to attempt to reuse any of the archived classifiers.

The first method is applied by making structural or statistical changes to classifiers whereas the second requires another decision on whether any existing archived classifier could be reused or not. To implement the second method, it is necessary that outdated classifiers be put into an archival classifier pool right after a concept change detection. At the same time, it is necessary to ensure that duplicate classifiers are not added to this pool in order to avoid unnecessary computation and memory consumption. The decision to reuse any existing classifier can be made with reference to meta statistics that estimates classifier performance or the accuracy on the current set of instances if true class labels are available. A criterion that sets the threshold on the estimated performance or actual accuracy has to be defined to choose the archived classifier that best fits the current concept. This is the second crucial step in the process. If the threshold is set to a low value, there is a possibility of being repeatedly stuck with poor selection of archived classifiers. This would prevent the construction of new classifiers on unseen concepts thus affecting the performance of the system. At the same time, high value of this threshold decreases the usage of archival pool. Thus, the aim to avoid relearning on previously seen concepts may not be achieved. In the event that there is no suitable archived classifier for a reasonable value of this threshold, a new classifier should be constructed.

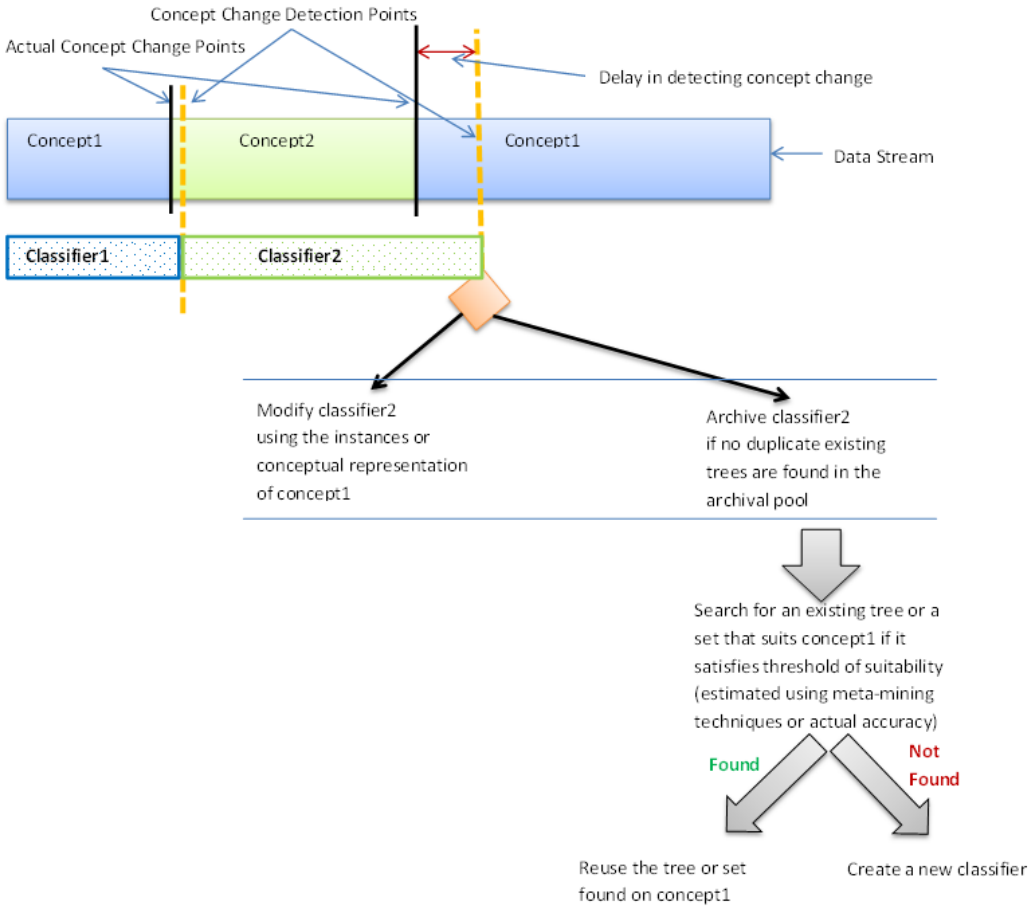


Figure 2.2: The strategy for recurrence capture in a data stream environment

The next complication arises when memory consumption is constrained. This constraint limits the number of archived trees in the pool. A mechanism has to be used to ensure that the set of frequently reusable classifiers are retained while obeying the memory limitation.

The problem of recurrence capture will become more complex if stream instances arrive at very high speed. To cope with high speed, the classifiers must be able to learn and process instances with minimal processing time. In addition, archival pool management and the strategy for classifier selection from the pool should be optimized to minimize the time.

As mentioned earlier in this chapter, the design to capture recurring concepts in exact form has very limited applicability in practice because data streams produce similar, rather than exactly recurring concepts. Therefore, outdated classifiers should be modified only to remember the most significant trends of a concepts rather than being too specific to a certain concept. This goal may be achieved through a pruning or compression mechanism.

This research proposes two different strategies that support the above framework in addition to two novel algorithms to detect concept changes in effective and efficient ways with respect to accuracy, memory usage and computational complexity.

2.5 Summary

In a nutshell, this chapter has presented an overview of the problem *Recurrent Concept Capturing* and its sub-problems with the use of a framework that connects a number of components to provide a comprehensive solution. The crucial steps involved in addressing this problem are *the precise capture of concept boundaries with minimum of delay* and *the maximization of archival classifier pool usage*. A scheme that effectively addresses these issues will improve stability and accuracy in dynamic environments. The next five chapters of this thesis address the major challenges identified in this Chapter.

Chapter 3

Change Detection in High Speed Data Streams

3.1 Introduction

In this chapter, the problem of concept change detection is introduced and its role in a data stream environment is discussed. Concept change occurs when the underlying stochastic data distribution changes, causing changes to concepts and ultimately causing models to degrade in accuracy. In a supervised learning context, two general approaches to concept change detection exist. Firstly, a multivariate approach that tracks whether statistically significant changes have occurred in the set of predictor features taken together. Such a change is taken to signal that a concept change has taken place. The other approach takes explicit advantage of the supervised nature of the learning process and is based on the class itself. The classification error is tracked on a per instance basis and a vector of binary values (0 denoting a correct classification and 1 as an error) is fed to a change detector that detects whether a statistically significant shift has occurred in the error rate. The rationale behind this approach is that a change in error rate signals the arrival of a new concept which is not embedded in the present version of the classifier.

The latter approach has been used in the vast majority of studies and is the approach that is used in this research as well.

3.2 Rationale for Change Detection

Concept change detection has been studied extensively by both the statistical and machine learning communities. The main incentive within the statistical community has been in the manufacturing and process control applications, whereby changes in equipment due to wear and tear over time can cause changes in the quality of products. The machine learning community has a different interest: whether models induced from historical data perform equally well on newly arriving data or whether performance has degraded due to changes in the underlying data distribution. In a data stream environment, data arrives on a continuous basis and concept change causes changes in patterns, thus requiring models to be changed on an ongoing basis and hence a need arises for the automation of the concept change detection process.

The fundamental issue with data stream mining is to manage the sheer volume of data which grows continuously over time. A standard method of coping with this issue is to use a fixed size window of width w, where only the most recent w instances are used to update the model built [Wang 2003a]. While this method is conceptually appealing on account of its simplicity, the major limitation is that concept change can occur at intervals that are quite distinct from window boundaries. If rapid changes occur within a window, then these multiple changes will be not be detected by the mining algorithm thus reducing the effectiveness of the model generated. Ideally a data stream algorithm should use long periods of stability to build a more detailed model whereas in time of rapid change the window needs to be shrunk at each change, the data representing the old concept purged and the model updated with the new concept. Concept change detection with variable-sized adaptive windows has received very little attention compared to the well established area of algorithm development for data stream mining.

The methods proposed for concept change detection all tend to suffer from

limitations with respect to one or more key performance factors such as high computational complexity and memory consumption, poor sensitivity to gradual change or drift, or the opposite problem of high false positive rate. In this research, two novel sequential approaches to change detection, namely SeqDrift1 and SeqDrift2 are proposed with the objective of achieving overall improvement in all the above key performance factors.

The following section formally defines the problem of *Concept Change Detection* before going into the details of related work.

3.3 Change Detection Problem Definition

Let $S_1 = (x_1, x_2, ..., x_m)$ and $S_2 = (x_{m+1}, ..., x_n)$ with 0 <m<n representing two samples of instances from a stream with population means μ_1 and μ_2 respectively. Then the change detection problem can be expressed as testing the null hypothesis H_0 *that* $\mu_1 = \mu_2$ *that the two samples are drawn from the same distribution* against the alternate hypothesis H_1 *that they arrive from different distributions with* $\mu_1 \neq \mu_2$. In practice, the underlying data distribution is unknown and a test statistic based on sample means needs to be constructed by the change detector M to test H_0. If the null hypothesis is not rejected when a change has occurred, then, a false negative is said to have taken place. On the other hand, if M rejects H_0 when no change has occurred in the data distribution, then, a false positive is said to have occurred. Since the population mean of the underlying distribution is unknown, sample means need to be used to perform the above hypothesis tests. A further issue is that the population variance is also unknown in practice and once again it is resorted to estimation, this time by using the sample variance as an estimator. In section 3.6, it is shown that the sample variance converges rapidly to the population variance as sample size increases.

Now, the change detection problem is formally defined as: Reject the null

hypothesis H_0 whenever $Pr(|\hat{\mu_{S_1}} - \hat{\mu_{S_2}}|) \geq \varepsilon) > \delta$, where δ lies in the interval $(0, 1)$ and is a parameter that controls the maximum allowable false positive rate, while ε is a function of δ when test statistics based on the Hoeffding or Bernstein type bounds are used to model the difference between the sample means.

The five evaluation measures that are used in the study are detection delay, false positive rate, false negative rate, memory consumption and processing time. These measures, taken together, cover all aspects of performance pertinent to change detection and hence have been widely used in previous research.

Detection Delay: Detection delay can be expressed as the distance between c and m, where c is the instance at which the change occurred and m is the instance at which change is detected. In other words, detection delay equals: $(m - c)$. It should be noted that in practice the true change point may not be known and in such cases it is only possible to record the difference in delay times between different change point detectors.

False Positive Rate: The false positive rate is the probability of falsely rejecting the null hypothesis for a given test. defined as: $\frac{icc}{nh}$, where icc represents the number of concept changes incorrectly signaled by the change detector measured across a given segment of the stream and nh is the number of hypothesis tests conducted across that same segment.

False negative Rate: The false negative rate is the probability of falsely accepting the null hypothesis when it is in fact true. defined as: $1 - \frac{ccd}{acc}$, where ccd represents the number of concept changes correctly signaled by the change detector measured across a given segment of the stream and acc is the total number of concept changes in the stream that actually occur across the same sized data segment.

Processing Time: Processing time is the time taken by the change detector in performing hypothesis testing to detect possible concept changes in the given stream segment.

3.4 Related Work

The concept change detection problem has a classic statistical interpretation: given a sample of data, does this sample represent a single homogeneous distribution or is there some point in the data (i.e the concept change point) at which the data distribution has undergone a significant shift from a statistical point of view? All concept change detection approaches in the literature formulate the problem from this viewpoint but the models and the algorithmics used to solve this problem differ greatly in their detail.

[Basseville 1993] present extensive coverage of methods for detection of abrupt changes. They categorized change detection into four classes of methods: Control Charts, Filtered Derivative Algorithms, CUSUM based methods and finally methods based on Bayesian inference. All four classes of methods use sliding windowing schemes to compute test statistics that are expressed in terms of a log likelihood ratio that computes the probability of change. The first three classes of methods differ mainly in the choice of threshold used for detection, with the Filtered Derivative and CUSUM approaches using adaptive thresholds. In addition, the Bayesian approaches assume a certain a priori distribution which is used in conjunction with Bayes theorem to compute a posteriori probability of change.

[Sebastiao 2009] present a concise survey on change detection methods. They point out that methods used fall into four basic categories: Statistical Process Control (SPC), Adaptive Windowing, Fixed Cumulative Windowing Schemes and finally other classic statistical change detection methods such as

the Page Hinkley test [Page 1954], Martingale frameworks [Ho 2005], kernel density methods [Aggarwal 2003] and support vector machines [Klinkenberg 2000].

[Gama 2004] adapt SPC methods to the change detection and formulate an algorithm in a data stream context. They use two thresholds for this purpose: when the classification error rate exceeds the lower of the two thresholds an alarm is activated and the system stores a time stamp t_w at which the alarm was generated. If the error rate in the subsequent instances decreases then the warning is canceled, else if the error rate exceeds the upper threshold value at time t_d then a change is declared. The mean error rate p_i and its standard deviation s_i is used to implement the *Warning* and *Change* states. The general form of the model is $(p_i + s_i \geq p_{min} + \kappa s_{min})$ where $\kappa = \alpha$ for change alarm and $\kappa = \beta$ is for warning change, α and β are user defined parameters. Gama's method performs well for abrupt changes but is poor at detecting gradual changes [Baena-García 2006].

A subsequent approach, called Early Drift Detection Method or EDDM [Baena-García 2006] was formulated by Baena-Garcia et al. to address this problem. EDDM tracks the mean distance and mean standard deviation between errors. EDDM was shown to outperform Gama's SPC based method proposed in [Gama 2004] on certain datasets but did not show significant improvement in detecting gradual changes on some of the other datasets.

[Kifer 2004] proposed an implementation of the fixed cumulative windowing scheme. They used two sliding windows, a reference window which was used as a baseline to detect changes and a current window to gather samples. The Kolmorogov Smirnoff (KS) test statistic computed through the use of a KS Tree was used to determine whether the samples arrived from the same distribution. The major issue is the high computational cost of maintaining a balanced form of the KS tree.

[Nishida 2007] and [Kuncheva 2013a] also used the two window approach for change detection. In [Kuncheva 2013a] a semi-parametric log-likelihood

change detector is proposed based on Kullback-Leibler statistics. The authors show that change detection through K-L distance and Hotelling t^2 test can be accommodated in a log likelihood framework. Since the objective was not to propose an optimal detection threshold evaluation the area under the curve was used in place of standard measures such as the true and false positive rates, detection delay and processing time.

A scheme for tackling change detection based on the use of martingale tests was proposed by Ho in [Ho 2005]. Two martingale tests, *Martingale Values* and *Martingale Difference* were proposed based on the use of parametric tests and the impact of these parameters were analyzed. The authors claimed that the method is feasible on high dimensional, numerical/categorical and multi class data. Furthermore, neither is the base classifier monitored for the detection nor is there a requirement for a sliding window as in [Kifer 2004]. The weakness is the setting of appropriate values for the user non-understandable parameters.

[Bifet 2007] proposed an adaptive windowing scheme called ADWIN that is based on the use of the Hoeffding bound to detect concept change. The ADWIN algorithm was shown to outperform the SPC approach and has the attractive property of providing rigorous guarantees on false positive and false negative rates. The initial version, called ADWIN0, maintains a window (W) of instances at a given time and compares the mean difference of any two sub windows (W_0 of older instances and W_1 of recent instances) from W. If the mean difference is statistically significant, then ADWIN0 removes all instances of W_0 considered to represent the old concept and only carries W_1 forward to the next test.

However, as mentioned in [Bifet 2009], ADWIN0 suffers from the use of Hoeffding Bound which greatly over estimates the probability of large deviations for distributions of small variance. As such, a much tighter Bernstein Bound was used in a follow up method, titled ADWIN [Bifet 2007]. ADWIN0

also suffers from high computational cost due to $(n-1)$ hypothesis tests that need to be conducted in a window containing n elements in W.

ADWIN was proposed which used a variation of exponential histograms and a memory parameter to limit the number of hypothesis tests done on a given window. ADWIN was shown to be superior to Gama's method and fixed size window with flushing [Kifer 2004] on almost all performance measures such as the false positive rate, false negative rate and sensitivity to slow gradual changes [Bifet 2007]. Despite the improvements made in ADWIN, some issues remain namely, the fact that multiple passes on data are made in the current window and an improvement in the false positive rate for noisy data environments.

[Ross 2012] propose a method for drift detection based on the use of exponentially weighted moving average (EWMA) chart which is a classical statistical technique for detecting an increase in the mean of a sequence of random variables. EWMA that uses Monte Carlo simulation to find a key control limit parameter L that determines the extent of change in the mean before a concept drift is flagged. As Monte Carlo simulation is computationally expensive it is used for only for a limited number of L values and produces a look-up table, thus enabling the method to be one pass. Ross et al did not conduct a study on the false positive rate, and so the difference between the actual false positive rate and the theoretical false positive rate (as determined by the L parameter) is unclear. Apart from this, the other limitation is that the method's applicability is limited to a small set of alternative formulations for L in the look-up table; a change will require the use of Monte Carlo simulation, thus increasing computational overhead.

Change detection in a non-classification context is also a promising approach in data streams. This approach is found to be more challenging in multi dimensional data streams. Two windows of multi dimensional data stream elements need to be compared to decide whether those are from the

same underlying distribution. Kullback-Leibler distance or relative entropy is the distance measure in [Inglada 2007]. This method uses bootstrapping method to establish statistical significance of the measurements.This method is nonparametric and requires no assumption on the distributions of the elements.In addition, the subregions of the highly varying data are also found in this method. Song et al in [Song 2007] claim that the previous method [Inglada 2007] is not scalable for high dimensional data as it relies on discretization of data space. Song et al proposes a method called density test which avoids space partitioning. In order to infer the baseline distribution a unique Expectation Maximization algorithm with kernel density estimator is used. In [Kuncheva 2013b] proposes a semiparametric log-likelihood change detectors using Kullback-Leibler statistics. The authors show that change detection through K-L distance and Hotelling t^2 test can be accommodated in a log likelihood framework. A computationally simple criterion called semiparametric log likelihood detector is also mentioned in [Kuncheva 2013b].

3.5 Research Contributions

The following major contributions are made in this research with respect to change detection are:

1. Two change detectors, SeqDrift1 and SeqDrift2 that have significantly better false positive rates than the Page Hinkley [Page 1954], EWMA [Ross 2012] and ADWIN [Bifet 2007] change detectors are proposed while maintaining processing times that are competitive with the Page Hinkley detector.

2. Bernstein bound [Bernstein 1946] for detecting changes within the change detection window is used in the detectors proposed. Although the Bernstein bound has been used before in ADWIN, this

research presents a different formulation of the change detection problem to compute a detection threshold that results in significantly better execution time and false positive rate when compared to widely used concept change detectors.

3. The experiments are done with reservoir sampling for implementing the change detection window and demonstrate its ability in improving detection with data that has low gradient of change. Furthermore, instead of using a fixed size reservoir, the size of the reservoir is dynamically varied according to the rate of change in the data stream.

4. The change detectors proposed have strict theoretical guarantees on false positive and false negative rates.

5. A new scheme for compensating for false positive error arising out of repeated hypothesis testing instead of the overly conservative Bonferroni correction is proposed. When combined with the optimized cut threshold value based on variable reservoir size, the new correction factor for false positives enabled the detection delay to be reduced by more than half in certain cases.

6. An enhanced empirical study that subjected the two SeqDrift detectors and ADWIN to varying levels of noise and abrupt concept shifts in order to assess their robustness with respect to the false positive rate is carried out.

3.6 Use of Bernstein Bound in Bounding Deviation of Population Mean from Sample Mean

The approach proposed in this research relies on well established bounds for the difference between the true population and sample mean. A number of such bounds exist that do not assume a particular data distribution. Among them are the Hoeffding, Chebyshev [Hardy 1988], Chernoff [Hardy 1988] and Bernstein inequalities [Bernstein 1946]. The Hoeffding inequality has been widely used in the context of machine learning but has been found to be too conservative [Bifet 2007], over estimating the probability of large deviations for distributions of small variance.

The Hoeffding inequality states that

$$Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - E[X]\right| > \varepsilon\right) \leq 2\exp(-2n\varepsilon^2) \tag{3.1}$$

where $X_1, ..., X_n$ are independent random variables, $E[X]$ is the expected value or population mean.

The Bernstein inequality takes into account the variance and is defined as:

$$Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - E[X]\right| > \varepsilon\right) \leq 2\exp\left(\frac{-n\varepsilon^2}{2\sigma^2 + \frac{2}{3}\varepsilon(c-a)}\right) \tag{3.2}$$

where $X_i \in [a, c]\ \forall i$ and σ^2 is the population variance. In the classification context, $X_i \in [0, 1]$ and thus $a = 0, c = 1$. Also, since $X_i \in [0, 1]$ the maximum value that the population variance can take is $\frac{1}{4}$.

From equations (3.1) and (3.2), when $\sigma^2 \leq \frac{1}{4} - \frac{\varepsilon}{3}$ is satisfied, the Bernstein Bound is guaranteed to be tighter than the Hoeffding Bound. Therefore, for distributions of low variance it is highly likely that the Hoeffding Bound overestimates the probability of large deviations, given that ε is small, as mentioned in [Bifet 2007].

In the preliminary experimentation in this research, the performance of the Hoeffding and Bernstein bounds re-contrasted and found that the latter

produced much smaller detection delays than with the Hoeffding bound, thus influencing the decision to use the Bernstein bound, just as is done with ADWIN.

Population variance, in general, is unknown in a real world data stream environment. An empirical form of the Bernstein inequality has however being used in a number of recent studies in a machine learning context including [Shivaswamy 2010], [Audibert 2007], [Mnih 2008], [Maurer 2009]. In all of these studies, the Bernstein bound was expressed in terms of the observable sample variance rather than the population variance. The empirical Bernstein inequality expressed in probabilistic form is given by [Shivaswamy 2010].

$$Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n}X_i - E[X]\right| > \varepsilon\right) \leq 2\exp\left(\frac{-n\varepsilon^2}{2\sigma^2 + \frac{7}{3}\varepsilon}\right) \tag{3.3}$$

Comparing expressions 3.3 and 3.2, it can be seen that a penalty factor of $\frac{7}{2}$ has been introduced into the bound to compensate for the use of the sample variance. However, this penalty is in practice extremely conservative since it was designed to be applicable for small sized data segments. In the two concept change detectors that are proposed in this research, the minimum data segment size that the variance is sampled from is 200 (this is the block size b, and therefore the minimum size of data buffer). With these segment sizes, the normality distribution assumption holds well and hence a $(1-\delta)$ confidence interval for the population variance is given by: $\left(\frac{(n-1){\sigma_s}^2}{\chi^2_{\frac{\delta}{2}}(n-1)}, \frac{(n-1){\sigma_s}^2}{\chi^2_{1-\frac{\delta}{2}}(n-1)}\right)$, where ${\sigma_s}^2$ is the sample variance, n is the segment size, with $\chi^2_{\frac{\delta}{2}}(n-1)$ and $\chi^2_{1-\frac{\delta}{2}}(n-1)$ being the critical values of the chi-squared distribution at significance levels $\frac{\delta}{2}$ and $1-\frac{\delta}{2}$ respectively. With $\delta = 0.1$ and $n = 200$ the ratio between the two limits of the interval is 1.39, giving an expected deviation from the median of 0.19. With $n = 400$ the ratio is 1.26; with $n = 560$ it is 1.20 and with $n = 800$ it converges to 1.0. Given that both the change detectors proposed use a minimum segment size of 200, the population variance can be approximated very well with the sample variance. The alternative would

be to use the empirical formulation of the bound but as can be seen from the confidence interval limits this option would have been far too conservative and would have resulted in an unnecessary lengthening of the detection delay.

3.7 Summary

This chapter has introduced the problem of Change Detection with the problem definition and a literature review on the recent and relevant works. The contributions of this research with respect to change detection have also been outlined. Moreover, Bernstein Bound that defines the difference between population and sample mean has also been explained. In the following chapters 4 and 5, the two change detectors namely SeqDrift1 and SeqDrift2 that make use of Bernstein Bound are described.

Chapter 4

SeqDrift1: An Algorithm Based on Sliding Window Approach

4.1 Introduction

In this Chapter, the first of two algorithms that were developed for change detection is introduced. The algorithm, called SeqDrift1 accumulates a binary stream of instances representing classification decisions as input to a buffer. In common with ADWIN, the SeqDrift1 change detector uses the Bernstein bound to derive a detection threshold to determine when change takes place in the data stream. Unlike ADWIN, the SeqDrift1 uses a sequential approach to change detection, in the sense that it only examines whether the data in the current block has a statistically different mean value from the data in the rest of its buffer. Thus, for every block of data, only one candidate cut point is examined unlike ADWIN which examines all possible combinations of cut points in its current window. Further, SeqDrift1 never re-examines previous candidates. This approach meant that a new detection threshold had to be formulated. This sequential approach to change detection had two important outcomes. Firstly, a dramatic improvement in execution time results as a direct result of not having to examine multiple cut points at a new block boundary. Secondly, the false positive rate reduced significantly, once again due to the fact that lesser number of cut points are being examined.

4.2 Core Algorithm Overview

First, a basic sketch of the algorithm proposed is presented before discussing details of hypothesis testing. The following simple example is used to illustrate the working of the algorithm. SeqDrift1 accumulates data instances into blocks of size b. When attached to a classifier that uses SeqDrift1 to detect change points, input data instances consists of a binary sequence of bits where binary 1 denotes a misclassification error and binary 0 denotes a correct classification decision. A block of data instances are considered as the basic unit instead of instances as it would both be very inefficient and unnecessary from a statistical point of view to test for concept changes at the arrival of every instance.

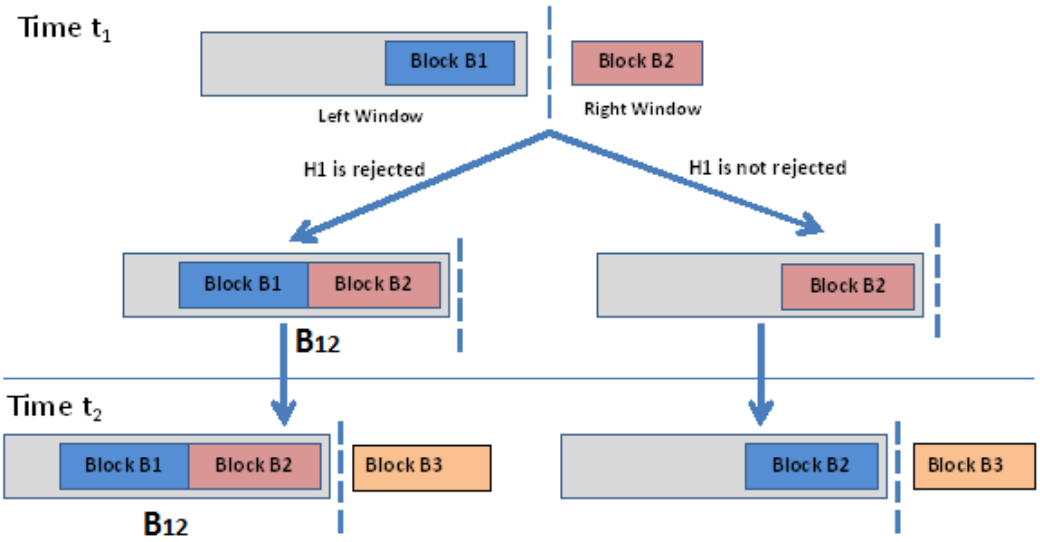


Figure 4.1: A sequential block based approach to change detection

Figure 4.1 shows the strategy briefly. Suppose that at time t_1 blocks B_1 and B_2 have arrived. SeqDrift1 then checks whether a concept change has occurred at the $B_1|B_2$ boundary by testing H_1 above. If H_1 is rejected then blocks B_1 and B_2 are concatenated into one single block B_{12} and H_1 is next

tested on the $B_{12}|B_3$ boundary. In this check, the sample mean of sub-window B_{12} is computed by taking the average value of a random sample of size b from the sub-window of size $2b$. This sample mean is then compared with the sample mean computed from block B_3, also of size b. This process continues until H_1 is accepted, at which point a concept change is declared; instances in the left sub-window are removed and the instances in the right sub-window are transferred to the left. At all testing points, equal sized samples are used to compare the sample means from the two sides of the window. The use of random sampling accelerates the process of the computation of the sample mean while maintaining robustness. The use of the averaging function as seen in the experimentation helps to smooth variation in the data and makes SeqDrift1 more robust to noise than ADWIN.

In essence, SeqDrift1 does a single forward scan through its memory buffer without the use of expensive backtracking as employed ADWIN. While the use of random sampling ensures that sample means can be computed efficiently, a memory management strategy is required to ensure efficient use of memory as the left sub-window has the potential to grow indefinitely during periods of long stability in the stream.

4.3 Memory Management in SeqDrift1

As SeqDrift1 never re-examines previous candidate cut points, it does not need to maintain a history of such cut-points and thus does not need to store memory synopses in the form of exponential histograms as ADWIN does. Instead, SeqDrift1 only requires the means of its left and right sub-windows. In order to efficiently support the computation of sample averages a random sampling strategy is employed.

In addition to improving efficiency, random sampling is also necessary to satisfy the independence requirement for data used in the computation of

the Bernstein bound. In a data stream environment, independence between data instances in the same locality may not always be true as changes in the underlying data causes instances arriving after such a change to have very similar data characteristics, thus violating the independence property. One simple and effective method of addressing this dependence effect is to perform random sampling.

The memory management strategy used in SeqDrift detectors is based on the use of arrays to store blocks of data. An array enables fast access to specific data blocks that are sampled via the use of random sampling. The array is used to capture data in SeqDrift1's memory buffer. The memory buffer is divided into a left sub-window and a right sub-window, each of which uses an array for storage. When a new data block arrives, the block is temporarily inserted into the right sub-window and the sample means from the two sub-windows are compared to check for statistically significant differences. If no such difference exists, data in the right sub-window is copied into the left sub-window and is then removed from the right sub-window. Essentially this means that the left sub-window consists of a set of largely homogeneous blocks. In this context, it is more efficient from a memory point of view to slide the oldest $\frac{w}{b}$ block from the sub-window, where w is the width of the window and b is the data block size.

In certain circumstances, the right sub-window may hold more than one data block. This happens when SeqDrift1 enters a warning state after which newly arriving data blocks are added to the right sub-window instead of the left sub-window. A warning state is triggered when the mean of the data block in the right sub-window is not significantly different from the mean in the left sub-window on the basis of the change confidence value $1-\delta_{change}$ but is significantly different with respect to a warning confidence value $1-\delta_{warning}$. In cases when a warning state is entered, a sliding window scheme is used for the right sub-window as well.

Given the SeqDrift1's worst case memory requirements are bounded above by $2w$ as two memory buffers are allocated of size w for each of the two sub-windows. Different values of w are chosen for the experiments and it is shown that the quality of change detection (false positive rate, false negative rate and detection delay) is largely insensitive to the size of w, provided that w exceeds the block size b.

4.4 Computation of Cut Point Threshold ε

The value of the cut threshold is established against a null hypothesis that the data in the left and right sub-windows are drawn from the same population. The null hypothesis is expressed as: H_0 is $H_0 : \mu_l = \mu_r = \mu$ and the alternate hypothesis as $H_1 : \mu_l \neq \mu_r$. Let S_l = a random sample $\{z_1, z_2, \ldots, z_l\}$of size l from $\{x_1, x_2, ..., x_m\}$ which comprise the m blocks in the left sub-window and let S_r = a random sample $\{z_1, z_2, \ldots, z_r\}$ of size r from $\{x_{m+1}, x_{m+2}, \ldots, x_n\}$ which comprises the (n-m) blocks in the right sub-window. With the application of the union bound on expression 4.1, the following is derived for every real number $k \in (0, 1)$:

$$Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right] \leq Pr\left[|\hat{\mu}_l - \mu| \geq k\varepsilon\right] + Pr\left[|\mu - \hat{\mu}_r| \geq (1-k)\,\varepsilon\right] \tag{4.1}$$

Applying the Bernstein inequality on the R.H.S of Equation 4.1 results:

$$\begin{aligned} Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right] \quad & \leq 2\exp\left(\frac{-b(k\varepsilon)^2}{2\sigma_s^2 \frac{2}{3}k\varepsilon(c-a)}\right) \\ & +2\exp\left(\frac{-b((1-k)\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}(1-k)\varepsilon(c-a)}\right) \end{aligned} \tag{4.2}$$

In the classification context, the bounds a and c for the Bernstein bound take values $a = 0$, $c = 1$. Substituting this in 4.2 results:

$$Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right] \leq 2\exp\left(\frac{-b(k\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}k\varepsilon}\right) + 2\exp\left(\frac{-b((1-k)\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}(1-k)\varepsilon}\right) \tag{4.3}$$

The probability $Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right]$ represents the false positive rate δ and hence:

$$\delta = Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right] \leq 2\exp\Big(\frac{-b(k\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}k\varepsilon}\Big) + 2\exp\Big(\frac{-b((1-k)\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}(1-k)\varepsilon}\Big) \quad (4.4)$$

The RHS of 4.4 needs to be minimized in order to minimize the upper bound δ for the false positive rate. Given the two exponential terms, the RHS of 4.4 can be minimized when:

$$\frac{-b(k\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}k\varepsilon} = \frac{-b((1-k)\varepsilon)^2}{2\sigma_s^2 + \frac{2}{3}(1-k)\varepsilon} \quad (4.5)$$

Note that $k = \frac{1}{2}$ satisfies 4.5 above. Substituting $k = \frac{1}{2}$ in 4.4 gives:

$$\delta \leq 2\exp\Big(\frac{-b\frac{1}{4}\varepsilon^2}{2\sigma_s^2 + \frac{2}{3}.\frac{1}{2}\varepsilon}\Big) + 2\exp\Big(\frac{-b\frac{1}{4}\varepsilon^2}{2\sigma_s^2 + \frac{2}{3}.\frac{1}{2}\varepsilon}\Big) \quad (4.6)$$

Solving 4.6 to find ε gives:

$$\varepsilon = \frac{2}{3b}\left\{p + \sqrt{p^2 + 18\sigma_s^2 bp}\right\} \quad (4.7)$$

where $p = ln\left(\frac{4}{\delta}\right)$. If $|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon$, concept change is declared at instance $(m+1)$ and S_l, S_r can be considered to be from different distributions with probability $(1-\delta)$, otherwise, hypothesis H_0 is accepted that there is no concept change in the window of instances S_n.

A change detection algorithm, by its very nature, needs to test multiple cut points before an actual change point is detected. Each test involves a hypothesis test applied at a certain confidence level. The effect of multiple tests is to reduce the confidence from δ to $\delta^{'}$ which represents the effective (overall) confidence after n successive hypothesis tests have been carried. However, it needs to be noted that the hypothesis tests in the change detection scenario are not independent of each other as the probability of a false positive (i.e incorrectly accepting hypothesis H1 that the means across the left and and right sub-windows are different) at a particular test has an influence on whether a false positive occurs at subsequent tests and hence methods such

as Bonferroni do not apply. This research introduces a new error correction factor that best suits the sequential hypothesis testing in change detection. The following section explains the error correction factor.

4.5 Compensating for Repeated Hypothesis Testing

Repeated hypothesis testing carries with it the risk of increased type 1 errors, which in the concept change context is the increased risk of rejecting the null hypothesis H0 (that the means across the two data repositories are equal) when it is in fact true. The commonly adopted solution to this problem is to use the Bonferroni correction whereby the given δ value is divided by the number of hypotheses (n) tested since the last concept change point detected. However, the Bonferroni correction has been widely acknowledged to be too conservative in nature [Bender 1999], [Narum 2006], by erring on the side of caution in setting the δ value too high and thus decreasing the sensitivity of the detection process. This is due to the fact the hypotheses tested in the change detection problem context are not independent of each other. The motivation is to derive a less conservative correction factor and to give an expression for its update as each hypothesis test is carried out.

A general scenario is considered, where $n+1$ blocks B_1, B_2, ..., B_n, B_{n+1} (with mean values μ_1, μ_2, ..., μ_n, μ_{n+1} respectively) have arrived at time point $n+1$ when n hypothesis tests have been carried out.

Theorem 4.1 *The correction factor $CF(n)$ to be used for the n^{th} hypothesis*

test is given by:

$$CF(n) = \frac{1}{\frac{1}{CF(n-1)} + \frac{1}{2^{n-1}}} \tag{4.8}$$

Proof In the first test (1) on $|\mu_2 - \mu_1|$ on B_1, B_2 boundary, the false positive error resulting from a cut is $\delta^{'} = \delta$. In the second test (2), there are three cases to consider.

Case (1) corresponds to the case when no cut was made on the B_1, B_2 boundary. This case does not contribute to a false positive error rate and hence can be ignored.

Case (2) arises when a cut was made on the B_1, B_2 boundary and when $|\frac{\mu_1+\mu_2}{2} - \mu_3| > |\mu_3 - \mu_2|$. It can be seen that Case (2) does not contribute to the false positive rate as the cut made on the B_1, B_2 boundary has ensured that a reduction in the mean difference $|\mu_3 - \mu_2|$ between the two samples used in the second hypothesis test. This is due to the fact that $|\frac{\mu_1+\mu_2}{2} - \mu_3| > |\mu_3 - \mu_2|$.

Case (3) which corresponds to the situation where a cut was made on the B_1, B_2 boundary and $|\frac{\mu_1+\mu_2}{2} - \mu_3| < |\mu_3 - \mu_2|$. This case increases the false positive rate. The degree of increase has been estimated by enumerating the fraction of all possible scenarios possible with the arrival of 3 blocks in the stream.

With the arrival of 3 blocks there are a total of 6 possible orderings for μ_1, μ_2 and μ_3. These orderings are (in ascending order of value): (μ_1, μ_2, μ_3), (μ_1, μ_3, μ_2), (μ_2, μ_1, μ_3), (μ_2, μ_3, μ_1), (μ_3, μ_1, μ_2), (μ_3, μ_2, μ_1). Out of these exactly half (3 out of 6) cause an increase of $|\mu_3 - \mu_2|$ over $|\frac{\mu_1+\mu_2}{2} - \mu_3|$, and thus contribute to the increase in the false positive rate.

With an assumption of uniformity (equal priors in the general case) of the probability of occurrence of these 6 triples it can be inferred that the false positive error δ in test (1) has contributed an amount $\frac{\delta}{2}$ to the false positive error in test (2), thus giving an overall false positive error of $\delta + \frac{\delta}{2}$ and a correction factor $CF(1)$ of $\frac{2}{3}$ to be applied.

Generalizing this situation to test (3) the overall false positive error at this test of $\delta + \frac{\delta+\frac{\delta}{2}}{2} = \delta + \frac{\delta}{2} + \frac{\delta}{4}$ and a correction factor $CF(2)$ of $\frac{4}{7}$ which is $\frac{\delta}{\frac{1}{CF(1)}+\frac{1}{4}}$. Thus, in general after n hypothesis tests the false positive error is: $\delta + \frac{\delta}{2} + \frac{\delta}{4} + ... + \frac{\delta}{2^{n-1}}$. and the $CF(t)$ to be applied is:

$$CF(n) = \frac{1}{\frac{1}{CF(n-1)} + \frac{1}{2^{n-1}}} \tag{4.9}$$

which proves the theorem.

It can be observed that the correction factor computed above is much less conservative than the Bonferroni correction and converges to $\frac{1}{2}$ for large values of n.

This, in the model the change significance level δ is scaled by the correction factor given by Theorem 4.1 to control the false positive probability. The correction factor computed by Theorem 4.1 is much less conservative than the Bonferroni correction and converges to $\frac{1}{2}$ for large values of n.

4.6 SeqDrift1 Change Detection Algorithm

This section presents the core algorithms used in the change detector system. S_r and S_l denote the right and left sub windows. Algorithm 4.1 decides the change type given the mean values $\hat{\mu_r}$ and $\hat{\mu_l}$ of S_r and S_l respectively, ε_{change} (the threshold mean difference for δ_{change}) and $\varepsilon_{warning}$ (the warning threshold mean difference for $\delta_{warning}$). ε_{change} and $\varepsilon_{warning}$ are calculated using the

equation 4.7. Though SeqDrift1 detects changes in any variation in the mean, algorithm 4.1 only reports the change when mean increases ($\hat{\mu_r} > \hat{\mu_l}$). In the event of a concept change, Algorithm 4.2 transfers the contents of the right sub-window into the left (Step 2 to 8). When a warning state is triggered, it increases the sample size (Step 14), in expectation of a subsequent concept change. This increase has the effect of increasing precision in sampling and the algorithm may become more sensitive to slow gradual change.

`Algorithm 4.1 GetChangeType()`

Input: $\hat{\mu_l}, \hat{\mu_r}, \varepsilon_{Change}, \varepsilon_{Warning}$
Output: $Change \mid\mid Warning \mid\mid Internal$

```
1 if ε_Warning ≤ |μ̂_l − μ̂_r| then
2     if ε_Change ≤ |μ̂_l − μ̂_r| then
3         if μ̂_r > μ̂_l then
4             return Change
5         return Internal
6     return Warning
7 return None
```

```
Algorithm 4.2 IsChange()
Input: An instance(Ins), BlockSize, S_l, S_r
Output: True/False
Increment the instance counter
S_r ∪{Ins}
if At the block boundary then
    ChangeType = GetChangeType()
    if (DriftType is Change or Internal ) then
        Remove all elements from S_l
        Copy all elements of S_r to S_l
        Remove all elements from S_r
        Set SampleSize to BlockSize
        if (DriftType is Change) then
            return True
        return False
    else if (DriftType is Warning) then
        Double the sample size
        return False
    Copy all elements of S_r to S_l
    SampleSize = BlockSize
    return False
```

4.7 SeqDrift1 versus ADWIN: Similarities and Differences

Two major design differences exist between the two change detectors. The first lies in the policy used in determining cuts. When new data arrives, ADWIN creates a new bucket and adds it to its memory buffer. It then searches through all buckets currently stored in its memory buffer for a possible cut point. A cut point in ADWIN lies on the boundary between buckets. With N buckets currently in storage, ADWIN will examine a total of $(N - 1)$ possible cut points. Furthermore, as each new bucket arrives previous bucket

boundaries that were examined before will be re-examined for possible cuts. Effectively, ADWIN makes multiple passes through its memory buffer. In contrast, SeqDrift1 never re-examines previous block (equivalent of ADWIN's bucket) boundaries for cuts and only examines the boundary between the newly arrived block and the collection of blocks that arrived previously for a possible cut. In this sense, SeqDrift1 can be said to do a single pass through its memory buffer when searching for cuts.

The second major difference lies in the estimation strategy for assessing means of data segments. ADWIN relies on exponential histograms for estimating mean values, whereas SeqDrift1 uses random sampling base on an efficient array structure to estimate means. The problem with exponential histograms is that some of the buckets, typically the more recent ones may be too small in size to yield accurate estimations for mean values. This is due to the fact that in ADWIN a bucket is created whenever a 1 appears in the stream, and when data has high variation bucket size will vary widely. For buckets that are too small in size to support accurate estimation, ADWIN will end up overestimating the true mean and false positives may then result.

4.8 Empirical Study

There were two objectives of the empirical study presented in this section. The first objective was to compare SeqDrift1 and ADWIN on key performance criteria such as the true positive rate, the false positive rate, the time delay in detecting changes and the execution time overheads involved in change detection.

In the second part of the experimentation SeqDrift1 was subjected to a sensitivity analysis of the effects of block size, warning level and sliding window size on the delay detection time for SeqDrift1.

Dataset was generated as a Bernoulli distribution, consisting of 1s and 0s

in all experiments to simulate classifier outputs though SeqDrift1 is a general drift detector for any distribution.

4.8.1 Comparative Performance Study

The first experiment was designed to test SeqDrift1's false positive rate vis-a-vis ADWIN. A stationary Bernoulli distribution could create a dataset that has a single concept. Therefore, a stationary Bernoulli has been used for this experiment. To statistically generalize the results, different stationary distributions with different mean values for different experiments have been used. The mean values are shown in Table 4.1. In addition, with each mean value, the parameter of SeqDrift1 **significance level** has also been adjusted to test the effect of this parameter.

The other parameter **block size** of SeqDrift1 was set to its default value of 100 and ADWIN's internal parameter M was also set to its default value.

A total of 100 trials has been conducted for each combination of μ and δ and the average false positive rate for each combination was recorded.

Table 4.1: False Positive Rate of SeqDrift1 and ADWIN for all stationary Bernoulli Distributions and Significance level values

	One Pass Sampler			ADWIN		
μ	δ =0.05	δ =0.1	δ =0.3	δ =0.05	δ =0.1	δ =0.3
0.01	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.0000	0.0000	0.0000	0.0001	0.0002	0.0018
0.3	0.0000	0.0000	0.0001	0.0008	0.0017	0.0100
0.5	0.0000	0.0000	0.0001	0.0012	0.0030	0.0128

Table 4.1 shows that both SeqDrift1 and ADWIN have good false positive rates that are substantially lower than the confidence level set. However, it is interesting to note that as the variance in the data increases with the

increase in the μ value (for a Bernoulli distribution, the variance is $\mu \times (1-\mu)$) that ADWIN starts to register false positives. ADWIN's false positive rate increases progressively with the increase in the variance as well as the lowering of confidence (ie higher significance (δ) values). On the other hand, SeqDrift1 retains a virtually zero false positive rate except when the confidence is low at 0.3 when it registers a rate of 0.01%, compared to the ADWIN rate of 1.28% at $\mu = 0.5$ and $\delta = 0.3$. As the confidence becomes lower, the ε value decreases and this results in an increase in the false positive rate for ADWIN. However, SeqDrift1 is virtually insensitive to the decrease in ε due to the fact that the mean value can be estimated more accurately through the combined use of random sampling and the use of the aggregated running average mechanism.

The second experiment was designed to test the true positive (detection) rates of SeqDrift1 and ADWIN over data that was also generated from a Bernoulli distribution. Four different data streams of length $L = 10,000$, 50,000, 100,000 and 1,000,000 bits from a Bernoulli distribution have been generated. The data generated was stationary with mean 0.01 in the first $L - 2300$ time steps and the distribution has been varied in a linear fashion with different gradients in the last 2300 time steps. Different data stream lengths help understanding the history effect on change detectors.

A total of 100 trials were conducted for each combination of data length and slope values. The key performance indicators such as the true detection rate, average execution time and the detection delay time have been tracked. Both SeqDrift1 and ADWIN managed to achieve a true detection rate of 100% for all combinations of data length and change gradients.

Figure 4.2 also illustrates that ADWIN was much slower in stream processing than SeqDrift1. Furthermore, the gap between the two processing times becomes wider as the length of the stable segment of the stream becomes longer. This was expected as ADWIN spends much time doing repeated scans through the histogram and examines every possible combination of cuts de-

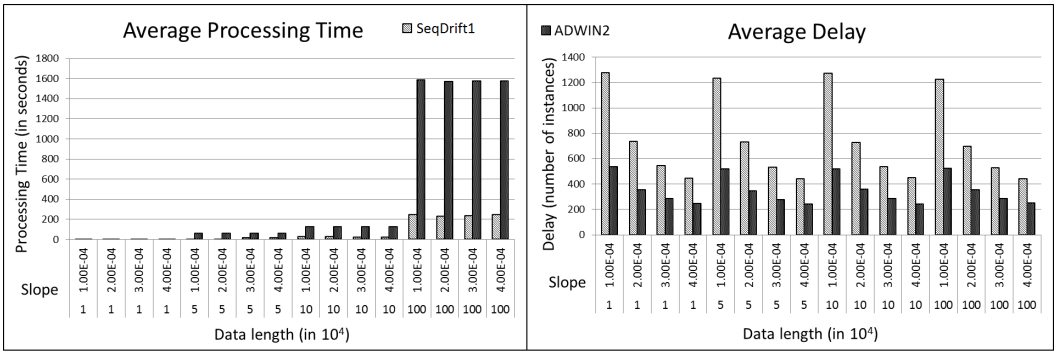


Figure 4.2: Comparative Change Detection Performance of SeqDrift1 and ADWIN

fined by the buckets. SeqDrift1, on the other hand does a single pass through the window segment and at each block of data it assesses whether the newly arrived block is sufficiently different from the previous blocks in its memory buffer.

However, it is clear from Figure 4.2 that ADWIN has better mean detection delay when compared to SeqDrift1. SeqDrift1 needs a relatively larger window segment before it can decide whether a newly arrived block is sufficiently different due to the sampling strategy that it uses. As expected, the delay times reduced with increasing gradient of change, although it is expected that SeqDrift1 reduces at a faster rate than ADWIN with the gap between the two closing for higher gradients of change. Section 4.8.2 shows that SeqDrift1's detection delay can be reduced with proper use of warning level and particularly block size on which it is most sensitive with respect to delay.

The final part of the experimentation involved an investigation of the sensitivity of SeqDrift1's key parameters on detection delay time. From previous experimentation with Bernoulli data, it was observed that SeqDrift1 had a higher detection delay time than ADWIN and thus the motivation was to determine parameter settings that minimize SeqDrift1's detection delay time.

4.8.2 Sensitivity Analysis on SeqDrift1

In the first experiment, the effect of block size is investigated on Bernoulli data streams with different gradients. Figure 4.3 shows that as block size increases, delay time initially decreases, reaches a minimum value and then starts to rise once again. In order to detect changes in data distribution, a sample of sufficient size is required, which in turn is determined by the block size.

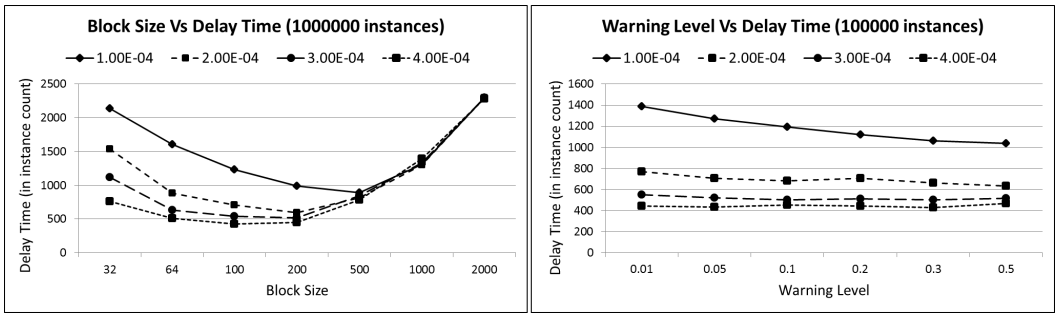


Figure 4.3: Effects of Block Size and Warning Level on Detection Delay Time for SeqDrift1

If the size of the block (sample size) is too low, then in common with other statistical tests of significance, a statistical difference cannot be determined until a greater change occurs with time, thus delaying the detection. On the other hand, if the block size is too large then the probability increases that a change occurs too late within a given block for the change to be detected and so the change will go undetected until at least a new block arrives, thus giving rise to an increased detection delay. A block size of 200 appears to be optimal across a range of different change gradients, except when the change is very gradual, in which case 500 gives a slightly lower delay.

Next, the effect of warning level on delay was checked. Figure 4.3 shows that warning level has a much smaller effect on delay than block size. With a

slope of $1.00E-04$ the warning level setting has a negligible effect on delay and thus a pragmatic setting that is twice the significance level should suffice in most cases to reduce the delay. The other graphs describing the effect of the above two parameters for the other data lengths are included in Appendix A.

In the next experiment, the effect of sample size increment was assessed. Whenever the warning level is triggered the sample size is incremented in the hope of trapping an impending change earlier. A range of increments has been investigated and as Figure 4.4 shows, a doubling of sample size produces optimal results across the entire spectrum. Please refer Appendix A for the results with various other data lengths. As with the warning level, too large an increase results in an increase in the detection delay.

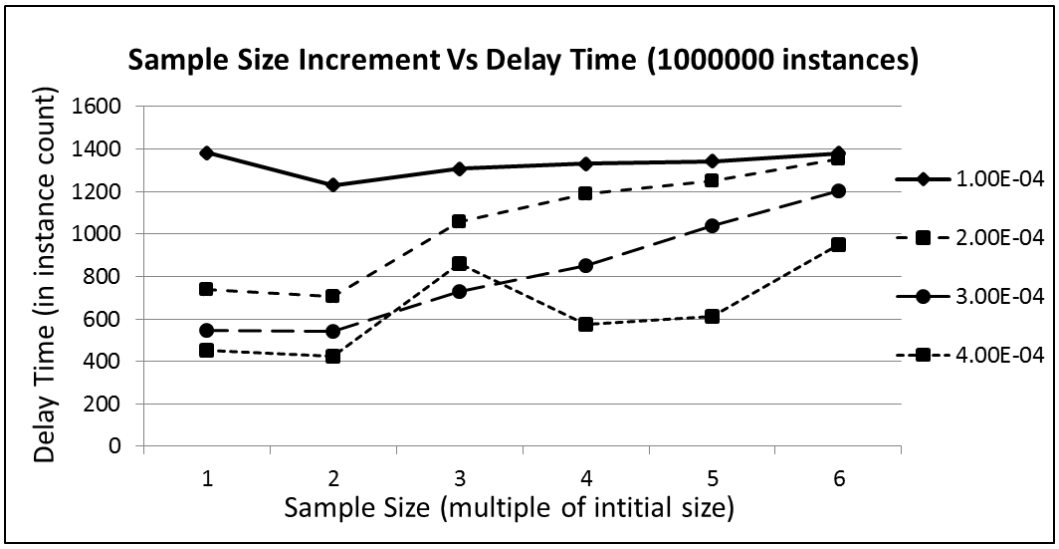


Figure 4.4: Effects of Sample Size Increment on Detection Delay Time for SeqDrift1

Overall, it appears that block size is of prime importance in minimizing delay time; a block size of 200 works well for a range of datasets with different change dynamics. The other two parameters have a much smaller effect in

Table 4.2: Detection delay for varying window sizes

Sliding Window Size	500	1000	2000	4000	6000	8000	10000
True Positive Rate	100	100	100	100	100	100	100
Delay Time	1300	1330	1350	1320	1350	1370	1330
False Positive Rate	0.00000	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000

general but can also contribute to smaller delay times with settings that were discussed above, especially in the case of slowly varying data.

Finally, the effects of the sliding window size on true positive rate have been assessed, false positive rate and delay time. The sliding window size has been varied in the range 500 to 10,000. For each window size, 30 trials were conducted on data from a Bernoulli distribution and the average for each of the performance measures were recorded. The detection delay for the smallest change gradient of $1.00E-4$ is only presented because the results for the other change gradients followed very similar trends. As Table 4.2 shows, the detection delay is largely insensitive to window size. In addition, all window sizes recorded a true positive rate of 100%. The false positive rate was in line with the other two measures, virtually no change in rate was observed across the entire range of window sizes used. The case with mean value 0.3 and delta 0.3 is only shown here. All other combinations of mean and delta returned virtually identical results.

These results indicate that window size when set at a reasonable multiple of block size has no significant effect on key factors such as the true positive rate and delay time. These results are to be expected as data that is slid out of the window consists of a set of homogeneous instances from SeqDrift1's left sub-window.

4.9 Summary

In conclusion, it is clear that SeqDrift1 signals relatively very less false alarms compared to ADWIN even in a high variance environment while achieving 100% true detection rate. Both warning level and sliding window parameters produces negligible effect on SeqDrift1's performance for reasonable parameter values. The key disadvantage of SeqDrift1 is the high detection delay compared to ADWIN. The reasons for the high detection delay would be as follows:

- Insensitivity to small variations in means of both windows due to high ε value
- Inability to capture good history of a distribution in left sub window due to limited memory and first come first out policy.
- Difficulty in finding optimal parameter values manually

Therefore, SeqDrift1 is optimized addressing the above issues as SeqDrift2 in the next chapter.

Chapter 5

SeqDrift2 Change Detector

5.1 Introduction

In the previous Chapter, it was seen that although SeqDrift1 had significant execution time and false positive rate advantages over ADWIN. ADWIN outperformed in the area of detection delay, particularly for slowly varying data. This is a direct consequence of restricting cut point determination to block boundaries instead of per instance. In order to increase sensitivity of SeqDrift1, two strategies are presented in this Chapter. The first is based on a new derivation of the evaluation cut point threshold. Essentially, false positive rate is traded off for improved detection delay. Secondly, reservoir sampling is employed to implement the detection window. With reservoir sampling, older instances tend to persist in the detection window thus allowing for a better contrast between the window and the newly arriving block of data which in turn helps to trigger the detection threshold earlier.

5.2 SeqDrift2 Design Fundamentals

SeqDrift2 uses the same basic sequential hypothesis testing strategy as SeqDrift1 (refer Chapter 4 but, contains a number of important enhancements, including the use of reservoir sampling for memory management and the use of a much tighter bound for the ε cut threshold.

5.3 Memory Management within SeqDrift2

Whilst memory management in SeqDrift1 is efficient, the integrity of the sampling process may be compromised by sliding out data instances older than the chosen window size w. At any given time point in the progression of the stream, all instances that have arrived since the last cut point should have an equal chance of being sampled. This is not the case with SeqDrift1 as older instances are not available, resulting in a possible loss of sensitivity in detection of concepts that change very gradually. With slowly varying data, older instances should be preserved as much as possible in the left repository L so as to provide the best possible contrast with newer instances that arrive in the right repository R. The higher the contrast in the means, the greater is the chance of detecting change.

In SeqDrift2, an adaptive sampling strategy based on reservoir sampling is adopted. The reservoir sampling algorithm was proposed by [Vitter 1985] and is an elegant one pass method of obtaining a random sample of a given size from a data pool whose size is not known in advance. Thus, this algorithm is ideally suited to a data stream environment.

In reservoir sampling, a data repository of a certain size is first filled with instances that arrive in the stream. Thereafter, each subsequent instance will replace a randomly chosen existing instance in the repository with a probability that diminishes with each new instance that arrives. Over a period of time, the repository will consist of a mix of older and newer instances, with the exact mix being determined by the length of the stream segment measured from the last cut point. Unlike with the sliding window approach adopted in SeqDrift1, there is a non-zero probability of an instance surviving that arrived more than s instances prior to the current instance. Thus, the left repository L is implemented as a reservoir and use the reservoir sampling algorithm for its maintenance. The right repository R, as in SeqDrift1 remains as a temporary

staging area to store the newly arrived block in the stream.

In addition to improving sensitivity, another advantage of the reservoir sampling approach is the computational efficiency in maintaining and sampling it against the complexity of maintaining and accessing structures like exponential histograms [Datar 2002] and this was the main reason why this strategy of implementing the data repositories was adopted.

SeqDrift2 also uses Bernstein Bound to compute test statistic to compare the mean values of L and R. The use of the Bernstein bound in SeqDrift2 is reviewed in the next section.

5.4 Use of Bernstein Bound in SeqDrift2

Bernstein Bound described in Chapter 3 is again the fundamental mechanism on which the test statistic is based. However a different derivation is used to increase its sensitivity, while assuring that the false positive rate does not exceed a user defined threshold. Moreover, the use of Bernstein Bound in SeqDrift2 also requires the population variance. The same argument in the previous chapter that proves that sample variance is an unbiased estimation in Change Detection context holds for SeqDrift2 as well.Therefore, sample variance of the two segments of data streams replaces population variance in calculating test statistic.

5.5 Cutpoint Threshold for SeqDrift2

Let $\hat{\mu_l}$, $\hat{\mu_r}$ represent the sample means on the left and right repositories.

Theorem 5.1 ***False Positive Guarantee for SeqDrift2.***
When $Pr\left[|\hat{\mu_l} - \hat{\mu_r}| \geq \varepsilon\right] < \delta$ *with* $\varepsilon = \frac{1}{3(1-k)n_r}\left(p + \sqrt{p^2 + 18\sigma_s^2 n_r p}\right)$ *the proba-*

bility that SeqDrift2 makes a false cut is at most δ, where $p = ln\left(\frac{4}{\delta}\right)$, $k = \frac{n_r}{n_l+n_r}$ and n_l, n_r are the sizes of the left and right repositories respectively.

Proof In the computation of the ε cut point threshold for SeqDrift1, the value of k that weighs the relative contributions of the means from the left and right repositories to be $\frac{1}{2}$ are fixed. However, this value of k may not be optimal and in SeqDrift2, an alternative method for computing the value of k is presented.

The Bernstein inequality is defined as:

$$Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - E[X]\right| > \varepsilon\right) \leq 2\exp\left(\frac{-n\varepsilon^2}{2\sigma^2 + \frac{2}{3}\varepsilon(c-a)}\right) \tag{5.1}$$

where $X_i \in [a, c]\ \forall i$ and σ^2 is the population variance. Applying the union bound, the following can be derived for every real number k $\in (0, 1)$:

$$Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right] \leq Pr\left[|\hat{\mu}_l - \mu| \geq k\varepsilon\right] + Pr\left[|\mu - \hat{\mu}_r| \geq (1-k)\,\varepsilon\right] \tag{5.2}$$

Applying the Bernstein inequality on the R.H.S of expression 5.2 results:

$$Pr\left[|\hat{\mu}_l - \hat{\mu}_r| \geq \varepsilon\right] \leq 2\exp\left(\frac{-n_l k^2\varepsilon^2}{2\sigma_l^2 + \frac{2}{3}k\varepsilon}\right) + 2\exp\left(\frac{-n_r(1-k)^2\varepsilon^2}{2\sigma_r^2 + \frac{2}{3}(1-k)\varepsilon}\right) = \delta' \tag{5.3}$$

where δ' represents the empirical false positive rate based on an ε threshold computed; σ_l^2, σ_r^2 represent population variances across the left and right repositories respectively.

The optimization of the false positive rate involves finding an ε value that is larger than the difference in means between any two randomly selected data segments from a stable stream where no concept change occurs. From

equation 5.3, it can be understood that k, σ_l^2 and σ_r^2 contribute to the value of ε for a given $\delta^{'}$. Note that the contribution made by the two variance terms is proportionately less in low variance environments, thus requiring an increase in the relative proportion contributed by k in such environments. At the same time, in low variance environments an underestimation in the value of ε will increase the risk, δ', of false positives occurring. In the limit, as variance approaches zero, the risk δ' becomes higher unless k is appropriately set to offset the loss in contribution from σ_s^2 to the ε value. This suggests that the determination of both k and ε should be driven by low variance environments. In the limit as σ_l^2 and σ_r^2 approach zero in expression 5.3 results:

$$\delta' = 2\exp\Big(\frac{-3n_l\varepsilon k}{2}\Big) + 2\exp\Big(\frac{-3n_r\varepsilon(1-k)}{2}\Big) \tag{5.4}$$

Equation 5.4 represents an equation with two variables k and ε. The goal is to find functions for k and ε that optimizes δ' for a given window configuration having sizes n_l and n_r. To do this, partial derivatives of δ' with respect to k and ε are taken and each of them are set to zero, giving:

$$\frac{\partial\delta'}{\partial\varepsilon} = 2\Big(-\frac{3n_l k}{2}\Big)\exp\Big(\frac{-3n_l\varepsilon k}{2}\Big) + 2\Big(-\frac{3n_r(1-k)}{2}\Big)\exp\Big(\frac{-3n_r\varepsilon(1-k)}{2}\Big) = 0 \tag{5.5}$$

$$\frac{\partial\delta'}{\partial k} = 2\Big(-\frac{3n_l\varepsilon}{2}\Big)\exp\Big(\frac{-3n_l\varepsilon k}{2}\Big) + 2(-1)\Big(-\frac{3n_r\varepsilon}{2}\Big)\exp\Big(\frac{-3n_r\varepsilon(1-k)}{2}\Big) = 0 \tag{5.6}$$

Equations 5.5 and 5.6 yields, $k = \frac{n_r}{n_r+n_l}$. In other words, k is determined by equating the two exponent terms in expression 5.3. Note that this solution for k is a generalization of $k = \frac{1}{2}$ used for SeqDrift1.

An alternative way of formulating the optimization problem is to visualize a function consisting of the sum of two exponent terms in three dimensional space. The graphic in Figure 5.1 shows clearly that the sum of two negative exponent terms, $\exp(-X)+\exp(-Y)$ is minimized when (X, Y) lie on the line

segment $X = Y$, as shown by the highlighted line in the Figure 5.1 This provides geometrical support for the algebraic derivation of k and ε done through expressions (5.2 - 5.6).

However, it should be pointed out that both formulations of the optimization problem for determining k is based on asymptotic behavior and hence the value of k determined is an approximate one.

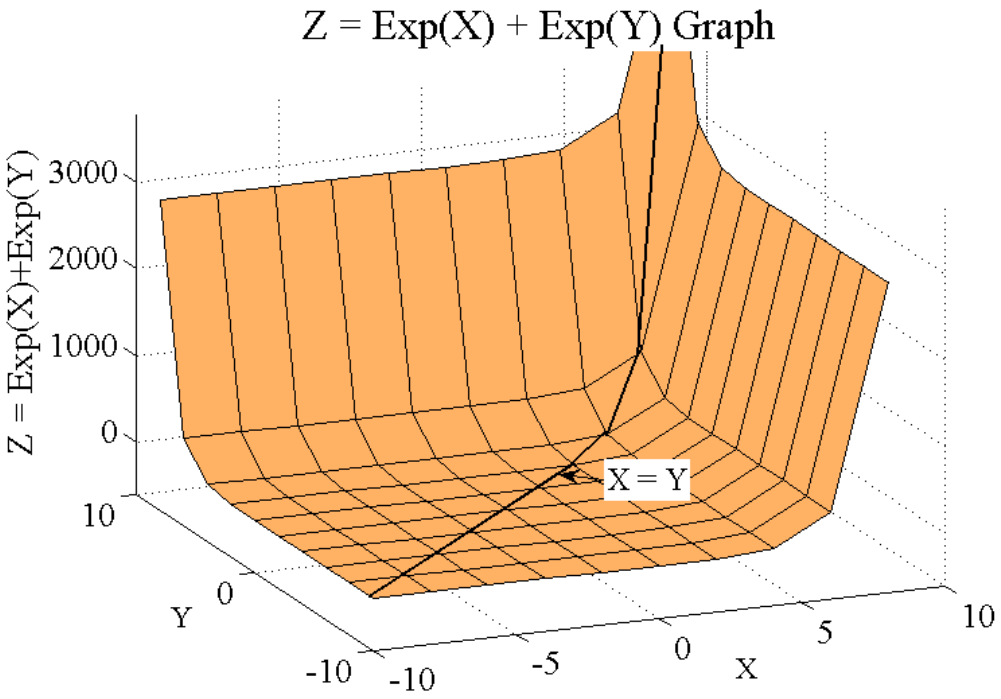


Figure 5.1: Minimization of the sum of two negative exponents

Using the expression for k derived above and equating the two terms in expression 5.3 yields:

$$n_l\sigma_l^2 = n_r\sigma_r^2 \tag{5.7}$$

An expression for ε can be formulated now as k is determined. Equating δ' in expression 5.3 to the user-assigned δ and setting the two exponent terms to

be of equal value:

$$\delta = 4\exp\Big(\frac{-n_r(1-k)^2\varepsilon^2}{2\sigma_r^2 + \frac{2}{3}(1-k)\varepsilon}\Big) = 4\exp\Big(\frac{-n_r(1-k)^2\varepsilon^2}{2\frac{n_l\sigma_l^2}{n_r} + \frac{2}{3}(1-k)\varepsilon}\Big) \tag{5.8}$$

δ could have been equated to the left exponential term in 5.3 instead of the right to obtain ε, but as the terms asymptotically approach each other in a low variance environment, an equivalent expression will result which yields approximately the same numerical value.

Solving 5.8 above for ε and the use of expression 5.7 yields:

$$\varepsilon = \frac{1}{3(1-k)n_r}\Big(p + \sqrt{p^2 + 18\frac{n_l\sigma_l^2}{n_r}n_r p}\Big) \tag{5.9}$$

where $p = \ln\frac{4}{\delta}$ and $k = \frac{n_r}{n_r+n_l}$.

In the expression above involves the population variance σ_l^2 across the reservoir. As noted earlier in Chapter 3 with the segment sizes used for the reservoir, the sample variance provides a good approximation for population variance even for the worst-case scenario where the reservoir size is 200, while improving progressively with reservoir size. Thus, henceforth, in this research sample variance σ_{sl}^2 is used as a good approximation of the population variance σ_l^2. For simplicity of notation σ_s^2 is used in place of $\frac{n_l\sigma_{sl}^2}{n_r}$, thus giving:

$$\varepsilon = \frac{1}{3(1-k)n_r}\Big(p + \sqrt{p^2 + 18\sigma_s^2 n_r p}\Big) \tag{5.10}$$

Comparing 5.10 with the corresponding ε cut value for SeqDrift1 defined in Theorem 5.1, the following observations can be made:

- The ε cut threshold for SeqDrift2 is more flexible and accommodates a greater range of values for k other than the single value $\frac{1}{2}$, which SeqDrift1 always uses.

- The ε cut threshold for SeqDrift2 is tighter than its SeqDrift counterpart, and thus can be expected to yield better delay times for SeqDrift2. In SeqDrift2, the right repository size is set to the block size b, so essentially the difference amounts to the factor $\frac{1}{3(1-k)}$ in SeqDrift2 replacing the factor $\frac{2}{3}$ in SeqDrift1. Thus for all values of $k < 0.5$, SeqDrift2 would yield a tighter ε threshold value.

Theorem 5.2 ***False Negative Guarantee for SeqDrift2***
With ε defined as in Theorem 5.1, and $|\mu_l - \mu_r| > 2\varepsilon$, then the probability of occurrence of a false negative with SeqDrift2 is $< \delta$.

Proof $Pr(2|\mu_l - \mu_r| \leq 2\varepsilon) = Pr(|\mu_l - \hat{\mu}_l + \hat{\mu}_l - \mu_r + \mu_l - \hat{\mu}_r + \hat{\mu}_r - \mu_r| \leq 2\varepsilon)$
$= Pr(\mu_l - \hat{\mu}_l + \hat{\mu}_l - \mu_r + \mu_l - \hat{\mu}_r + \hat{\mu}_r - \mu_r| \leq k\varepsilon + (1-k)\varepsilon + k\varepsilon + (1-k)\varepsilon)$
$= Pr(|\mu_l - \hat{\mu}_l| \leq k\varepsilon) + Pr|(\mu_l - \hat{\mu}_r)| \leq (1-k)\varepsilon)$
$+ Pr(|\mu_l - \hat{\mu}_r| \leq k\varepsilon) + Pr(|\mu_r - \hat{\mu}_r|) \leq (1-k)\varepsilon)$ for some $k \in (0,1)$

From the union bound: $Pr(|\hat{\mu}_l - \hat{\mu}_r| \leq \varepsilon) \leq Pr(|\mu_l - \hat{\mu}_l| \leq k\varepsilon) + Pr|(\mu_l - \hat{\mu}_r|) \leq (1-k)\varepsilon)$ and $Pr(|\hat{\mu}_l - \hat{\mu}_r| \leq \varepsilon) \leq Pr(|\mu_l - \hat{\mu}_r| \leq k\varepsilon) + Pr(|\mu_r - \hat{\mu}_r| \leq (1-k)\varepsilon$

But from the starting assumption, the following is true: $Pr(|\hat{\mu}_l - \hat{\mu}_r| \leq \varepsilon) > 1 - \delta$

Thus $Pr(2|\mu_l - \mu_r| \leq 2\varepsilon) > 2Pr(|\hat{\mu}_l - \hat{\mu}_r| \leq \varepsilon) > 2(1-\delta)$, or in other words: $Pr(|\mu_l - \mu_r| \leq 2\varepsilon) > (1-\delta)$which leads to a contradiction since $Pr(|\mu_l - \mu_r| > 2\varepsilon)$ with probability 1.

Thus the assumption of $Pr(|\hat{\mu}_l - \hat{\mu}_r| \leq \varepsilon) > 1 - \delta$ is false and so $Pr(|\hat{\mu}_l - \hat{\mu}_r| > \varepsilon) > 1 - \delta$ which means that the probability of a false negative is $< \delta$, which in turn proves the theorem.

5.6 Optimizing SeqDrift2 Detection Delay

The determination of the ε cut threshold so far has been completely determined by the false positive rate, with no consideration given to sensitivity. A method whereby sensitivity is explicitly taken into account by reducing ε as long as the estimated false positive rate remains well below the user defined permissible rate of δ.

An equally important objective of the optimization procedure is to automatically determine the size of the reservoir (left repository) to be used as concept changes occur in the stream. In the SeqDrift detectors, change point detection is activated at intervals of the block size. Thus, if detection delay was the only consideration then block size should be set as small as possible. If the right repository was a multiple m (> 1) of the block size, then the minimum possible detection delay would be mb, as cut point determination is only done when a new block arrives. Thus the only choice left to minimize detection delay would be to either make b as small as possible or m as small as possible. The latter choice is opted as making b too small would increase the risk that the sample mean taken from the right repository will have higher deviations from the true population mean, especially when the natural variance in the data is relatively high. Assigning $m = 1$ is opted, thus effectively making the size of the right repository equal to the block size. In terms of an adequate size for b, a value of 200 is taken. A smaller value such as 100 might be a choice, but the experimentation showed that a slightly better false positive rate could be achieved with a size of 200.

Having set the block and right repository sizes, the left repository size then becomes an important parameter that affects both the false positive rate and the detection delay time. The probability that a randomly chosen instance in the left repository being replaced by a new instance arriving in the stream is inversely proportional to its size [Vitter 1985]. Thus the smaller the size of

the left repository the smaller is the capacity of the left repository to keep an accurate record of its past; put simply its memory of the past is limited. Thus it is clear that the size of the left repository is crucial to improving sensitivity. This suggests that an optimization procedure for reservoir sizing that takes into account the various trade-offs between false positive rate, detection delay and memory overheads is needed. Determination of the ε cut threshold has ensured that the false positive rate is minimized. It is now necessary to ensure that sensitivity is increased by increasing the size of the reservoir while ensuring that the false positive rate does not increase above the user-defined δ threshold.

A logical starting point for the optimization procedure is to begin the process with equal sized repositories (effectively making k =0.5) and to progressively reduce k (and thus increase n_l) until the false positive rate is smaller than δ. In a real world environment, especially in a high speed data stream environment concept changes cannot be signaled externally and thus an estimate of the false positive rate is needed. This estimate is a function of the current k value, the variance in the stream and ε.

5.6.1 Convergence of Algorithm 5.1

In this section, Algorithm 5.1 is described and examined for its speed of convergence. The procedure runs for each new data block and finds the optimal value of k subject to the constraint expressed by 5.10. In order to ensure that memory usage remains within reasonable bounds a maximum size Nl_{max} is set for the reservoir according to the system memory available, just as is done in ADWIN. Starting with an initial configuration of $n_l = n_r = b$ a value for ε is calculated using equation 5.10 with $k = 0.5$ and the sample variance (line 3). The value of k is then reduced by a factor of $\frac{3}{4}$ and the ε value is updated (line 5). Whenever the constraint expressed by 5.11 is not violated,

k is decreased further and ε is updated as in the previous iteration. When the constraint expressed by 5.11 is violated, the value of k is reset by undoing the latest update to k and a function AdjustForDataRate$(k, \hat{\mu}_l, \hat{\mu}_r)$ is called that further adjusts k according to the data rate.

The rationale behind Algorithm 5.1 is that SeqDrift2's sensitivity can be enhanced by decreasing the value of its cut point threshold ε as long as the estimated false positive rate δ' does not exceed the maximum user defined permissible rate δ. This is accomplished by progressively decreasing the value of k until the constraint $\delta' < \delta$ is violated. As the starting value of k is 0.5, the initial value of δ' is much smaller than δ. However, with each iteration δ' will increase with each decrease in the value of k and will approach the value of δ. In practice convergence based on the violation of $\delta' < \delta$ can be quite slow and so to increase the speed of convergence, the following condition is used:

$$\frac{\delta'_{i-1} - \delta'_i}{\delta'_{i-1}} < t \tag{5.11}$$

instead, where t is a tolerance value, set to a small value such as 0.0001. Convergence governed by 5.11 ensures that the risk estimate has stabilized at the convergence point and while it may be possible for the δ' value to increase by small amounts beyond the convergence point, the gains in sensitivity gained by further iterations are marginal and outweighed by the increased execution overhead. Lemma 1 establishes that 5.11 can be replaced by the constraint 5.12 which can be enforced more conveniently.

$$\frac{\varepsilon_{i-1} - \varepsilon_i}{\varepsilon_{i-1}} < t \tag{5.12}$$

Algorithm 5.1 GetOptimalEpsilon()

Input: Block Size b, $\hat{\mu_l}, \hat{\mu_r}$, δ, σ_s, Nl_{max}, t
Output: optimized value for ε_{opt}

/* $\hat{\mu_l}$ is the sample mean of reservoir */
/* $\hat{\mu_r}$ is the sample mean of repository */
/* δ is the user-defined maximum false positive rate */
/* σ_s is the sample variance within the reservoir */
/* Nl_{max} is the maximum permissible memory size of the reservoir */

1 $n_l = b, n_r = b, k_1 = 0.5$

2 $i = 2$

/* calculate ε using equation 5.10 */

3 $\varepsilon_{i-1} = \frac{1}{3(1-k_{i-1})n_r}\left(\ln\left(\frac{4}{\delta}\right) + \sqrt{p^2 + 18\sigma_s^2 n_r \ln\left(\frac{4}{\delta}\right)}\right)$

/* decrement k in small steps */

4 $k_i = \frac{3}{4}k_{i-1}$

5 $\varepsilon_i = \frac{1}{3(1-k_i)n_r}\left(\ln\left(\frac{4}{\delta}\right) + \sqrt{p^2 + 18\sigma_s^2 n_r \ln\left(\frac{4}{\delta}\right)}\right)$

6 **if** $\frac{\varepsilon_{i-1}-\varepsilon_i}{\varepsilon_{i-1}} < t$ **then**
7 $\quad i = i + 1$
8 $\quad$ go to Step 3
9 **else**
$\quad$ /* reset k as it decreased beyond the optimal value */
10 $\quad k_i = \frac{4}{3}k_{i-1}$
$\quad\quad$ /* Now adjust k according to the rate of change in data */
11 $\quad k = AdjustForDataRate(k_i, \hat{\mu_l}, \hat{\mu_r})$
12 $\quad \varepsilon_{opt} = \frac{1}{3(1-k)n_r}\left(\ln\left(\frac{4}{\delta}\right) + \sqrt{\ln\left(\frac{4}{\delta}\right)^2 + 18\sigma_s^2 n_r \ln\left(\frac{4}{\delta}\right)}\right)$

/* Set the left reservoir size according to new k value */

13 $n_l = n_r * \frac{1-k}{k}$

14 **if** $n_l > Nl_{max}$ **then**
15 $\quad n_l = Nl_{max}$

16 **return** ε_{opt}

Algorithm 5.2 AdjustForDataRate()

Input: k_{curr} ,$\hat{\mu}_l$, $\hat{\mu}_r$
Output: Optimal value of k

/* Determine the current rate of change */
1 $rate = (|\hat{\mu}_l - \hat{\mu}_r|)$
/* Calculate the optimal k value based on rate of change */
2 $k = k_{curr} + (-rate^4 + 1) * k_{curr}$

return k

Lemma 1 is stated below and then it is used to examine Algorithm 5.1's convergence properties.

Lemma 1. Whenever $\frac{\varepsilon_{i-1}-\varepsilon_i}{\varepsilon_{i-1}} < t$, then $\frac{\delta_i^{'}-\delta_{i-1}^{'}}{\delta_{i-1}^{'}} < t$ where i is the iteration number.

Proof: $\frac{\varepsilon_{i-1}-\varepsilon_i}{\varepsilon_{i-1}} < t$ can be rewritten as: $\frac{\varepsilon_{i-1}}{\varepsilon_i} < \frac{1}{1-t}$

From equation 5.10, the following is true:

$$\varepsilon_{i-1} = \frac{\alpha_{i-1}}{3n_r(1-k_{i-1})} \tag{5.13}$$

and

$$\varepsilon_i = \frac{\alpha_i}{3n_r(1-k_i)} \tag{5.14}$$

where $\alpha_{i-1} = \left(\ln\left(\frac{4}{\delta_{i-1}^{'}}\right) + \sqrt{\ln\left(\frac{4}{\delta_{i-1}^{'}}\right)^2 + 18\sigma_s^2 n_r \ln\left(\frac{4}{\delta_{i-1}^{'}}\right)}\right)$ and $\alpha_i = \left(\ln\left(\frac{4}{\delta_i^{'}}\right) + \sqrt{\ln\left(\frac{4}{\delta_i^{'}}\right)^2 + 18\sigma_s^2 n_r \ln\left(\frac{4}{\delta_i^{'}}\right)}\right)$ where $\delta_{i-1}^{'}$, $\delta_i^{'}$ are the estimated false positive rates associated with iterations $i-1$ and i respectively

Dividing equation 5.13 by equation 5.14 yields,

$$\frac{\varepsilon_{i-1}}{\varepsilon_i} = \frac{(1-k_i)}{(1-k_{i-1})} * \frac{\alpha_{i-1}}{\alpha_i} \tag{5.15}$$

Rearranging equation 5.15,

$$\frac{\alpha_{i-1}}{\alpha_i} = \frac{(1-k_{i-1})}{(1-k_i)} * \frac{\varepsilon_{i-1}}{\varepsilon_i} \tag{5.16}$$

Now $\frac{(1-k_{i-1})}{(1-k_i)} < 1$ as $k_{i-1} = \frac{1}{f}k_i \Rightarrow k_{i-1} > k_i$

Thus from equation 5.16 and expression 5.11, the following can be derived: $\frac{\alpha_{i-1}-\alpha_i}{\alpha_{i-1}} < t$ which in turn means that $\frac{\delta'_i-\delta'_{i-1}}{\delta'_{i-1}} < t$, thus proving the Lemma.

Theorem 5.3 *Algorithm 5.1's convergence is* $O(log(log(\frac{1}{\delta})))$ *and* $O(log(\sigma_s^2))$.

Proof Denote ε_1 as the cut threshold used at the first iteration with $k = k_1 = 0.5$. Now ε_1 is given by: $\varepsilon_1 = \frac{1}{3(1-0.5)n_r}\Big(p + \sqrt{p^2 + 18\sigma_s^2 n_r p}\Big)$ and $\varepsilon_2 = \frac{1}{3(1-k_2)n_r}\Big(p + \sqrt{p^2 + 18\sigma_s^2 n_r p}\Big)$. Thus $\varepsilon_2 = \frac{1}{2}\Big(\frac{1}{(1-k_2)}\Big)\varepsilon_1$. Generalizing the derivation of ε to the $(i-1)^{th}$ iteration yields:

$$\varepsilon_i = \frac{1}{2}\Big(\frac{1}{(1-k_i)}\Big)\varepsilon_1 \tag{5.17}$$

From 5.10, the following can be stated:

$$\varepsilon_{i-1} = \frac{1}{3(1-k_{i-1})n_r}\Big(p + \sqrt{p^2 + 18\sigma_s^2 n_r p}\Big) \tag{5.18}$$

Applying the result from Lemma 1 on equations 5.15 and 5.16, the following quadratic is obtained:

$$3{\varepsilon_1}^2 n_r q^2 - 4(1-t)\varepsilon_1 pq - 24(1-t)\sigma_s^2 p = 0 \tag{5.19}$$

where $q = \frac{(1-k_{i-1})}{(1-k_i)}$and t is the tolerance factor.

Equation 5.19 when solved yields

$$q = \frac{(1-t)(2p + \sqrt{(4p^2 + 72n_r\sigma_s^2 p)}}{3\varepsilon_1 n_r} \tag{5.20}$$

Therefore:

$$\frac{(1-k_{i-1})}{(1-k_i)} = \frac{(2p + \sqrt{(4p^2 + 72n_r\sigma_s^2 p)}}{3\varepsilon_1 n_r} \tag{5.21}$$

Also, $k_{i-1} = \frac{1}{k} k_i = 0.5 f^{i-2}$

Substituting for k_{i-1} and k_i in (24) above and replacing the right hand side of (24) by q for simplicity, gives:

$$\frac{1 - f0.5^{i-2}}{1 - 0.5^{i-2}} = q \tag{5.22}$$

which when solved yields:

$$i = 2 + \log_f\left(2\frac{(q-1)}{(q-f)}\right) \tag{5.23}$$

Equation 5.23 shows that the number of iterations i in which Algorithm 5.1 converges is logarithmic (to base f) in q, and combined with equation 5.19 which expresses q as a square root function of σ_s^2, it is concluded that i is a logarithmic function (to base f) of σ_s^2. With respect to δ, i is logarithmic (again with base f) with respect to p, which in turn is inverse logarithmic (to base e) with respect to δ. Thus overall, with adjustment of logarithmic bases, $i = O(log(log(\frac{1}{\delta}))$

With a f value of 0.75, a p value of 5.99 (corresponding to a δ value of 0.01), a σ_s^2 value of 0.2, and $n_r = 200$, the use of (25) and (22) yields convergence at the 20^{th} iteration which corresponds to $\delta' = 0.0099$ (see Table 5.1) which is very close to the desired value of 0.01. Using the same f value of 0.75, and p value of 5.99, a σ_s^2 value of 0.02, and $n_r = 200$, convergence was signaled by (25) and (22) at the 18^{th} iteration, corresponding to δ' value of 0.0099 (see Table 5.1) which again is very close to the desired value of 0.01.

Although the value of 0.75 is used for f in Algorithm 5.1(line 4), in principle other values such as 0.8 or 0.9 can be used to produce very similar results. Different values of f were experimented and found that the difference in results is insignificant. Even an extreme value for f of 0.1 yields an ε value of 0.1199 (very similar to the 0.1202 value achieved with $f = 0.75$) at convergence for

the high variance case, a very similar result to the one that is achieved with $f = 0.75$. The same holds true for the low variance case; with $f = 0.1$, convergence was achieved with an ε value of 0.0460 (again, very similar to the 0.0461 achieved with $f = 0.75$. Therefore it is concluded that the optimization process is not sensitive to the value of f used, provided that a reasonable value for f is used, say in the range $[0.1, 0.9]$.

σ_s^2 =0.2					σ_s^2 =0.02				
ε_{curr}	k_{curr}	k_{new}	δ'	ε_{new}	ε_{curr}	k_{curr}	k_{new}	δ'	ε_{new}
0.2398	0.500	0.375	0.00050	0.1919	0.0920	0.500	0.375	0.00086	0.0736
0.1919	0.375	0.281	0.00176	0.1668	0.0736	0.375	0.281	0.00235	0.0640
0.1668	0.281	0.211	0.00328	0.1520	0.0640	0.281	0.211	0.00392	0.0583
0.1520	0.211	0.158	0.00471	0.1424	0.0583	0.211	0.158	0.00530	0.0547
0.1424	0.158	0.119	0.00591	0.1360	0.0547	0.158	0.119	0.00641	0.0522
0.1360	0.119	0.089	0.00688	0.1316	0.0522	0.119	0.089	0.00728	0.0505
0.1316	0.089	0.067	0.00763	0.1285	0.0505	0.089	0.067	0.00794	0.0493
0.1285	0.067	0.050	0.00821	0.1262	0.0493	0.067	0.050	0.00845	0.0484
0.1262	0.050	0.038	0.00865	0.1246	0.0484	0.050	0.038	0.00883	0.0478
0.1246	0.038	0.028	0.00898	0.1234	0.0478	0.038	0.028	0.00912	0.0473
0.1234	0.028	0.021	0.00923	0.1225	0.0473	0.028	0.021	0.00934	0.0470
0.1225	0.021	0.016	0.00942	0.1218	0.0470	0.021	0.016	0.00951	0.0468
0.1218	0.016	0.012	0.00957	0.1213	0.0468	0.016	0.012	0.00963	0.0466
0.1213	0.012	0.009	0.00967	0.1210	0.0466	0.012	0.009	0.00972	0.0464
0.1210	0.009	0.007	0.00976	0.1207	0.0464	0.009	0.007	0.00979	0.0463
0.1207	0.007	0.005	0.00982	0.1205	0.0463	0.007	0.005	0.00984	0.0462
0.1205	0.005	0.004	0.00986	0.1204	0.0462	0.005	0.004	0.00988	0.0462
0.1204	0.004	0.003	0.00990	0.1202	0.0462	0.004	0.003	0.00991	0.0461
0.1202	0.003	0.002	0.00992	0.1202	0.0461	0.003	0.002	0.00993	0.0461
0.1202	0.002	0.002	0.00994	0.1201	0.0461	0.002	0.002	0.00995	0.0461
0.1201	0.002	0.001	0.00996	0.1200	0.0461	0.002	0.001	0.00996	0.0461
0.1200	0.001	0.001	0.00997	0.1200	0.0461	0.001	0.001	0.00997	0.0461
0.1200	0.001	0.001	0.00998	0.1200	0.0461	0.001	0.001	0.00998	0.0460

Table 5.1: Optimization of k value by Algorithm 5.1

Table 5.1 clearly shows the utility of the optimization process. In the high

variance scenario the ε value starts off at a high value of approximately 0.24 for an equal-sized data repository configuration and decreases by around 0.12 at the point of convergence (it is defined that convergence is achieved when the difference between consecutive δ' values is less than or equal to 0.0001, a tolerance factor) to reach a value of 0.1202, thus greatly increasing sensitivity while ensuring that the false positive rate is kept within user-defined bounds. A similar behavior is observed for the low variance case, although the reduction in the ε value at 0.045 is not so drastic in absolute terms. As expected, the ε values at convergence are very much different from each other, as the determination of ε is dominated by the variance term, as shown in equation 5.10.

The optimization provided by Algorithm 5.1 in reducing k does not explicitly take into account changes in the classifier error rate The value of k determined by Algorithm 5.1 cannot be further decreased as this would entail a higher risk of false positives. On the other hand, fine tuning, involving small increases to the k value can be made to improve the false positive rate in accordance with the error rate trajectory. Smaller gradients of change require higher increases to the k value in order to minimize the false positive rate - in the limit when the gradient is 0, then k should receive its highest boost. By the same token higher gradients require smaller increases to the k value as sensitivity is less of a concern in such cases. In the limit when the gradient is 1, no increase in k value is needed from the false positive perspective as concept change is occurring, and hence there is no need to increase k to counter false positives. At the same time, from the sensitivity point of view detection times will be lesser for steeper gradients of change. With the above principles and boundary conditions established, the concave function defined in 5.11 is proposed to increment k and make it sensitive to the error trajectory.

$$\Delta k = (-rate^4 + 1) \tag{5.24}$$

where rate refers to the difference in sample means between the right and left repositories.

It can be seen that the function defined in 5.24 satisfies the necessary boundary conditions have been established. In reality, any algebraic function that decreases with increasing rate will suffice, but a concave function is deliberately chosen in preference to a linear function. The concavity in the function introduces a bias towards false positive minimization vis-a-vis detection delay minimization. When the rate increases from zero, a concave function will always provide a higher boost to the k value than a linear function as shown in the graphic in Figure 5.2. Thus a linear function will introduce the opposite bias: i.e. towards detection delay minimization as opposed to minimization of the false positive rate.

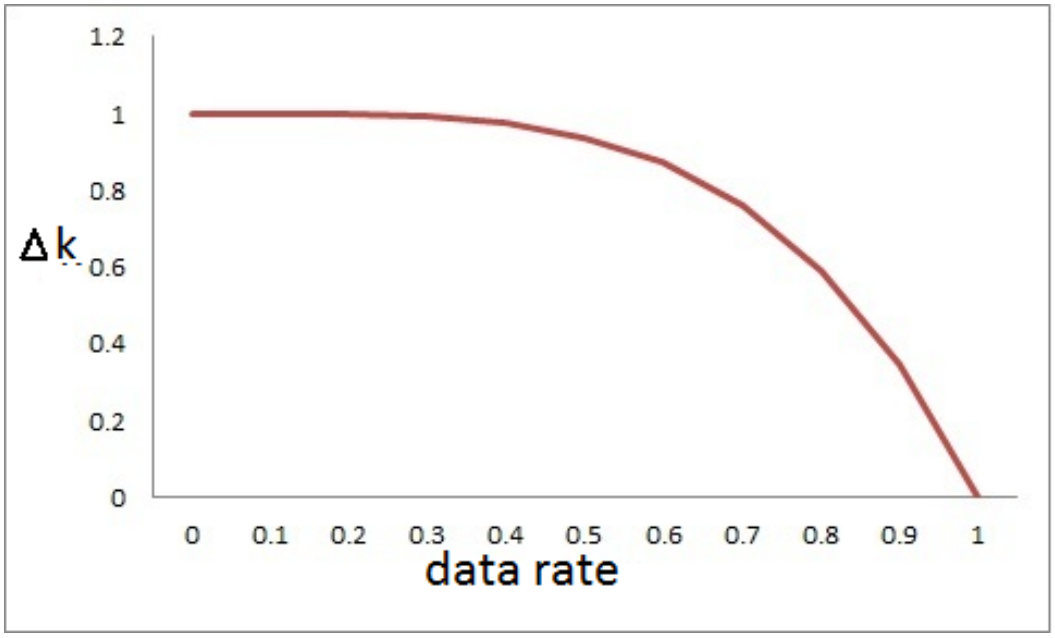


Figure 5.2: Fine adjustment to k based on the data rate using the concave function

Having decided on a concave function to implement the policy of biasing

the detection process in favor of false positives, the exact composition of the function is relatively unimportant as long as concavity is preserved. In principle, the function in 5.24 can be replaced by any other concave function that has the same *shape* and satisfies the boundary conditions but will lead to very similar results.

5.6.1.1 Motivating Example

The above mechanism has the effect of making the ε detection threshold sensitive to the changes in the stream data rate. Consider a virus checker application running on a Web server that processes tens of thousands of instances arriving per second. The virus checker is implemented through a classifier that learns a set of signatures that indicates the presence of viruses. The virus checker classifies each incoming instance (file request) as a threat (class 1, indicating the presence of a virus) or as a non threat (class 0, no virus). The classifier receives periodic updates of class labels from the virus checker vendor and users. In general these periodic updates will be available at the vendor site with varying frequency, but it is reasonable to assume that they will be distributed to client sites in average frequency intervals of the order of minutes rather than seconds. In between updates from the central server the classifier will use its current signature database to classify new instances that arrive. In this type of application the two priorities are processing speed and minimization of false positives. Processing speed is critical as the virus checker's speed of classification needs to match with the rate of arrival of instances. If it is below, load shedding will be forced on the server, thus increasing the risk of infection. At the same time the false positive rate needs to be as small as possible in order to reduce system overheads. Each instance wrongly tagged as a virus will require the generation of an alert to the user who made the file request to desist from downloading the file concerned, thus

not only generating additional system overhead but also potentially denying the user access to a legitimate file.

In this application concept change signifies that a new strain of viruses have been introduced into the stream. The longer the detection delay, the longer the time taken by the virus checker in discovering a new strain of virus. In this type of application detection delay will be determined primarily by the frequency of updates that arrive from the central site, which as discussed earlier can be expected to be in the order of minutes. It is thus established for this type of application (and there are many others like it, such as Spam detection, detecting changes in behavioral patterns in social media applications, etc) that maximization of throughput is essential, Likewise, minimization of the false positive rate is also a priority in such types of applications in order to minimize system overheads as well as to avoid unnecessarily burdening users with false alerts. Apart from the above mentioned applications the field of radio astronomy has opened up a vast opportunity for knowledge discovery on distant galaxies through the use of data mining. Radio Astronomy data represents an ultra high speed data stream, is highly noisy, has slow rate of concept change and has potentially high false positive rate [Das 2009], [Borne 2007].

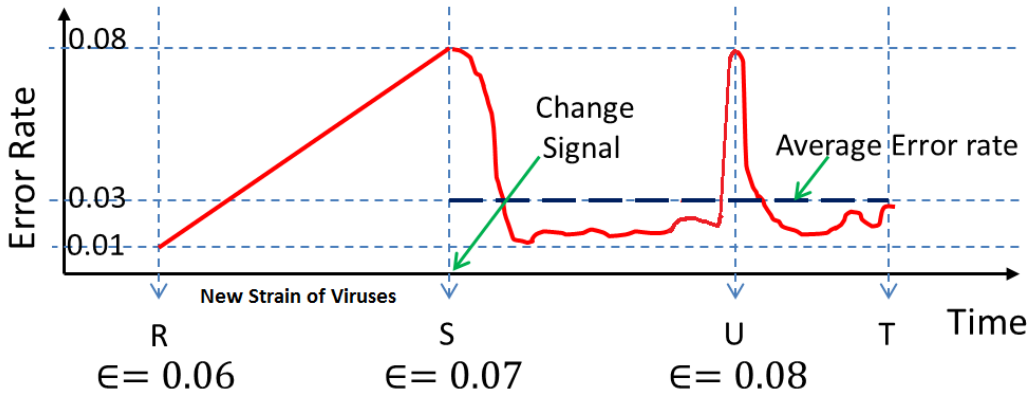


Figure 5.3: Error rate Vs Time with optimized values of ε

A virus checker application is used to illustrate the need for varying the detection threshold in order to minimize the false positive rate. Figure 5.3 illustrates the error rate vs time along with the optimized cut point threshold value ε. Suppose that the AdjustForDataRate() method increases ε to 0.07 to protect against false positives in (R,S]. Similarly, due to the stability of the error rate, with a mean value of approximately 0.03 in (S,T], (except for the short lived spike at U), AdjustForDataRate() increases ε to 0.08. No concept change is triggered at U unlike at point S which also had an error rate of 0.08 due to the higher setting of the detection threshold by AdjustForDataRate(), which will only trigger changes at 0.11 or above. Thus coupling the detection threshold to data rate has enabled the change detector to avoid false positives in the segment $(S, T]$. This example also illustrates that the necessity of different probabilities of detection for the same magnitude of error rate change that is registered over different segments of the data stream.

While it could be argued that an increase in variance in sub segments of $(S, T]$ would by itself contribute to an increase in value of the detection threshold, in fact this will not happen as the data in segment $(S, T]$ as a whole is stable, and hence an external mechanism such as AdjustForDataRate() is needed.

Algorithm 5.3 GetChangeType()

Input: $\hat{\mu}_l, \hat{\mu}_r, \varepsilon_{Change}$
Output: *Change* || *Internal* || *Homogeneous*

```
1 if ε_Change ≤ |μ̂_l − μ̂_r| then
2 |  if μ̂_r > μ̂_l then
3 |  |  return Change
4 |  return Internal
5 return Homogenous
```

Algorithm 5.4 UpdateUsingReservoirSampling()

Input: R, L
Output: void

```
1  CurrentSize = Get the current size of the reservoir L
2  MaxSize = Get the maximum size of the reservoir L
3  for Each instance Ins of R do
4      Increment the TotalInstancesSeen of L
5      if CurrentSize is less than MaxSize then
6          Add the element to L
7          Increment the CurrentSize of L
8      else
9          RandomIndex = Generate a random index in 0 and
           TotalInstancesSeen
10         if RandomIndex is less than MaxSize then
               /* With probability MaxSize / TotalInstancesSeen   */
11             Replace the instance of L at RandomIndex by Ins
12     Remove Ins from R
13     update the mean μ̂_l and variance σ_s^2
```

Algorithm 5.5 SeqDrift2: IsChange()

Input: An instance (Ins), BlockSize b, Left Repository R_l, Right Repository R_r
Output: Change / NoChange

```
1  Increment the instance counter
2  Insert Ins into R
3  Update the mean μ̂_r of R
4  if At the block boundary then
5      ε_Change = GetOptimalEpsilon(b, μ̂_l, μ̂_r, δ, σ_s^2, Nl_max)
6      μ̂_l = Get the mean of L
7      ChangeType = GetChangeType(μ̂_l, μ̂_r, ε_Change )
8      if (ChangeType is Change or Internal ) then
9          Remove all elements from L
10         Copy the elements from R to L
11         Remove all elements from R
           if (ChangeType is Change) then
12             return Change
13         return NoChange
14     UpdateUsingReservoirSampling(R,L)
15     return NoChange
```

5.7 Driver Routines for SeqDrift2

Algorithms 5.3, 5.5 and 5.2 show the pseudo code for the driver routines for the SeqDrift2 change detector. Algorithm 5.5 is the main driver routine that starts off by inserting a new instance into R, incrementally maintaining its (R) mean, and computing the sample variance, taken across all instances in L and R (lines 2 to 3). At a block boundary it calls Algorithm 5.1 (line 5) to optimize the values of k and ε. Having obtained the optimal ε and optimal L value it obtains the mean across L (line 6) and then calls Algorithm 5.3 to determine if concept change has occurred. Algorithm 5.3 performs hypothesis testing (lines 1) and returns the change type, as appropriate. Three possible

states exist: *homogeneous*, when no concept change has occurred, *change* when change has occurred, and finally *internal* when the ε threshold is triggered but the error rate of the classifier (i.e. when $\hat{\mu_r} \leq \hat{\mu_l}$) decreases.

If the change type returned by Algorithm 5.3 is of type *change* or *internal*, then Algorithm 5.5 flushes L, copies R into L and then flushes R (lines 9 to 11). If no change has occurred then it calls Algorithm 5.4 to perform an update on L using the reservoir sampling algorithm (line 14).

5.8 Time Complexity for SeqDrift2

The time complexity of SeqDrift2 is O(1) with respect to a stream instance. SeqDrift2's detection strategy is broken down into 3 major steps and analyze the time complexity involved in each step.

1. As each new instance is received it is buffered in the right repository and an incremental update is made to the sample mean in the right repository. Overall, the time complexity of this step is O(1).

2. When b instances are received (where b is the block size), Algorithms 5.1 and 5.2 are executed to optimize the value of k. As Algorithms 5.1 and 5.2 only require already calculated summary measures such as means and variance, no iteration over already read instances is required, so time complexity remains at O(1).

3. The final step involves hypothesis testing and updating the reservoir with instances from the right repository. The time complexity for the hypothesis testing operation is O(1) as no iteration through past instances is required. If the null hypothesis H_0 holds true that no significant difference in means exist between the left and right repositories then the instances buffered in the right repository update the reservoir in the left repository. This update requires a

single further pass through the instances in the right repository as it involves computing a random number in the range $[1..n_l+1]$ (line 9 in Algorithm 5.4), and then determining whether an instance in the reservoir needs to be replaced (line 10).

Thus the first two steps have time complexity O(1), while the third step is also O(1) but requires a further scan through the right repository, thus making SeqDrift2 a two pass algorithm with respect to its internal buffer. The internal buffer is much more compact than the raw data as in a classification context the buffer consists of a bit stream as each data instance once classified will end up as a 0 for a correct classification or 1 in the case of a miss-classification. Also, in common with SeqDrift1, it is one-pass with respect to cut point examination, as it never reexamines previous cut points. The theorem 5.2 is stated to establish a theoretical guarantee on the false negative rate for the SeqDrift2 detector.

5.9 Space, Time and Detection Delay Expectations

Before undertaking the empirical study on the five change detectors, namely SeqDrift1, SeqDrift2, ADWIN, PHT and EWMA, the following are summarized in Table 5.2: the space, time and detection delay complexities. In order to provide easy reference to the methods named parameters have been used, as proposed by the authors of the methods concerned.

	Memory Complexity	Time Complexity	Best Case Detection Delay
PHT	O(1)	O(1)	0
EWMA	O(1)	O(1)	0
SeqDrift1 and SeqDrift2	O(W)	O(2)	O(b)
ADWIN	$O(Mlog\frac{W}{M})$	$O(log(W)$ passes	O(g)

Table 5.2: Complexity analysis of change detectors

As shown in Table 5.2 PHT and EWMA are essentially memory less methods and are thus the most space efficient. The SeqDrift and ADWIN detectors store data instances in the detection window of size W. The latter uses compression in the form of an exponential histogram and is in general more space efficient than the SeqDrift detectors. The PHT,and EWMA detectors make a singe pass through the data, the SeqDrift detectors make two passes (one for buffering instances in the window and the other for updates), while ADWIN in the worst case makes $O(logW)$ passes as it checks each possible combinations of cuts in its window. In the case of detection delay, PHT and EWMA have best case delays of 0 for the scenario that the change takes place at the first instance of the current block/grace period, as they check for changes with the arrival of each new instance whereas the SeqDrift detectors and ADWIN check in intervals of b (block size) and g (grace period) respectively.

On the basis of the above measures both PHT and EWMA are attractive but other measures such as false positive rate and actual processing times need to be evaluated empirically before final judgments can be made on the relative effectiveness of these different approaches. The empirical evaluation has shown that both PHT and EWMA have significantly higher false positive rates than SeqDrift2 and ADWIN. In fact, EWMA had such high false positive

rates that could render it unusable as a change detector in its present form.

5.10 Empirical Study

The Empirical study had five broad objectives to test the following that are listed below:

- A five-way comparative study between SeqDrift1, SeqDrift2, ADWIN, PHT and EWMA on the false positive rate by recording the number of false positives made on stationary data.
- The robustness of the detectors to noise by injecting data with varying amounts of noise and recording the number of changes flagged by each of the detectors.
- Tolerance of the detectors to abrupt changes and the impact of upstream changes that are flagged by the detectors.
- The sensitivity of the change detectors by subjecting them to data that varied by different amounts over time and recorded the detection delay time and the processing time.
- The impact on classification accuracy, mining time and memory usage

5.10.1 False Positive Rate Assessment

This section elaborates the experimental procedure and the outcome in term of average false detection rates of SeqDrift1, SeqDrift2 and ADWIN.

5.10.1.1 Experimental Setup

The first experiment was designed to compare the false positive rates of the two SeqDrift detectors against ADWIN, Page Hinkley Test (PHT) and the Exponential Weighted Moving Average (EWMA)methods. A stationary Bernoulli

distribution of 200,000 instances was used for this and tested the effect of various combinations of mean value (μ) and maximum allowable false positive rate (δ). For this experiment the block size for the SeqDrift detectors was set to the default value of 200 and ADWIN's internal grace period parameter (the equivalent of the SeqDrift detector's block size parameter) was also set to its default value of 32. In the case of PHT, its internal parameters, drift level=10 and delta=0.02 were set to achieve a detection delay that was approximately equal to that of ADWIN and SeqDrift2 so that an assessment of its performance could then be on the basis of its false positive rate. In the case of EWMA, a setting of its control parameters lamda=0.2, ARL_0=1000 and warning level=0.1 was used that resulted in the lowest possible false positive rate. With this setting it turned out that its detection delay was similar to that of SeqDrift2, ADWIN, PHT thus enabling it to be compared with the others as well on the false positive rate. The implementations of PHT and EWMA from MOA Extensions by Paulo Mauricio Gonçalves Jr [1] were used.

A total of 100 trials for each combination of μ and δ were conducted and the average false positive count for each combination was recorded for the three change detectors. In order to obtain statistically reliable results a separate Bernoulli stream segment was generated for each trial, with the statistical properties mentioned above.

5.10.1.2 Comparison of SeqDrift1, SeqDrift2 and ADWIN

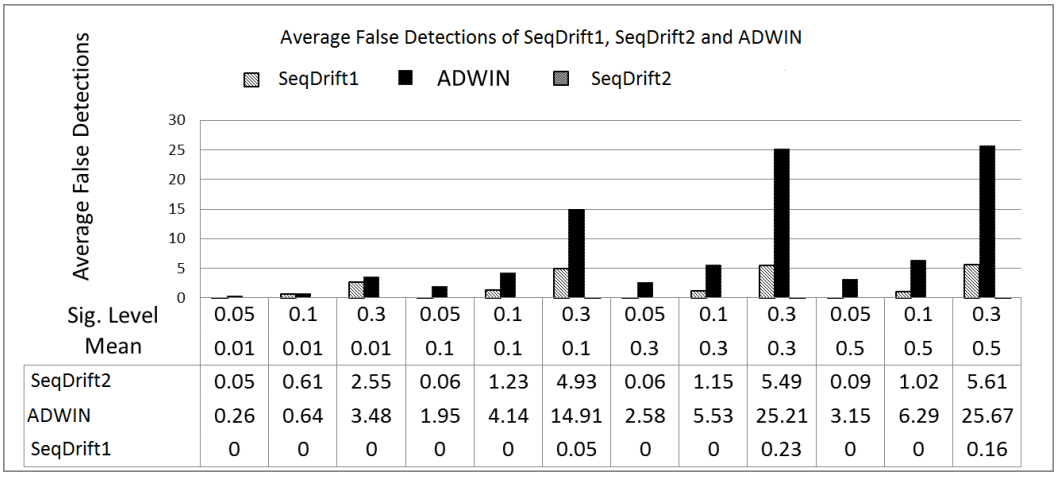
The comparison of SeqDrift with ADWIN is reported first as they both explicitly make use of a significance level δ and so the effects of this parameter can be assessed for the two different types of detectors. Figure 5.4 shows that all three detectors have good false positive rates that are substantially lower than the user defined maximum permissible level (δ) set. However, it is observed

[1] From sites.google.com/site/moaextensions/

that as the variance in the data increases with the increase in the μ value (for a Bernoulli distribution, the variance is $\mu \times (1-\mu)$) the gap between ADWIN and the SeqDrift detectors widens. The ADWIN false positive rate increases progressively with the increase in variance as well as the lowering of confidence (i.e higher δ values). On the other hand the SeqDrift detectors retain a virtually zero false positive rate except when the confidence is low at 0.7 ($\delta = 0.3$), with SeqDrift1 and SeqDrift2 returning rates of .0028%, compared to the ADWIN rate of .0128% at $\mu = 0.5$. As the confidence becomes lower, the ε value decreases and this results in an increase in the false positive rate for ADWIN. The relatively higher false positive rate for ADWIN is due to the use of compression to reduce storage size of its buffer.

ADWIN uses exponential histograms to compress its buffer and estimates true population means from the compressed version, thus introducing a degree of imprecision in the estimation of the population mean as explained in detail in Section 5.10.1.5. As the variance in the data increases so does the degree of imprecision in estimating the mean. In contrast, the SeqDrift detectors do not employ compression and estimate population means using random sampling techniques described in section 5.2.

While both SeqDrift detectors exhibit better false positive rates than ADWIN, it is clear from Figure 5.4 that SeqDrift1's rate is better than that of SeqDrift2. This is to be expected as SeqDrift2 was specifically optimized for detection delay through the use of Algorithm 5.1 which trades off detection delay time with the false positive rate, while ensuring that the false positive rate does not rise above the user defined permissible rate of δ.



Sig. Level	0.05	0.1	0.3	0.05	0.1	0.3	0.05	0.1	0.3	0.05	0.1	0.3
Mean	0.01	0.01	0.01	0.1	0.1	0.1	0.3	0.3	0.3	0.5	0.5	0.5
SeqDrift2	0.05	0.61	2.55	0.06	1.23	4.93	0.06	1.15	5.49	0.09	1.02	5.61
ADWIN	0.26	0.64	3.48	1.95	4.14	14.91	2.58	5.53	25.21	3.15	6.29	25.67
SeqDrift1	0	0	0	0	0	0.05	0	0	0.23	0	0	0.16

Figure 5.4: Average False Detections of SeqDrift1, SeqDrift2 and ADWIN

5.10.1.3 Overall Assessment of False Positive Performance

Since ADWIN and the SeqDrift detectors use an explicit significance parameter it was necessary to choose a significance level for these detectors. A level of 0.1 was chosen as levels below will be in favor of these detectors while a level of 0.3 would bias against them as the ε threshold for these detectors would then be too loose, thus increasing their false positive rate unfairly. Table 5.3 shows the average number of false positives recorded across the stable segment of the stream of length 200,000.

Mean	0.05	0.1	0.3	0.5
SeqDrift1	0	0	0	0
SeqDrift2	0.61	1.23	1.15	1.02
ADWIN	0.64	4.14	5.53	6.29
PHT	0	26.96	189.95	252.02
EWMA	1722.25	2727.79	549.86	16371.77

Table 5.3: Average False Change Comparison across all chosen detectors

Table 5.3 shows that the false positive count for PHT and EWMA detectors is orders of magnitude higher than for the rest of the detectors. Such a high number of false positives would result in significant degradation of performance for classifiers using these two detectors, both in terms of computational time (to adjust models) and in terms of classifier accuracy (in wrongly restructuring existing accurate models). The results clearly show that the default behavior of these detectors in operating on a per instance basis is not desirable. A better performance from them can be expected when they check for change in intervals, rather than individual instances. This will enable all detectors to be compared on an equal footing.

Table 5.4 shows that the gap between the (PHT, EWMA) pair and the rest has narrowed but it is clear that they have much higher false positive rates (except for PHT at mean 0.05) than the rest. Methods such as ADWIN and the SeqDrift detectors use detection thresholds that are specifically designed to minimize the false positive rate for a given significance level and this is the key reason for the difference in robustness of these methods.

Interestingly, it is observed that EWMA's false positive rate for mean 0.3 is much smaller than its rate for mean 0.05 but the rate climbs very sharply for mean 0.5. Several more trials were done for EWMA but this trend was always observed, strongly suggesting that this is an inherent feature of the method, possibly due to increased precision in the Monte Carlo simulation for mean values around 0.3.

Next, memory utilization across a stable stream was measured as this represents the worst case scenario for methods such as the SeqDrift and ADWIN which are not memory less unlike PHT and EWMA. False positives were suppressed in order to eliminate the confounding effect of memory flushing by SeqDrift2 and ADWIN. As expected, PHT and EWMA had trivial memory consumption with 64 bytes and 48 bytes for PHT and EWMA respectively. For SeqDrift2 its memory consumption increased until its reservoir filled and

then remained constant at 26k (with a reservoir of size of 50,000), while for ADWIN it increased throughout and reached 2.8K at the end of the 200,000 stream segment. The three detectors that had the best false positive rates are carried forward, that is SeqDrift1, SeqDrift2 and ADWIN for further analysis.

Mean	0.05	0.1	0.3	0.5
SeqDrift1	0	0	0.00005	0
SeqDrift2	0.00061	0.00123	0.00115	0.00102
ADWIN	0.00039	0.00158	0.00266	0.00257
PHT	0	0.00983	0.07198	0.09665
EWMA	0.06501	0.05270	0.00290	0.72522

Table 5.4: Average False Positive Rates across all chosen detectors

The experiments above reveal that SeqDrift change detectors are superior with respect to the number of false positive signals compared to the other three algorithms while consuming a constant memory in worst case. The next objective was to test the robustness to noisy data. The results are presented in the following section.

5.10.1.4 Robustness to Noisy Data

This experiment was designed to test the robustness of the change detectors to noise. Noise causes sudden fluctuations in sample means, thus creating the potential to increase the false positive rate for any given change detector. The higher the tolerance to noise, the more robust is the change detector. Ideally, a change detector should be immune to noise and maintain its false positive rate in the presence of noise, but in practice any change detector will be affected by noise, given a sufficiently high level and duration of the noise signal. The objective, then, of this experiment was to measure the level of tolerance to noise in the three change detectors that were experimented.

A noisy environment was simulated by introducing spikes in the error rate distribution that would typically arise from a classifier operating in a noisy environment. The key challenge was to sufficiently differentiate noise from the true signal so that noise could not be interpreted as true concept change. This was achieved by superimposing a distribution consisting of spikes on a stationary Bernoulli distribution.

The noise distribution was simulated by a Bernoulli distribution, but was sufficiently differentiated from the underlying stationary one by choosing a significantly higher population mean. It is noted that for a Bernoulli distribution its mean and variance are related by the expression $\sigma^2 = \mu(1-\mu)$, thus giving $\mu = \frac{(1-\sqrt{(1-4\sigma^2)})}{2}$. Thus, by introducing a factor k into the variance term σ^2 it was able to compute the corresponding mean value, μ_n value required to generate new distributions that were sufficiently different from the stationary (base) distribution with mean μ_b that corresponds to a k value of 1. Different noise distributions were generated to test the effects of different levels of noise by varying k in the range 1 to 2.5 in increments of 0.1. A base mean value of 0.01 was used for the stationary signal (with $k = 1.0$) which meant that the expected value of the variance was very low at 0.0099. For each value of k, 100 separate trials were conducted and the average false positive rate for the three change detectors were For each value of k, instances are drawn from a Bernoulli distribution with a mean value chosen at random between the base mean μ_b and μ_n.

As k increases, the variance of the noise signal increases and this has the effect of elongating the spike duration as well as the amplitude of the noise signal, as the noise trajectory for $k = 2.5$ shows in Figure 5.5. On the other hand the noise trajectory for $k = 1.5$ is barely above that of the baseline value (with $k = 1.0$), thus explaining why both ADWIN and SeqDrift detectors did not register an increase in the false positive rate.

An alternative way of generating noise would have been to increase the

population mean from its base value by certain predetermined amounts, but manipulating variance instead gave a higher level of control as it was possible to ensure that on average, fluctuations from the mean in the noise distribution exceeded the usual statistically inherent fluctuations present in the stationary (base) signal. A base mean value of 0.01 was used for the stationary signal (with $k = 1.0$) which meant that the expected value of the variance was very low at 0.0099. For each value of k, 100 separate trials were conducted and the average false positive rate for the three change detectors were recorded.

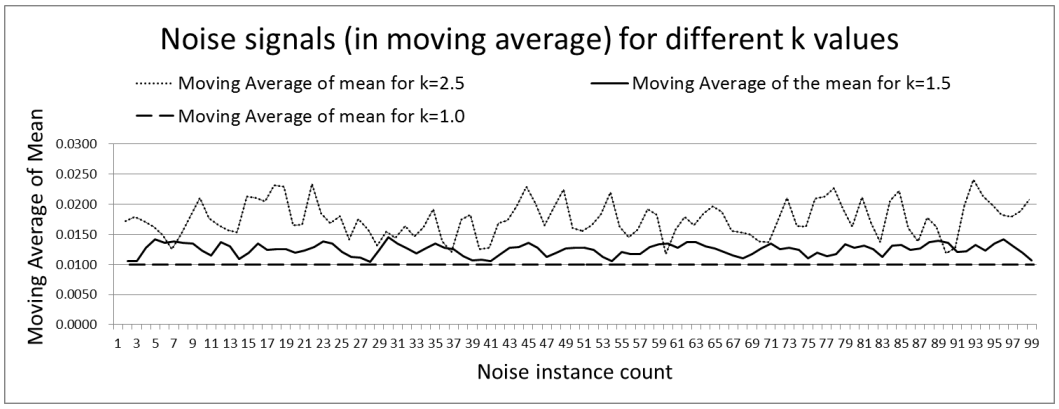


Figure 5.5: Effects of Noise Injection

k	Max Mean Increment	SeqDrift1	SeqDrift2	ADWIN
1	0	0	0	0
1.1	0.001	0	0	0
1.2	0.002	0	0	0
1.3	0.003	0	0	0
1.4	0.004	0	0	0
1.5	0.005	0	0	0
1.6	0.006	0	0	0
1.7	0.007	0	0	0.03
1.8	0.008	0	0	0
1.9	0.009	0	0	0.03
2.0	0.010	0	0	0.05
2.1	0.011	0	0	0.02
2.2	0.012	0	0	0.09
2.3	0.013	0	0.02	0.11
2.4	0.014	0	0	0.21
2.5	0.015	0	0	0.27

Table 5.5: Average Number of Changes Detected on a Noisy Stream

Table 5.5 shows that both SeqDrift1 and SeqDrift2 were very stable on noisy stream segments whereas ADWIN signaled a significant number of false positives when the level of noise increased. Given that noise is inevitable in most real world environments, this experiment clearly illustrates that both SeqDrift1 and SeqDrift2 are more robust to noise and hence better choices with respect to change detection.Values of k in the range $[1.0, 1.6]$ yielded a zero false positive rate for all three detectors. A comparison of ADWIN's false positive rate for values of k in the range 2.0 to 2.5 reveals a significant increase, thus exposing a fundamental weakness of ADWIN to noisy data distributions

containing spikes.

The underlying reasons for the higher tolerance of the SeqDrift detectors to noise are basically the same as for the experimentation with a pure stationary signal that was discussed in Section 5.10.1.2 above. The spikes in the noise signal have a much lower effect on the SeqDrift detector's false positive rate as the data representing such spikes is very short-lived and is added in general to a much larger pool (the reservoir) from which it is smoothed through the use of the averaging process.

5.10.1.5 Flow-on Effect of Abrupt Change on False Positive Rate

Abrupt changes are relatively easily detected by most change detectors in contrast to more gradual changes. In an environment where abrupt changes occur it is essential that such changes are detected with the minimum possible delay while ensuring that no false detections are made as a result of the abrupt change. When a sudden increase in the mean error rate for a classifier occurs its model needs to be updated to reflect such a change. Once the classifier's model is updated to reflect the new concept the expectation is that its performance will improve and thus further updates to its model will be unnecessary until a further actual change occurs. However, if as a result of the sudden change the detector falsely signals further changes, then unnecessary overhead will be incurred by the classifier in performing updates to its model.

In order to test the vulnerability of the detectors to the flow on effects of abrupt changes, a Bernoulli stream of 100,000 instances was generated with initial mean = 0.01 and then another 100,000 instances were drawn from a distribution with an increased mean value. Thus, abrupt change was injected into the stream at the $100,000^{th}$ instance and the concept changes flagged by each of the detectors were observed after processing the first 100,000 instances.

Initial Mean	End Mean	SeqDrift1 Changes	Det. Point	SeqDrift2 Changes	Det. Point	ADWIN Changes	Det. Points
0.01	0.02	0	N/A	0	N/A	5	102911,103135,103199 103295,103583
0.01	0.04	0	N/A	1	100399	7	100607,100927,100959 101311,101343,101439 101727
0.01	0.08	1	101199	1	100199	6	100159,100191,100223 100319,100351,100543
0.01	0.16	1	100199	1	100199	6	100095,100127,100159 100191,100255,100319
0.01	0.32	1	100199	1	100199	4	100031,100063,100127 100351
0.01	0.64	1	100199	1	100199	2	100031,100063
0.01	0.83	1	100199	1	100199	2	100031,100063

Table 5.6: Comparison of the change detectors SeqDrift1, SeqDrift2 and ADWIN on an abrupt drift of various mean increments

As Table 5.6 clearly shows ADWIN flags false changes upstream from its first detection point. For example, when the mean changes from the baseline value of 0.01 to 0.04, ADWIN first detects the change at point 100607 and then goes to flag 6 additional points further upstream from this point. In contrast, both SeqDrift detectors consistently report a single change point, with the exception of the less abrupt change scenario when a modest change was made to the mean from the 0.01 baseline to 0.02, for which no change was reported by either of the two detectors.

As with the stationary distribution scenario the cause for false detections with ADWIN lies with its use of exponential histograms in approximating the true data distribution. In the case of abrupt shift, however the estimation error is much greater due to the speed of change. ADWIN approximates the state of a window segment by maintaining at most M buckets for a given value 2^i, for integers i ranging from 0 upwards. In the case of an abrupt increase in the mean value of the data distribution, the probability of the appearance of 1s is subject to rapid change due to increase in variance that accompanies an increase in the mean value. This rapid change in the frequency of 1s in the stream causes the most recent buckets generated by ADWIN - i.e buck-

ets containing a single 1 to occur with widely different buckets lengths. This in turn causes the segments encapsulated by these buckets to have significantly different means, even when the underlying data distribution remains stationary after the change point. ADWIN stabilizes and stops detecting false change points only after a sufficient number of instances have been generated after the change point. When the number of buckets with value 1 exceeds the M threshold, merging of buckets occurs and sharp differences in mean values between buckets is smoothed due to the merging operation. Of course, a decrease in ADWIN's M parameter will help to alleviate this problem, but as mentioned before this will come at a heavy price in terms of computational overheads.

Interestingly, ADWIN's false alarms seem to disappear at the higher end of the μ range with no false positives reported for μ values 0.16 and greater. Although higher variability occurs in the data for such μ values, this is compensated by the increased frequency of the merge operation which is triggered by the higher frequency of 1s in the stream that occur with distributions having higher mean values.

The SeqDrift detectors do not artificially slice the stream into discrete storage units but represent the entire window segment after the last cut point as one single storage unit for sampling and are thus in a better position to avoid errors arising from comparing units (buckets) of arbitrary and insufficient length to estimate mean values accurately.

5.10.1.6 Effect of ADWIN's Grace Period on False Positive Rate

Mean	Significance	ADWIN Grace Period				SeqDrift1	SeqDrift2
	Level	32	100	200	300		
0.01	0.05	0.26	0.14	0.17	0.10	0	0.05
0.01	0.10	0.64	0.51	0.35	0.42	0	0.61
0.01	0.30	3.48	2.51	1.90	1.76	0	2.55
0.10	0.05	1.95	1.17	0.40	0.73	0	0.06
0.10	0.10	4.14	2.24	1.34	1.31	0	1.23
0.10	0.30	14.91	9.42	5.52	4.98	0.05	4.93
0.30	0.05	2.58	1.94	1.00	0.86	0	0.06
0.30	0.10	5.53	3.56	1.92	1.72	0	1.15
0.30	0.30	25.21	15.33	7.41	7.48	0.23	5.49
0.50	0.05	3.15	1.76	0.98	0.98	0	0.09
0.50	0.10	6.29	3.34	1.87	2.06	0	1.02
0.50	0.30	25.67	16.25	7.89	7.22	0.16	5.61

Table 5.7: Average false change detections over different stationary Bernoulli distributions

The ADWIN change detector is claimed to have no design parameters except for an internal parameter grace period parameter set to its default value of 32 instances [Bifet 2007]. The grace period parameter is the closest analogue to SeqDrift detector's block parameter and controls the processing time, the higher the value, the smaller the processing time. With its default setting, ADWIN checks for change in intervals of 32 instances. Higher values for this parameter could possibly improve ADWIN's false positive rate since false positives are checked at longer intervals of time. As such, the interest was in adjusting this parameter value in order to study its effect on ADWIN's performance. In addition, this enabled the comparison of SeqDrift detectors

with ADWIN on an equal footing. This is the motivation behind equalizing the grace period parameter to block size.

In order for the comparison to be fair SeqDrift1 and SeqDrift2 are not designed to capture changes on small sample sizes such as 32 due to insensitivity of Bernstein Bound. Therefore, the block size = 32 for SeqDrift1 and SeqDrift2 was not a suitable solution for a fair comparison to ADWIN. Given that both SeqDrift detectors had significantly better false positive rates than ADWIN with its default grace period setting of 32, it was more reasonable to increase the grace period to match the SeqDrift detector's block size of 200, rather than downsizing the block size to match the default grace period setting of 32. Accordingly, the false positive experiment was repeated in Section 5.10.1.3 with grace period set to 200. Table 5.7 shows that grace period has a significant effect on false changes as shown in Table 5.7. There was a substantial reduction in the false positive rate when the parameter was change from 32 to 200. No such reduction was observed when it was further increased to 300. Even though the grace period increase proved to be beneficial to ADWIN, Table 5.7 shows that both SeqDrift detectors (false positive rates reproduced for easy comparison from Figure 5.4) still outperform ADWIN even for higher grace period values. Intuitively, the improvement in ADWIN's false positive rate at higher grace period values is likely to come at a price of higher detection delay for ADWIN as it is now forced to check for change in larger intervals. The experimentation in Section 5.10.1.3 confirms that this is the case.

It can be observed that SeqDrift1 and SeqDrift2 outperformed ADWIN's best average false changes with grace period 200,thus being superior to the State of Art Change Detector ADWIN, in minimal average false changes generation.

5.10.2 Detection Delays and False Negative Rate

In addition to the false positive rate, detection delay is an important performance measure as minimal delay in detecting changes will assist the classifier in responding to concept changes quickly. However detection delay needs to be assessed against false negative rate. A low false negative rate combined with a small detection delay ensures that the change detector is responsive to concept changes, thus helping to improve classification accuracy.

In a high speed data stream environment another crucial performance factor is the processing speed of the change detector. A high processing time can lead to a bottleneck in the classifier, slowing down the speed with which it can classify instances. This is due to the fact that concept change detectors are invoked very often by the classifier (which for ADWIN will be 32 instances and for the SeqDrift detectors, every 200 instances). Each invocation requires the detector to compare means for one or more divisions of its buffer in its search for potential cut points. In general, the more extensive the search, i.e. the greater the number of candidates examined, the greater is the potential for detecting changes earlier but with a corresponding increase in processing time. In summary, a trade-off exists between detection delay and processing time. Given that the SeqDrift detectors and ADWIN have quite different change detection strategies it will be interesting to examine the trade-off in environments with different gradients of change.

Moreover, the effect of history in delay was also assessed.

The first experiment to compare the delays was designed in the following manner. A stationary Bernoulli data stream of different lengths 10000, 50000, 100,000 and 1,000,000 with mean = 0.01(initial mean) were generated in separate trials. Different data lengths were used to ascertain the effects of history, if any.

With each data length, concept change was injected into the stream by

generating the last 2300 instances with different gradients of change corresponding to 1 $\times 10^{-4}$, 2 $\times 10^{-4}$, 3 $\times 10^{-4}$ and 4 $\times 10^{-4}$. The processing time and the delay in detecting concept change injected were measured for each combination of data length and gradient of change. Each combination was repeated 100 times to boost reliability. In order to remove the confounding effect of reading time, the processing time measured only the actual time taken by the respective change detectors in executing their change detection algorithms.

ADWIN was used with its default grace period of 32, SeqDrift1 and SeqDrift2 with default block size of 200, and a significance level = 0.01 was used for all three change detectors.

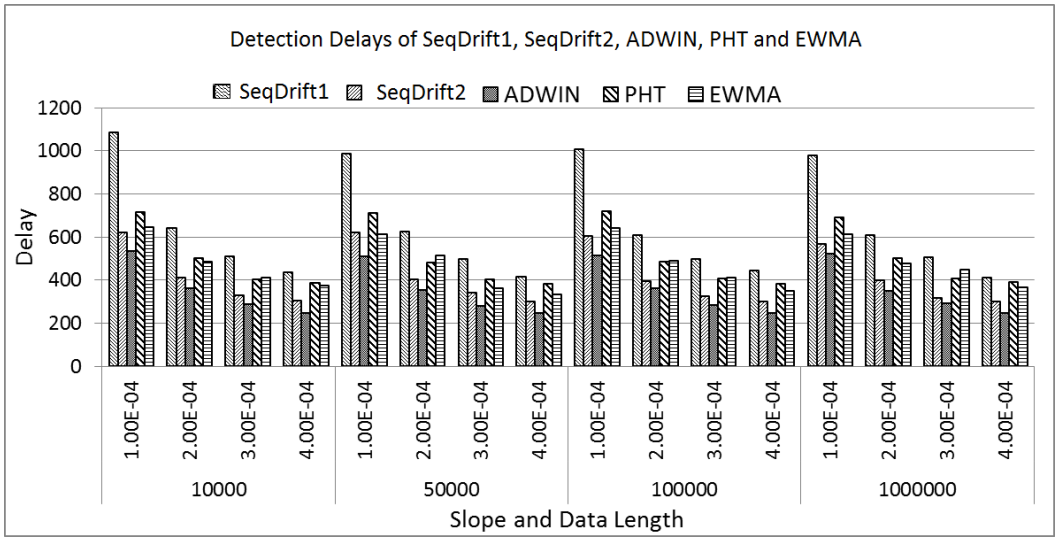


Figure 5.6: Detection delays of SeqDrift1, SeqDrift2 and ADWIN on streams with various slopes and lengths

Slope in (10^{-4})	Data Length 10000				100000				10000000			
	1	2	3	4	1	2	3	4	1	2	3	4
SeqDrift1	3.6	3.5	3.3	3.5	42.5	44.0	45.7	43.4	460.8	455.9	454.9	454.1
SeqDrift2	1.5	1.2	1.2	1.5	16.5	16.2	15.6	14.6	174.9	175.1	164.9	165.0
ADWIN	7.3	7.5	7.5	7.5	124.7	125.8	127.4	126.4	1635.7	1638.2	1639.7	1655.7
PHT	1.5	1.4	1.4	1.3	12.4	11.0	11.4	10.3	121.7	121.9	120.5	120.5
EWMA	2.6	2.8	3.4	3.2	42.8	44.5	48.6	45.3	418.9	426.7	418.1	425.0

Table 5.8: Processing times of SeqDrift1, SeqDrift2 and ADWIN on streams with different slopes and lengths

Figure 5.6, together with Table 5.8, clearly illustrates the detection delay versus processing time trade-off. In terms of processing time the two SeqDrift detectors were far superior to ADWIN. At the smallest data length of 10,000 ADWIN was around 3 times slower than SeqDrift1 and around 6 times lower than SeqDrift2. The gap between ADWIN and SeqDrift detectors widens as the data length increases: the processing speed of the SeqDrift detectors is essentially linear in data segment length while ADWIN is super-linear, as can be seen from the growth in processing times when the segment length is increased by factors of 10 and 100. This super-linear growth in ADWIN's processing time is only to be expected as the number of hypotheses it tests is $n\frac{(n-1)}{2}$, versus $n-1$ for the SeqDrift detectors on a buffer of size n. Given that the processing time for the SeqDrift detectors start off at a much lower level than ADWIN and that the latter's growth in time is super-linear in segment length, the SeqDrift detectors are by far the better choice for a high speed data stream environment.

In terms of detection delay, however, it is clear from Figure 5.6 that ADWIN has better mean detection delay when compared to the SeqDrift detectors. The SeqDrift detectors only examine a single candidate cut point for concept change that corresponds to the current boundary between the left and right repositories, whereas ADWIN examines all possible combinations of points in its buffer, and as such can be expected to detect change points

sooner. As expected, the delay times reduced with increasing gradient of change, although it is observed that the SeqDrift detector's delay reduces at a faster rate than ADWIN with the gap closing for higher gradients of change. In terms of detection rate, all five detectors returned a value of 100% for all combinations of data length and gradient of change.

It is also observed from Figure 5.6 and Table 5.8 that SeqDrift2's detection delay times are much closer to that of ADWIN, PHT and EWMA than SeqDrift1, while maintaining a superior processing speed advantage. The slower speed of SeqDrift1 over SeqDrift2 is due to the fact that it requires $\frac{n_l}{b}$ passes through its left repository to perform sub-sampling, as opposed to two passes for the latter. On the other hand, SeqDrift2 does not use sub-samples but computes means across the two repositories in a one-pass incremental manner. In fact, SeqDrift2's processing times are competitive with PHT and better than EWMA throughout the data length range. Despite the fact that EWMA is one pass, it has higher processing times than SeqDrift2 due to the large number of floating point calculations required in the computation of the control parameter L which is needed in flagging concept change.

Thus, from all the results examined so far, it is concluded that SeqDrift2 is superior to SeqDrift1 as it maintains a competitive false positive rate to the latter while exhibiting superior detection delay and processing times. Henceforth in the experimentation, it focus will be exclusively on the SeqDrift2 and ADWIN change detectors.

5.10.3 Effects of Reservoir Sampling

In Section 5.10.2, it was observed that replacing the sliding window by a reservoir in the implementation of the reference window reduces processing by around 50%, as shown in Table 5.8. Now another potential benefit of the reservoir is investigated which is improving sensitivity with respect to slowly

varying data. Data from a stationary Bernoulli distribution was generated with mean 0.01 for 200,000 instances. Thereafter, concept change was injected into the stream in two stages. In the first stage, change with a slope of 10^{-6} was injected for the next 10,000 instances. This was followed by injecting change at a higher rate of 10^{-5} for the next 70,000 instances. This experiment models a real-world scenario where data is first in a stable state after detection of the previous concept change. Thereafter in stage 2, due to the emergence of a new concept, data is subject to a slow rate of change, followed by an increase in the rate of change after emergence of the new concept in stage 3. It would be of interest to investigate the effect of memory retention of the oldest data samples from stage 1 on detection sensitivity measures. The performance measures of interest here are the detection rate, detection delay, stability of detection, as measured by the standard deviation of the detection delay over the 100 trials conducted, and the memory retention capacity of each of the two memory management schemes.

Ref. Window Size	Detection Rate (SW)	Detection Rate (Res)	Delay Ratio (SW/R)	Std. Dev. Ratio (SW/R)	% Replacement Res, SW
5000	10	100	3.34	12.58	0.16,100
10000	83	100	1.84	6.9	0.3,100
20000	100	100	1.1	1.71	0.6, 73
40000	100	100	1.05	0.96	1.19, 33.6
50000	100	100	0.99	1.39	1.5, 25.6

Table 5.9: Sensitivity of Reservoir over Sliding Window Approach

Table 5.9 shows that with reference windows of size 5,000 and 10,000 the reservoir scheme significantly outperformed the sliding window scheme on all measures. When the window size is less than 10,000 the sliding window approach was unable to store any samples from the oldest (stage 1) state and its buffer consisted entirely of samples from stage 2 and stage 3 thus severely affecting its sensitivity, causing it to return an average detection delay time which are 3.34 and 1.84 times that of the reservoir for sizes 5000 and 10,000

respectively. Furthermore, its detection behavior is highly unstable as the corresponding standard deviations of its detection times are 12.58 and 6.9 times that of the reservoir approach. However, as the reference window size increases the sensitivity of the sliding window approach improves and converges to that of the reservoir at around the 40-50,000 window size setting. As expected, Table 5.9 also shows that the relative performances of the two approaches is highly correlated to the memory retention capacity which is the percentage of data samples retained from the original stable state (stage 1). In a real world setting the length of concept formation (the sum of stage 2 and stage 3 lengths) may be much greater than memory available in the reference window and in such situations the reservoir approach is by far the superior choice.

5.10.4 Effects of Detection Thresholds and Window Management Strategies

Given that SeqDrift2 had the best false positive performance and that ADWIN maintains a good balance between detection delay and false positive rate it is of interest to examine whether each of these two best performing methods can benefit from using features implemented in each other.

Measure	ADWIN with SeqDrift2 Epsilon Threshold	SeqDrift2 with ADWIN Epsilon Threshold
Average False Detections (200000 instances)	Increased by 8.78	Decreased by 0.79
Average Delay	Decreased by 0.18%	Increased by 57%

Measure	SeqDrift2 with unlimited repository size
Average False Detections (200000 instances)	Increased by 0.19
Average Delay	Decreased by 0.03%

Measure	ADWIN with one cut point
Average False Detections (200000 instances)	Decreased by 14.82
Average Delay	Increased by 40%

Table 5.10: Further Experimentation on SeqDrift2 and ADWIN

Table 5.10 shows clearly that neither of the two change detectors benefit from using each other's features. When ADWIN uses SeqDrift2's detection threshold its false positive rate increased (measured for mean 0.3 and $\delta = 0.1$) while only a very marginal improvement resulted for detection delay (measured for slope $1 \times (10)^{-4}$). The opposite was observed when SeqDrift2 used ADWIN's threshold: its false positive decreased marginally but its detection delay increased quite considerably, showing that SeqDrift2's detection threshold is more sensitive than that of ADWIN. Unlike SeqDrift2, ADWIN cannot benefit from SeqDrift2's more sensitive detection threshold as it maintains multiple cut points in its window thus contributing to an increased false positive rate. These results are not unexpected as each of the two change detectors are optimized to work with their own detection thresholds.

The third experiment showed once again the effectiveness of the reservoir sampling strategy. Replacement of the reservoir with a buffer that grows in an unbounded fashion led to a very marginal improvement in detection delay while increasing the false positive rate. It should be noted that an unbounded buffer may be impractical in many situations as in a stable stream segment of large size the left repository could exceed the memory available. Apart

from that the computational cost of sampling the left repository would also increase considerably.

The final experiment with ADWIN operating on a single cut point does not yield any clear material benefits: although its false positive rate decreased quite considerably its detection delay increased sharply yielding values that were much higher than SeqDrift2 Thus the overall conclusion is that each of the two change detectors are best served by keeping them in their default configurations with their native detection thresholds.

5.10.5 Integration with Adaptive Hoeffding Tree Classifier

In a typical data stream environment, a change detector operates in conjunction with a classifier. To the extent that a change detector is able to detect concept changes efficiently in the minimal possible time with the least number of false detections, it will support the classifier in processing the data stream quicker and return higher classification accuracy. To investigate this premise, SeqDrift2 and ADWIN were integrated with the Hoeffding Adaptive Decision tree [Bifet 2009] and conducted a series of experiments on datasets generated with various different stream generators, sea concepts[2] and the real world data sets, airline and poker hand [3]. For the synthetic data the degree of concept change was controlled by specifying the number of features (f) that were subject to change. More details of the characteristics of these data generators is available from [Bifet 2010b].

For each experiment, the four performance measures were tracked: classification time, classification accuracy, the Kappa statistic and memory, measured in terms of tree size. These statistics represent averages over a stream

[2]From www.liaad.up.pt

[3]from moa.cms.waikato.ac.nz

length which varied for each dataset.

Data Generator	Parameters	Performance		Hoeffding Tree Size	
		ADWIN	SeqDrift2	ADWIN	SeqDrift2
SEA Concepts		(A)81.76	**82.23**	(N)**34**	37
		(K)79.53	**80.05**	(L)**17**	19
		(T)0.28s	**0.23s**		
Wave Form	-Drift Attr 10	a(A)80.06	**80.47**	(N)**2437**	2444
Generator		(K)70.10	**70.70**	(L)**1218**	1222
		(T)241.20s	**139.48s**		
Wave Form	-Drift Attr 20	(A)79.71	**80.18**	(N)2381	**2378**
Generator		(K)69.57	**70.27**	(L)1190	**1189**
		(T)228.47s	**127.96s**		
Wave Form	-Drift Attr 30	(A)74.28	**75.15**	(N)2432	**2412**
Generator		(K)62.23	**62.72**	(L)1216	**1206**
		(T)237.05s	**129.29s**		
LED	-Drift Attr 5	(A)73.41	**73.44**	(N)486	**466**
Generator		(K)70.46	**70.49**	(L)243	**233**
		(T)199.93s	**107.50s**		
RBF	-Speed Change 0.2	(A)52.91	**56.34**	(N)4975	**4798**
Generator		(K)5.83	**12.68**	(L)2487	**2399**
		(T)136.54s	**123.16s**		
Rotating	-Total Attr 20	(A)87.23	**90.09**	(N)7383	**5318**
Hyperplane	-Drift Attr 2	(K)74.46	**80.18**	(L)3691	**2659**
Generator	-Mag Change 0.1	(T)97.87s	**90.07s**		
Rotating	-Total Attr 20	(A)78.68	**85.47**	(N)6044	**5579**
Hyperplane	-Drift Attr 6	(K)57.37	**70.95**	(L)3022	**2789**
Generator	-Mag Change 0.1	(T)113.16s	**109.79s**		
Rotating	-Total Attr 20	(A)77.81	**78.99**	(N)**5494**	5681
Hyperplane	-Drift Attr 10	(K)55.63	**57.99**	(L)**2747**	2840
Generator	-Mag Change 0.1	(T)156.92s	**121.41s**		
Airline		(A)57.23	**58.90**	(N)82555	**48367**
		(K)13.49	**15.49**	(L)82206	**48153**
		(T)17.99s	**11.56s**		
Poker Hand		(A) **59.08**	58.85	(N) 227	**191**
		(K) **5.63**	3.82	(L)123	**104**
		(T)5.21s	**4.63s**		

Table 5.11: Integration of Change Detectors with Adaptive Hoeffding Tree. A - Accuracy, T- Mining Time, K - Kappa coefficient, N - Total number of nodes and L - Number of leaf nodes

Table 5.11 shows that SeqDrift2 clearly outperforms ADWIN in terms of all four measures that were tracked. In terms of classification accuracy, SeqDrift2 was better in 10 out of 11 experiments conducted. In the experimentation with the rotating hyperplane (with 6 drifting attributes) substantial improvements in accuracy of around 5% or greater was achieved. SeqDrift2's better accuracy is attributed to its superior false positive rate. When detections are signaled at a given node in the tree, the sub-tree rooted at that node is removed as it is thought to represent an old or outdated concept. If the detection is false, then the removal of the sub-tree will reduce accuracy as the concept it represents is actually current. Given that ADWIN registers a higher false positive rate than SeqDrift2 in general, the incidence of erroneous pruning will be proportionately higher than in SeqDrift2, thus resulting in a lower accuracy. The learning curves given in Figure 5.7 support this line of reasoning. The learning curves in Figure 5.7 were generated using the holdout evaluation method in MOA [Bifet 2010b] and represents 3 of the 11 datasets used. The curves for the rest follow the same trends and have been omitted to conserve space.

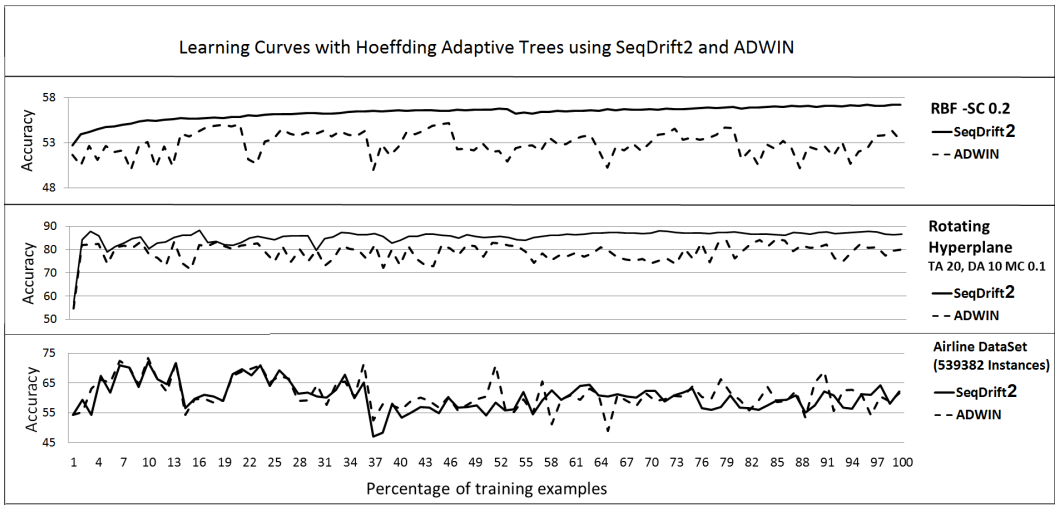


Figure 5.7: Variation of Accuracy with Training Set Size

As Figure 5.7 shows, the learning curves for ADWIN fluctuate significantly throughout the range of training set size. In contrast, SeqDrift2's curves are smoother and behave closer to the ideal scenario of smooth incremental growth in accuracy with increasing training set size.

The improvements in classification accuracy for SeqDrift2 are closely mirrored by improvements in Kappa value. The Kappa statistic for a given classifier measures the degree of improvement of the classification decisions over pure chance. SeqDrift2's Kappa values are consistently greater than that of ADWIN (with the exception of the Poker Hand dataset), thus inspiring greater confidence that the decisions taken with it in operation are better than chance.

The most significant improvements in performance for SeqDrift2, however, occur in the area of classification time. In some cases the mining time more than halved with the use of SeqDrift2 in place of ADWIN. The major reason for this reduction in time is due to SeqDrift2's efficient change detection strategy that employs a single forward sequential scan of its memory buffer instead of repeated backward scans and checks for cut points at every bucket boundary, as ADWIN does.

Finally, it is noted that SeqDrift2's induced smaller trees than ADWIN in 8 of 11 cases that were considered. As with accuracy the major cause for higher memory utilization with ADWIN as the change detector is its higher false positive rate. When a sub-tree is pruned as a result of concept change being signaled, a new alternate sub-tree is grown and maintained.

5.11 Summary

In this chapter, a novel scheme for concept change detection that employs a sequential one pass strategy has been presented along with extensive empirical evidence to proof its effectiveness in terms of various performance measures. In a nut shell, SeqDrift2 outperformed ADWIN with respect to false positive

rate and processing time, while maintaining competitive detection delay times. Sequential hypothesis testing as opposed to multiple hypothesis tests for a given pair of left and right buffers of stream instances is the reason for the gain in processing time in SeqDrift detectors. Sequential hypothesis testing ensures that no previously examined cut points are revisited.

In addition, it has been shown that SeqDrift2 has outperformed Page Hinkley detector with respect to false positive rate while maintaining competitive processing and detection delay times.

Now, the underlying reasons for these improvements are summarized. There are two major factors that caused such improvements. Low false positive rates were achieved with the help of reservoir sampling. In maintaining a buffer for change detection, there are two important considerations that need to be taken into account. First and foremost is the need for maintaining recent memory.Given that the memory is maintained within defined bounds, good representation of the trends in data streams need to be remembered even on an indefinite stable data segments. SeqDrift2 resolves this challenge through reservoir sampling, rather than by data compression, as employed by ADWIN. The use of the reservoir enabled SeqDrift2 to keep a representative sample of the data from the stream without relying on data compression/data aggregation strategy such as used by ADWIN that causes the latter to return higher false positive rates. Therefore, Reservoir sampling can be stated as the key to the success of SeqDrift2 superior performance over ADWIN. In an ultra high speed data stream environment, reservoir sampling effectively gathers a representative sample with minimal processing efforts avoiding bottlenecks in coping with high input rate. This sample helps both the change detector and classifier to avoid buffering or loss of information by simply deleting instances to cope up with the input. Such deletions will probably lead to loss of information thus distorting the underlying concepts.

Once concepts boundaries are identified precisely with the help of change

detectors such as seqDrift, current classifiers can be updated or newly created to respond to new concept. At the same time, outdated classifiers can also be archived to be reused when similar concept recurs in future. This is the next problem that this research proposes solutions.The next chapter focuses on recurring concepts capturing with the use of drift detectors and Fourier compression mechanism of decision trees.

Chapter 6

Capturing Recurrent Concepts Using Discrete Fourier Transform

6.1 Introduction

In the previous chapters, two effective methods have been proposed to detect changes in data streams. Changes in the underlying concepts cause current classifier models to become obsolete and unsuitable, thus leading to a potential loss in classification accuracy. Rapid changes in concept require classifiers to perform more frequent checks to monitor accuracy. On the other hand, in periods of relative stability, the frequency of checks can be decreased drastically, while allowing learning to take place in a more incremental fashion.

Machine learning applications that model, capture such concepts and recognize their re-occurrence gain significant efficiency and accuracy advantages over the systems that simply re-learn concepts each time they re-occur. When such applications include safety and time critical requirements, the need for concept re-use to support decision making becomes even more compelling. Such applications are the hardest ones to be replaced by fully autonomous systems due to unbearable cost of damage made by mistakes. One such application is the auto-pilot system in an aircraft or drone. An auto-pilot system needs to be extremely dependable and accurate.

Auto-pilot systems sense environmental changes and take appropriate action (classifiers, in the supervised machine learning context) to avoid disasters

and to fly smoothly. As environmental conditions change, appropriate actions must be taken in the shortest possible time in the interest of safety. Thus for example, a situation that involves the occurrence of a sudden low pressure area coupled with high winds (a concept that would be captured by a classifier) would require appropriate action to keep the aircraft on a steady trajectory. A machine learning system that is coupled to a flight simulator can learn such concepts in the form of classifiers and store them in a repository for timely re-use when the aircraft is on live flying missions. In live flying mode, the autopilot system can quickly re-use the stored classifier when such situations re-occur. Additionally, in live flying mode, new potentially hazardous situations not experienced in simulator mode can also be learned and stored as classifiers in the repository for future use.

In a real world setting, there is an abundance of applications that exhibit such recurring behavior in the financial area such as stock market and sales applications where timely decision making results in improved productivity. The research setting here is a data stream environment where the objective is to capture concepts as they occur, store them in highly compressed form in a repository and to re-use such concepts for classification when the need arises in the future.

A number of methods have been proposed that deal with the capture and exploitation of recurring concepts [Gama 2011], [Gomes 2010], [Hosseini 2012], [Alippi 2013] and [Morshedlou 2009]. Although achieving higher accuracy as expected during phases of concept recurrence in the stream is a challenge, the a major issue is the setting of user defined parameters to determine whether a current concept matches with one from the past with existing approaches

Such parameters are difficult to set, particularly due to the drifting nature of real world data streams. The proposed approach avoids this problem by applying the Discrete Fourier Transform (DFT) as a meta learner. The DFT, when applied on a concept (Decision Tree model) results in a spectral

representation that captures the classification power of the original models. One very attractive property of the Fourier representation of Decision Tree is that most of the energy and classification power is contained within the low order coefficients [Kargupta 2006]. The implication of this is that that when a concept C recurs as concept C* with relatively small differences caused by noise or concept change, then such differences are likely to manifest in the high order coefficients of spectra S and S* (derived from C and C* respectively), thus increasing the likelihood of C* being recognized as a recurrence of C.

The DFT, apart from its use in meta learning, has a number of other desirable properties that make it attractive for mining high speed data streams. This includes the ability to classify directly from the spectra generated, thus eliminating the need for expensive traversal of a tree structure.

The experimental results in section 6.6 clearly show the accuracy, processing speed and memory advantages of applying the DFT as opposed to the meta learning approach proposed by Gama and Kosina in [Gama 2011].

6.2 Related Research

While a vast literature on concept change detection exists, [Pears 2014] only a small body of work exists so far on exploitation of recurrent concepts. The methods that exist fall into two broad categories. Firstly, methods that store past concepts as *models* and then use a meta learning mechanism to find the best match when a concept change is triggered [Gama 2011], [Gomes 2010]. Secondly, methods that store past concepts as an *ensemble of classifiers*.

Lazarescu in [Lazarescu 2005] proposes an evidence forgetting mechanism for data instances based on a multiple window approach and a prediction module to adapt classifiers based on an estimation of the future rate of change. Whenever the difference between the observed and estimated rates of change is above a user defined threshold, a classifier that best represents

the current concept is stored in a repository. Experimentation on the STAGGER [Schlimmer 1986] dataset showed that the proposed approach outperformed the FLORA [Widmer 1996] method on classification accuracy with re-emergence of previous concepts in the stream.

Ramamurthy and Bhatnagar [Ramamurthy 2007] use an ensemble approach based on a set of classifiers in a global set G. An ensemble of classifiers is built dynamically from a collection of classifiers in G if none of the existing individual classifiers are able to meet a minimum accuracy threshold based on a user defined acceptance factor. Whenever the ensemble accuracy falls below the accuracy threshold, then the global set G is updated with a new classifier trained on the current chunk of data.

Another ensemble based approach by Katakis et al. is proposed in [Katakis 2008]. A mapping function is applied on data stream instances to form conceptual vectors which are then grouped together into a set of clusters. A classifier is incrementally built on each cluster and an ensemble is formed based on the set of classifiers. Experimentation on the Usenet dataset showed that the ensemble approach produced better accuracy than a simple incremental version of the Naive Bayes classifier.

Gomes et al [Gomes 2010] used a two layer approach with the first layer consisting of a set of classifiers trained on the current concept while the second contains classifiers created from past concepts. A concept change detector is used to flag changes in concept and when a warning state is triggered incoming data instances are buffered in a window to prepare a new classifier. If the number of instances in the warning window is below a user defined threshold, then the classifier in layer 1 is used instead of re-using classifiers in layer 2. One major issue with this method is the validity of the assumption that explicit contextual information is available in the data stream.

Gama and Kosina also proposed a two layered system in [Gama 2011] designed for delayed labeling, similar in some respects to the Gomes et al.

[Gomes 2010] approach. In their approach, Gama and Kosina pair a base classifier in the first layer with a referee in second layer.. Referee learns regions of feature space which its corresponding base classifier predicts accurately and is thus able to express a level of confidence on its base classifier with respect to a newly generated concept. The base classifier which receives the highest confidence score is selected, provided that it is above a user defined hit ratio parameter; if not, a new classifier is learned.

Just-in-Time classifiers is the solution proposed by Allipi et al. [Alippi 2013] to deal with recurrent concepts. JIT classifiers operate differently in stationary and non-stationary data streams. Concept change detection is carried out on the classification accuracy as well as by observing the distribution of input instances. The models consist of a set of concept representations and operators. Their concept representation stores a sequence of supervised instances, a set of concept defining features and drifting features. In stationary environments, supervised methods help enhance the performance whereas in non-stationary streams, concept change detection deactivates the current model and activate the best suitable model. The drawback is that this model is designed for abrupt drifts and is weak at handling gradual changes whereas both changes are handled well in the model proposed in this chapter.

Prior to describing the algorithmic aspects of the model proposed, the key technique that is the *Application of Discrete Fourier Transform* is described in the following section to illustrate its attractive properties that are exploited in this research.

6.3 Application of the Discrete Fourier Transform on Decision Trees

The Discrete Fourier Transform (DFT) has a vast area of application in very diverse domains such as time series analysis, signal processing, image processing and so on. It turns out as Park [Byung-Hoon 2001] and Kargupta [Kargupta 2006] show that the DFT is very effective in terms of classification and compression when applied on a decision tree model.

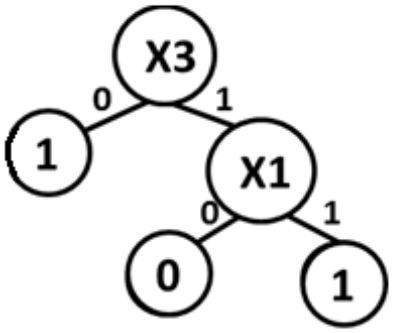
Kargupta et al. [Kargupta 2006] working in the domain of distributed data mining showed that the Fourier spectrum consisting of a set of Fourier coefficients fully captures a decision tree in algebraic form, meaning that the Fourier representation preserves the same classification power as the original decision tree. In addition, as a decision tree is converted into algebraic form, it becomes a straightforward aggregation to add any two decision trees in Fourier space. A decision tree has a self-explanatory property meaning that it is not a black box approach like neural network. Even in Fourier space, visualization of Fourier spectrum can be useful as shown in [Kargupta 2006]. Moreover, Kargupta et al. presents an algorithm to reconstruct a decision tree from Fourier spectrum, which again adds strength to the Fourier Spectrum representation. All the above properties can be exploited in data stream mining as well, especially in recurrent concept mining. The process of transforming a decision tree into Fourier spectrum is illustrated using an example in the next section.

6.4 Transforming a Decision Tree into Fourier Spectrum

A decision tree can be represented in compact algebraic form by applying the DFT to the paths of the tree. The process is illustrated by considering a binary tree for simplicity, but in practice the DFT can be applied to non binary trees as well [Kargupta 2006]. Each Fourier coefficient ω_j is given by:

$$\omega_j = \frac{1}{2^d} \sum_x f(x) \psi_j^{\overline{\lambda}}(x); \tag{6.1}$$

$\psi_j^{\overline{\lambda}}(x) = \prod_m \exp^{\frac{2\pi i}{\lambda_m} x_m j_m} = (-1)^{j.x}$(for binary data) where j and x are strings of length d, x_m and j_m are m^{th} attribute value in j and x, $f(x)$ is the classification outcome of path vector x and $\psi_j^{\overline{\lambda}}(x)$ is the Fourier basis function. x_m may have a wild card character $'*'$ if m^{th} attribute is not present in the path. The wild card character $'*'$ enables groups of coefficients that share attribute values to be defined with ease; thus for example ω_{*1*} denotes the group of coefficients that take values either 0 or 1 for attribute x_1 and x_3, value 1 for attribute x_2. The *order* of a coefficient is equal to the number of $1s$ in j vector.



X (Path Vector)			
X1	X2	X3	f(x)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

ω_j	Value
ω_{000}	3/4
ω_{001}	1/4
ω_{010}	0
ω_{011}	0
ω_{100}	-1/4
ω_{101}	1/4
ω_{110}	0
ω_{111}	0

Figure 6.1: Decision tree with 3 binary features, truth table of classification and its Fourier Spectrum representation

Figure 6.1 shows a simple example with 3 binary valued features x_1, x_2 and x_3, out of which only x_1 and x_3 are actually used in the classification. For example, for the path $x3 \Rightarrow 1$, x is $(**0)$, $f(x)$ is 1, $j \in \{000, 001, 010, \ldots, 111\}$ and $j.x$ is the inner product of vectors j and x.

The coefficient values ω_{000} and ω_{010} are calculated as:

$$\begin{aligned}\omega_{000} =& \frac{1}{2^d} f(000)(-1)^{000.000} + \frac{1}{2^d} f(001)(-1)^{000.001} \\ &+ \frac{1}{2^d} f(010)(-1)^{000.010} + \frac{1}{2^d} f(011)(-1)^{000.011} + \\ &+ \frac{1}{2^d} f(100)(-1)^{000.100} + \frac{1}{2^d} f(101)(-1)^{000.101} \\ &+ \frac{1}{2^d} f(110)(-1)^{000.110} + \frac{1}{2^d} f(111)(-1)^{000.111} = \frac{3}{4}\end{aligned} \tag{6.2}$$

$$\begin{aligned}\omega_{010} =& \frac{1}{2^d} f(000)(-1)^{010.000} + \frac{1}{2^d} f(001)(-1)^{010.001} \\ &+ \frac{1}{2^d} f(010)(-1)^{010.010} + \frac{1}{2^d} f(011)(-1)^{010.011} \\ &+ \frac{1}{2^d} f(100)(-1)^{010.100} + \frac{1}{2^d} f(101)(-1)^{010.101} \\ &+ \frac{1}{2^d} f(110)(-1)^{010.110} + \frac{1}{2^d} f(111)(-1)^{010.111} = 0 \end{aligned} \tag{6.3}$$

As shown in [Byung-Hoon 2001], only the coefficients for paths that are defined by attributes that actually appear in the tree need to be computed as all other coefficients are guaranteed to be zero in value (for example ω_{010}). The coefficient ω_{*1*} will be zero since attribute x_2 does not appear in the tree in Figure 6.1.

The *energy* contained in each coefficient ω_j is defined as,

$$E_{\omega_j} = ||\omega_j||^2 \tag{6.4}$$

Thus the total energy contained within all *order 1* coefficients (E_1) for the spectrum in Figure 6.1 is given by:

$$\begin{aligned} E_1 =& ||\omega_{001}||^2 + ||\omega_{010}||^2 + ||\omega_{100}||^2 \\ =& \frac{1}{4}^2 + \frac{1}{4}^2 + 0^2 = 0.125 \end{aligned} \tag{6.5}$$

Similarly, the energies of all other *orders* can be calculated. For the spectrum in Figure 6.1, $E_0 = 0.5627$, $E_1 = 0.125$, $E_2 = 0.0625$ and $E_3 = 0$. This example illustrates that most of the energy is contained in the low order coefficients as in Figure 6.2.

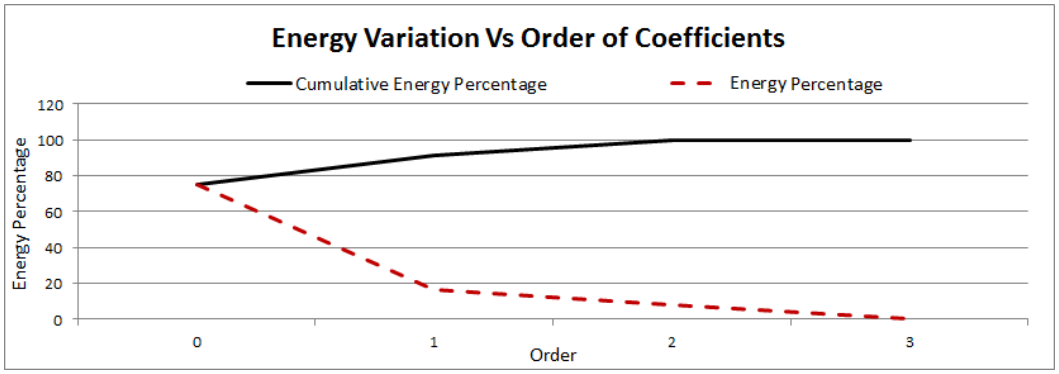


Figure 6.2: This graph shows an example based on Figure 6.1, Energy contained in low order coefficients decreases exponentially as shown in this graph. Therefore, low order coefficients are capable of capturing most of the energy contained in a tree

Kargupta et al in [Kargupta 2006] showed that the Fourier spectrum of a given decision tree has two very useful properties that make it attractive as a tree compression technique. [Kargupta 2006], [Linial 1993]:

1. All coefficients corresponding to partitions not defined in the tree are zero.

2. The magnitudes of the Fourier coefficients decrease exponentially with their order, where the order is taken as the number of defining attributes in the partition.

Taken together these properties mean that the spectrum of a decision tree can be approximated by computing only a small number of low order coefficients, thus reducing storage overhead. With a suitable thresholding scheme in place, the Fourier spectrum consisting of the set of low order coefficients is thus an ideal mechanism for capturing past concepts.

Furthermore, classification of unlabeled data instances can be done directly in the Fourier domain as it is well known that the inverse of the DFT defined

in expression 6.1 can be used to recover the classification value, instead of the use of a tree traversal which can be expensive in the case of an ensemble of trees. The inverse Fourier Transform is given by:

$$f(x) = \sum_j \omega_j \overline{\psi}_j^{\overline{\lambda}}(x) \tag{6.6}$$

where $\overline{\psi}_j^{\overline{\lambda}}(x)$ is the complex conjugate of $\psi_j^{\overline{\lambda}}(x)$.

An instance can be transformed into binary vector through the symbolic mapping between the actual attribute value and mapped value (either 0 or 1 in binary case). It can then be classified using the inverse function in equation 6.6. Suppose the instances are 000 and 010, the classification values $f(000)$ and $f(010)$ can be calculated as follows:

$$\begin{aligned} f(000) =& \frac{1}{2^d}(-1)^{000.000}\omega_{000} + \frac{1}{2^d}(-1)^{001.000}\omega_{001} \\ &+ \frac{1}{2^d}(-1)^{010.000}\omega_{010} + \frac{1}{2^d}(-1)^{011.000}\omega_{011} \\ &+ \frac{1}{2^d}(-1)^{100.000}\omega_{100} + \frac{1}{2^d}(-1)^{101.000}\omega_{101} \\ &+ \frac{1}{2^d}(-1)^{110.000}\omega_{110} + \frac{1}{2^d}(-1)^{111.000}\omega_{111} = 1 \end{aligned} \tag{6.7}$$

$$\begin{aligned} f(010) =& \frac{1}{2^d}(-1)^{000.010}\omega_{000} + \frac{1}{2^d}(-1)^{001.010}\omega_{001} \\ &+ \frac{1}{2^d}(-1)^{010.010}\omega_{010} + \frac{1}{2^d}(-1)^{011.010}\omega_{011} \\ &+ \frac{1}{2^d}(-1)^{100.010}\omega_{100} + \frac{1}{2^d}(-1)^{101.010}\omega_{101} \\ &+ \frac{1}{2^d}(-1)^{110.010}\omega_{110} + \frac{1}{2^d}(-1)^{111.010}\omega_{111} = 1 \end{aligned} \tag{6.8}$$

In terms of its application to decision trees, the DFT is not restricted to nominal valued attributes as splits on numeric attributes can be mapped easily to a set of discrete nominal values.

Due to thresholding and loss of some high order coefficient values, the classification value $f(x)$ for a given data instance x may need to be rounded to the nearest integer in order to assign the class value. For example, with binary classes a value for f is rounded up to 1 if it is in the range $[0.5, 1)$ and rounded down to 0 in the range $(0, 0.5)$.

6.5 Exploitation of the Fourier Transform for Recurrent Concept Capture

In this section, the application of DFT in the context of recurrence capture is elaborated.

In data streams, a variant of decision tree that learns incrementally and splits on a threshold defined with Hoeffding Bound is used [Domingos 2000], [Hulten 2001] and [Gama 2005]. In this research, DFT is applied on Hoeffding Tree implemented in CBDT [Hoeglinger 2009] forest.

The basic algorithm termed Fourier Concept Trees (FCT) is presented first in section 6.5.1 and then an optimization used for energy thresholding is described in section 6.5.2.

6.5.1 The FCT algorithm

CBDT is used as the base algorithm which maintains a forest of trees as mentioned earlier. This forest of trees is dynamic in the sense that it can adapt to changing concepts at drift detection points. Thus memory consumed by this forest is defined as *active*. In regular intervals, a winner tree that has the highest accuracy is chosen to represent the CBDT system. Classification output of CBDT depends on the winner chosen. In order words, winner tree is considered to be the best representation of the current concept in the underlying data stream.

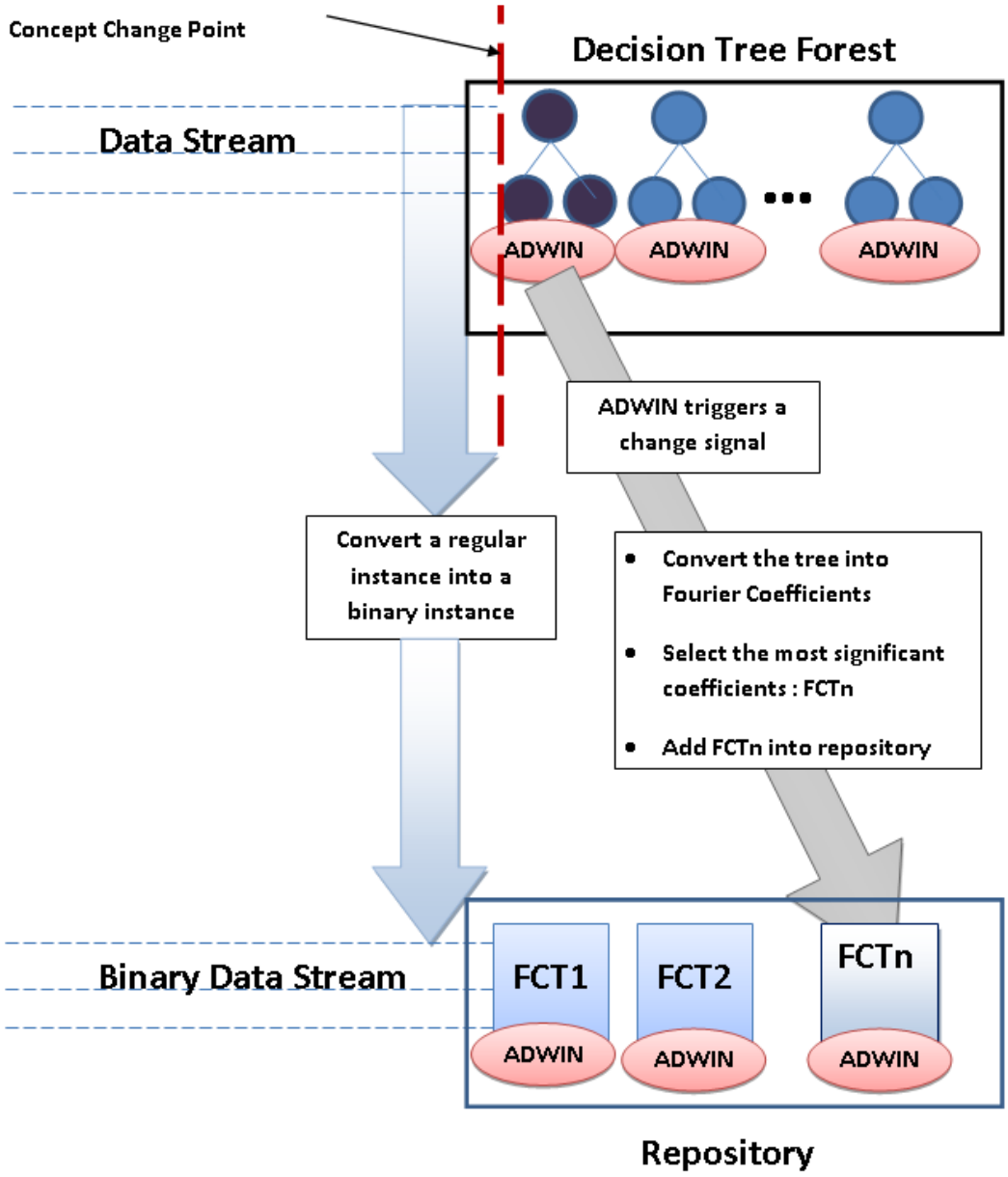


Figure 6.3: An architecture for concept re-use with the FCT approach

The basic CBDT algorithm 6.1 is integrated with the ADWIN [Bifet 2007] change detector to signal concept change. ADWIN serves two purposes in FCT model: detection of concept change depending on the error rate of the winner tree and showing the accuracy of a Hoeffding Tree over the current concept.

Traditionally, accuracy values are calculated historically from the beginning of a data stream. These historical accuracy values are poor representations of the performance of an algorithm due to time related changes in a data stream. ADWIN and SeqDrift change detectors store a buffer consisting of 1's and 0's (refer Chapters 4 and 5). The average of this buffer is the current error rate over current concept as the above change detectors flush the buffer belonging to all previous concepts. Therefore, this error rate is an ideal performance metric and it is used to calculate all accuracy values in this research.

Algorithm 6.1 fct algorithm

Input: Energy Threshold E_T , Accuracy Tie Threshold τ
Output: Best Performing model M that suits current concept

```
1  Plant a decision tree rooted on each attribute found in the data
   stream
2  M is set to a randomly selected decision tree model from the
   forest
3  Initialize the repository to null
4  read an instance I from the data stream
5  while change is not detected by the ADWIN instance of the current best model
   M do
6      if best model M is in repository call
       ClassifyUsingFourierSpectrum (Algorithm 6.2) to classify
       I
7      append 0 to ADWIN's window for M if classification is
       correct, else append 1
8      if M is from active memory then
9          identify best performing model F in repository
10         if (accuracy(M)-accuracy(F))> τ then
11             apply DFT on model M to produce F* using energy
               threshold E_T
12             if F* is not already in repository then
13                 insert F* into repository
14 Identify best performing model M by polling active memory and
   repository
15 GoTo 4
```

Algorithm 6.2 ClassifyUsingFourierSpectrum()

Input: Instance I, Fourier Spectrum F

Output: class value

1 **for** *all coefficients in* F **do**

2 If $f(x)$ calculated using equation 6.6 is greater than 0.5, return class1, otherwise class2

CBDT initializes d number of trees if there are d number of features present in a data stream (step 1). A random tree is chosen at the beginning until the first concept change is detected (step 2). At the first concept change point, the best performing tree (line 5) (in terms of accuracy) is identified and the DFT is applied after energy thresholding. Then, the resulting spectrum is stored in the repository for future use if the current concept recurs (line 11 to 13). The spectra stored in the repository are fixed in nature as the intention is to capture past concepts. At each subsequent change point, a winner model is chosen by polling both the active memory and the repository. If the winner emerges from the active memory, two checks are made before the DFT is applied. First of all, a test is made to see whether the difference in accuracy between the winner tree in active memory (T) and the best performing model in the repository is greater than a tie threshold τ (line 10). This is to avoid storing two similar performing classifiers (redundant) on a particular concept. If this check is passed, then the DFT is applied to T and a further check is made to ensure that its Fourier representation is not already in the Repository. This situation arises when two structurally similar trees are converted into Fourier Spectra. During the conversion, the minor differences between the two trees may disappear in Fourier Spectrum that consists of only the high energy coefficients. Therefore, to eliminate the possibility of redundant Fourier Spectra, the above test is necessary. If the winner model at a change point emerges from an already existing spectrum in the Repository, no Fourier

conversion is applied on any of the trees in active memory. Whichever model is chosen as the winner, it is applied to classify all unlabeled data instances until a new winner emerges at a subsequent change point. The least performing model M having the lowest weighted accuracy function is deleted if the repository has no room for new models. The weighted accuracy of M is defined by: $weight(M) = winner\_tally(M) * accuracy(M)$, where $winner\_tally$ is the number of times that M was declared a winner since it was inserted into the repository. FCT procedure is visually depicted in Figure 6.3.

6.5.2 Optimizing the Energy Thresholding Process

In order to avoid unnecessary computation of higher order coefficients which yield increasingly low returns on classification accuracy, energy threshold is highly desirable. To threshold on energy a subset S of the (lower order) coefficients needs to be determined such that $\frac{E(S)}{E(T)} > \varepsilon$, where $E(T)$ denotes the total energy across the spectrum and ε is the desired energy threshold value. In the optimized thresholding, the cumulative energy CE_i at order i is computed first and it is given by: $CE_i = \sum_{j=0}^{i} \sum_k ({w_k}^2 | order(k) = j)$.

Given an order i, an upper bound estimate for the cumulative energy across the rest of the spectrum is given by: $(d+1-(i+1)+1)CE_i$, as the exponential decay property ensures that the energy at each of the orders $i+1, i+2, \cdots, d$ is less than energy E_i at order i, where d is number of attributes in the dataset. Thus a lower bound estimate for the fraction of the cumulative energy CEF_i at order i to the total energy across all orders can then be expressed as:

$$CEF_i = \frac{CE_i}{CE_i + (d - i + 1)E_i} \tag{6.9}$$

where E_i is actual (computed) energy at order i. The lower bound estimate allows the specification of a threshold ε based on the energy captured by a given order i which is more meaningful to set rather than an arbitrary threshold.

The scheme expressed by equation (6.9) enables the thresholding process to be done algorithmically. If the cumulative energy fraction $CEF_i \geq \varepsilon$, then *the actual energy captured is at least* ε can be guaranteed, since CEF_i is a lower bound estimate. On the other hand, if $CEF_i < \varepsilon$, then CEF_{i+1} can be expressed as:

$$CEF_{i+1} = \frac{CE_{i+1}}{CE_{i+1} + (d-i)E_{i+1}} = \frac{CE_i + E_{i+1}}{CE_i + dE_{i+1}} \tag{6.10}$$

Thus equation (6.10) enables the cumulative fraction to be easily updated incrementally for the next higher order $(i+1)$ by simply computing the energy at that order while exploiting the exponential decay property of Fourier spectrum. The thresholding method guarantees that no early termination will take place. This is because CEF_i is a lower bound estimate and hence the order that it returns will never be less than the true order that captures a given fraction ε of the total actual energy in the spectrum.

6.6 Experimental Study

This section elaborates on the empirical study involving the following learning systems: CBDT, FCT and MetaCT. Gama's meta learning approach [Gama 2011] with CBDT as the base learner is implemented as MetaCT. The main focus of the study is to assess the extent to which recurrences are recognized using old models preserved in classifier pools.

6.6.1 Parameter Values

All experimentation was done with the following default parameter values:

- **Hoeffding Tree Default Parameters**
  - The desired probability of choosing the correct split attribute=0.99
  - Tie Threshold=0.01

  - Growth check interval=32

- **Tree Forest Default Parameter Values**
  - Maximum Node Count=5000
  - Maximum Number of Hoeffding Trees=50

- **Fourier Pool Default Parameter Values**
  - Accuracy Tie Threshold τ=0.01 which is the minimum accuracy difference between a new candidate Fourier Tree and any existing Fourier Tree in the Fourier pool
  - Maximum Fourier Trees = 50

- **Fourier Tree Default Parameter Values**
  - Energy Threshold = 95%

- **ADWIN Default Paramater Values**
  - drift significance value=0.01
  - warning significance value=0.3 (for MetaCT only)

All experiments were done on the same software with C# .net runtime and hardware with Intel i5 CPU and 8GB RAM, clearning the memory in each run to have a fair comparison.

6.6.2 Datasets Used for the Experimental study

The experimentation was done with data generated from 3 synthetic data generators commonly used in change detection and recurrent concept mining, namely SEA concept [Street 2001], RBF (generated using RBF kernels) and Rotating hyperplane [Hulten 2001] generators. In addition, 2 real-world

datasets *Spam* and the *NSW electricity* which were commonly used in previous research was also used in this research to compare the performances of the models.

For the synthetic datasets, each of the 4 concepts spanned 5,000 instances and reappeared 25 times in a data set, yielding a total of 500,000 instances with 100 true concept change points.

In order to challenge the concept recognition process, a 10% noise level is added for all synthetic data sets to ensure that concepts recur in similar, but not exact form.

6.6.2.1 Synthetic Data Sets

MOA [Bifet 2010b] was as the tool to generate these datasets.

1. **SEA:** The concepts are defined by the function $feature1 + feature2 > threshold$. The concepts are ordered as concept1, concept2, concept3 and concept4 generated using threshold values 8,7,9 and 9.5 respectively on the first data segment of size 20,000. 96 recurrences of a modified form of these concepts have been generated by using different seed values in MOA for each sequence of recurrence. Thus, the version of this dataset differed from the one used by Gama and Kosina [Gama 2011]. who simply used 3 concepts with the third being an exact copy of the first.

2. **RBF:** Number of centroids parameter was adjusted to generate different concepts for the RBF dataset. Concept1, concept2, concept3 and concept4 were produced with the number of centroids set to 5, 15, 25 and 35 respectively. Similar to the SEA dataset, the seed parameter helped producing similar concepts for a given centroid count value. This dataset had 10 attributes.

3. **Rotating hyperplane:** The number of drifting attributes was adjusted to 2, 4, 6, and 8 in a 10 dimensional data set to create the four

concepts. The concept ordering, generation of similar concepts and concatenation were exactly the same as in the other data sets mentioned above.

6.6.2.2 Real World datasets

1. *Spam Data Set:* The Spam dataset was used in it original form [1] which encapsulates an evolution of Spam messages. There are 9,324 instances and 499 informative attributes, which was different from the one version used by Gama that had 850 attributes.

2. *Electricity Data Set:* NSW Electricity dataset is also used in its original form [2]. There are two classes *Up* and *Down* that indicate the change of price with respect to the moving average of the prices in last 24 hours.

6.6.3 Tuning MetaCT Key Parameter

In the preliminary experiments, optimal value for the parameter *hit percentage threshold value* was as 80%. This parameter reflects the estimated similarity of the current concept with one from the past and thus controls the degree of usage of classifiers from the pool.

MetaCT was observed having good accuracy for the *delay in receiving true class labels* for the instances in short term memory, when it was set to 200.

6.6.4 Comparative Study: CBDT vs FCT vs MetaCT

The focus in this series of experiments was to assess the models in terms of accuracy, memory consumption and processing times. None of the previous studies reported in the recurrent concept mining literature undertook a comparative study against other approaches and so this empirical study is the

[1] from *http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift*
[2] from *http://moa.cms.waikato.ac.nz/datasets/*

first such effort. Furthermore, all of the previous studies focused exclusively on accuracy without tracking memory and execution time overheads and so this study would also be the first of its kind.

Before analyzing the performance of FCT, the exponential decay property of Fourier trees are investigated empirically. Figure 6.4 shows the average spectral energy captured by each *order*. Average value was calculated on RBF dataset (the same trend exists on the other datasets as well) by taking energy readings in an interval of 100 instances. It is clear that the low order coefficients capture most of the energy. This empirically validates the claim made by [Kargupta 2006]. Therefore, only a few low order coefficients need to be kept to achieve competitive classification power as a complete decision tree.

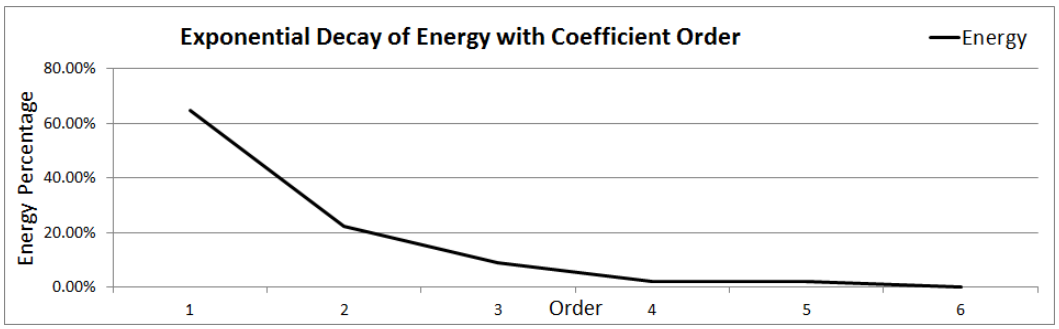


Figure 6.4: Exponential Decay of Energy with Coefficient Order in FCT on RBF dataset

The next section focuses on evaluation metrics accuracy, memory and processing speed.

6.6.4.1 Accuracy

A delay period of 200 was used with all three approaches in order to perform a fair comparison. Figure 6.5 clearly shows that overall, FCT significantly

outperforms its two rivals with respect to classification accuracy. The major reason for FCT's superior performance was its ability to re-use previous classifiers as shown in the segment 20k-25k on the RBF dataset where the concept is similar to concept1 that occurred in interval 1-5K. This is in contrast to MetaCT which was unable to recognize the recurrence of concept1. A similar situation occurs in the interval 25k-35k where the concept is similar to the previously occurring concepts, which are concept2 and 3. As expected CBDT, operating on its own without support for concept recurrence had a relatively flat trajectory throughout the stream segment.

A similar trend to the RBF dataset was observed in Rotating Hyperplane and SEA datasets as well. It can be clearly seen that FCT was successful in reusing the models learned before on data segments from 20k to 25k and from 30k to 35k. Though a preserved model was reused on the data segment from 25k to 30k (corresponding to concept3), the accuracy was not as high as in the above two segments. On the segment from 35k to 40k, concept recurrence was not picked up by either FCT or MetaCT resulting in a new classifier being used.

In summary, FCT outperformed MetaCT over 90 recurring concepts in RBF dataset whereas MetaCT did better in 6 occurrences, maintaining the similar trend as with the other 2 synthetic datasets.

The next experiment was on the NSW Electricity data set. Figure 6.5 shows that in overall, FCT was the winner here as well, outperforming MetaCT at 25 segments out of 35 that were tracked. Sudden fall in accuracy of MetaCT is occasional but due to incorrect selection of winner which was a decision stump.

MetaCT grows new classifier pair (base and referee) using its buffer populated during a warning phase before a concept change, if no existing classifier in its pool is reusable. This buffer has labeled and unlabeled instances. Whereas both types of instances help deciding on re-use of an existing model, only

the labeled instances are used to train and develop a new classifier. When MetaCT runs with classifier such as Hoeffding Tree that requires a large number of data instances to learn, MetaCT is only able to plant decision tree stumps as the number of data instances accumulated during a warning phase is small except over slow gradual change. These decision stumps acting as referee and base classifier often introduce instability in MetaCT by triggering a large number of concept change indications with fluctuating error rate. This was one of the causes of the poor performance of MetaCT. In addition, MetaCT has no mechanism to capture *only the essence of a concept* rather than the full representation, as opposed to DFT application in FCT. This also caused the lower accuracy of MetaCT compared to FCT.

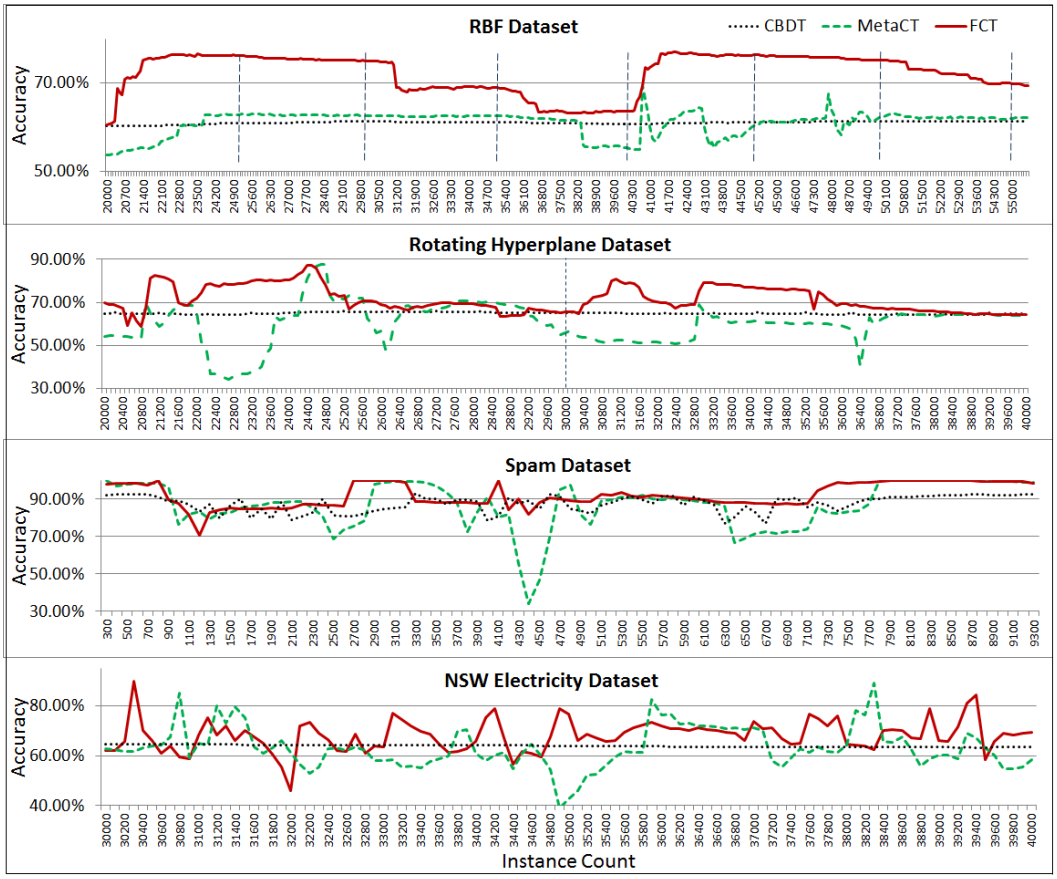


Figure 6.5: Classification Accuracy for CBDT, FCT and MetaCT

6.6.4.2 Memory

The experimentation on accuracy has revealed, especially in the case of FCT, the key role that concept capture and re-use has played in improving accuracy. The question is, what price has to be paid in terms of memory overhead in storing these recurrent concepts? Table 6.1 clearly shows that the Fourier transformed trees consume a small fraction of the memory used by the pool of trees kept in FCT's active memory, despite the fact that collectively these models outperform their decision tree counterparts at a greater number of points in the stream progression.

Comparison of overall memory consumption across FCT and MetaCT is complicated by the fact that the latter tended to have immature trees in its classifier pool that under fits concepts. Despite this, Table 6.1 reveals that FCT's memory consumption is competitive to that of MetaCT. The only case where MetaCT had a substantially lower consumption was with the Spam dataset with a lower overhead for active memory.

Datasets	Memory FCT		Memory MetaCT	
	Tree Forest	Fourier Pool	Tree Forest	Pool
RBF	97.9	24.8	122.7	14.9
Rot. Hy/plane	187.4	59.7	148.7	43.4
SEA	29.3	34.8	28.0	18.1
Spam	1712.8	18.8	878.0	15.3
Electricity	48.4	39.9	19.8	18.9

Table 6.1: Average Memory Consumption (in KBs) Comparison

Figure 6.6 shows memory profile over rotating hyperplane dataset. As seen on Figure 6.6, the growth rate of pool memory is much less than that of tree forest. Tree forest releases memory when a tree is transfered to pool or when a tree or leaf node is collapsed. Pool virtually consumes the same amount of memory after all different concepts have been captured. As mentioned earlier,

MetaCT was not successful in capturing some concepts due to the limitations of the use of Hoeffiding trees. Therefore, after first 10K instances, MetaCT stopped growing its pool. This would have enabled MetaCT to remember the decision trees representing the first two concepts only. FCT continued to refine its pool by capturing the other two concepts as well. Note that pool size can be limited to a maximum value specified as a parameter. Therefore, there is no possibility of an infinite growth of pool size.

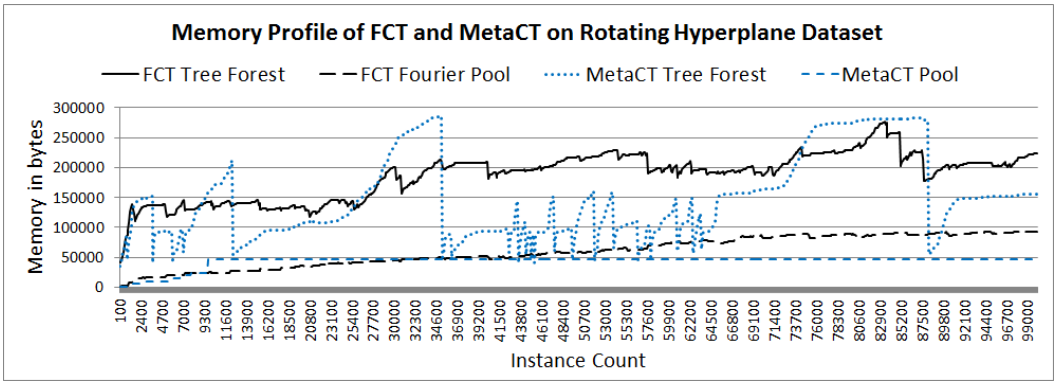


Figure 6.6: Memory profiles of FCT and MetaCT on Rotating Hyperplane Dataset

Please refer Appendix B for the memory profiles on RBF, SEA, Electricity and Spam datasets.

6.6.4.3 Processing Speed

FCT routes an instances to all classifiers to update statistics. In addition, FCT has potentially more overhead at concept change points if the winner tree is one that is selected from the active forest as this tree needs to be converted into its Fourier representation. MetaCT has one active classifier pair consisting of base classifier and its referee for classification. It has to maintain its buffer during warning phase. After a concept change, *hit percentage* needs to be computed using cached instances for all classifiers in its pool. In addition,

new classifier needs to be grown using the labeled instances in its buffer. Moreover, use of Hoeffding Tree in MetaCT introduces overhead by repeating the above operations often due to fluctuating accuracy. Thus it is interesting to contrast the run time performances of the two approaches.

Datasets	Processing Speed FCT	Processing Speed MetaCT
RBF	3540.6	2662.5
Rot. Hy/plane	2686.2	2180.1
SEA	11368.2	10125.8
Spam	4.1	4.3
Electricity	5705.7	7191.42

Table 6.2: (Processing Speed Instances per second) Comparison

Table 6.2 shows that in general FCT has a higher processing speed (measured in instances processed per second); the only exception was with the Electricity dataset where MetaCT was faster. The electricity data contains a relatively larger number of concept change points in comparison to the other datasets and this in turn required a greater number of DFT operations to be performed, thus slowing down the processing.

Discrete Fourier Transform (DFT), as expected, was able to capture the *essence* of a concept to the extent that when it reappeared in a modified form in the presence of noise, it helped recognizing similar recurrences and was able to classify it accurately.

6.6.5 Sensitivity Analysis on FCT

Having established the superiority of FCT, it would be interesting to explore the sensitivity of FCT's accuracy on two key factors: *Energy Threshold* and *Noise level in data stream.*

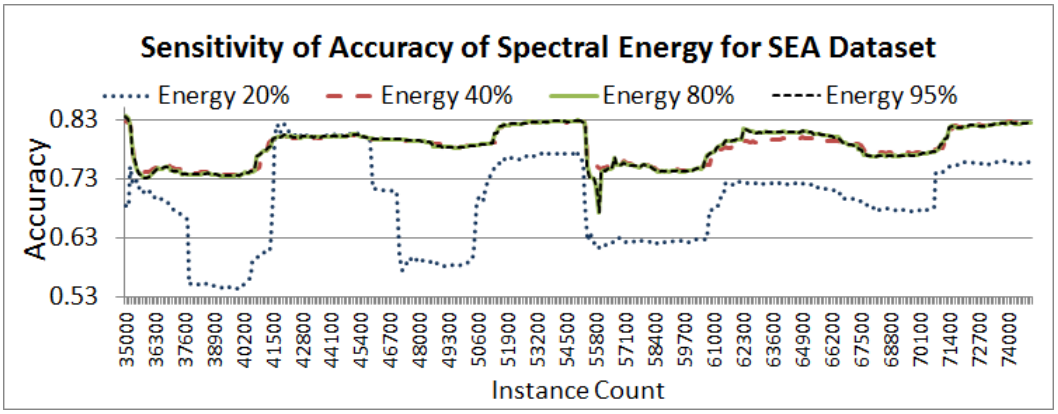


Figure 6.7: Sensitivity of Accuracy on Spectral Energy

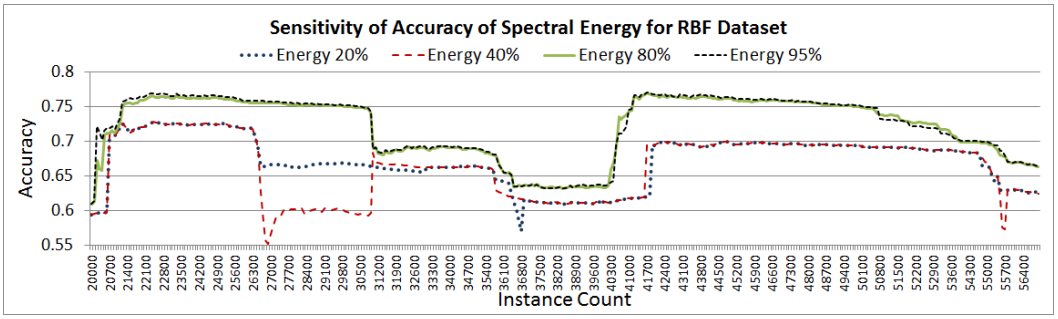


Figure 6.8: Sensitivity of Accuracy on Spectral Energy

6.6.5.1 Energy Threshold

FCT's energy threshold parameter controls the extent to which it captures recurring contexts. All datasets have been used in experiments and accuracy has been tracked across four different thresholds: 95%, 80%, 40% and 20%. Figure 6.7 clearly shows that very little difference in accuracy exists between the trajectories for 40% and 95%, showing the resilience of the DFT in capturing the classification power of concepts at low energy levels such as 40%. Similarly, Figure 6.8 also confirms that the accuracy drop for a lower value of energy is not very significant. The average difference of accuracy between 40%

and 95% on RBF dataset is 0.058. Thus the low order Fourier coefficients that survive the 40% threshold hold almost the same classification power of spectra at the 80% or 90% levels which contain more coefficients. Such higher energy spectra would represent larger decision trees in which some of the decision nodes would be split into leaf nodes, thus enabling them to reach a slightly higher level of accuracy. Interestingly, over short data segments, trees with 20% energy show better performance than that of 40%.

6.6.5.2 Noise Level

In section 6.6.4.1, it was observed that FCT outperformed MetaCT by recognizing concepts from the past even though the concepts did not recur exactly in their original form due partly to noise and partly due to different data instances being produced as a result of re-seeding of the concept generation functions. In this experiment the resilience of FCT to noise level is explicitly tested by subjecting it to three different levels of noise - 10%, 20% and 30%.

Figure 6.9 reveals three interesting pieces of information. Firstly, FCT is still able to recognize recurring concepts at the 20% noise level even though the models it re-uses do not have quite the same classification power (when compared to the 10% noise level) on the current concept due to data instances being corrupted by a relatively higher level of noise.

Secondly, FCT's concept recurrence recognition is essentially disabled at the 30% noise level as shown by its flat trajectory, thus essentially performing at the level of the base CBDT system. It is able to avoid drops in accuracy on account of the forest of trees that is maintained and is able to switch quickly and seamlessly from one tree to another when concepts change occurs.

Thirdly, although MetaCT is not the focus of this experiment it can be seen that the ability of MetaCT to recognize recurring concepts is disabled at the 20% level, showing once gain the resilience of the DFT to noise. At the

30% level its accuracy drops quite sharply at certain points in the stream.This is due to the fact that it learns a single new classifier and relies on it to classify instances in the current concept. In contrast, FCT exploits the entire forest of trees and switches from one tree to another tree in its active forest in response to concept change.

As mentioned earlier, FCT relies on the choice of a change detector to identify concept boundaries. Delay in concept change detection or false positive signals impact FCT's performance. Higher delay in detection would force FCT to postpone classifier update. This causes drop in accuracy over the new concept. At the same time, false positive change signals will cause unnecessary archival of a well-performing classier and fluctuations in accuracy. ADWIN has been chosen for all the previous experiments due to its popularity in data stream mining applications. As this research proposes a better change detection strategy called SeqDrift2, FCT is again evaluated with SeqDrift2 in place of ADWIN. The next section presents the results and comparative study with the ADWIN version.

6.7 Empirical Study on FCT with SeqDrift2 Change Detector

In this section, the impact of a change detector on FCT is analyzed. Two versions of FCT have been tested on all the datasets used in the previous section. The results for accuracy evaluation only includes the datasets RBF, Rotating Hyperplane and Electricity as the other two follow the same trend as the ones presented here. The first version is the default FCT with ADWIN change detector namely FCT+ADWIN. The second one is the FCT with SeqDrift2 namely FCT+SeqDrift2.

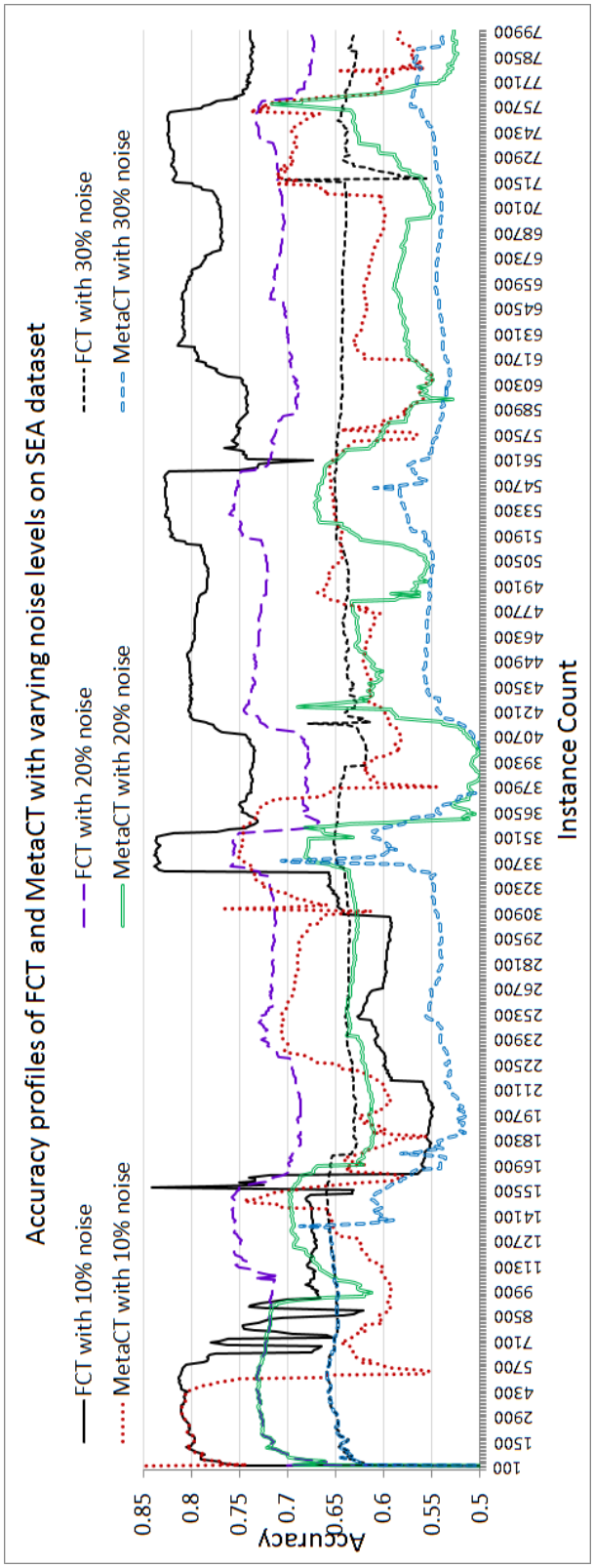


Figure 6.9: Sensitivity of Accuracy for FCT and MetaCT on Noise Level

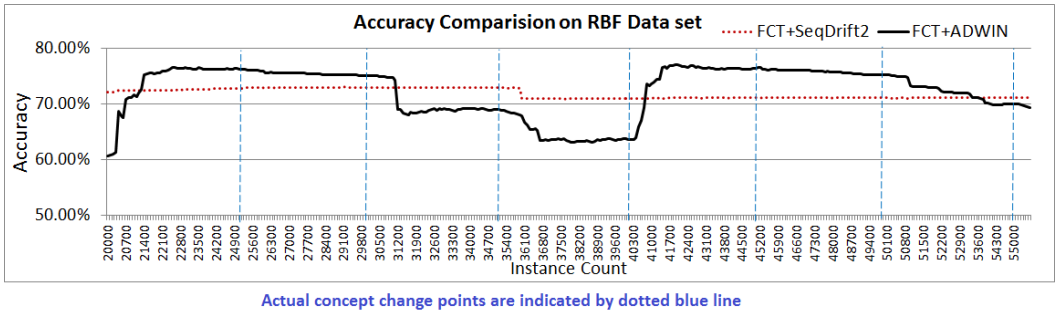


Figure 6.10: Accuracy comparison between FCT+ADWIN and FCT+SeqDrift2 on RBF dataset

6.7.1 Accuracy Comparison

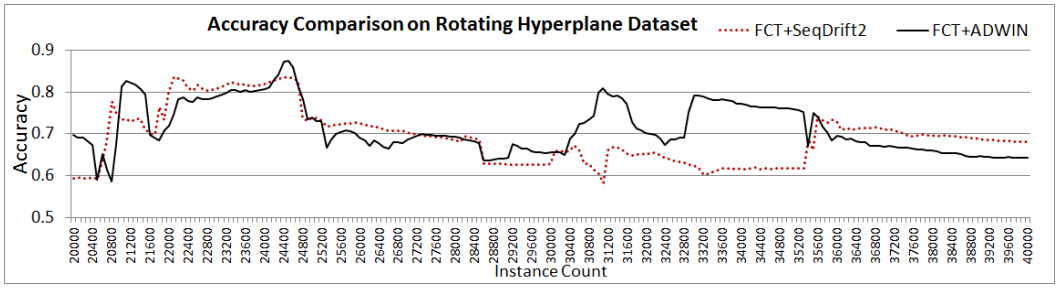


Figure 6.11: Accuracy comparison between FCT+ADWIN and FCT+SeqDrift2 on Rotating Hyperplane dataset

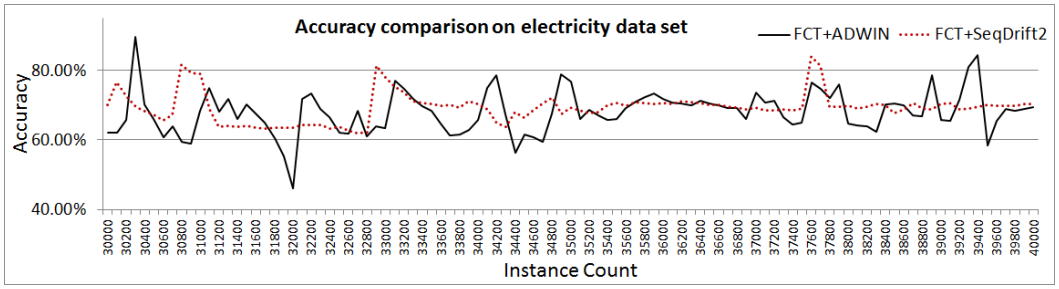


Figure 6.12: Accuracy comparison between FCT+ADWIN and FCT+SeqDrift2 on NSW Electricity dataset

Figures 6.10, 6.11 and 6.12 show that the accuracy fluctuation in FCT+SeqDrift2 is not as severe as FCT+ADWIN. On each dataset, there are regions of stability while FCT+ADWIN drops or temporarily gains accuracy. This trend is an evidence that ADWIN has triggered false positive signals though there was no true concept change. Due to the robustness of SeqDrift2, the mining model is better at deciding when to update a classifier. This is an important observation in this research as the two algorithms that are proposed are a better combination over exiting algorithm.

The other two metrics chosen for this empirical study are the processing speed and the memory consumption.

6.7.2 Processing Time and Memory Comparison

Datasets	Memory FCT+ADWIN		Memory FCT+SeqDrift2		Processing Speed	Processing Speed
	Tree Forest	Fourier Pool	Tree Forest	Fourier Pool	FCT+ADWIN	FCT+SeqDrift2
RBF	97.9	24.8	98.2	20.9	3540.6	3652.6
Rot. Hy/plane	187.4	59.7	190.4	52.6	2686.2	2659.2
SEA	29.3	34.8	26.5	33.6	11368.2	12006.4
Spam	1712.8	18.8	1816.4	17.3	4.1	4.1
Electricity	48.4	39.9	56.2	31.6	5705.7	6945.2

Table 6.3: Average Memory Consumption (in KBs) and Processing Speed Instances per second Comparison

Table 6.3 presents the average memory and processing speed values over the entire data set taken in an interval of 100 instances. There are three clear trends here. Except SEA dataset, tree forest of FCT+SeqDrift2 consumes slightly larger amount of memory than its counterpart. This is expected because SeqDrift2 has lower false positive rate than ADWIN, therefore, trees in the forest are not collapsed or transfered less frequently compared to FCT+ADWIN. Tree forest in FCT+SeqDrift2 learns the current instances and grows its trees to produce a more precise representation of a concept.

The other trend is the compactness of pool in FCT+SeqDrift2 compared to that of FCT+ADWIN. The same explanation follows here as well. Low false positive rate implies lower number of Fourier Spectra thus smaller memory consumption.

FCT+SeqDrift2 is again the winner because of its higher processing speed. Lower number of change signals reduces computationally expensive task of transforming a tree into its Fourier coefficients. In addition, less number of Fourier Trees also decreases processing time in polling to select a winner and in processing an instance.

6.8 Summary

In this chapter, a novel mechanism for mining data streams by capturing and exploiting recurring concepts is proposed. The experimentation showed that the Discrete Fourier Transform when applied on Decision Trees captures concepts very effectively, both in terms of information content and conciseness. The Fourier transformed trees were robust to noise and were thus able to recognize concepts that reappeared in modified form, thus contributing significantly to improving accuracy. Overall the proposed approach significantly outperformed the meta learning approach by Gama and Kosina [Gama 2011] in terms of classification accuracy while being competitive in terms of memory and processing speed.

As the last experiment, ADWIN change detector was replaced by SeqDrift2. There is noticeable gain in stability, memory consumption and processing speed when SeqDrift2 is used. The gain is mainly due to SeqDrift2's ability to avoid many false positive signals which are triggered by ADWIN.

The derivation of the Fourier spectrum is optimized by an efficient thresholding process but there is further scope for optimization. Currently all significant concepts are stored in the repository in the form of Fourier spectra. While energy thresholding was shown to be very effective in producing compact spectra the overhead of storing such spectra will grow over time and could potentially flood memory in highly dynamic high dimensional environment. Thus a mechanism for repository memory management is needed to ensure that memory overflow does not take place. On simple solution would be to characterize concepts into some importance measure based on their previous classification performance and their usage frequency and then use a window on the importance measure to slide out the least important concepts when memory is running low. Although attractive from the viewpoint of conceptual simplicity, such an approach has serious drawbacks. The major issue

is one of currency, concepts are poorly rated in the past could assume more importance in the future and purging such concepts would lead to a loss in accuracy upstream. In the next Chapter, an ensemble approach that effectively addresses this problem is proposed. In the ensemble approach, Fourier are aggregated into single entities thus reducing memory overhead while preserving concept integrity.

Chapter 7

The Role of Fourier Ensembles in Capturing Recurring Concepts

7.1 Introduction

Ensemble of classifiers are widely used in machine learning as ensembles have been shown to reduce learning bias and improve accuracy [Brown 2010]. In [Wang 2003b], ensemble approaches are claimed to be effective in avoiding over fitting problems and conflicting concepts. In addition, ensemble avoids the dependence on a single classifier and incorporates suggestions from a number of models that could reduce the need for the high degree tuning required for single models. Ensembles have the potential to perform better than a single classifier when the correlation of errors of individual classifiers are low [Freund 1996]. Gavin in [Brown 2010] mentions that ensemble learning have been often found to be superior than single models in numerous empirical and theoretical studies. Unstable learners such as Decision Trees can have high sensitivity to training conditions that result in high variance in its performance. High variance can be averaged out in ensemble learning. Therefore, it can be expected that the proposed method in this chapter that relies on the Decision Tree would benefit from an ensemble approach.

Physical aggregation of classifiers have the advantages of lower redundancy, lesser memory and lesser computational complexity than a group of classifiers in an ensemble. Ensemble learning may produce classifiers that are

not comprehensible or transferable and therefore would benefit from mechanisms that provide transparency in the decision making process. The work in [Iqbal 2012] attempts to extract common rules by aggregating decision trees. The approach presented in this chapter is a similar approach in extracting and weighting common patterns found in a group of decision trees in Fourier space through the use of Fourier coefficients.

This chapter investigates the effectiveness of using ensembles of Fourier spectra in capturing recurring concepts. In the case of classifiers produced by Fourier spectra, ensemble learning can be implemented by taking a linear weighted sum of spectra as described later in this Chapter. However, given that the objective is to capture recurring concepts from the past, the approach to ensemble learning will be to maintain a pool of ensembles rather than one single ensemble, with each ensemble capturing different concepts. Each ensemble will embed variants on the same concept that arise due to drift that causes a concept to change with time. In this respect, the method chosen to aggregate concepts into an ensemble is of vital importance to preserve the integrity of a concept and ensure that any given ensemble is not polluted by aggregating with a spectrum belonging to another very different concept. At the same time, a certain amount of diversity is maintained within any given ensemble in order to improve its generalization capability and thus to make it more robust to concept change and noise in the stream.

In a high speed highly volatile environment, classifier stability becomes an important issue as there is always a delay for a classifier to change its structure to suit changes taking place in a data stream. If changes take place at a higher speed than its (i.e. the classifier's) ability to learn new concepts, then both accuracy and stability will be severely compromised unless classifiers are robust enough to generalize to new changes taking place. In this respect, an ensemble of classifiers can be expected to more stable than a single classifier approach. Such highly volatile environments exist in many real world

applications such as highly volatile stock markets, social networks etc.

On the other hand, when data instances arrive at low speed, a single classifier can also suffer, due to a different reason altogether. In this case, the delay associated with detecting change is the cause of the problem. As an example, suppose that a sensor network sends data at a rate of one instance per day. Suppose that the delay with tracking change is 200 instances; then, a change in this data stream can only be recognized after 200 days during which time the current classifier will be outdated and will suffer from a loss in accuracy. Since the change was not detected there will not be an opportunity to re-use a classifier from the past that is better suited to the new concept as re-use is only triggered at concept change detection for reasons of efficiency. Thus, as with the high speed scenario, an ensemble classifier that is not overly dependent on sensitivity of the change detector and is able to generalize is expected to perform better in this scenario as well.

In order to meet the above challenges, the work proposed in the previous Chapter is extended. Firstly, instead of encoding each concept using its own Fourier spectrum, an ensemble approach is used to aggregate individual spectra into a collection of unified spectra. This has the two above mentioned advantages: firstly, memory overhead is further reduced as Fourier coefficients that are in common between different spectra can be combined into a single coefficient, thus eliminating redundancy. Secondly, due to the presence of ensembles, new concepts that manifest as a combination of previously occurring concepts have a higher likelihood of being recognized, resulting in better accuracy and stability over large segments of the data stream.

The second extension, as presented in Theorem 7.1, is the derivation of an efficient method for spectral energy thresholding that directly controls the degree of compression that can be obtained in encoding concepts in the repository.

The third major extension is optimization of the DFT encoding process for

which a potentially expensive inner product operation is required. Theorem 7.2 in section 7.2.3. reduces the computational overhead by identifying two cases where the inner product can be computed in O(1) time.

The empirical study covers two different types of environments where concepts recur in different fashion. The first environment is a simulation of a large number of concepts recurring in similar form. This is to analyze how well highly volatile environments are handled by ensembles operating under a memory constrained environment. In addition, two different ensemble formulation schemes are compared to understand the influence of a similarity measure on classification accuracy and stability. The scheme with the best aggregation strategy has then been taken to further experiment with the Seq-Drift2 change detector that has been proposed in this research to understand the influence of a change detector on performance.

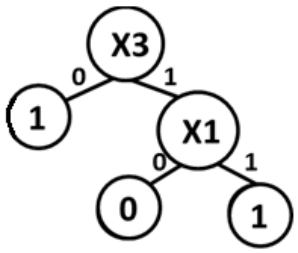
The next section covers the theoretical aspects of decision tree aggregation in Fourier space.

7.1.1 Aggregation of Fourier Spectrum

The fundamental aggregation operation is straightforward, as described by Kargupta et al. in [Kargupta 2006]. Kargupta et al. use a weighted linear combination of spectra to produce an ensemble.

$$s_c(x) = \sum_i W_i \sum_i s_i(x) = \sum_i W_i \sum_{j \in P_i} {\omega_j}^{(i)} \overline{\psi_j^{\lambda}}(x) \tag{7.1}$$

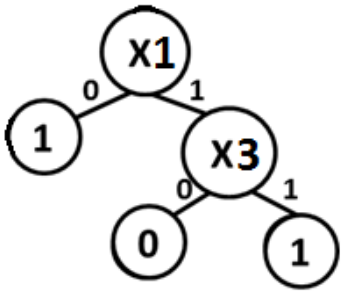
where $s_c(x)$ denotes the ensemble spectrum produced from the individual spectra $s_i(x)$. W_i is the relative accuracy of the corresponding spectrum and P_i is the set of partitions for non zero coefficients in spectrum s_i.



X (Path Vector)			
X1	X2	X3	f(x)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

ω_j	Value
ω_{000}	3/4
ω_{001}	1/4
ω_{010}	0
ω_{011}	0
ω_{100}	-1/4
ω_{101}	1/4
ω_{110}	0
ω_{111}	0

Figure 7.1: Decision Tree 1 with 3 binary features



ω_j	Value
ω_{000}	3/4
ω_{001}	-1/4
ω_{010}	0
ω_{011}	0
ω_{100}	1/4
ω_{101}	1/4
ω_{110}	0
ω_{111}	0

Figure 7.2: Decision Tree 2 with 3 binary features

With reference to Figures 7.1 and 7.2 , the aggregation of coefficient ω_j can be done as follows:

$$\omega_{c_j} = \frac{Acc_{s1}}{Acc_{s1} + Acc_{s2}} \omega_{(s1)_j} + \frac{Acc_{s2}}{Acc_{s1} + Acc_{s2}} \omega_{(s2)_j} \tag{7.2}$$

where ω_{c_j} denotes the aggregated j^{th} coefficient and Acc_{s1} and Acc_{s2} are the accuracies of the Fourier spectrum generated from Figures 7.1 and 7.2 respectively.

This process needs to be repeated for all coefficients that are present in both $s1$ and $s2$.

7.2 Exploitation of the Fourier Transform for Recurrent Concept Capture

The basic algorithm used in Section 7.2.1 is also presented in Figure 7.3 as a structural diagram. The discussion on an optimization used for energy thresholding is explained in Section 7.2.3.

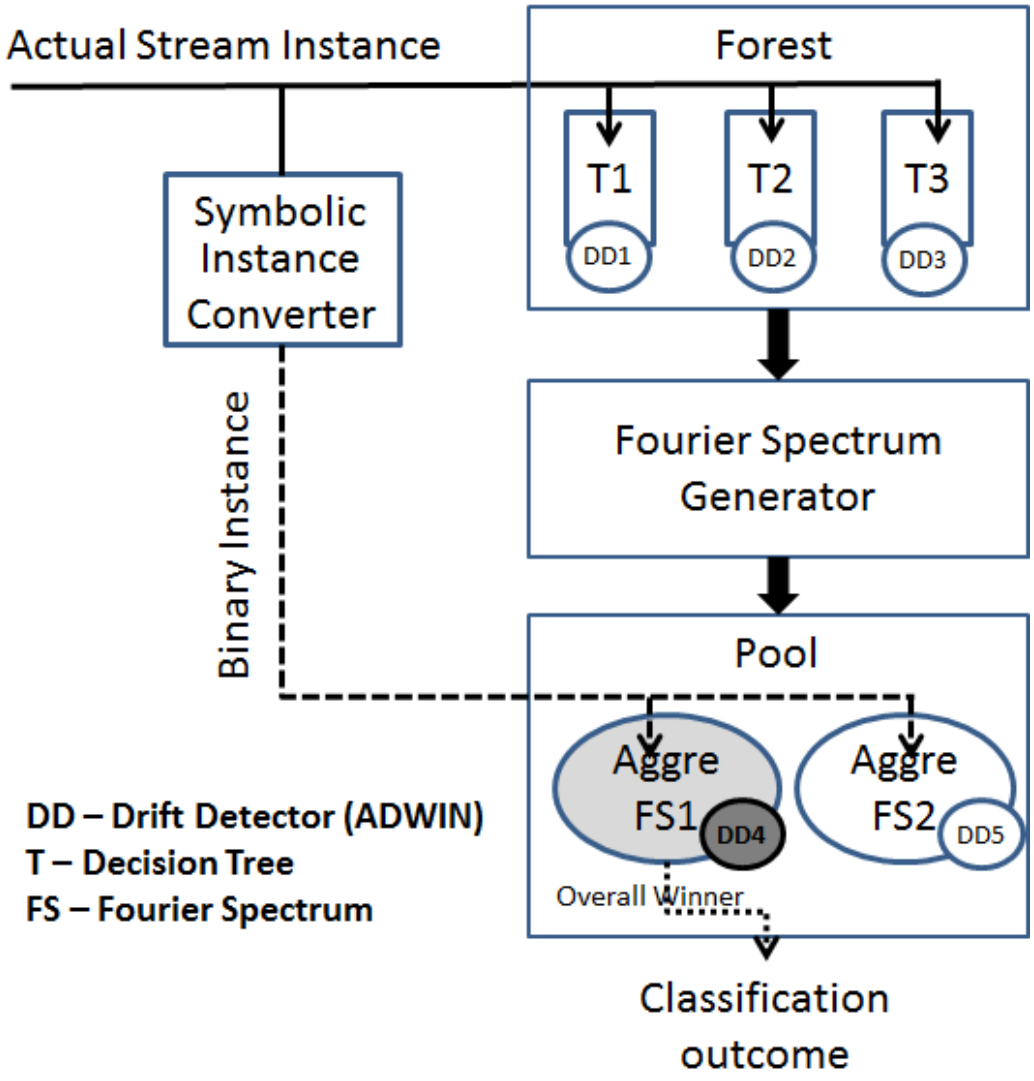


Figure 7.3: EP Structural Diagram

CBDT [Hoeglinger 2009] has been as the base classifier which maintains a forest of Hoeffding Trees [Hoeglinger 2007], as in the previous chapter.

As shown in Figure 7.3, the memory is divided into two segments: firstly to support the growth of a forest of decision trees; and secondly, to support a collection of Fourier Ensemble Spectra encoded and aggregated from several Decision Trees. Each of the decision trees is encoded into a Fourier Spectrum that had the best classification accuracy across the forest at a particular concept change point, as with the operation of the FCT algorithm in the previous Chapter. Each Hoeffding Tree and Fourier Spectrum in the pool is equipped with an instance of the change detector, ADWIN [Bifet 2007]. In FCT each Fourier spectrum is represented individually as a Fourier Concept Tree (FCT). In this work, spectra are aggregated as described and maintained in a pool of ensemble spectra known as Ensemble Pool (EP). The EP creation process is described in Algorithm EP (7.1) and how FCT can be generated from it as a special case is also discussed.

In practice, any incremental decision tree approach that uses a forest of decision trees can be used to create a system like EP.

7.2.1 The EP Algorithm

Algorithm 7.1 EP Algorithm

Input: Energy Threshold E_T , Accuracy Tie Threshold τ
Output: Best Performing model M that suits current concept

1 Plant a decision tree rooted on each attribute found in the data stream
2 M is set to a randomly selected decision tree model from the forest
3 Initialize the repository to null
4 Read an instance I from the data stream
5 **while** *change is not detected by the ADWIN instance of the current best model* M **do**
6 Apply all models in the forest and the repository on I to classify I
7 **if** *the overall winner is a tree from the forest* **then**
8 Increase the *counter* of each ensemble in the pool if the classification output matches with that of the current active winner
9 Append 0 to the embedded ADWIN's window for each model if classification is correct, else 1
10 **if** *M is from decision tree forest* **then**
11 Identify the best performing model F in repository
12 **if** *(accuracy(M)-accuracy(F))*$> \tau$ **then**
13 apply DFT on model M to produce F* using energy threshold E_T
14 **if** *F* is not already in repository* **then**
15 Call *Aggregation* algorithm 7.2 and insert F* into the repository
16 Reset the *counter* of ensemble trees in the pool
17 Identify best performing model M by polling active memory and repository
18 GoTo 4

In step 1 of Algorithm EP, 7.1 a decision tree rooted on each attribute is created. In step 2 a randomly selected decision tree that is created in step 1 is chosen as the best performing model M. Next, an empty pool is created in step 3. Each incoming instance is routed to all trees in the forest and the pool, until a concept change signal is triggered by the ADWIN instance attached to

Algorithm 7.2 Aggregation

Input: Spectrum S derived from best performing tree T in active memory, set of existing ensembles E, concept length L, similarity threshold

Output: Aggregated Spectrum F*

1 Select the ensemble E* that has the maximum number of matching classification outputs with T

2 Calculate matching percentage (MP_{E*}) of E* over L

3 **if** MP_{E*} > *Similarity Threshold or no room for a new ensemble in the pool* **then**

4 Aggregate E* and S using equation 7.10 to produce F*

5 **else**

6 Return S as F*

the winner tree (steps 6 to 9). At the same time, the *counter* of each ensemble tree is increased whenever the classifications of an ensemble and an active tree matches in the event that an active tree emerges as the overall winner at the current concept change point. This counter remembers the number of times both an ensemble and active winner tree agrees in classification.

The empirical study tests another aggregation strategy based on similarity of accuracy as the aggregation criterion. With this strategy, there is no need to keep a counter for each ensemble and the selection of best matching ensemble is based on an accuracy tie threshold.

At the first concept change point, the best performing tree (in terms of ADWIN's estimate of accuracy) is identified and the DFT is applied after energy thresholding. Thereafter, the resulting spectrum is stored in the repository for future use if the current concept recurs. The spectra stored in the pool are fixed in nature as the intention is to capture past concepts. A new winner is then identified as shown in step 17.

At each subsequent change point, if the winner model at a change point emerges from an already existing spectrum in the pool, then no Fourier conversion is applied on any of the trees in active memory. The winner model is applied to classify all unlabeled data instances until a new winner emerges at

a subsequent change point, Otherwise, if the winner emerges from the active memory, a check is made before the DFT is applied. First of all, a comparison is made to see whether the difference in accuracy between the winner tree in active memory (T) and the best performing model in the repository (from step 11) is greater than a tie threshold τ (step 12). If this check is passed then the DFT is applied to T (step 13) and a further check is made to ensure that its Fourier representation is not already in the pool (step 14). If this check is also passed, algorithm *Aggregation* 7.2 is called to integrate the current Fourier representation (Fourier Spectrum) into an existing Fourier Spectrum or plant it as a separate Fourier spectrum in the pool (step 2 to 6).

The algorithm 7.2 searches for the spectrum having the greatest structural similarity to the currently generated spectrum (step 1). Step 1 evaluates the degree of agreement (d) between the classification decisions (c) for S and E* over the entire concept length using counters that remembers the matching classification outputs of S and E*. As mentioned earlier, an alternative strategy to aggregating structurally similar spectra would be aggregation based on similar accuracy. To distinguish with ensembles created with the structural similarity criterion, these ensembles are referred to as EP_a.

As stated before, the FCT algorithm is a special case of the EP algorithm that omits the call to the aggregation algorithm made in step 15. The next section describes the three optimizations done on EP to enhance the performance of EP especially in terms of processing speed and memory.

7.2.2 Optimizing the Energy Thresholding Process

Sakthihasan et al. in [Sripirakas 2014a] showed that classification accuracy is sensitive to spectral energy; the higher the energy the greater is the classification accuracy in general, although the relationship is by no means a linear one as energy thresholding at values of 40% and 60% yielded accuracy

profiles similar to the 95% energy level. Thresholding on spectral energy is thus an effective method of obtaining a compact spectrum while retaining the classification power inherent in the decision tree counterpart.

A solution described in [Sripirakas 2014a] was to iterate through each order of the spectrum and compute the cumulative energy fraction as in the equation 7.3.

$$CEF_{i+1} = \frac{CE_i + E_{i+1}}{CE_i + dE_{i+1}} \tag{7.3}$$

where CE_i is the cumulative energy calculated from *order* 0 to i, E_{i+1} is the energy of order $i+1$, d is the number of orders that are not included in the calculation of CE_i. This thresholding depends on the estimation computed by $d.E_{i+1}$. Though, this will ensure that the actual fraction at *order* i will be at least the estimated value CEF_{i+1}, due to the over estimation of total energy $CE_i + dE_{i+1}$, the thresholding procedure is not guaranteed to terminate at the optimal number of *orders* that captures the energy percentage specified as a parameter. An alternative approach is to calculate $\frac{E(O_i)}{E(O_{i-1})}$ where $E(O_i)$, $E(O_{i-1})$ are the energies at orders i and $i-1$ respectively. Thresholding can then be implemented at order O when the ratio is less than some small tolerance value, say 0.01. The drawback of this simple solution is that it does not guarantee that the cumulative energy up to order O contains a proportion (ε) of the total energy. Fortunately, a solution exists for this problem. Theorem 7.1 proves that E(T) equals to $\omega_{\overline{0}}$ (The 0^{th} coefficient). Thus total energy can be computed efficiently, without having to enumerate all coefficients in the spectrum.

Theorem 7.1 *The total spectral energy* $E = \sum_j \omega_j^2 = \omega_{\overline{0}}$, *where* $\omega_{\overline{0}}$ *denotes the coefficient with order 0, which is easily computed as its Fourier basis function is unity.*

Proof Induction is used for the proof. The case when the number of attributes in the tree is 1 is proved first. Suppose that there are c classes in

total. Without loss of generality, a binary tree can be used for classification with x_1 as input and a classification of 0 if $x_1 = 0$ and 1 otherwise. If the class outcome is not 0 then the binary tree is simply reused with x_1 to obtain a result. If the 0 class outcome was obtained after n iterations ($n \leq c$) then the class outcome can be deduced as n. This means that a single binary tree is sufficient. For this binary tree $\omega_0 = \frac{1}{2}$ and $\sum_j \omega_j^2 = \frac{1}{2}$, thus giving $\omega_0 = \sum_j \omega_j^2$ and proving the theorem when the number of attributes is 1.

Now it is assumed that the theorem is true when the number of attributes is k. This means that for any tree T_i with k attributes $x_0, x_1, \cdots, x_{k-1}$, thus,

$$E(T_i) = {\omega_0}^{T_i} \tag{7.4}$$

λ_k sub-trees of a larger tree LT with an additional (new) attribute x_k as the root attribute is considered, where λ_k is the cardinality of the new attribute x_k

For any T_i, $E(T_i) = \sum_{p=0}^{\lambda_i - 1} E(T_i)_p$ as the energy of any given tree is simply the sum of the energies along each of its path vectors p. For tree LT with $k+1$ attributes, each of its constituent sub-trees T_i will contribute a proportion $\frac{\lambda_i}{\lambda_k}$ to the total energy of tree LT. Thus:

$$E(LT)_i = \frac{\lambda_i}{\lambda_k} E(T_i) \tag{7.5}$$

After sub-trees $T_0, T_1, \cdots, T_{k-1}$ are combined into LT, the following must be

true,

$$
\begin{aligned}
E(T) &= E(LT)_0 + E(LT)_1 + \cdots + E(LT)_{k-1} \\
&= \left(\frac{\lambda_0}{\lambda_k}\right)E(T_0) + \left(\frac{\lambda_1}{\lambda_k}\right)E(T_1) + \cdots + \left(\frac{\lambda_{k-1}}{\lambda_k}\right)E(T_{\lambda_k-1}) \text{ from eq 7.5} \\
&= \left(\frac{\lambda_0}{\lambda_k}\right)\omega_0^{T_0} + \left(\frac{\lambda_1}{\lambda_k}\right)\omega_0^{T_1} + \cdots + \left(\frac{\lambda_{k-1}}{\lambda_k}\right)\omega_0^{T_{\lambda_k-1}} \text{ from eq 7.4} \\
&= \frac{\sum f^{T_0} + \sum f^{T_1} + \cdots + \sum f^{T_{\lambda_{k-1}}}}{\lambda_k} \text{as} \\
&\sum f^{T-i} \text{ as the classificationsum is } \lambda_i \omega_i^{T-i} \\
&= \omega_0^{LT} \text{ as } \omega_0^{LT} \text{ is the sum of the classification over} \\
&\text{all paths dividedby the cardinality } \lambda_k
\end{aligned}
$$

This optimization significantly increased processing speed in the preliminary experiments, especially when a large number of attributes were present in the data stream.

The next optimization is applied on the inner product calculation. The Fourier Basis function calculation is optimized ($(-1)^{j.x}$) in equation 6.1 in Chapter 6 especially when wild card characters are present in path vector x. The following optimization shows a strategy to directly calculate the Fourier basis function, not the inner product itself.

7.2.3 Optimizing the Computation of the Fourier Basis Function

The computation of a Fourier basis function for a given partition j in generic $n-ary$ ($n \geq 2$) domain is given by:

$$\sum_{x \in S}(\psi_j(x)) = \sum_{x \in S}\prod_m \exp^{\frac{2\pi i j_m x_m}{\lambda_m}} \tag{7.6}$$

Thus, it can be seen from 7.6 that the computation of $\sum_{x \in S}\psi(j)$ over a set of schema S requires the computation of an expensive inner product operation between the x and j. However, it is possible to optimize this inner product computation as defined in Theorem 7.2.

Theorem 7.2 *The computation of* $\sum_{x \in S} \psi_j(x)$ *can be optimized as follows:*

- *Case 1: If there exists at least one* $(p, *)$ *combinations with* $p \in j$, $p \neq 0$ *and* $*$ *a wildcard character defining a set of schema* S, *then* $\sum_{x \in S} \psi_j(x) = 0$.
- *Case 2: else if there exists* n *combinations of* $(0, *)$ *pairs in the* j *and* x *vectors respectively, then*

$$\sum_{x \in S} \psi_j(x) = \lambda \prod_{k=n}^{\lambda_k - 1} \exp^{\frac{2\pi i j_k x_k}{\lambda_k}} \text{ where } \lambda = \prod_{l=0}^{n-1} \lambda_l$$

Proof For case 1, the result is proved when exactly one such combination exists and the extension to the case when more than one combination is present is discussed. Without loss of generality, the proof is illustrated when the wild card characters occur at the beginning of vector x; if they occur in any other position, then a simple reordering operation can be used without affecting the validity of the proof.

Suppose that the cardinality of the attributes after reordering are λ_i where $i \in [0, d-1]$, where d is the dimensionality of the dataset.

$$\begin{aligned}
\sum_{x \in S} \psi_j(x) &= \exp\Big(\frac{2\pi i j_0 0}{\lambda_0}\Big) \times \exp\Big(\frac{2\pi i j_1 x_1}{\lambda_1}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \\
&+ \exp\Big(\frac{2\pi i j_0 1}{\lambda_0}\Big) \times \exp\Big(\frac{2\pi i j_1 x_1}{\lambda_1}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \\
&\vdots \\
&+ \exp\Big(\frac{2\pi i j_0 (\lambda_0 - 1)}{\lambda_0}\Big) \times \exp\Big(\frac{2\pi i j_1 x_1}{\lambda_1}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \\
&= \exp\Big(\frac{2\pi i j_1 x_1}{\lambda_1}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \sum_{k=0}^{\lambda_0 - 1} \Big(\exp\Big(\frac{2\pi i j_0 k}{\lambda_0}\Big)\Big) \\
&= \exp\Big(\frac{2\pi i j_1 x_1}{\lambda_1}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \sum_{k=0}^{\lambda_0 - 1} \Big(\psi_o \psi_k\Big) \text{ as } \psi_0 = 1 \\
&= 0 \text{ due to orthogonality of the Fourier basis functions}
\end{aligned}$$

When n wild cards are present, the extension is straightforward and results in:

$$\sum_{x \in S} \psi_j(x) = \exp\Big(\frac{2\pi i j_n x_n}{\lambda_n}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \prod_{l=0}^{n-1} \sum_{k=0}^{\lambda_l - 1} \Big(\exp\Big(\frac{2\pi i j_l k}{\lambda_l}\Big)\Big) \tag{7.7}$$

$$\sum_{x \in S} \psi_j(x) = \exp\Big(\frac{2\pi i j_n x_n}{\lambda_n}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \prod_{l=0}^{n-1} \sum_{k=0}^{\lambda_l - 1} \Big(\psi_0 \psi_k\Big) \tag{7.8}$$

The value of Case 1 is that a simple scan of the j and x vectors will save a total of d multiplications and $d-1$ additions.

Now Case 2 is considered.

For convenience of proof, a re-ordering operation is performed to bring all the n $(0, *)$ pairs to the beginning, just as with Case 1. Following expression (7.8), thus,

$$\sum_{x \in S} \psi_j(x) = \exp\Big(\frac{2\pi i j_n x_n}{\lambda_n}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \prod_{l=0}^{n-1} \sum_{k=0}^{\lambda_l - 1} \exp\Big(\frac{2\pi i j_n 0}{\lambda_k}\Big) \tag{7.9}$$

$$= \exp\Big(\frac{2\pi i j_n x_n}{\lambda_n}\Big) \times \cdots \times \exp\Big(\frac{2\pi i j_{d-1} x_{d-1}}{\lambda_{d-1}}\Big) \prod_{l=0}^{n-1} \lambda_l$$

as each summation term i in equation 7.9 reduces to 1 Since $\prod_l \lambda_l$ is a constant for all possible values of j and y, the value of Case 2 is that a scan of the two vectors will avoid the overhead of n multiplications and $n-1$ additions.

7.2.4 Localized Approach to Ensemble Learning in the Fourier Domain

In order to realize the full benefits of ensemble learning in the Fourier domain, individual spectra that represent different concepts which manifest at different points in the stream are aggregated. The aggregation operation is

straightforward, as described in section 7.1.1.

$$s_c(x) = \sum_i A_i \sum_i s_i(x) = \sum_i A_i \sum_{j \in P_i} {\omega_j}^{(i)} \overline{\psi}_{j(x)} \tag{7.10}$$

where $s_c(x)$ denotes the ensemble spectrum produced from the individual spectra $s_i(x)$ produced at different points i in the stream; A_i is the classification accuracy of its corresponding spectrum and P_i is the set of partitions for non zero coefficients in spectrum s_i.

Park in [Byung-Hoon 2001] used ensemble learning with Fourier spectra in a different setting to this research. They considered a distributed system with each node i producing its own spectrum $s_i(x)$ with aggregation taking place at a central node. In a data stream environment setting, all spectra are not present in advance but the same principle can be used due to the distributive nature of the linear weighted sum expressed by (7.10). Hence:

$$s_c^{(i+1)}(x) = s_c^{(i)}(x) + A_{i+1} s_{i+1}(x) \tag{7.11}$$

where $s_c^{(i+1)}(x)$, $s_c^{(i)}$ represent the ensemble spectra at concept change points $i+1$ and i respectively in the stream and $s_{i+1}(x)$ is the spectrum produced at change point $i+1$ with accuracy A_{i+1}.

The expression (7.11) is used for implementing ensemble learning but with one essential difference. A direct application of (7.11) using the entire (global) set of attributes G comprising the dataset would be inefficient. As there are exponential number of coefficients with respect to the number of attributes present in the stream, this would be a bottleneck in high dimensional environments. One practical solution is to populate the spectrum only using the attributes present in a given decision tree. The major advantage of this approach is smaller computational overhead as the Fourier computational effort is directly proportional to the size of the attribute set used. Then this initial spectrum can be extended to a full length spectrum containing the attributes that are absent in the given tree, using a simple transformation scheme.

An attribute set of a Decision Tree is defined as that subset of attributes which define splits in the tree. Suppose that the spectra from decision trees D_1 and D_2 having attribute sets L and M respectively are integrated. The DFT is applied on D_1 to obtain S_1 using *only the attributes in its attribute set L and not all attributes in G*. SimilarlyS_2 is generated from D_2 using only the attributes defined in M.

Now, in order to integrate S_1 with S_2 the differences in the attribute sets L and M are needed to be accounted for. To do this, S_1 is taken and expand the spectrum by incorporating attributes in the set $M \setminus L$. The expansion is defined by a single operation:

For each schema instance in the spectrum (say S_1) expand the spectrum by adding 0 to all attribute index positions in set $M \setminus L$. The *coefficient value after expansion will remain the same* as the classification f value for all of these added index positions remains unchanged. Now the two spectra produced from their own localized set of attributes can be integrated. Essentially, this means that this is a more efficient method of implementing ensemble learning using expression (7.11). The next section presents the empirical outcomes of the proposed model with the above mentioned three optimizations.

7.3 Experimental Study

The main focus of the study is to assess the effectiveness of the ensemble EP approach vis-a-vis FCT in respect of accuracy, memory consumption, processing speed, and tolerance to noise.

7.3.1 Parameter Values

All experimentation was done with the following default parameter values:

- **Hoeffding Tree Default Parameters**

  - The desired probability of choosing the correct split attribute=0.99
  - Tie Threshold=0.01
  - Growth check interval=32

- **Tree Forest Default Parameter Values**
  - Maximum Node Count=5000
  - Maximum Number of Hoeffding Trees=50

- **Fourier Pool Default Parameter Values**
  - Accuracy Tie Threshold τ=0.01 which is the minimum accuracy difference between a new candidate Fourier Tree and any existing Fourier Tree in the Fourier pool
  - Maximum Fourier/Ensemble Trees = 3
  - Similarity Threshold = 30% (for EP)

- **Fourier Tree Default Parameter Values**
  - Energy Threshold = 80%

- **ADWIN Default Paramater Values**
  - drift significance value=0.01
  - warning significance value=0.3 (MetaCT only)

All experiments were done on the same software with C# .net runtime and hardware with Intel i5 CPU and 8GB RAM, with a memory flush at each run in order to have a fair comparison.

7.3.2 Datasets used for the experimental study

The experimentation has been done with 3 synthetic data generators commonly used in change detection and recurrent concept mining, namely SEA, RBF and Rotating hyperplane generators. The same data generators were used in the previous chapter as well. All synthetic datasets were generated within the MOA data stream tool [Bifet 2010b]. Please refer Chapter 6 for more details on the data generation process. The difference between the datasets used in Chapter 6 and this Chapter is the method by which multiple concepts are concatenated together. The target environment of the experimentation to evaluate aggregation is a highly volatile environment. Therefore 10 different concepts were generated using RBF and Rotating Hyperplane data generators, each of which spanned 5,000 instances and each occurred a total of 3 times at different points in the stream. SEA generator is designed to create 4 concepts only, therefore, SEA data set has been limited only to 4 concepts. In order to challenge the concept recognition process, 10% noise was added for all synthetic data sets to ensure that concepts recur in similar, but not exact form.

Each of those 10 concepts has been generated by adjusting one influential parameter of each data generator. 10%, 20% and 30% random noise have been added to the entire dataset to produce each noisy dataset. Noisy datasets play two roles. It simulates a real world data stream in addition to being a similar (but not exact) reproduction of the original concept. The details of parameters adjusted are given below:

- **SEA [Street 2001]:** The concepts are defined by the function $feature1 + feature2 > threshold$. The concepts are ordered as concept1, concept2, concept3 and concept4 generated using threshold values 8,7,9 and 9.5 respectively on the first data segment of size 20,000. Three recurrences of a modified form of these concepts were generated by using different

seed values in MOA for each sequence of recurrence.

- **RBF:** The number of centroids parameter was adjusted to generate different concepts for the RBF dataset. Concept1... 10 were produced with the number of changeing centroids set to 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 respectively and the total number of centroids was set to 50. Each of the 10 concepts was repeated three times.

- The number of changing attributes was set to 2 and sigma percentage was adjusted to 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 in a 10 dimensional dataset to create the four concepts.

7.3.2.1 Real World datasets

1. *Spam Data Set:* The Spam dataset was used in it original form [1] which encapsulates an evolution of Spam messages. There are 9,324 instances and 499 informative attributes.

2. *Electricity Data Set:* NSW Electricity dataset is also used in its original form [2]. There are two classes *Up* and *Down* that indicate the change of price with respect to the moving average of the prices in last 24 hours.

The next section describes the three models that were used to evaluate the effectiveness of ensemble (aggregated) approach over single Fourier Trees (Fourier Spectra). All three models were evaluated on the above five datasets.

7.3.3 Models used in empirical study

The experiments included two aggregation based models and a non-aggregation approach. FCT, the proposed approach in Chapter 6 is one of the chosen and it is based on a non-aggregation strategy. Two different aggregation methods

[1] from *http://www.liaad.up.pt/kdus/products/datasets-for-concept-change*
[2] from *http://moa.cms.waikato.ac.nz/datasets/*

were tested using Algorithm 7.1 as the base. One method, namely EPa that aggregates similar performing classifiers based on the accuracy values whereas the other method, EP focuses on structural aggregation.

The hypothesis behind EPa is that *the aggregation of similar performing spectra produces better spectra in terms of performance and memory.* As long as enough space for a new ensemble tree is available in Fourier Pool, aggregation is strictly carried out between the best ensemble and new Fourier Spectra if and only if an accuracy tie threshold is satisfied. Please refer Algorithm 7.2.

EP performs aggregation on the basis of structural similarity. Structural similarity can be indirectly checked by comparing classifications (not dependent on whether classification is correct or not) of an ensemble and a new Fourier spectrum for a given set of data instances. Algorithm 7.1 incrementally increases a counter at each of the ensemble classifier in the pool if its classification matches that of current winner tree in the forest. This can be done with an instance buffer that gathers data instances from the beginning of a concept, but it will be memory and computationally intensive. Therefore, counters are updated as each ensemble processes each instance of the current concept. This is a memory-less operation as opposed to a buffer based strategy and is computationally inexpensive.

Comparison of EPa and EP against FCT reveals the contribution of aggregation in a recurring concept data stream whereas the comparison between EPa and EP helps analyze the influence of different aggregation methods.

7.3.4 Comparative Study : FCT Vs EPa Vs EP

The focus here is on a comparative study of the Ensemble versus single spectrum approach. With this in mind, three types of experiments have been designed to study the impact of aggregation of spectra via an ensemble.

Firstly, the effects of constraining the number of objects that could be

stored in the repository are investigated. In a highly volatile real world stream where new concepts emerge very often, a limited number of concepts can be accommodated in a repository and so the purpose was to assess classification accuracy against repository pool size for each approach.

Secondly, the effects of noise on accuracy is studied. Noise is an integral part of any real world data stream and as the level of noise increases, concepts will increasingly deviate from their original form. A robust mechanism is then needed to recognize partial recurrence of concepts. The hypothesis here is that *ensemble spectra are better equipped to handle partial recurrences.*

Thirdly, the effects of the energy thresholding level are investigated. As the threshold is increased, accuracy is expected to increase as spectra contain more coefficients, thus improving the chances of reproducing classification decisions made by the original decision tree. However it is hypothesized that *the ensemble approach will have better resilience than FCT at low energy levels due to different spectra complementing each other.*

7.3.5 Effects of Pool Size

In this experiment, the effects of a highly volatile stream are simulated by constraining the number of objects that could be stored in the repository. In terms of the EP method, the number of objects correspond to the number of ensemble spectra whereas with FCT it refers to the number of individual spectra. The object sizes used were 3, 5 and 10 and for each size and the classification accuracy for all 5 datasets were recorded with the progression of the stream.

At size 3, memory is severely limited as 10 concepts are present in the stream. This limitation is felt more severely by FCT as at a given concept recurrence point, it has on the average $\frac{3}{10}$ probability of having the correct concept stored in its repository. Thus on the average (assuming uniform priors

for concept recurrence) $\frac{7}{10}$ of the time it will need to rely on trees in active memory for classification. The drawback of using such trees is that they may not be suited to the current concept until such trees have adapted to the new concept, which will take time as sufficient number of training samples will need to be gathered. EP, on the other hand does not suffer from this problem to the same extent as FCT does as the concept (along with others) will be embedded in one or more of the 3 ensembles especially when the threshold to find the best matching pair of Fourier Spectra is slightly relaxed.

In a highly volatile high speed environment, the disparity between EP and FCT will be much larger than with the experimentation carried out here as the probability of a cache hit in the repository for FCT will in general be of the order of $\frac{N}{M}$ where N is the size of the repository and M is the number of concepts that can recur at a given point in time. In practice, $M \gg N$, thus favoring EP over FCT.

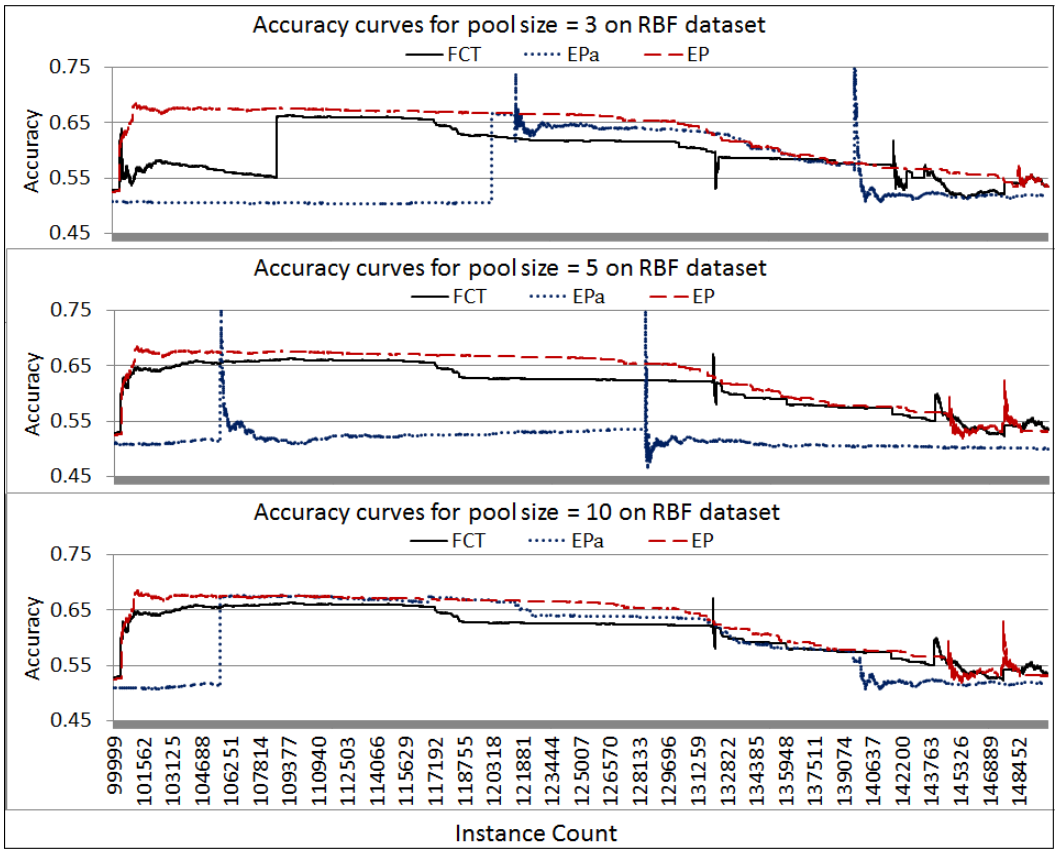


Figure 7.4: Accuracy Profile of FCT, EPa and EP on RBF dataset for pool sizes 3,5 and 10

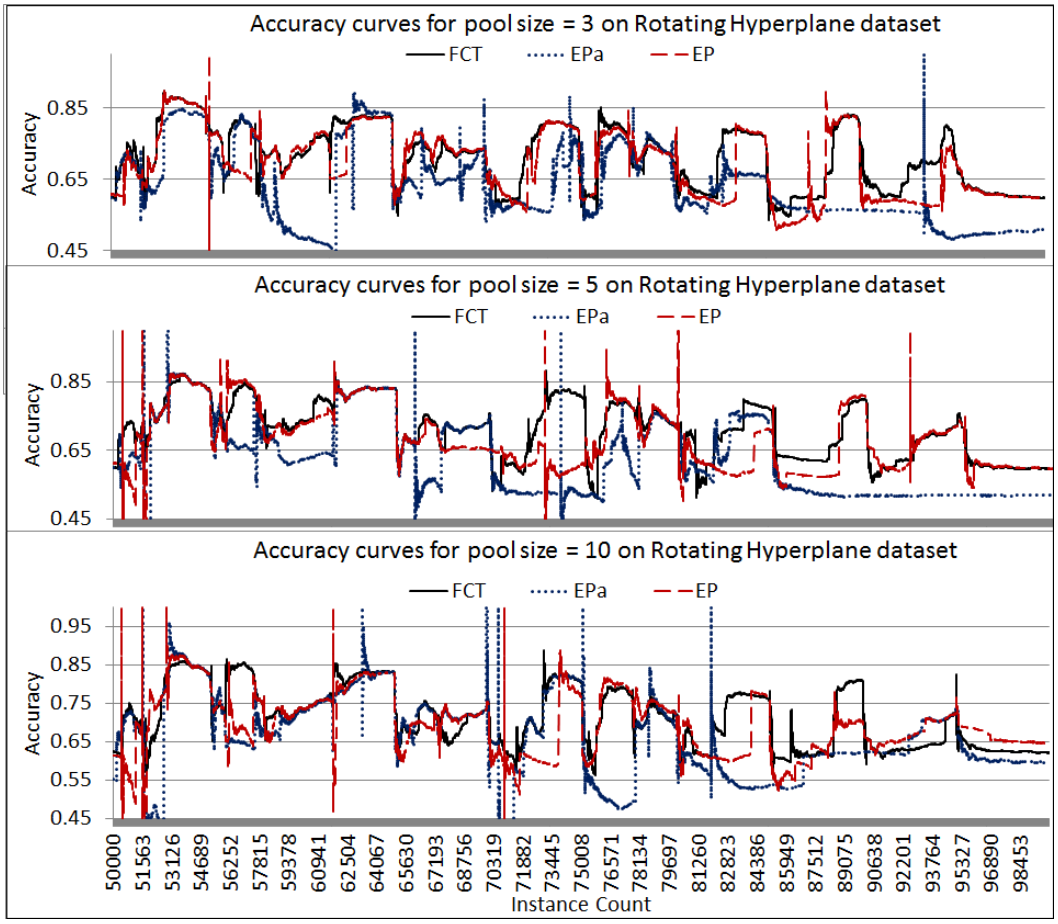


Figure 7.5: Accuracy Profile of FCT, EPa and EP on Rotating Hyperplane dataset for pool sizes 3,5 and 10

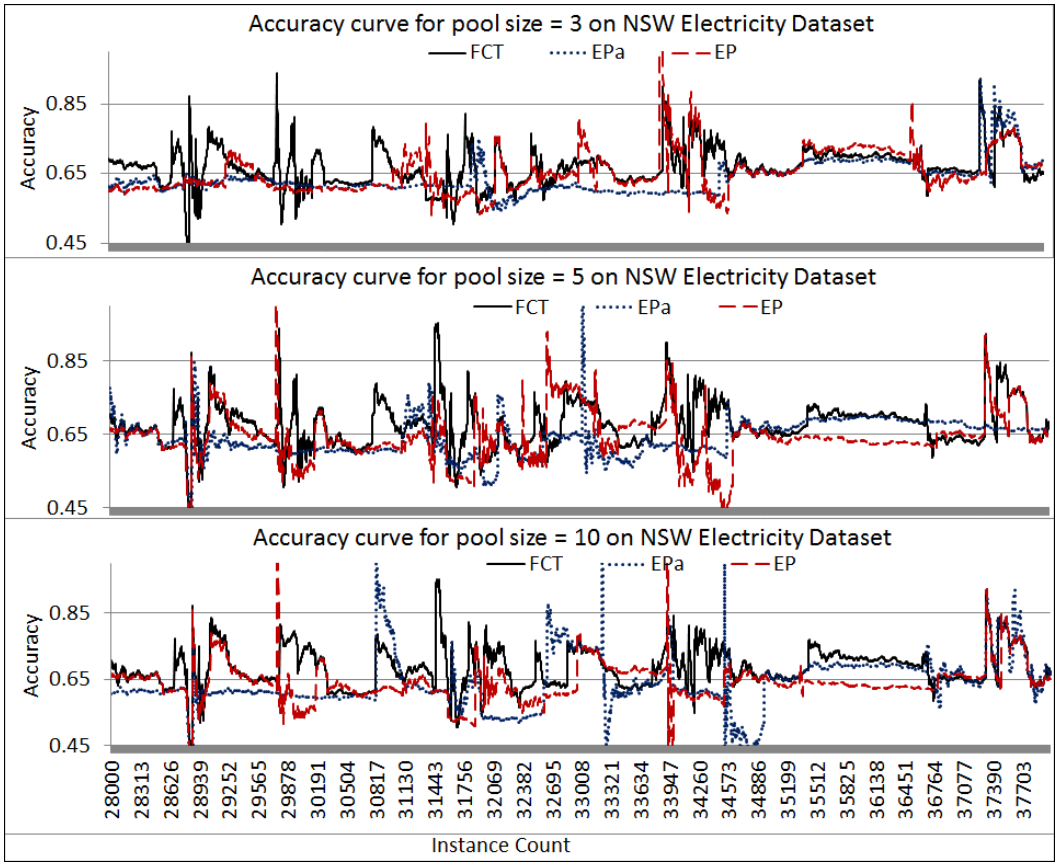


Figure 7.6: Accuracy Profile of FCT, EPa and EP on NSW Electricity dataset for pool sizes 3,5 and 10

Figures 7.4, 7.5 and 7.6 trace the behavior of the two ensemble methods EP_a and EP against FCT for each of the 3 pool sizes 3, 5 and 10. Accuracy values have been recorded on a per instance basis. In addition, ADWIN's estimation referring to the accuracy over the current concept has been monitored and shown in the graphs. Therefore, these values are free from any history effect of past concepts. Accuracy profile on RBF dataset is shown 100K to 150K which is the period of third round of occurrence of all 10 concepts. For the Rotating Hyperplane dataset, the trajectory of accuracy was tracked across a concept recurrence period spanning instances $50K$ to around $100K$. This pe-

riod spanned the second round of occurrence of the 10 concepts injected into the dataset. For the Electricity dataset the data segment spanning instances 28K to around 38K has been tracked.

Although experimentation was done on the other 2 datasets, selected data segments for only 3 algorithms are presented because of interesting contrasts and insights. The performance on the other two datasets were similar to the selected ones or did not show any significant difference among the three algorithms. Please refer Appendix C for the results on the other datasets.

First of all, it could be noted that the other two datasets presented in Figures 7.4, 7.5 and 7.6 have very contrasting volatility profiles compared to Electricity. The Electricity dataset is highly volatile as evident from Fig 7.6. For the same size of data segment, accuracies of all algorithms with all pool sizes on RBF dataset did not fluctuate as much as on Rotating Hyperplane. This is due to the nature of the data streams generated by these generators.

There is a very clear pattern on RBF dataset. EP is superior most of the time over all 10 concepts. EPa, the accuracy based aggregation method fails at a number of instances to reuse or even relearn concepts. EP had very similar accuracy values for any pool size values whereas FCT lost in its competition to EP especially when pool size is three. This is an evidence that aggregated ensembles remembers history well even memory is very limited.

The Rotating Hyperplane dataset did not show high contrast as with the RBF dataset. Accuracy values fluctuated more frequently than that of RBF. In general, it can be observed from the graphs in Fig 7.5 that the differences between EP and FCT decreases for pool size=3. FCT did not suffer from a large drop in accuracy as seen in RBF dataset.

However, with the Electricity dataset the trend is very different. Again the trend seen on RBF dataset is clear on Electricity dataset. FCT slightly outperforms when it has more memory to remember past models on Electricity dataset whereas it loses its classification power in a memory constrained envi-

ronment. This illustrates the essential difference between the two approaches: as expected, an ensemble provides stability in periods of concept change or noise in the stream. For FCT to gain the degree of stability that EP attains it would need to use a larger pool size (10 for this dataset). In a memory-challenged highly volatile environment FCT is less likely to be able to afford the use of additional memory.

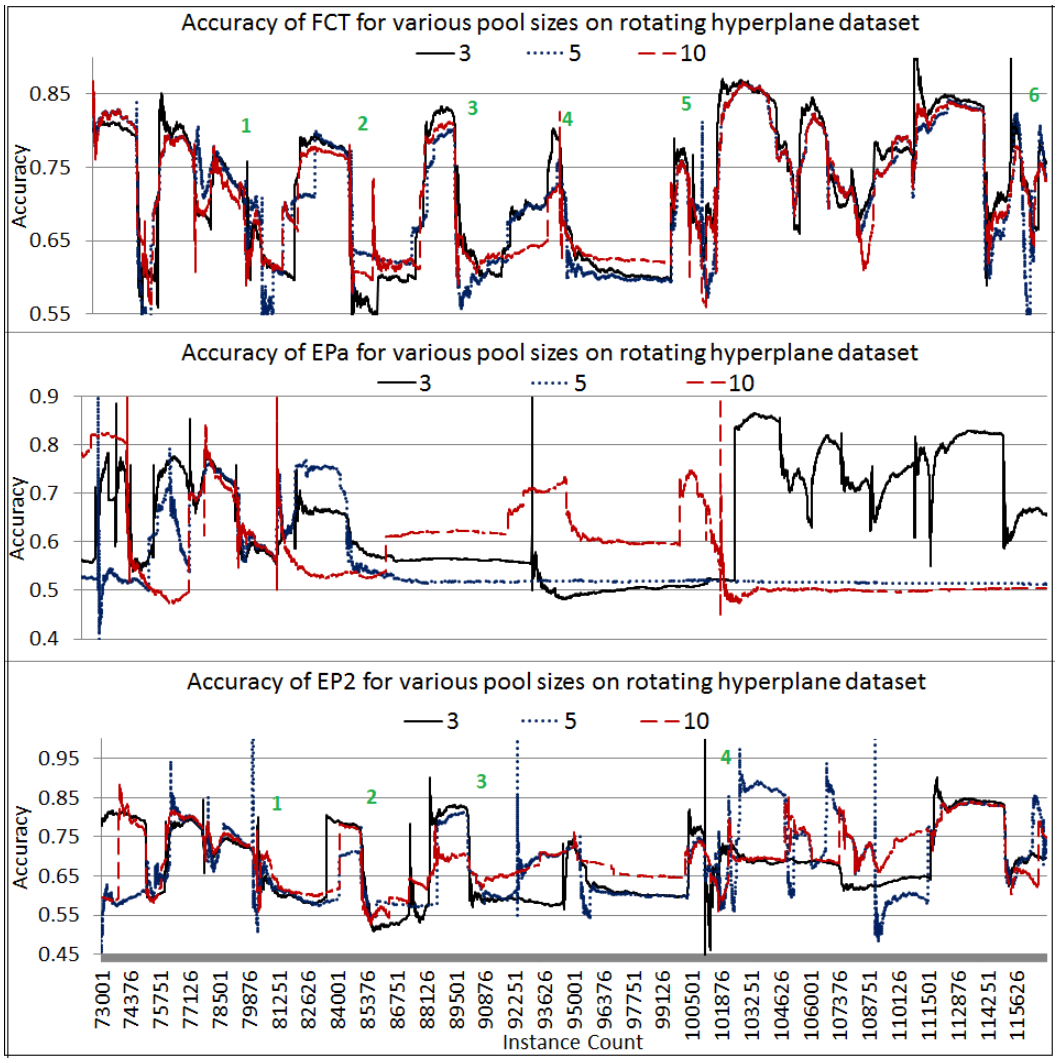


Figure 7.7: Accuracy profile comparison of FCT, EPa and EP **by algorithm** for pool sizes 3,5 and 10 on rotating hyperplane dataset

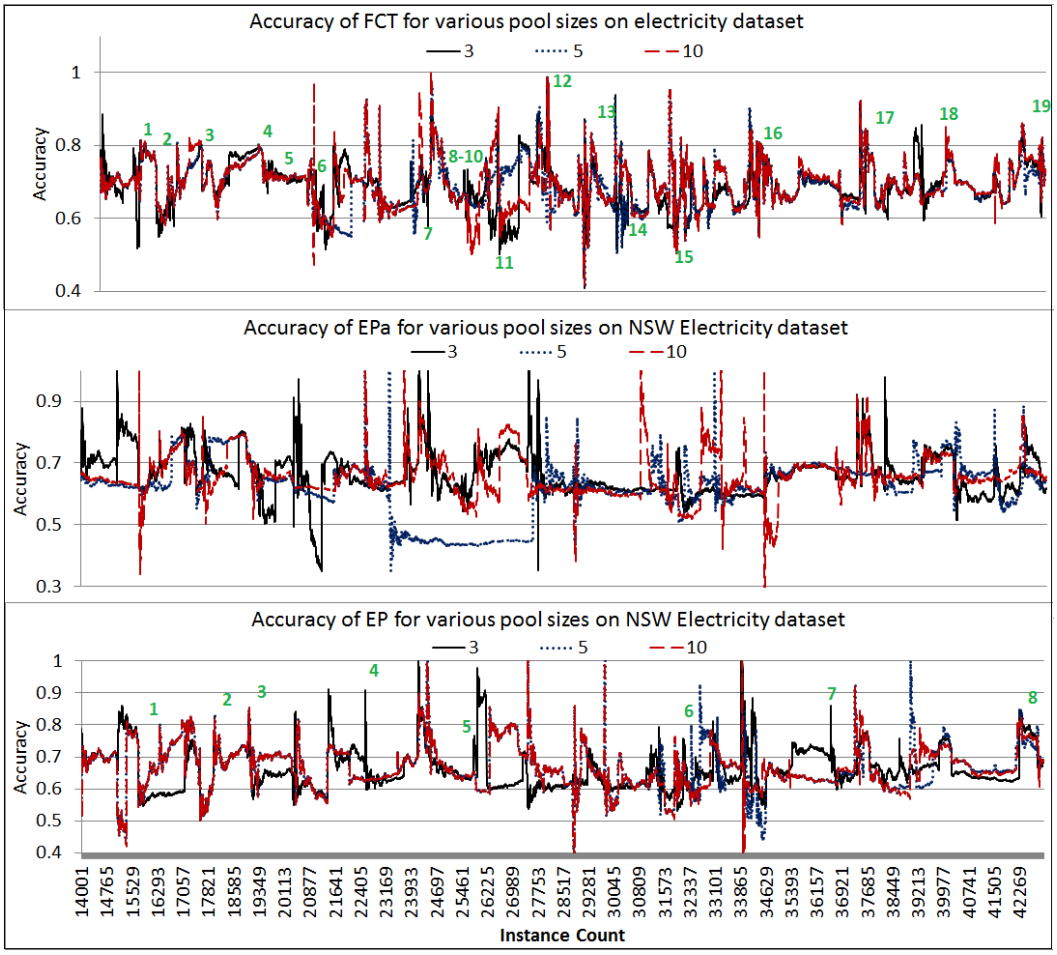


Figure 7.8: Accuracy profiles of FCT, EPa and EP by algorithm for pool sizes 3,5 and 10 on NSW Electricity dataset

Figures 7.7 and 7.8 show the accuracy comparison by algorithm for the pool sizes of 3,5 and 10. Accuracy on rotating hyperplane and NSW Electricity datasets are only shown in the Figures 7.7 and 7.8 due to interesting and contrasting characteristics.

The Electricity dataset is highly volatile as evident from Figure 7.8. This makes the comparison interesting as it provides empirical evidence for the research premise that ensemble spectra can cope better with volatile data streams. To characterize volatility, the regions where a near vertical drop in

accuracy occurs (as shown in annotations on the graph) are identified. For the less volatile Rotating Hyperplane dataset, the number of volatility shifts for both EP and FCT at pool size of 3 is small with no significant difference in number between the two approaches. However, with the Electricity dataset the trend is very different. For FCT, the number at pool size 3 is more than double the corresponding number for EP. This illustrates the essential difference between the two approaches: as expected, an ensemble provides stability in periods of concept change or noise in the stream. For FCT to gain the degree of stability that EP attains, it would need to use a larger pool size (10 for this dataset). In a memory challenged highly volatile environment, FCT is less likely to be able to afford the use of additional memory. EP is more resilient at small pool sizes as any given concept that recurs can be approximated by a linear combination of spectra embedded in the ensemble, just as a waveform of arbitrary shape can be approximated by a large enough sum of sine or cosine functions in signal processing.

Based on empirical evidence from the pool size adjustment experiments, EP is more resilient at small pool sizes as any given concept that recurs can be approximated by a linear combination of spectra embedded in the ensemble, just as a waveform of arbitrary shape can be approximated by a large enough sum of sine or cosine functions in signal processing.

7.3.6 Effects of Noise

Having established the difference in behavior of the approaches at the base 10% noise level, the next interest was in the degree of resilience at the higher noise levels of 20% and 30%. Realistically, expectations cannot be high at the 30% level, as even for a two class problem (Electricity), one would expect the signal to be buried in noise at certain points in the stream. However, if the average accuracy across the stream did not drop in proportion to the added

noise level of 20%, then the performance can be judged as good.

The experiments were with the default parameters on the datasets with randomly injected noise. It should be noted that the initial percentage of noise in Electricity and Spam datasets are unknown. Therefore, adding extra noise may have a severe impact than seen on synthetic dataset.

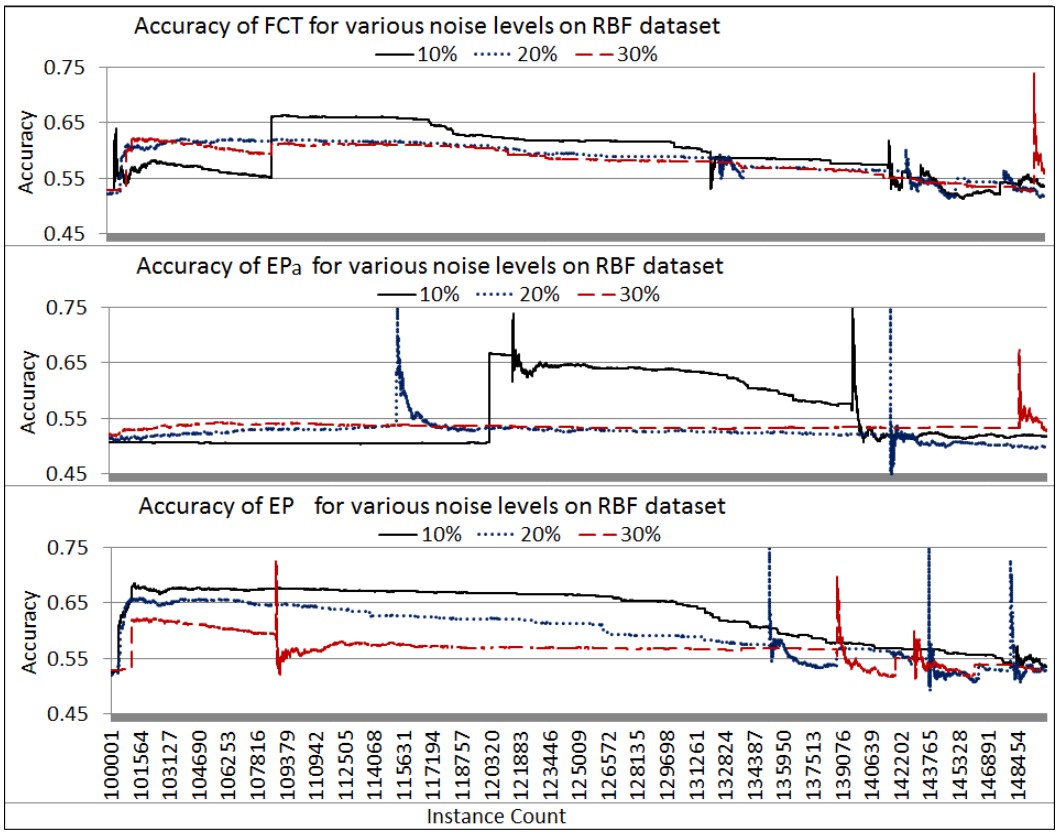


Figure 7.9: Noise resilience of FCT, EPa and EP on noisy RBF datasets

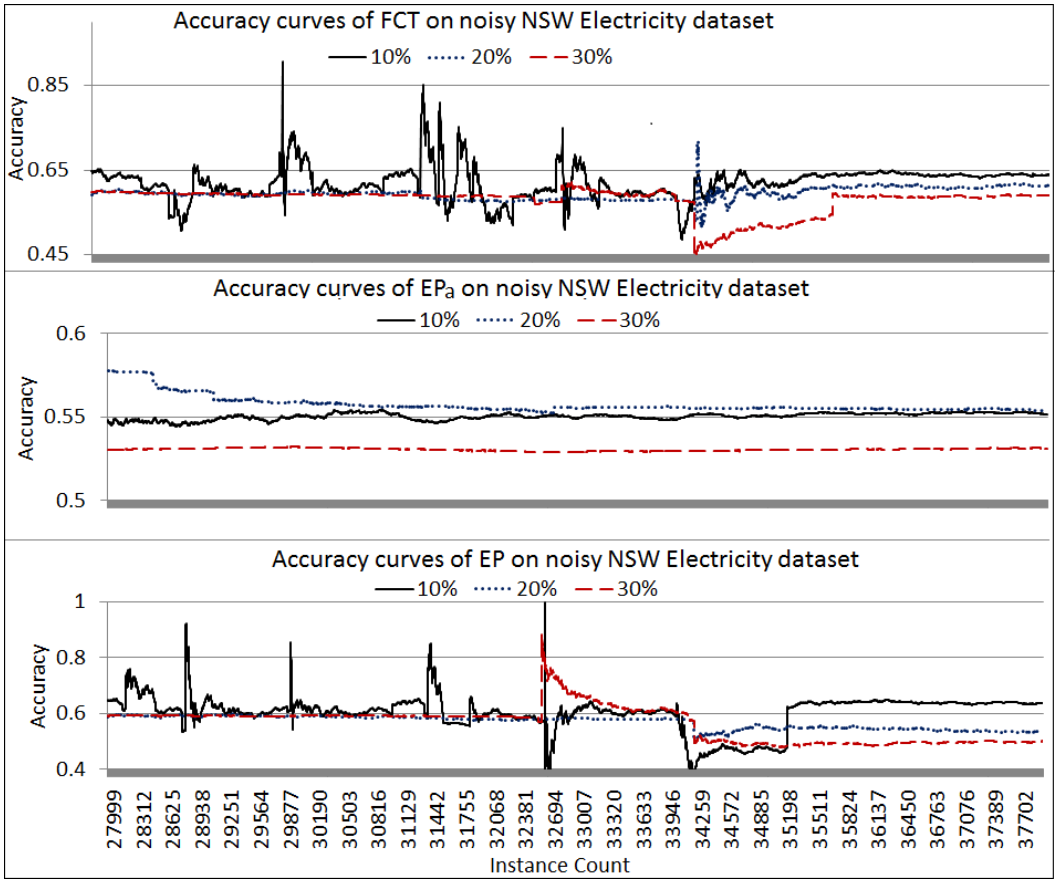


Figure 7.10: Noise resilience of FCT, EPa and EP on noisy NSW Electricity datasets

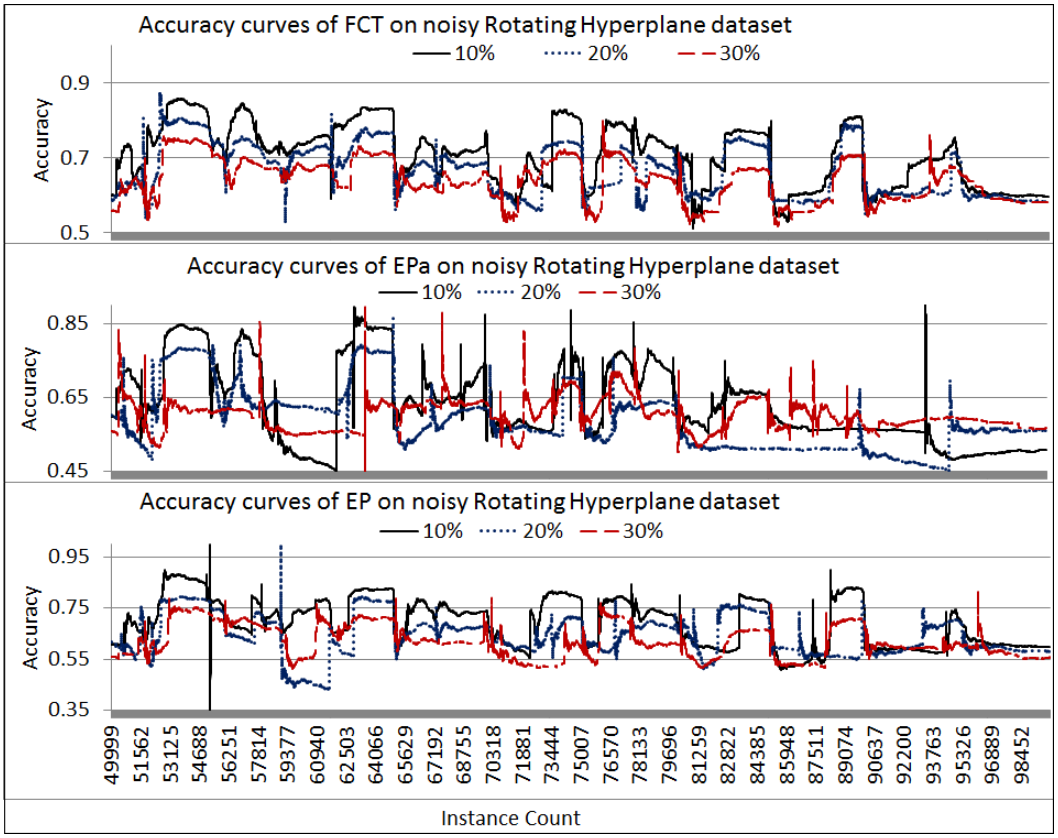


Figure 7.11: Noise resilience of FCT, EPa and EP on noisy Rotaing Hyperplane datasets

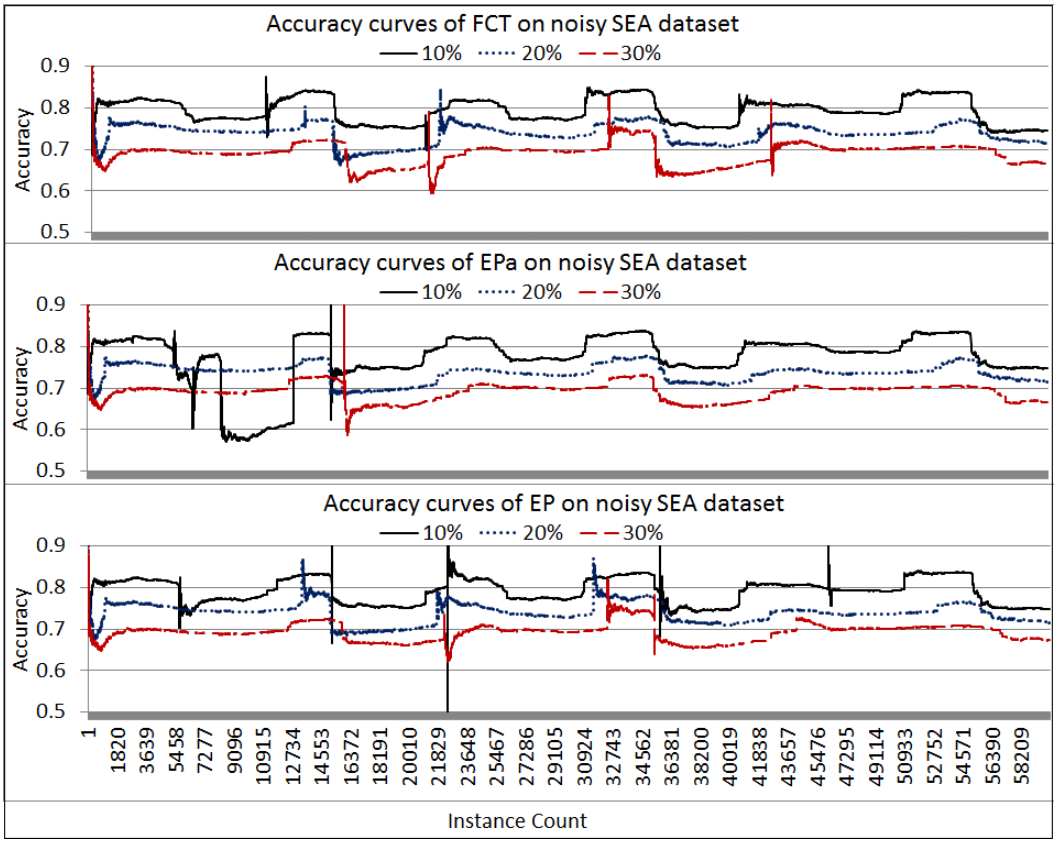


Figure 7.12: Noise resilience of FCT, EPa and EP on noisy SEA datasets

Figures 7.9, 7.10, 7.11 and 7.12 show a general trend as expected. There is a decrease in accuracy as more noise is added to a dataset. Interestingly, aggregation based methods EP and EPa have noticeable regions where there is a higher accuracy on 30% noise over 20% noisy dataset. This can be seen in the figures 7.10, 7.11, 7.9 clearly. FCT easily loses its classification power as the noise percentage increases as seen on all graphs.

7.3.7 Effects of Spectral Energy Thresholding

In this experiment, Spectral Energy has been thresholded at energy levels 40%, 60% and 80% and the accuracy of EP is compared with that of FCT.

EPa has been excluded due to its poor performance in previous experiment.

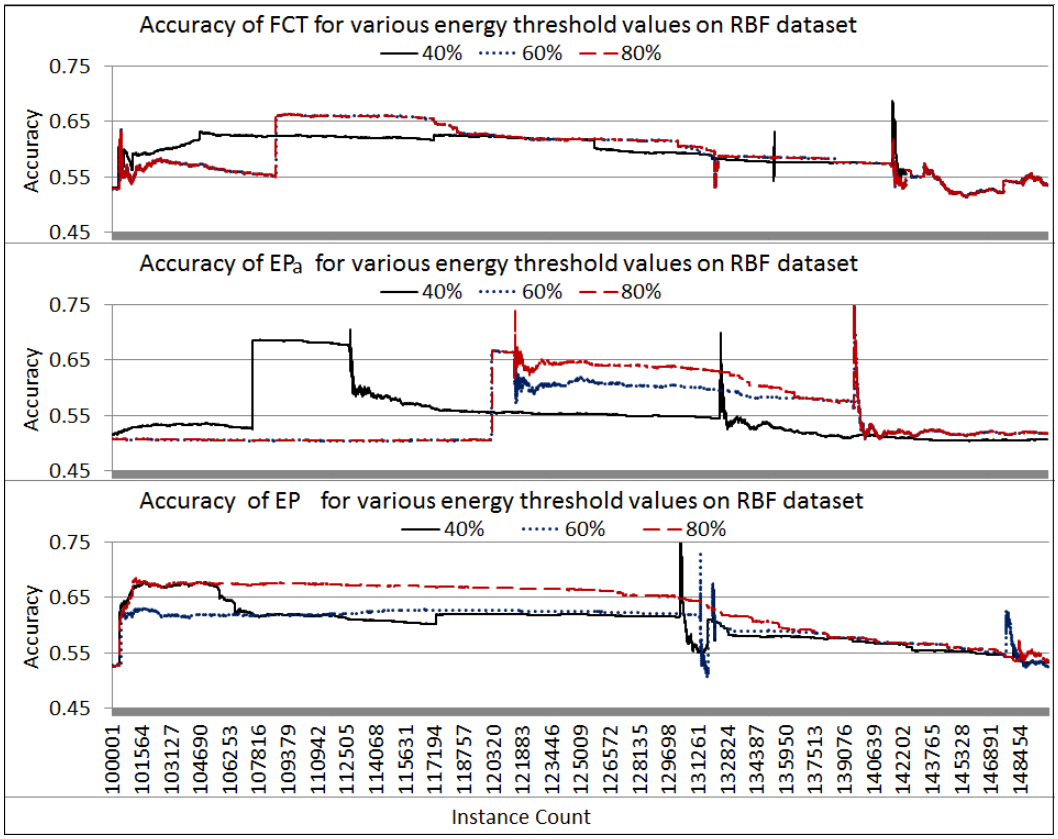


Figure 7.13: Accuracy profile of FCT, EPa and EP for various levels of energy thresholding on RBF dataset

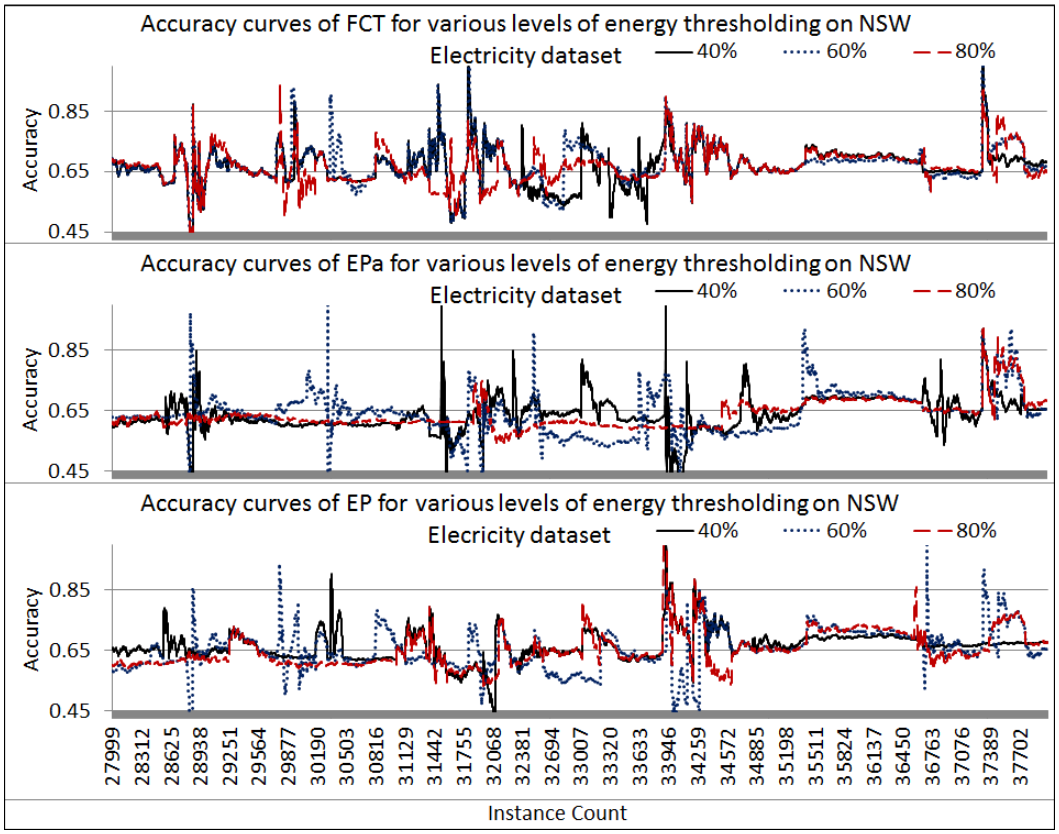


Figure 7.14: Accuracy profile of FCT, EPa and EP for various levels of energy thresholding on NSW Electricity dataset

Figure 7.13 and 7.14 show the difference in accuracy for a number of energy threshold values. In general, the noticeable trend is that the accuracy is higher at 80% energy threshold. At the same time the performance at 40% energy is also competitive in many segments of the data streams.

Out of the three algorithms, EP had many regions with higher or equal accuracy at 40% compared to that of 80%. This trend repeated across all datasets that were experimented with. At low energy levels, individual spectra will be generated from shallow decision trees with different root attributes. This situation is then analogous to better accuracy obtained by bagging a collection of small decision trees (such as decision stumps) in contrast to a

single large tree that may over-fit the current concept.

Also,from a pair-wise comparison between the accuracy profiles at the 40% spectra between the two approaches, it is quite evident that EP outperformed FCT on RBF dataset. There was a 1.3% increase in accuracy on RBF dataset in the region 100K to 150K where concept recurrences were in place for the third time. At the same time, the accuracy of EP was significantly higher in the region 100K to 107K for 40% energy threshold. This supports the hypothesis *the ensemble approach has better resilience than FCT at low energy levels due to different spectra complementing each other*. For Electricity dataset, there was a decrease in accuracy in the region 28K to 38K though there are many short regions (32.3K to 33K, 33.3K to 34.4K) where EP had steady or higher accuracy than that of FCT.

Please refer Appendix C for the graphs on Rotating Hyperplane dataset for various energy levels.

7.3.8 Effects of Structural Similarity Threshold

The impact of structural similarity threshold that controls aggregability of two Fourier spectra. The lower the value of this parameter, the higher the tendency for aggregation. Very low values cause very different spectra to be merged together. This could lead to the pollution of the concept representation kept by spectra in the aggregated form. High values discourage aggregation, thus producing model representations similar to that of FCT presented in Chapter 6. Therefore, it is interesting to study what values are best for which datasets. In general, a moderate value between 30% and 80% is recommended. The experiments conducted to analyse the impact of this parameter consist of three threshold values 30%, 50% and 70%.

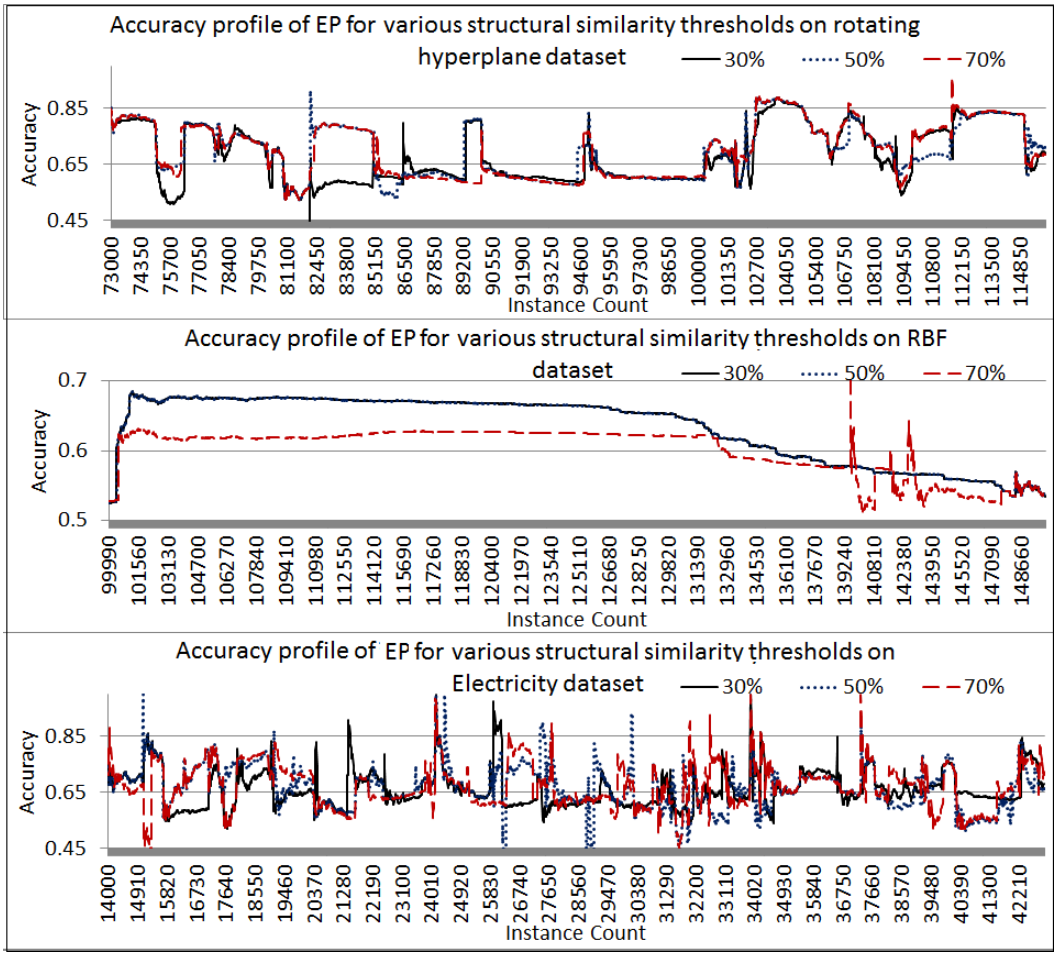


Figure 7.15: Accuracy profiles of EP for the structural similarity threshold values 30%, 50% and 70%

For the Rotating Hyperplane and NSW Electricity datasets, slightly better accuracy is observed for threshold values of 50% and 70%. On average, taken across the full length of Rotating Hyperplane dataset, there is around a 1% and 0.7% increase in accuracy for 50% and 70% threshold values respectively compared to that of the threshold value of 30%. Electricity dataset has 1.1% and 1.3% increase in accuracy for the structural similarity values 50% and 70% in comparison to 30%. For the RBF dataset, no difference has been observed between the values 30% and 50% for the structural similarity threshold. There

is a marginal decrease of 0.6% in accuracy found for 70% similarity threshold compared to the 30% one.

This experiment does not show a very significant impact of structural similarity threshold on average though there are short intervals of accuracy gain or decrease depending on the dataset. RBF dataset is an evidence that supports the hypothesis Accuracy gain for larger values for a given dataset indicates that the concepts present in the dataset are polluted by inappropriate aggregation.

7.3.9 Memory

The experimentation on accuracy has revealed, especially in the case of EP, concept re-use has the potential in improving accuracy. The research reported in Chapter 6 showed that Fourier spectra consume a small fraction of the memory used by trees kept in active memory; thus the focus in this study is to compare the performance of EP vis-a-vis FCT in terms of their spectra memory usage.

Average memory consumption has been measured in bytes per instance basis across the stream for each method and for each dataset. Tables 7.1, 7.2 and 7.3 show the raw memory values consumed by each algorithm for different pool sizes. Lower pool sizes artificially force the algorithms to consume limited memory whereas high values would show the true contrast in consumption. Table 7.1 and 7.3 clearly show that EP uses memory more efficiently than FCT as EP starts off at a lower base usage at pool size of 3 and the percentage difference is always in favor of EP (with on exception for the RBF dataset; the SEA dataset produced a compact spectrum which never grew for both approaches). In a very high dimensional dataset like Spam that has 499 attributes, EP consumes 36% less memory compared to FCT. For a larger pool size value of 20, the clear winner is EP which takes 30% less average

memory over all datasets compared to FCT. EP's memory consumption is 57% less than that of FCT on the high dimensional Spam dataset. Similar trends are found for the pool sizes 5 and 10. Furthermore the percentage differences widens at the larger pool sizes.

Percentage increases for a given algorithm are always lower at pool sizes of 5 and 10 whichever dataset is considered (the exception is SEA which did not increase for both algorithms). Thus it is evident that EP's memory consumption scales better than that of FCT. This is to be expected as, for a given pool size, EP will, in general, be more economical as it combines spectra with many coefficients in common, whereas for FCT every different spectrum having the same coefficient will require its own storage.

As noted in the experiments on accuracy, EP often acquires the same or higher accuracy at lower pool sizes and this further increases the disparity in memory consumption of EP vis-a-vis FCT. This proves the hypothesis *the aggregation of similar performing spectra produces better spectra in terms of performance and memory.*

Dataset	3		5		10		20	
	Forest	Pool	Forest	Pool	Forest	Pool	Forest	Pool
rbf	82.6	13.9	81.7	17.2	81.7	17.2	81.7	17.2
rh	113.6	14.5	120.6	18.8	109.9	29.2	111.5	47.3
sea	18.0	0.4	18.0	0.4	18.0	0.4	18.0	0.4
electricity	49.6	19.6	46.3	24.1	46.1	31.4	46.1	41.2
spam	35103.2	22.6	35103.2	29.0	35218.3	33.4	35218.3	33.4

Table 7.1: Raw memory values consumed (in KBs) by FCT on all five datasets for pool sizes 3,5,10 and 20. As FCT has a forest of tree and a Fourier pool, memory consumption is divided into two columns for each pool size experimented.

Dataset	3		5		10		20	
	Forest	Pool	Forest	Pool	Forest	Pool	Forest	Pool
rbf	106.3	12.9	107.4	16.2	106.6	18.9	106.6	18.9
rh	145.7	13.7	144.3	18.2	152.4	28.2	152.5	40.2
sea	18.0	0.4	17.6	0.4	18.0	0.4	18.0	0.4
electricity	58.1	12.6	63.9	17.2	60.9	25.3	59.6	29.3
spam	37075.0	20.3	37736.7	26.5	37316.2	32.4	37316.2	32.4

Table 7.2: Raw memory values consumed (in KBs) by EPa on all five datasets for pool sizes 3,5,10 and 20. As EPa has a forest of tree and a Fourier pool, memory consumption is divided into two columns for each pool size experimented.

Dataset	3		5		10		20	
	Forest	Pool	Forest	Pool	Forest	Pool	Forest	Pool
rbf	94.8	13.9	94.7	14.4	94.7	14.4	94.7	14.4
rh	121.0	13.3	113.8	17.7	116.5	25.8	115.8	35.0
sea	18.1	0.4	18.1	0.4	18.1	0.4	18.1	0.4
electricity	51.6	12.8	50.6	16.3	51.7	19.6	51.7	19.6
spam	35163.9	14.3	35163.9	14.3	35163.9	14.3	35163.9	14.3

Table 7.3: Raw memory values consumed (in KBs) by EP on all five datasets for pool sizes 3,5,10 and 20. As EP has a forest of tree and a Fourier pool, memory consumption is divided into two columns for each pool size experimented.

7.3.10 Processing Speed

Scalability in terms of processing speed has also been measured. This is an important metric to cope with the speed of instances in high speed data stream. The average processing speed was measured by sampling the stream on a per instance basis and by counting the number of instances processed per second and then averaging the speed. The processing speed was tracked for all pool sizes for each approach.

Dataset	3	5	10	20
rbf	6669.3	6991.3	6739.4	6998.9
rh	5828.2	5778.1	5627.7	5292.6
sea	59939.2	59051.8	59871.9	58149.8
electricity	10036.8	9803.1	9156.7	8680.8
spam	1.5	1.4	1.4	1.4

Table 7.4: Processing speed of FCT for the pool sizes 3, 5, 10 and 20 is shown in this table. It is measured as number of instances processed per second

Dataset	3	5	10	20
rbf	6298.7	6737.0	6732.0	6468.6
rh	5458.9	5212.1	4954.2	4742.1
sea	58816.2	59034.0	59606.0	57547.9
electricity	9889.3	9104.3	8482.7	8538.4
spam	1.4	1.4	1.3	1.4

Table 7.5: Processing speed of EPa for the pool sizes 3, 5, 10 and 20 is shown in this table. It is measured as number of instances processed per second

Dataset	3	5	10	20
rbf	6433.6	6726.4	6615.8	6778.1
rh	5990.2	5766.7	5590.5	5466.4
sea	59561.6	58446.4	60566.2	60686.9
electricity	10197.0	9796.2	9691.6	9493.6
spam	1.5	1.4	1.5	1.3

Table 7.6: Processing speed of EP for the pool sizes 3, 5, 10 and 20 is shown in this table. It is measured as number of instances processed per second

Based on the Tables 7.4, 7.5 and 7.6, it can be observed that processing speed does not vary by a large amount for various pool sizes. When there are many classifiers in memory, instance routing and classification slows down the speed of processing.

EPa is found to be having the highest processing speed in general over all datasets. Compared to FCT, EP takes lesser time to process instances based on the values in the above tables. At each concept change point FCT will require more computational overhead than EP as it needs to poll a larger number of objects in the repository to identify whether the best performing active tree is better or not than existing classifiers (spectra) in the repository. It has been noted from execution logs that FCT had more objects in its repository (for a given pool size FCT tended to fill its pool, whereas EP did not have to make use of all slots available). In addition, as each instance is routed to all classifiers, FCT needs higher processing time than EP due to the higher count of classifiers.

Therefore, it is evident that aggregated ensembles are faster than single Fourier trees.

In conclusion, aggregated ensemble based methods are memory efficient while achieving competitive accuracy to a single classifier. The decrease in

memory and processing time are significant especially in high dimensional data streams.

7.4 Empirical Study on EP with SeqDrift2 Change Detector

This section analyzes the impact of a change detector on EP. Two versions of EP have been tested on all datasets used in the previous section. The first version is the default FCT with ADWIN change detector namely FCT+ADWIN. The second is FCT with seqdrift2, namely FCT+seqdrift2. The seqdrift2 detector has lower false positive rate than ADWIN. Therefore, it could be expected that EP+seqdrift2 has lesser fluctuations over stable data segments. The results for accuracy evaluation includes all datasets.

The experiments did not involve any parameter adjustments. The default parameter values were chosen as the objective of this study is simply to assess the effect of replacing ADWIN with SeqDrift2.

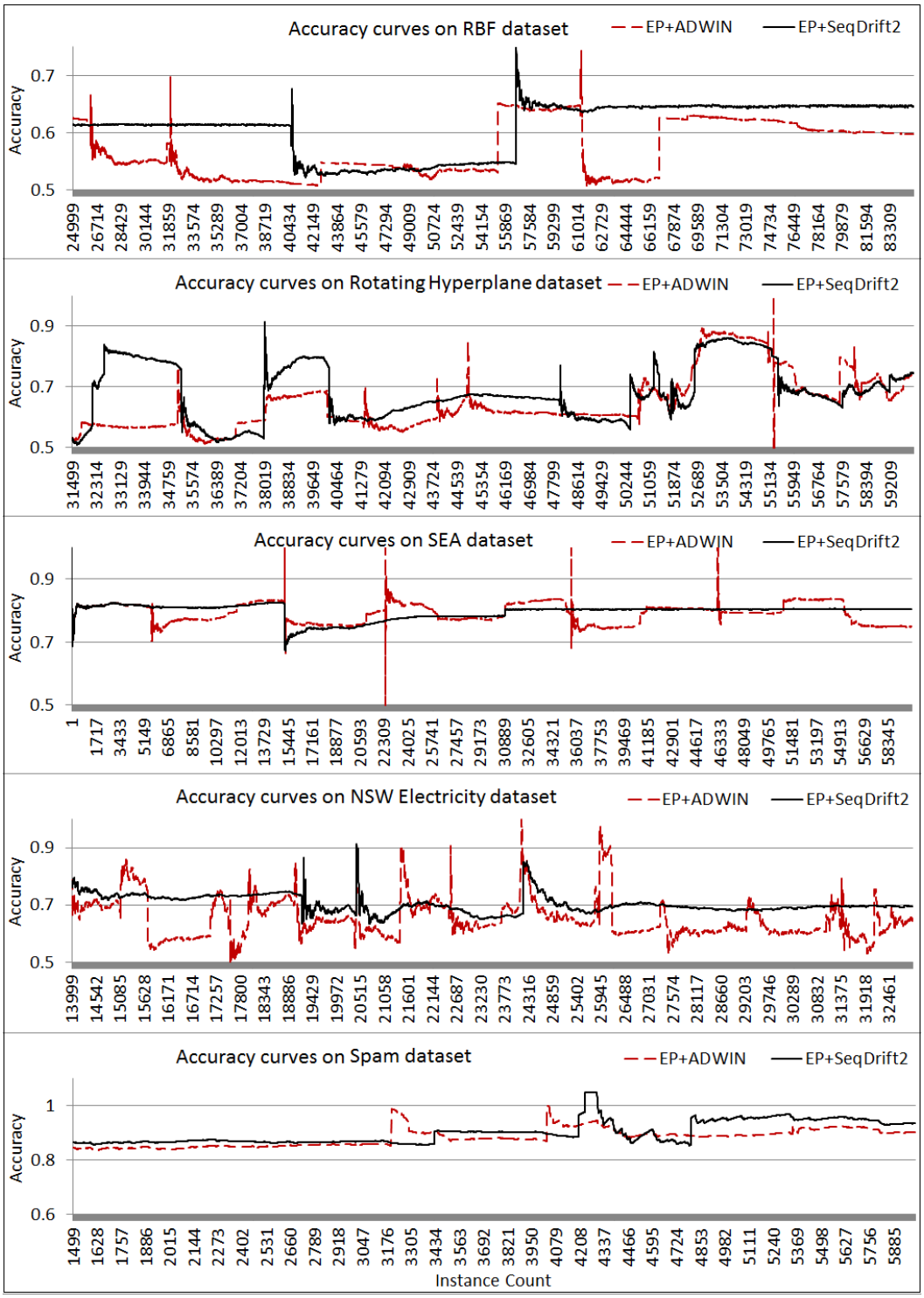


Figure 7.16: Accuracy comparison between EP+ADWIN and EP+seqdrift2 on all datasets

Similar to FCT+seqdrift2 in Chapter 6, EP+seqdrift2 also maintains its capability of being more stable as opposed to its counterpart EP+ADWIN. This is empirically shown in Figure 7.16 on all datasets. In addition to being more stable, it has higher accuracy over EP+ADWIN on RBF and Electricity datasets.

The same explanation holds for EP+seqdrift2 as FCT+seqdrift2 in Chapter 6 for its stability and higher accuracy. This result is due to the lower false positive rate of seqdrift2 change detector when compared to ADWIN. Low false positive rate helps keep the most appropriate classifier/ensemble as the winner until actual concept change occurs. False alarms trigger unnecessary archiving of current winner classifier into pool. This introduces fluctuations in accuracy.

7.4.1 Processing Speed and Memory Comparison

Table 7.7 contains the average memory consumption and processing speed values taken in per instance basis.

The main observation with respect to memory is that EP+seqdrift2 has compact pool compared to that of EP+ADWIN. Less false positive rate of seqdrift2 means less insertions of new Fourier Trees into the pool, thus less aggregation. This has resulted in smaller Fourier trees in EP+seqdrift2 model.

EP+seqdrift2 has advantage on high speed data streams with its higher processing speed compared to EP+ADWIN. Inherently, seqdrift2 is faster than ADWIN. Moreover, the lower false positive rate also contributed to a lower processing time.

Datasets	Memory EP+ADWIN		Memory EP+seqdrift2		Processing Speed	Processing Speed
	Tree Forest	Fourier Pool	Tree Forest	Pool	EP+ADWIN	EP+seqdrift2
rbf	94.8	13.9	95.9	12.6	6433.6	6529.2
rh	121.0	13.3	120.6	13.0	5990.2	6201.2
sea	18.1	0.4	19.2	0.4	59561.6	60936.7
electricity	51.6	12.8	51.6	12.6	10197.0	10263.4
spam	35163.9	14.3	34954.6	14.2	1.5	1.5

Table 7.7: Average Memory Consumption (in KBs) and Processing Speed (Instances per second) Comparison

7.5 Summary

In this Chapter, aggregation, a key property of Discrete Fourier Transform of decision trees has been exploited in two versions of algorithms called EP and EPa. Aggregation of classifiers, as expected introduced stability in memory constrained environments. This is due to the fact that with individual spectra, memory must be freed to plant new classifiers into the pool. This results in loss of some useful classifiers learned on past instances. Aggregation has the power to remember large number of past classifiers while at the same time removing redundancies.

Two different types of aggregation of Fourier spectra were tested on five commonly used datasets with respect to accuracy, memory, noise tolerance and processing time. EPa performs aggregation of an existing Fourier ensemble with a Fourier Spectrum if both has similar performance. The other algorithm EP combines the above two spectra if those are structurally similar. A heuristic based on a memoryless and computationally inexpensive method to check structural similarity has also been introduced.

Empirical results show that EP is more stable in its performance than FCT as expected while consuming less memory and processing instances faster. The

parameters, pool size and energy threshold have also been adjusted to study the influence on corresponding algorithms.

Lastly, the change detector of EP has been replaced by SeqDrift2 proposed in Chapter 5 and compared with the default change detector ADWIN. Interestingly, EP an SeqDrift2 combination is found to be generally superior than the default EP+ADWIN combination in accuracy, memory and processing speed aspects mainly due to lesser false positive change detections by SeqDrift2.

Chapter 8

Case Study

8.1 Introduction

This chapter presents a specific case study with experimental results to further validate the performance of the FCT and EP algorithms through an in-depth case study. Though concept recurrence is injected on the synthetic datasets and was thus able to trace recurrence exploitation by the algorithms, the same could not be said of the two real world datasets used, namely Electricity and Spam. Hence it would be of interest to validate performance against a real world dataset whose patterns of recurrence are known in advance.

The case study that is used revolves around the motivating example scenario on flight of an aircraft that was described in Chapter 7. In a flight, similar environmental and flight conditions cause sensors to produce similar data stream segments. For example, low pressure, rainy, cloudy etc. weather causes an aircraft to take certain well defined flight actions. Similarly take off, climb, cruise and landing also requires similar actions to be performed each time that they are actioned.

An aircraft could face low pressure environments a number of times during a flight time. If a classifier(s) are built on a certain scenario like low pressure and are kept in memory, those can be reused when the same environmental conditions recur. This avoids relearning and the delay in responding to environmental change. Minimum delay is crucial to prevent unfavorable events. Aircraft like small drones may have constraints on processing power

and memory. Moreover an automatic flight system must recognize changes in data streams autonomously in order to adapt its model. Therefore a model that captures recurrences in data streams and adapts itself with minimum delay is an absolute necessity in this safety-critical scenario.

8.2 Description of the dataset used

As a case study, a dataset obtained from a flight simulator has been selected and analyzed by applying the algorithms proposed in this research to study the effectiveness. This dataset is publicly available from this web link [1]. It has been generated with NASA's FLTz flight simulator [2]. It is available as a set of 20 separate files. Each file contains the data about a single flight with four scenarios: take off, climb, cruise and landing. Data is recorded every second and a data instance is produced. Based on the requirements of an auto pilot system, "Velocity" has been chosen as the target class based on the features and sensor readings of an aircraft. "Velocity" is the most appropriate feature as it needs to be adjusted in order to maintain aircraft stability during various manoeuvres such as take off and landing.

The dataset was formed by concatenating all 20 datasets after removing irrelevant features such as time stamp, latitude and longitude positional coordinates. The Velocity attribute was discretized into binary outcomes "UP" or "DOWN" depending on the directional change of the moving average in a window of size 10 data instances.

This dataset fits an appropriate testing environment for the models proposed in the following ways:

- It has four concepts generated during take off, climb, cruise and landing. Therefore this creates a well-defined concept change environment and it

[1] https://c3.nasa.gov/dashlink/static/media/dataset/FLTz_2.zip
[2] https://c3.nasa.gov/dashlink/projects/42/

can be used to test the effectiveness of the change detectors presented in this research.

- As 20 flights are concatenated together, each of the above concepts repeats 20 times. This provides an opportunity to assess whether recurrences are captured and models are appropriately switched to suit the current concept.

- As each flight is different from another, the four concepts would not recur in similar but not exact form. This is one of the objectives of this research. One of the hypotheses of applying the Discrete Fourier Transform is that *it results in capturing the essence of a concept without being too specific to a particular concept* so that it generalizes well to similar, but not exactly recurring concepts.

- The auto pilot scenario is a time and safety critical application.Therefore, it would be possible to evaluate how quickly the changes have been recognized and models chosen without relearning.

- This dataset closely reflect a real life situation rather than being synthetic

8.2.1 The models used for empirical study

Based on the previous analysis on all five datasets (refer Chapter 7), EP_a has been excluded due to its poor performance. Therefore, FCT and EP are selected to study the impact of a single classifier vs aggregated ensemble approach. Moreover, the influence of a change detector has also been evaluated by replacing the default ADWIN by SeqDrift2 change detector. The four models were named as FCT+ADWIN: the same algorithm referred to as FCT in Chapter 6, FCT+SeqDrift2: SeqDrift2 replaces ADWIN as change

detector, EP+ADWIN: the same algorithm referred to as EP in Chapter 7 and EP+SeqDrift2: SeqDrift2 replaces ADWIN as change detector.

8.3 Empirical Study

Empirical study has focused on the following performance metrics:

- Accuracy comparison
- Memory consumption and processing speed
- Robustness to Concept Change (Refer 8.3.4)

The last metric is defined in this research as novel measure that quantifies special performance considerations in recurrence capture. This metric is defined and explained in section 8.3.4.

The above metrics were assessed in a memory constrained environment by adjusting maximum pool size parameter for both FCT and EP in order to assess their relative effectiveness against each other.

8.3.1 Accuracy Comparison

This section presents an accuracy comparison of the four algorithms experimented in this research. The sensitive parameter *pool size* (equivalent to maximum memory) has been adjusted to investigate its effects on accuracy.

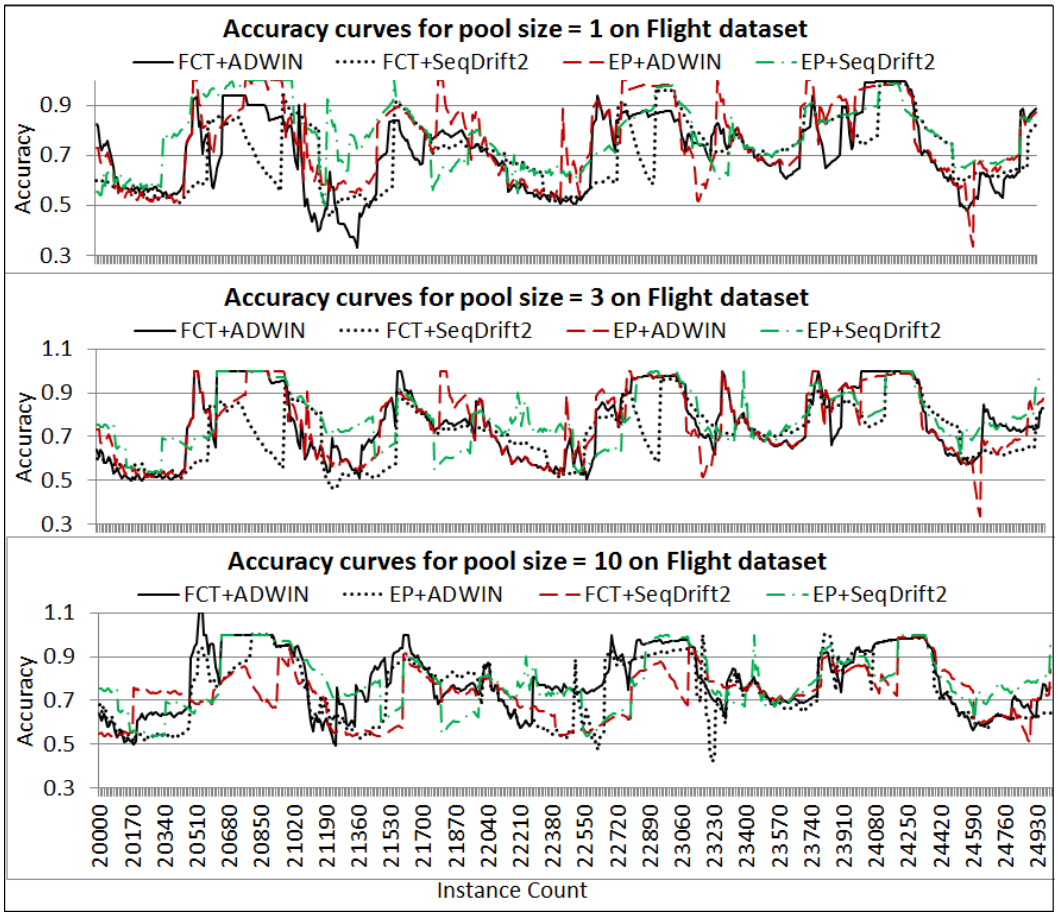


Figure 8.1: Accuracy profiles of all four algorithms on flight dataset for various pool size values

Figure 8.1 shows interesting patterns embedded in the accuracy profiles. The data segment 20K to 25K is presented in Figure 8.1 to avoid overcrowding of curves. Moreover, as this is the last segment of the dataset and has data on two separate flights, this provides an opportunity to contrast EP and FCT on their recurrence capture capability. EP has the opportunity to improve its ensemble by aggregating small-scale differences when similar concepts reappear. There are a number of clear observations that can be made from Figure 8.1:

- EP quickly adapted to changing concepts, compared to FCT when memory was constrained. This is visible on the first graph in Figure 8.1. After an accuracy drop, EP+SeqDrift2 was found to be the best algorithm as it regained its classification power quickly for pool size = 1 when memory was at an absolute premium. With this pool size setting, FCT must flush an existing Fourier spectrum in order to insert a new one into the pool. Therefore, FCT needs to relearn concepts often. Due to aggregation, EP has an advantage as it can keep a single ensemble containing spectra learned on many different concepts. This leads to quick adaptation.

- As pool size increases FCT's ability to adapt to changing concepts visibly improved, as expected. A couple of contrasting trends are apparent. First of all, FCT kept on generating more than 4 concept representations as Fourier Spectra in the pool even though only 4 basic concepts were inherent in the dataset. This was reflected in its higher accuracy at certain data segments with pool size of 10 when compared to the pool size of 3. This shows that FCT's generalization capability is weaker than that of EP. On the other hand there was virtually no change in performance between EP at levels 3 and 10, given that just 4 base concepts exist.

- EP in general had better stability when compared to FCT. Stability in performance was a direct result of aggregation. Aggregation helped avoid the removal of useful classifiers when memory is constrained.

Figure 8.1 also reveals two interesting trends with respect to the performance of the change detectors. With the ADWIN detector sharp drops in accuracy occurred with both FCT and EP at different points in the stream. These drops caused accuracy to dip below 50%, which for a two class problem was far from being desirable. In contrast, SeqDrift2 was stable at these

points in the stream. In Chapter 5, it was shown that ADWIN's false positive rate was higher than that of SeqDrift2 and false positives signaled by ADWIN can cause existing spectra that are performing well to be switched to a new spectrum which under performs. This problem was particularly acute at low memory setting of pool size 1. Both algorithms under ADWIN managed to avoid these sharp dips in accuracy at the higher pool sizes as other spectra present in the pool were able to compensate for the loss of the incorrectly removed spectrum. On the other hand, it was noted that both algorithms operating under SeqDrift2 did not suffer from this problem at any pool size setting.

At the higher memory settings, there was little to distinguish between the two detectors, both of them supported the classifiers equally well; each one outperforming the other at different points in the stream but overall returning around the same average accuracy.

To quantify the contribution of recurring concept capture over other adaptive data mining algorithms, a number of classifiers (the first 6 algorithms in Table 8.1 in MOA software) were chosen for a comparative study. These algorithms do not have a mechanism to capture recurrences but are capable of adapting their models to suit current concepts.

Algorithm	Average Accuracy
Adaptive Naive Bayes (NB)	66.61
Hoeffding Adaptive Tree	72.99
Hoeffding Option Tree	73.32
Hoeffding tree	72.96
Adaptive Decision Stump	60.08
Adaptive Hoeffding Option Tree with NB at leaves	74.29
EP+SeqDrift2	78.0
EP+ADWIN	75.6

Table 8.1: This table shows average accuracy of various classifiers that are designed for data stream mining. Each classifier is tested first and trained with each instance (Prequential Evaluation) with the sample frequency= *1 instance* thus, recording accuracy for each instance. Single classifier models were chosen for a fair comparison with both EP models because EP models were restricted to have only one Fourier tree in the pool for this comparison

This clearly provides empirical evidence that capturing recurring concepts using ensembles even under severe memory constraints has accuracy advantages. Of course, average accuracy figures mask much higher gains in accuracy at concept recurrence points where both FCT and EP re-used previously stored classifier models. The Figure 8.2 is an evidence to support the above claim. It is very clear to see from the graphs that EP+SeqDrift2 outperformed frequently over the other three adaptive classifiers. The accuracy profiles of the other three classifiers Hoeffding Tree, Adaptive Decision Stump, Adaptive Hoeffding Option Tree with NB at Leaves are not shown as they also had the similar trend as the algorithms presented in Figure 8.2.

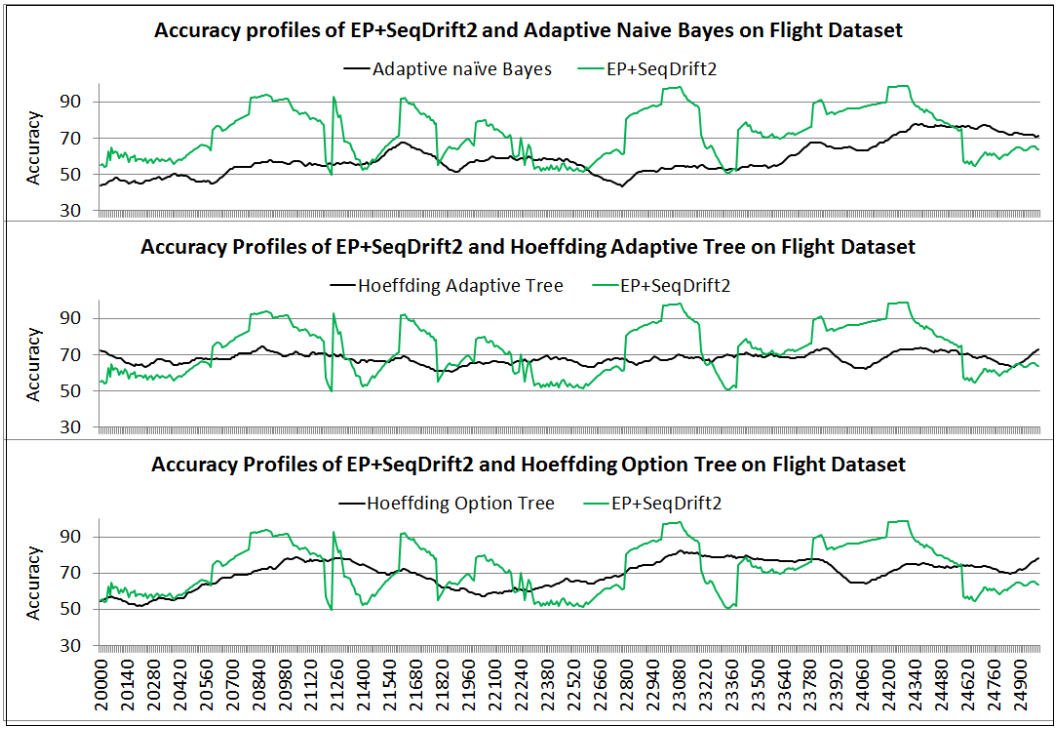


Figure 8.2: Accuracy comparison between each of the adaptive algorithms and EP+SeqDrift2 for pool size=1 on flight dataset

As this dataset includes data on 20 separate flights, each concept (take off, cruise etc.) in a single flight is repeated at least 20 times. Each recurrence is a similar manifestation of a previous concept. EP has an advantage to improve a concept representation by incrementally adding missing details found in similar concepts. Therefore, a net accuracy increase can be expected as many similar concepts are seen and ensembles are updated accordingly. This property has been analyzed by comparing the average accuracy over the second flight segment which has the first recurrence of previously seen concepts in the first flight. The FCT algorithm has no mechanism to improve an existing concept representation captured by a Fourier spectra unless it is replaced by a new spectrum altogether. EP+SeqDrift2 and EP+ADWIN had a net increase of around 5% increase in average accuracy whereas FCT does not

have an accuracy increase at all for pool size of 1. This difference for both EP algorithms over FCT was 4% with pool size=3 . The difference remained at the same value for pool size = 10. Thus, it can be concluded that aggregation of structurally similar classifiers helps to incorporate previously unseen subtle but important differences among similar concepts.

Having established accuracy and stability advantages, especially in memory constrained environments, it would be interesting to compare the actual memory consumed by each algorithm for different levels of pool size settings.

The next section presents the results in terms of memory consumption on Flight dataset.

8.3.2 Memory consumption comparison

This section compares the four algorithms on the basis of average memory consumption taken across the entire dataset with the stream sampled at each instance. Table 8.2 contains the actual memory consumed by the decision tree forest and Fourier pool for each algorithm for the pool sizes 1, 3 and 10. The memory values are in KBs. The key observation is that EP+SeqDrift2 consumes the least memory for its Fourier pool for all pool size parameter values. This is a direct result of aggregation that removes redundancy in Fourier spectra when structurally similar spectra are aggregated together. At the same time, EP+ADWIN is the runner up. Though EP is good at producing highly compressed models, higher memory consumption was due to ADWIN. Though ADWIN does not by itself consume a large amount of memory for keeping its buffer and statistics, its false alarm rate caused a higher number of Fourier spectra to be aggregated to ensembles in the Fourier pool. This results in larger ensembles than that of EP+SeqDrift2 combination as SeqDrift2 has a lower false alarm rate than ADWIN. Interestingly, both FCT models consumes almost equal amount of memory for their pools. As maximum pool

size parameter increases, there is an increase in average memory consumption by their respective pools.

Algorithm	1		3		10	
	Forest	Pool	Forest	Pool	Forest	Pool
FCT+ADWIN	582.2	27.8	572.6	31.5	570.7	36.9
EP+ADWIN	548.8	14.6	548.8	14.6	547.5	24.5
FCT+SeqDrift2	796.9	27.1	796.8	31.0	799.1	32.1
EP+SeqDrift2	771.2	14.1	792.9	13.8	792.9	18.1

Table 8.2: Raw memory values consumed by all four algorithms on Flight dataset for the pool sizes 1,3 and 10. As all algorithms have a forest of trees and a Fourier pool, memory consumption is divided into two columns for each pool size experimented.

In terms of decision tree forest, EP+SeqDrift2 and FCT+SeqDrift2 have consumed a larger amount of memory than with corresponding combinations with ADWIN. Again, the lower false positive rate of SeqDrift2 over ADWIN would be the explanation. A false alarm causes a winner decision tree to be converted into Fourier spectrum and to regrow from scratch. Less false alarms enable a winner decision tree to grow bigger by learning a concept in greater detail. Therefore, models with ADWIN change detector has a risk of partial learning over a stable segment of data stream instances. Higher memory consumption of the models with SeqDrift2 change detector is not of a major concern in memory constrained environments. The base classifier forest, CBDT has a mechanism to control its memory through least used decision path pruning and maximum memory calibration of each tree depending on its accuracy. This can shrink memory of the decision trees that have lower

accuracy than the current winner tree and increase maximum memory allowed for the current winner to enable it to grow to capture a concept precisely.

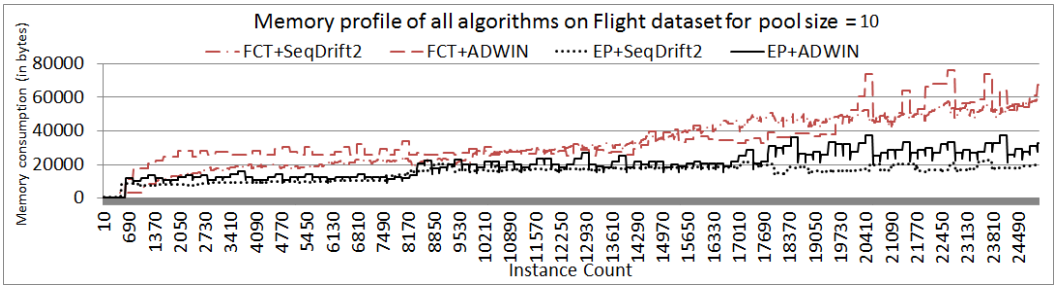


Figure 8.3: Memory profile of all algorithms on Flight dataset for pool size = 10

It would be interesting to study how memory consumption grows over the entire data stream. Figure 8.3 illustrates the trend in memory growth of Fourier pool over all algorithms for pool size 10 that doesn't restrict the models from maintaining a small number of spectra, thus allowing a clear contrast between FCT and EP approaches. As seen in 8.3, EP algorithm does not have a net growth in the interval 17K to 25K. This illustrates that EP has captured concepts fully and updated its ensembles using similar concepts seen from the start of the data stream to 17K. The EP+SeqDrift2 combination appears to have no memory growth after the arrival of the first 10K instances that span 4 different flights. The SeqDrift2 detector has enabled EP to capture concepts in detail by growing decision trees in the forest without signaling false alarms as opposed to ADWIN. Therefore, the Fourier spectrum of ensembles kept in EP+SeqDrift2 algorithm needed no significant update on recurring similar concepts.

The next performance metric is the processing speed of each algorithm.

8.3.3 Processing Speed Comparison

This is also an important property of an algorithm especially on high speed data streams produced by a Flight auto pilot system. In terms of processing speed, EP+SeqDrift2 is the clear winner with the highest processing speed for all pool size values as shown in Table 8.3. On average over all pool sizes, there is a 9.26% increase in processing speed for EP+SeqDrift2 over FCT+SeqDrift2. This increase is due to the reduced spectra that resulted through the aggregation process in EP. FCT has to route instances to a larger number of Fourier spectra than EP. In addition, redundancy reduction produces more compact structures in EP.

Algorithm	1	3	10
FCT+ADWIN	783.3	716.4	818.0
EP+ADWIN	829.4	863.1	851.8
FCT+SeqDrift2	823.8	818.9	729.2
EP+SeqDrift2	857.9	877.0	856.7

Table 8.3: Processing speed (number of instances processed per second) of all four algorithms on flight dataset for various pool size values.

Use of SeqDrift2 in place of ADWIN also has a contribution in processing time reduction in the EP+SeqDrift2 combination. This combination has 2% higher processing speed on average over all pool sizes when compared to EP+ADWIN. Though the increase may not seem very significant, in reality EP will benefit significantly on a very large data stream as the gap between EP+SeqDrift2 and EP+ADWIN will increase due to large number of false alarms that lead to expensive updates in the models kept.

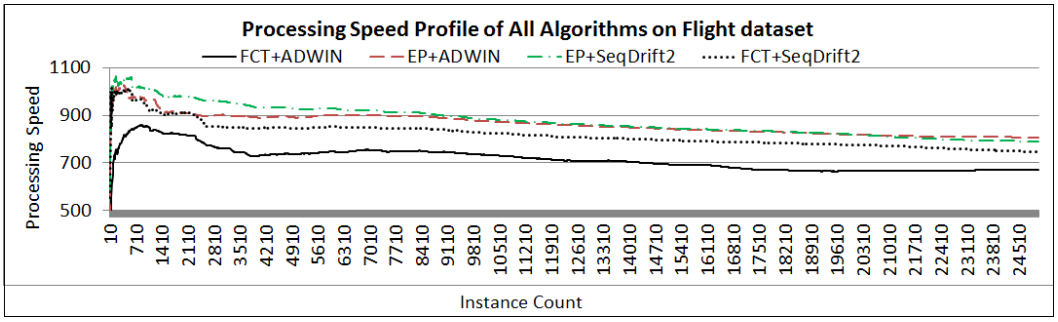


Figure 8.4: Processing speed profile of all algorithms on Flight dataset for pool size = 3

Figure 8.4 shows the profile of processing speed across the entire dataset. All four algorithms show a similar trend. Processing speed decreases as more and more instances are seen. This is a result of Fourier spectrum generation and memory management operations in all four models. FCT is expected to re-learn, create new Fourier spectra and remove the least performing Fourier spectra in memory constrained environment at high frequency. Therefore, a net decrease in processing speed can be anticipated throughout the data stream. Although, EP is designed to reduce relearning and memory management operations through aggregation, it too has a net decrease in processing speed. Analysis on Fourier spectrum generation has revealed that EP also has spent time in generating spectra even after its ensembles have captured most of the concepts. Though the newly generated spectra were ignored from being added to an existing ensemble due to higher level of structural similarity between the existing ensemble and new spectrum (refer Algorithm 7.1 in Chapter 7), EP suffers from spectrum generation. Future work will target at minimizing unnecessary Fourier spectrum generation if any given winner decision tree results in structurally similar Fourier ensemble spectra that already exist in the pool.This would stabilize the processing speed of EP algorithms after learning all the concepts in the underlying data stream.

At the same time, it should be noted that the speed of incoming instances in the Flight data stream is one instance per second. Therefore, all four models with around 750 times higher processing speed is more than adequate for this stream in practice.

8.3.4 Robustness to Concept Change

This is a new metric introduced in this research. In past literature, there has been no specific metric to compare the relative accuracy gain of any two given algorithms following a concept drift. Relative Accuracy Gain after a concept change indicates relative robustness of an algorithm. Here the specific interest is in tracking how different algorithms recover from changes in concept. Approaches that promote re-usability of models can be expected to recover faster and exhibit higher accuracy than other approaches that are forced to rely on re-learning the newly emerging concept instead of re-using appropriate models from the past.

This new metric is defined in this research and there has been no such metric in the literature to measure the relative ability to recover. This new metric is applicable on any dataset and it doesn't depend on true concept change points.

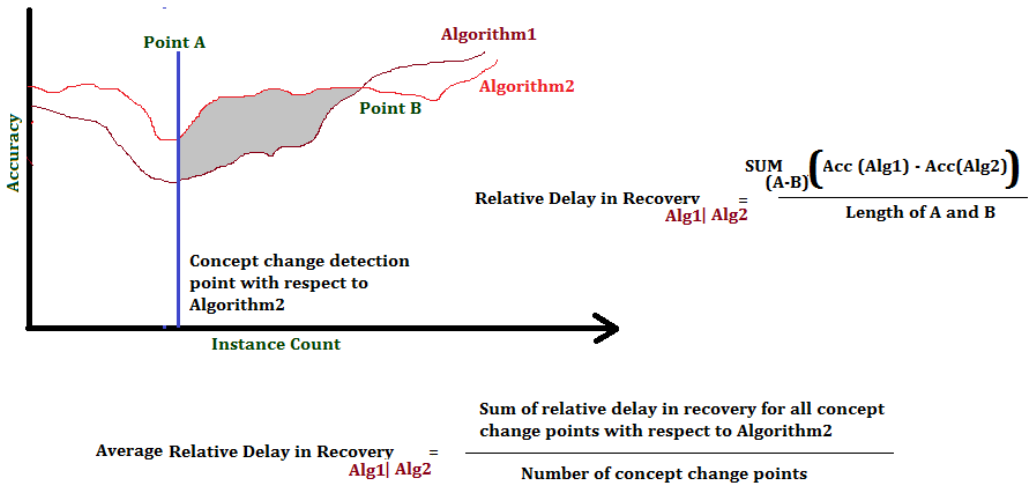


Figure 8.5: Accuracy recovery after concept change

The diagram shown in Figure 8.5 describes the data stream segment of interest (Point A and Point B) when computing the above metric. As an algorithm independently processes the stream elements and has a separate change detector, concept change detection points seen with the use of Algorithm 2 are not necessarily needed to be the same as the ones returned by Algorithm 1. Therefore, this metric is defined to be relative to an algorithm (say Algorithm 2) considering the concept change points associated with Algorithm 2. The metric quantifies the average accuracy gain of Algorithm 1 with respect to Algorithm 2 to meet Algorithm 2's accuracy for the first time after a concept change is detected by Algorithm 2. The larger the value, the more the stability of Algorithm 1 with respect to Algorithm 2.

Pool Size	FCT(ADWIN)/EP(ADWIN)	FCT(SeqDrift2)/EP(SeqDrift2)
1	0.0616	0.0592
3	0.006	0.0195
10	-0.012	0.002

Table 8.4: Average relative accuracy (scale of 0 to 1) gain until recovery after a concept change is detected

Table 8.4 shows the relative accuracy gain until recovery of FCT(ADWIN) against EP(ADWIN) and FCT(SeqDrift2) against EP(SeqDrift2) respectively. Each result indicates the average accuracy gain of an aggregated ensemble approach against a single Fourier Tree approach for a given change detector. When pool size is limited (memory constrained) there is a significant difference between an aggregated ensemble and single tree approaches for both change detectors. There is an 6% accuracy difference observed for pool size 1. As pool size increases, as expected, FCT's robustness to change improves as the metric values become smaller for both types of change detectors. With the maximum pool size of 10, FCT is almost as good or better than the EP method.

This metric values illustrates that an aggregated ensemble approach is preferable in a memory constrained environment to produce a more stable performance in terms of accuracy.

The next section concludes this case study with a summary of experimental observations.

8.4 Summary

This chapter has analyzed performance of four algorithms on a Flight dataset that reflects an environment with a large number of recurring similar concepts. Performance has been compared on the basis of accuracy, memory and

processing time. In addition, robustness to change in concept and the impact of memory constraints have also been evaluated.

The EP+SeqDrift2 combination is found to be the best choice, especially in a memory constrained environment with lesser delay in recovery, stable accuracy, lesser memory consumption and higher processing speed. The aggregation mechanism that helps to reduce redundancy, memory consumption and processing complexity has enabled EP to achieve superior results to FCT. Moreover, SeqDrift2 change detector also have contributed to lesser memory consumption and higher processing speed through its low false positive rate on stable data stream segments.

Improvements need to be made to avoid unnecessary Fourier spectrum generation of a winner decision tree if structurally similar ensembles already exist in the pool. Such improvements will further enhance the speed and stability of EP.

In a nutshell, the analysis in this chapter has revealed the effectiveness of the EP approach in recognizing recurring concepts in high speed data streams.

Chapter 9

Conclusion and Future Work

9.1 Research Accomplishments

Data stream mining while continuing to be an intensively researched problem, has reached a degree of maturity and is now applied to a large number of real world problems including stock markets, sensor networks, sales analysis, weather predictions, autonomous devices etc. Many real world applications produce data streams at high speed. In some application environments, typically wireless sensor networks, resources such as memory and processing power are constrained.

In addition, data streams usually embed changing concepts over time. Many application domains such as stock markets and weather show recurring trends over time. When concepts in the underlying data stream change, models need to be updated to cope with the change to continue to perform well. Moreover, when concepts recur, it would be advantageous to minimize relearning through the reuse of previously learned classifiers. This reduces sudden performance drops during relearning as a result of concept change.

This research has focused on capturing recurring concepts in high speed environments. The objective of this research was to propose a framework and associated algorithms to minimize relearning on similar recurring concepts while achieving accuracy benefits in a high speed environment with constrained memory.

Prior to formulating a framework for capturing recurring concepts, there

needs to be in place a change detector that does not assume that domain knowledge on stream elements is available, but is yet able to yield robust performance on high speed data streams. This research initially used the ADWIN change detector and suffered as a result of its relatively high false positive rate. Therefore, two novel changes detectors were introduced, namely SeqDrift1 and SeqDrift2.

SeqDrift1 was designed to maintain a sliding window of instances to detect changes using robust statistical bounds such as the Bernstein Bound and sequential hypothesis testing that assume no prior knowledge on stream characteristics. SeqDriftt1, similar to ADWIN, monitors a classifier's accuracy to find concept change points. The empirical study that compared SeqDrift1 with ADWIN has shown that SeqDrift1 had lesser false positive rate, higher noise tolerance and lesser processing time compared to ADWIN. However, detection delay of SeqDrift1 was higher than that of ADWIN. Therefore, SeqDrift1 was optimized into its second version, SeqDrift2, to reduce detection delay while maintaining the benefits of SeqDrift1.

In SeqDrift2, detection delay was minimized by improving the contrast between instances belonging to the previous and current concept with the use of a reservoir sampling strategy in place of a sliding window. In addition, the test statistic used in sequential hypothesis testing was also modified to enhance its sensitivity on slowly varying data. At the same time, the algorithm was designed to retain performance benefits of SeqDrift1 such as low false positive rate, high noise tolerance and high processing speed. Extensive experiments revealed that SeqDrift2 is superior to ADWIN in all of the above performance aspects while having comparable detection delay to that of ADWIN. With this improved change detector in place, this research moved on to a new phase of capturing recurring concepts in data streams.

Capturing recurrences of concepts has its own challenges. If models are to be remembered, there should be a mechanism to compress models to be effi-

ciently stored and reused in memory. The system has to be able to cope with the speed of the data stream and thus should operate with the lowest possible computational complexity without compromising on classification accuracy. . A robust strategy was designed to recognize concept recurrences and to select appropriate classifier models previously stored. Moreover, as concepts do not recur in exact form, it is necessary to remember the essence of a concept rather than being too specific to minor details of a concept. Having all of the above challenges in mind, this research designed two algorithms, namely FCT and EP that operate in conjunction with a base classifier, the decision tree forest CBDT.

The Fourier Concept Trees algorithm was optimized to store models in a highly compressed form in its memory. As the compression mechanism, the Discrete Fourier Transform was applied on Decision trees. Discrete Fourier Transform when applied on a decision tree yields a small number of low order Fourier coefficients that retain most of the classification power of a regular decision tree. Moreover, as DFT application captures only significant coefficients, the essence of a concept captured by a decision tree is only remembered. FCT learns a previously unseen concept using its decision tree forest and stores the best tree learned as a Fourier spectrum in its pool. When a concept similar to the one that has been captured by a Fourier spectrum recurs, the Fourier spectrum becomes the dominant classifier to perform classification. As FCT depends on a change detector to recognize concept changes, ADWIN and SeqDrift2 have been plugged in and tested. The empirical study has revealed the benefits of recognizing similar concepts compared to an existing algorithm called MetaCT. Accuracy gains and higher processing speed have been observed in experiments with a large number of synthetic and real world datasets over the use of MetaCT and the base classifier CBDT. The SeqDrift2 change detector has improved detection and helped reduce fluctuations in accuracy with its low false positive rate compared to ADWIN. FCT

has a memory management mechanism to maintain its pool in a memory constrained environment. In memory constrained environments, FCT is at risk of removing an existing well performing Fourier spectrum to make way for a new spectrum to be put in place. This may reduce the degree of reuse of classifiers and increase the chances of having to relearning. Moreover, FCT has a tendency to produce separate Fourier spectra with a lot of redundancy on similar concepts if parameters that govern insertion of spectra in the pool are not set appropriately. To resolve these issues, an improved version EP was proposed in this research.

EP exploits the DFT on decision trees by removing redundancies and optimizing the Fourier spectrum generation process. Fourier spectra being mathematical functions lend themselves easily to aggregation. EP takes advantage of this aggregation property of Fourier spectra. Aggregation removes redundancy by only storing unique coefficients which are weighted by relative accuracy which are in turn obtained from the spectrum's underlying decision tree. When similar concepts are captured by two different decision trees, aggregation of corresponding Fourier spectra results in common coefficients being integrated into a single spectrum. This strengthened the ability to capture recurring concepts in addition to reducing memory and computation complexity in classifier storage and classification.

The next problem that was investigated was a strategy to select a pair of spectra to aggregate. EP was studied with two aggregation strategies that are memoryless and efficient to execute. One strategy was based on the similarity in accuracy and the other was on the model structure. As Fourier coefficients do not preserve model structure explicitly, a mechanism was introduced to very effectively compare two different spectra. In addition to the above improvements, EP also has enhanced methods to calculate Fourier coefficient values efficiently. These methods are exact, not heuristics and are underpinned by two theorems presented in Chapter 7. Application of these

methods reduced processing time while returning results which are exactly the same as in the unoptimized version.

An empirical study was conducted on a different set of datasets than the ones used in the FCT experiments. A large number of concepts were concatenated and reintroduced in similar form to test EP. Both ADWIN and SeqDrift2 change detectors were again chosen to study the impact of change detection. A reduced number of fluctuations in accuracy, lower memory consumption and higher processing speed was seen in EP when compared to FCT no matter which change detector was used. However, it was also observed that SeqDrift2 helps to stabilize model performance compared to ADWIN, no matter which of FCT or EP was employed. A separate case study was done on a Flight data set that simulates the targeted application of the models proposed in this research. This case study also confirms the above benefits of EP and FCT over existing methods. Memory profile revealed that EP+SeqDrift2 had the ability to consume less than 50% lower memory compared to FCT+ADWIN that suffers from both the ADWIN's relatively high false positive rate and redundancy in Fourier spectra. In addition, EP+SeqDrift2 was shown to have around 15% gain in accuracy in memory constrained (pool size=1) environments in the case study.

Additionally, this research also has proposed a general framework to capture recurring concepts by archiving classifiers in a data stream. This framework includes classifiers, change detectors and model compressors.

9.2 Overall Reflection on Achievements

Overall, this research claims that the twin research objectives of formulating a new change detector and recurrent concept detector has been accomplished.

The research question that addressed the reduction in false positive rate in change detection compared to state of art methods is answered with two

different change detectors: SeqDrift1 and SeqDrift2. The second version of the change detector (SeqDrift2) has been shown to be better or competitive to the key performance metrics concerned. Theoretical guarantees on performance and extensive experimentation revealed that it outperforms several widely used change detectors. Therefore, this research concludes that the use of the Bernstein Bound, together with the use of Sequential hypothesis testing and Reservoir sampling form a strong combination that improves the false positive rate of a change detector while maintaining competitiveness with respect to other standard performance metrics used in change detection..

It has been shown that in the empirical study that the application of DFT has significantly reduced storage overhead with comparable or better computational overhead when compared to standard methods that use Hoeffding Trees in their original form, such as MetaCT. This provides empirical evidence in support for the second research question on whether the DFT application has the potential to improve the values of the above performance metrics.

Similarly the proposed recurrence capture algorithms comprehensively outperformed the current state-of-the-art meta learning algorithm in terms of recurrence capture.

Alternatives to encoding a Fourier spectrum to represent a concept are also explored with two different aggregation strategies and memory management mechanisms for the classifier pool. The empirical study showed that the strategy based on aggregating structurally similar spectra outperformed the other that was based on a greedy strategy of aggregating the pair of spectra that had the closest classification accuracy to each other.

In this sense, the goals that were set at the beginning of the research have truly been met. However, in the course of the research, a number of limitations were encountered with the methods that were proposed. In the following section these limitations are reflected on and some questions are posed for future research.

9.2.1 Limitations of this Research

The coupled approach with a change detector and a pool of classifier may not be the optimal choice for all types of data streams. This is due to the inherent limitations of change detectors, classifiers and algorithms employed for concept encoding and recurrence capture. For example, the proposed approach may not fit a data stream that has very frequently changing concepts with very short concept duration. This is due to the fact that inherent delays in detecting changes and learning concepts may lead to imprecision in the capture of concepts. The Hoeffding tree requires a sufficient number of instances to learn and any change detector requires a grace period to make statistically valid decisions. When concept duration is shorter than the grace period for learning and detecting changes, the proposed approach may not produce accurate outcomes. In such cases, decoupling concept detection from classifier learning may be a better solution, especially if the learner is able to adapt to changes quickly. However the danger is that in some cases, concepts that span very a few data instances may be noise, thus resulting in the learner erroneously capturing spurious concepts, leading to poor generalization capability.

The target data stream environment of the proposed approach is where concepts recur in reasonably lengthy durations so that change detector and classifiers will have sufficient statistics to evaluate the stream elements.

Other limitations are listed as follows:

- **The scalability of DFT application**. The number of Fourier coefficients in a spectrum is exponential with respect to the number of features in a data stream. Though only a few coefficients are significant from the viewpoint of classification accuracy, it is computationally expensive to extract those significant coefficients. This research reduced the severity of this problem by optimizing Fourier coefficient calculation

(in Chapter 7) and a localized approach was adopted that generated spectra from only the features actually present in a tree (as opposed to the data stream), without any attendant loss of information. In case, where all or most of the features are present in a tree constructed from a very high dimensional data stream, the DFT process will become a major bottleneck in processing the data stream. The FCT and EP algorithms thus have the potential to suffer from this limitation of DFT application in a high dimensional data stream.

- **The suitability of DFT application for continuous valued attributes**. As shown in [Byung-Hoon 2001], DFT transformation of a decision tree is not an optimal approach for continuous valued data features. The reason for this issue is again due to the exponential number of Fourier coefficients. Continuous valued features have a tendency to split on a large number of values in a decision tree. When calculating Fourier coefficients for such a tree, each of these values needs to be considered to be a different feature. Therefore, the number of features will drastically increase, thus giving rise to an exponential increase in computational complexity.

- **The delay in true class label/feedback**. The algorithms FCT and EP rely on classification accuracy in order to select a winner tree at stream change points. In addition, change detectors that monitor classifier accuracy to signal a change are dependent on the availability of class labels arriving on time. A delay in receiving class labels for data instances will lead to a delayed response in adapting to new concepts in underlying data streams. As a workaround, weighted classification across several different classifiers could be used as an alternative to depending on a sole winner tree. This could improve classification accuracy in intervals where labels are in short supply or are absent altogether.

Moreover, the approach used in MetaCT [Gama 2011] to learn the areas where a classifier performs well using another classifier (referee) can be adopted with the FCT and EP algorithms in environments where there is a long delay in receiving true class labels. As noted in the research, experimentation in Chapter 6 confirmed that MetaCT's performance is severely affected by the choice of a classifier such as the Hoeffding Tree that needs a large sample in order to learn a referee model. Therefore, an appropriate classifier that learns quickly at an acceptable rate of performance must be chosen as a solution to handle delay in obtaining class labels.

- **Recognition of very short concepts by SeqDrift algorithms**. SeqDrift algorithms check for concept changes in an interval of block size specified as a parameter. In a highly volatile environment, concept durations can be lesser than the size of block size. When such a short concept occurs within a block, it will not be recognized by the change detectors. Small values of block size reduce sensitivity of the change detectors. Therefore, an improved test statistic that can exploit statistics of a small sample and accompanying algorithmic enhancements need to be applied to the SeqDrift change detectors in such environments.

- **Dependence on Decision tree classifier**. Both FCT and EP explicitly depend on a decision tree structure to apply DFT. Therefore, the application of the above algorithms is limited to the environments where decision trees are suitable. In other environments operating with different classifiers, the design principles involved in the construction of the FCT and EP algorithms can be reused by replacing the DFT with a different compression scheme.

- **Dependence on a change detector**. Dependence on a change de-

tector algorithm propagates the error of a change detector to recurrence concept capture. The plausible solution would be to choose a change detector that has the best performance especially in terms of false positive and false negative rates.

In addition, this research opens up a number of research questions that need to be investigated in the future to understand a recurrence capture methodology using Discrete Fourier Transform applied on a decision tree.

9.2.2 Interesting Open Research Questions

A number of challenging research questions arise on the basis of this research.

- *Can the most significant Fourier coefficients be extracted in at least in linear computational complexity as opposed to exponential with respect to the number of features?* A solution to this problem will tremendously expand the applicability of DFT on any decision tree in any data stream environment including continuous valued features.

- *Can drastic improvement in change detector sensitivity be obtained without compromising on false positive rate?* Each change detector has its own inherent delay in recognition of a change. When delay is minimized, the data segments that result belong to a specific concept precisely. Such segments will directly be useful to describe concepts, to improve recurrence capture and to maximize the use of meta statistics to predict future concepts in real world applications with volatile data streams.

- *How can previously captured models be best exploited to handle totally unseen data stream environments where new learning is impossible or hard?* This problem is closely connected to the so called "Intelligent" approach that reuses previous knowledge to handle unseen situations.

- *How can a concept drift and concept recurrence best be predicted, as opposed to being detected reactively?* This would be an interesting problem that has never been addressed in the literature. In a large volatile data stream, meta statistics about concept change and recurrence capture could be used to make predictions about the future of data stream at a conceptual, rather than instance level. A meta classifier may be even learned to predict these occurrences to prepare other classifiers for future changes in data streams.

- *What is the best strategy to aggregate any two classifiers?* In ensemble based methods, classifier aggregation has a number of benefits including redundancy reduction in concept representation and decrease in computational and memory overheads as claimed by this research. Determination of the best aggregation strategy is a hard problem with lack of meta knowledge on the concepts present in a data stream.

- *How can a change detector or recurrence capturing algorithm be evaluated in a real world data stream with no domain knowledge on change points?* Performance metrics such as false positive rate, false negative rate assume that actual change points are known. The solution to this problem could benefit end users by helping them to choose the best models on their own without an expert's advice.

- *How can concept change be represented precisely in a form that can be interpreted by end users?* Concept change representation or description is one of the new research areas in data stream mining. This has potential benefits to the end users to understand the data generation process and even to find abnormalities in data. Although practical solutions such as decision trees exists to represent concepts, the accuracy of representation, relevance and usefulness of such representations needs to be

evaluated before a precise description of concept change can be made.

- *Can frequent pattern mining be extrapolated to capture concept recurrence? How can a concept representation be interpolated to extract frequent patterns found in data streams?* Concepts consist of patterns. Frequent pattern mining is aimed at extracting dominant patterns in given data segments. Thus, dominant patterns in a given recurring concept could be identified in terms of frequent patterns. Once these frequent patterns are extracted from different concepts then new concepts could be created by combining these patterns to tackle an unseen environment intelligently.

- *Can any classifier be used in the Decision Tree DFT framework used in this research?* The DFT requires a truth table that has all possible instances vectors and corresponding class values to generate Fourier Spectrum. It is not strictly limited to a decision tree. If a truth table can be generated for a classifier either in an incremental manner or by using heuristics, then the Fourier Spectrum of that classifier can be computed. Once it is in spectral representation, the reconstruction algorithm proposed in [Byung-Hoon 2001] to reproduce a decision tree from Fourier Spectrum can be used to obtain the representation of any classifier in a decision tree form. This has a potential advantage of revealing the knowledge captured by the so called **Black Box** classifiers such as neural networks.

9.3 Future Work

The future work is aimed at improving the proposed models in many different aspects.

- **Extension of the methods proposed**. The coupled approach which lays the foundation for the theoretical contribution can be improved in many ways. In the proposed method, there is a tight dependence between the change detection mechanisms and concept representation schemes. In other words, classifier error rate is the defining factor of a concept drift. In an unlabeled segments of the data stream, when class error rate is not relevant, change detection can be applied at the feature (attribute) level to locate concept changes, in addition to classifier error rate. To make the proposed methods better suited for frequent concept changes with short durations, improvements can be made to the test statistic and the algorithmic strategy employed by the change detector. Furthermore, classifiers that are fast learners may replace Hoeffding Tree when concept interpretation is not essential. Moreover, improvements to strategies on how past classifiers are aggregated, stored and managed will strengthen the theoretical contribution and expand the application domain.

- **Further Optimization on Fourier Coefficient Generation**. The total number of Fourier Coefficients is exponential in terms of the number of dimensions (features) present in a data stream. Even though energy thresholding can be used to restrict generation to low order coefficients, the time complexity is $O(n^m)$ where n is the number of dimensions (features) in the tree and m is the order at which the energy threshold has been reached. For $m > 2$ and sufficiently large n, spectrum generation will become a bottleneck. There are two possible approaches to dealing with this issue. First of all, a multi-dimensional version of the Fast Fourier Transform (FFT) that currents only exists on unidimensional data could be devised to speed up spectrum generation. An alternative strategy would be to use Bayesian methods to

estimate higher order coefficients from low order coefficients instead of using computationally expensive inner product operations This would enable application on a continuous valued attribute datasets which could generate very large number of attributes resulting from splits on a continuously valued numerical range

- **Parallel Processing and Pipelining Techniques**. Both FCT and EP support the use of parallel processing. Each of the major functional components such as learning a classifier, Fourier spectrum generation, memory management, concept change detection etc., can be assigned to different processing units. Similarly, Fourier spectrum generation by itself can also be pipelined to reduce processing time even further.

- **Improvements to SeqDrift Change Detectors**. Data structures and test statistics can be optimized to save memory and processing time in the two change detectors. The test static can be modified with a more tighter bound than the Bernstein Bound. Moreover, the reservoir that stores the input to change detectors explicitly can be modified to remember representative statistics.

- **Algorithmic Optimizations on FCT and EP**. FCT and EP should be enhanced to completely avoid the generation of a Fourier spectrum if it is expected to be similar to any existing Fourier spectrum that is already in the pool. This saves significant processing time spent on an unnecessary task. Aggregation in EP needs to be enhanced with a better weighting of Fourier coefficients and other aggregation methods. In addition to the above, delay in responding to recurring concepts also should be minimized.

- **Dynamic Energy Thresholding**. In the current models, energy threshold is set as a static value that does not change with data stream dy-

namics. Having such a static value is not optimal as new concepts that appear will vary in complexity thus requiring the use of different sized thresholds. In this respect it would be worth investigating the relationship between accuracy and energy threshold. If this relationship is modeled, then accuracy can be optimized by setting the right value for energy threshold. This would enhance the ability to capture and classify concepts precisely as well.

- **Use of Meta Mining** . Meta mining on model usage and change detection can be exploited to predict concept changes and the best classifiers to be used on future concepts. This will also ultimately lead to a method of describing concept changes in terms that can be explained to end users.

- **Modeling Intelligence of Brain**. Intelligence manifests in different ways. In a novel interpretation, it can be described using a recurring concept capturing framework. Taking humans as an example, a human has a number of sensors, processing units and memory. The sensors produce data streams that are received by brain. Over a lifetime, similar patterns or concepts are experienced or seen very often. The human brain can also be considered to be a model that reuses knowledge and classifiers built rather than re-learning concepts each time. In addition, it does prediction and handles unseen environments by relying on some form of model, which is presumably to a greater or lesser extent, computational. Once actual unfamiliar environments become familiar it stores information on it for future reuse. Therefore, a recurring concept capturing framework can be extended to simulate brain functions.

Appendix A

Appendix for Chapter 4

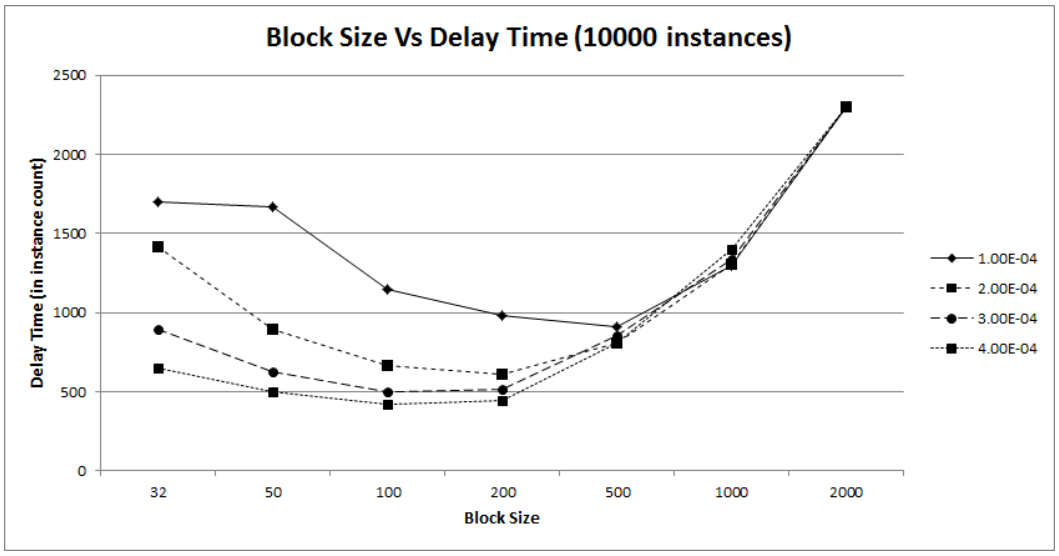


Figure A.1: Effects of Block Size on Detection Delay Time for SeqDrift1 for the data length 10,000

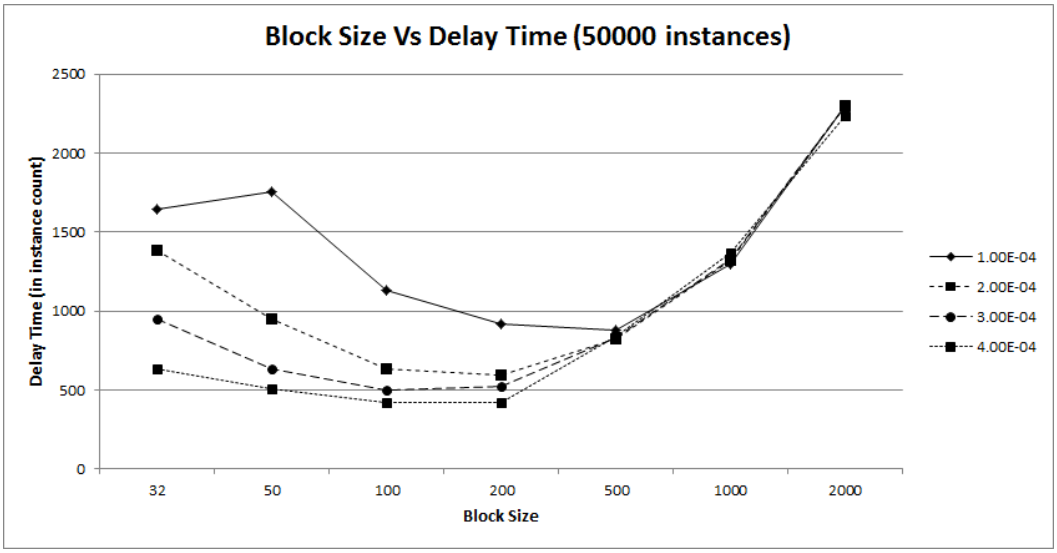


Figure A.2: Effects of Block Size on Detection Delay Time for SeqDrift1 for the data length 50,000

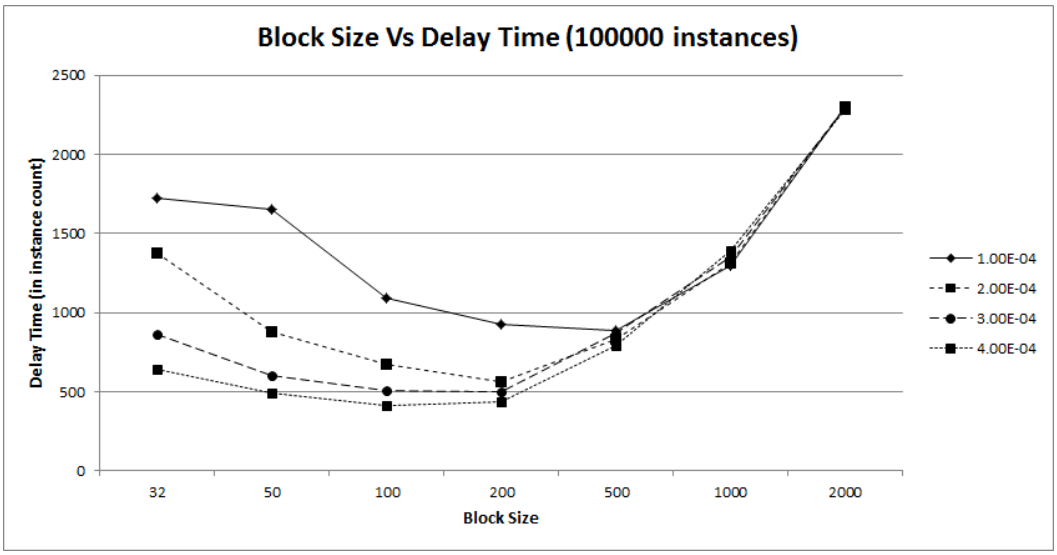


Figure A.3: Effects of Block Size on Detection Delay Time for SeqDrift1 for the data length 100,000

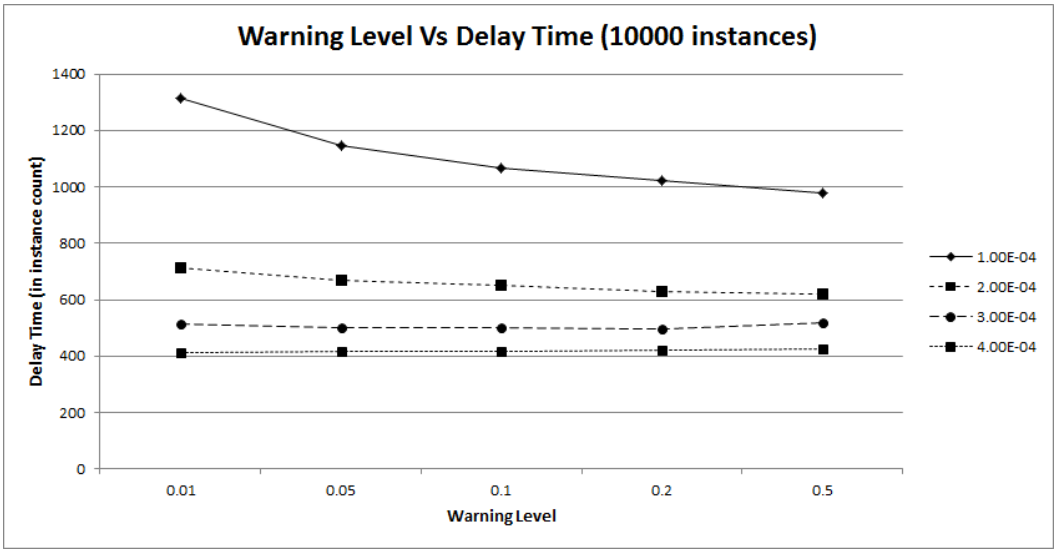


Figure A.4: Effects of Warning Level on Detection Delay Time for SeqDrift1 for the data length 10,000

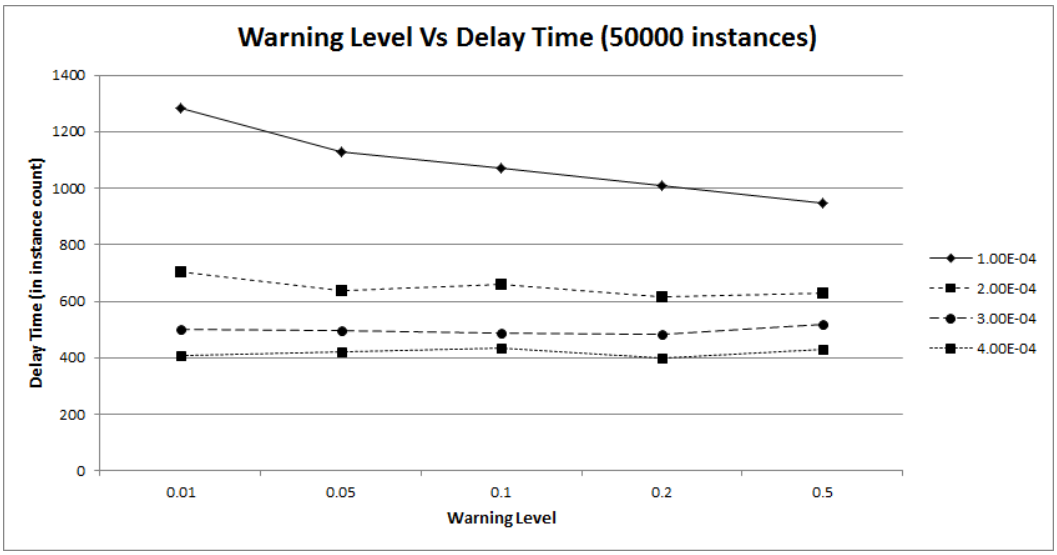


Figure A.5: Effects of Warning Level on Detection Delay Time for SeqDrift1 for the data length 50,000

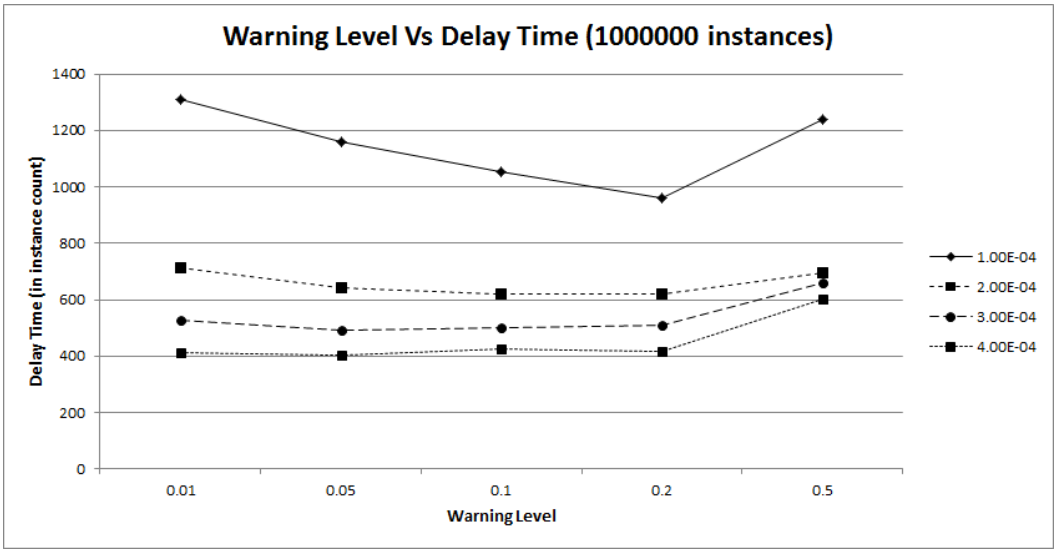


Figure A.6: Effects of Warning Level on Detection Delay Time for SeqDrift1 for the data length 1,000,000

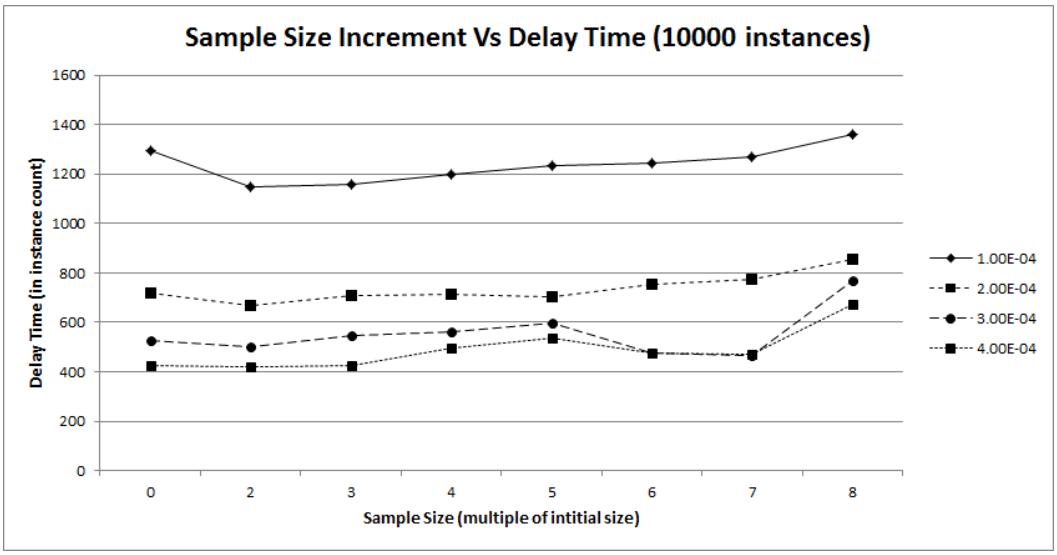


Figure A.7: Effects of Sample Size Increment on Detection Delay Time for SeqDrift1 for data length 10,000

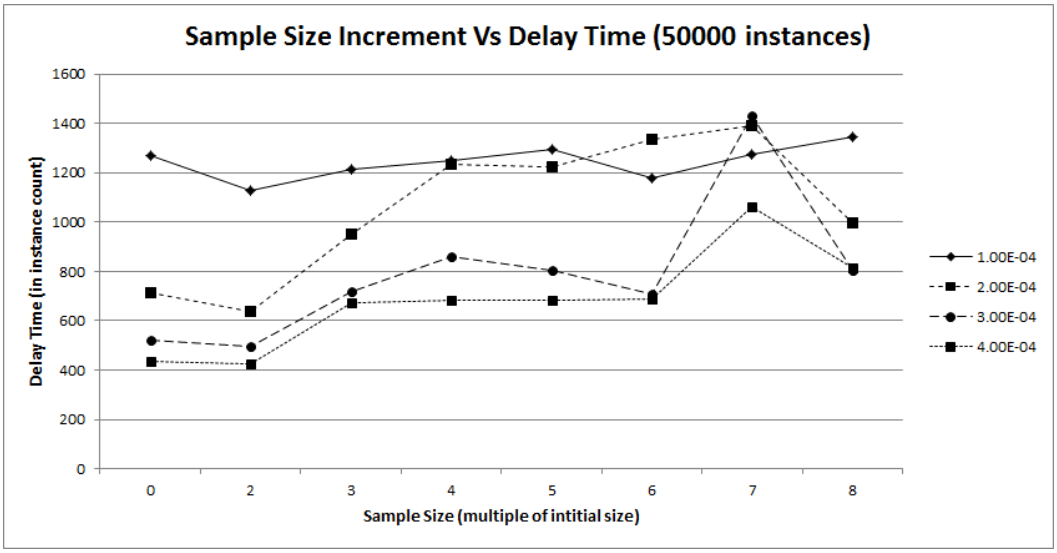


Figure A.8: Effects of Sample Size Increment on Detection Delay Time for SeqDrift1 for data length 50,000

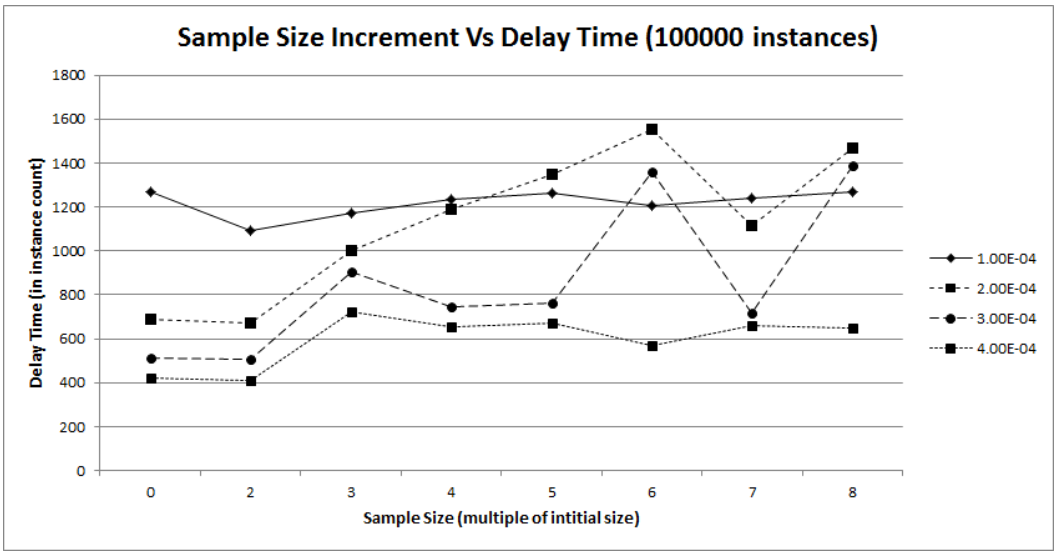


Figure A.9: Effects of Sample Size Increment on Detection Delay Time for SeqDrift1 for data length 100,000

Appendix B

Appendix for Chapter 6

Figure B.1 shows the memory profiles of FCT and MetaCT. Each of the algorithm has two segments of memory. Therefore, there are four in total namely FCT Tree Forest, FCT Pool, MetaCT Tree Forest and MetaCT Pool. Please refer 6.6.4.2 for more details on the experiment.

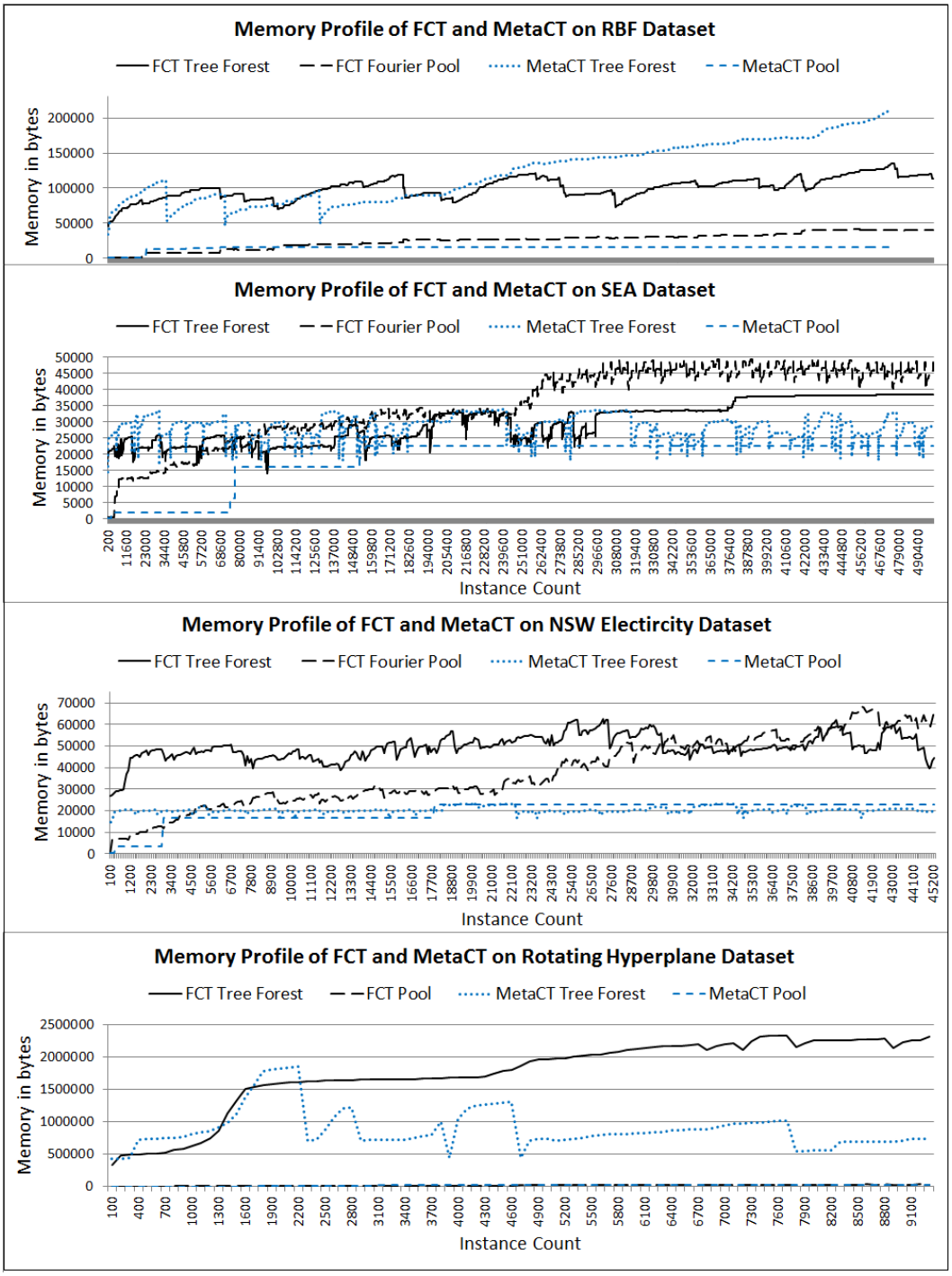


Figure B.1: Memory profiles of FCT and MetaCT on RBF, SEA, Electricity and Spam Datasets

Appendix C

Appendix for Chapter 7

Figure C.1 shows the accuracy profiles of FCT, EPa and EP on SEA dataset for the pool sizes 3, 5 and 10.

Please refer 7.3.5 for more details on the experiment.

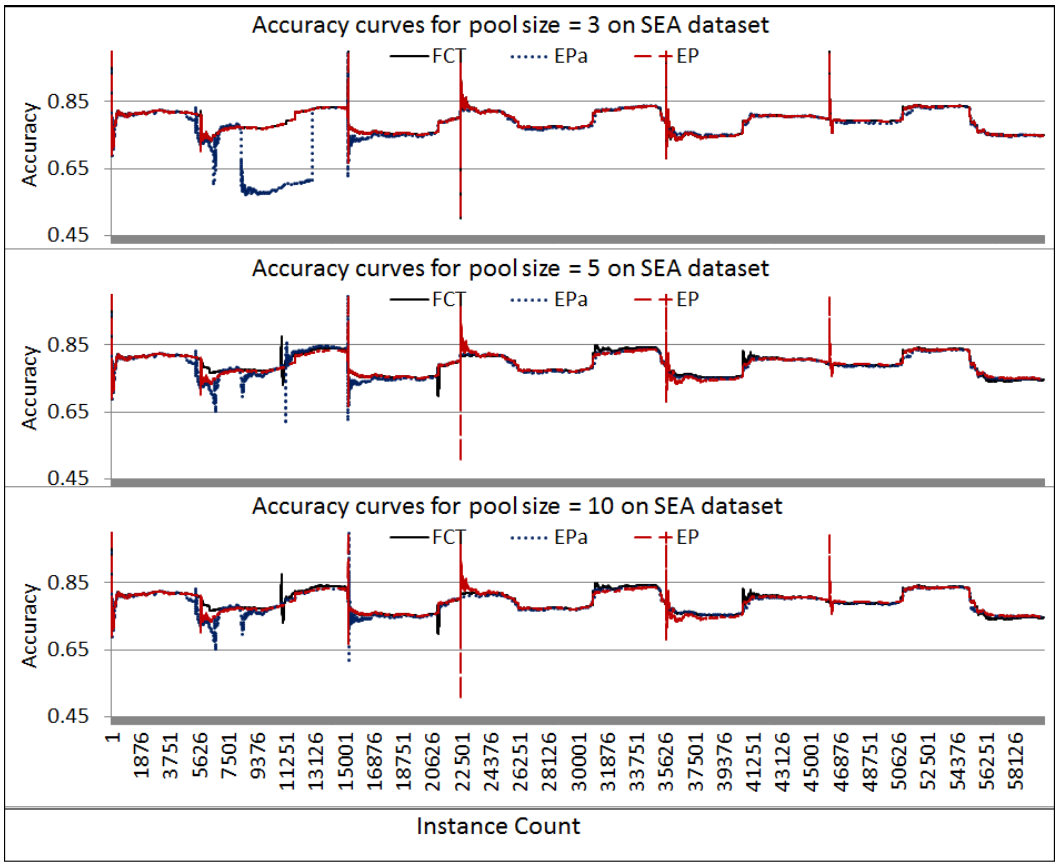


Figure C.1: Accuracy profile of FCT, EPa and EP on SEA dataset for pool sizes 3,5 and 10

Figure C.2 shows the accuracy profiles of FCT, EPa and EP on SEA dataset for the pool sizes 3, 5 and 10.

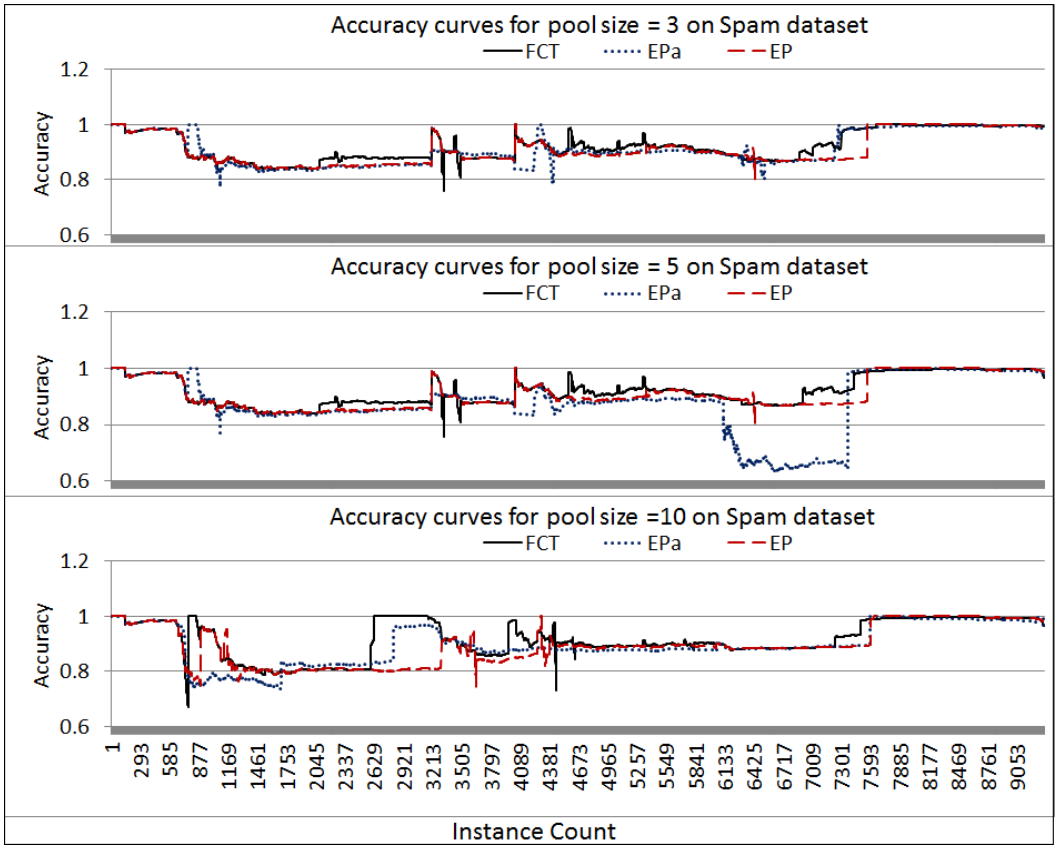


Figure C.2: Accuracy profile of FCT, EPa and EP on Spam dataset for pool sizes 3,5 and 10

Figure C.3 shows the accuracy profiles of FCT, EPa and EP for various energy thresholds on Rotating Hyperplane dataset.

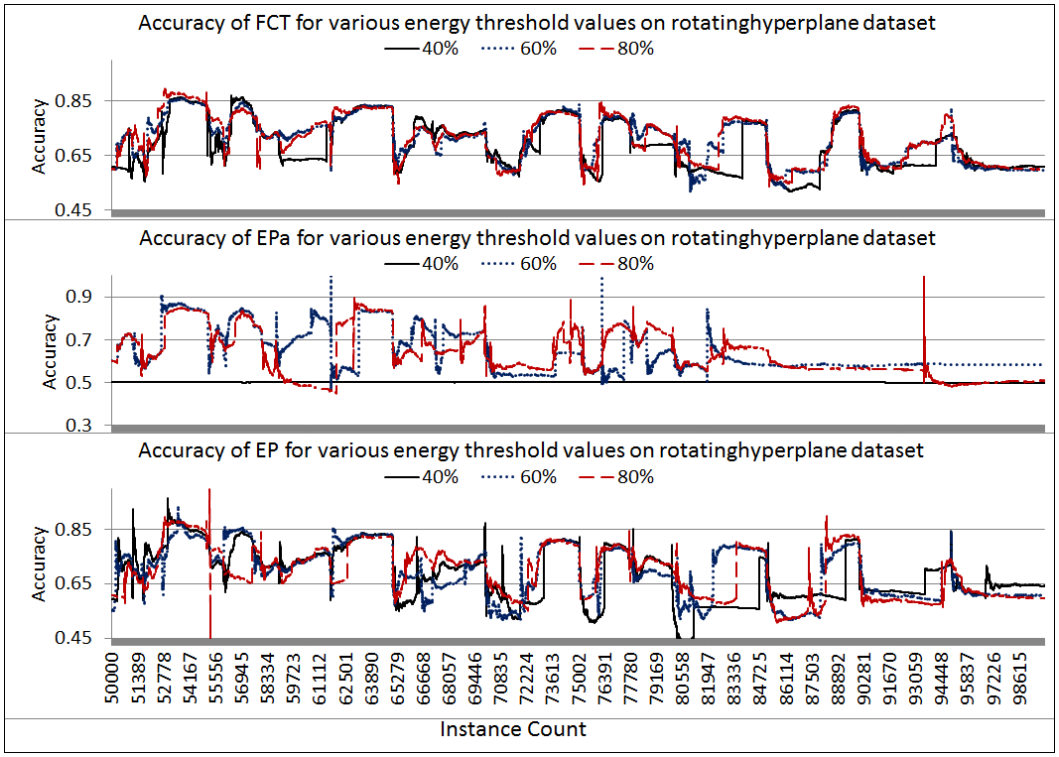


Figure C.3: Accuracy profile of FCT, EPa and EP on Rotating Hyperplane for various energy thresholds

Bibliography

[Aggarwal 2003] Charu C. Aggarwal, Jiawei Han, Jianyong Wang and Philip S. Yu. *A Framework for Clustering Evolving Data Streams*. In Proceedings of the 29th International Conference on Very Large Data Bases, volume 29 of *VLDB '03*, pages 81–92. VLDB Endowment, 2003. 36

[Alippi 2013] C. Alippi, G. Boracchi and M. Roveri. *Just-In-Time Classifiers for Recurrent Concepts*. IEEE Transactions on Neural Networks and Learning Systems, vol. 24(4), pages 620–634, 2013. 4, 119, 122

[Audibert 2007] Jean-Yves Audibert, Rémi Munos and Csaba Szepesvári. *Tuning Bandit Algorithms in Stochastic Environments*. In Proceedings of the 18th International Conference on Algorithmic Learning Theory, ALT '07, pages 150–165, Berlin, Heidelberg, 2007. Springer-Verlag. 42

[Baena-García 2006] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldá and Rafael Morales-Bueno. *Early Drift Detection Method*. In Proceedings of the 4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams, pages 77–86, Berlin, Germany, 2006. 5, 36

[Basseville 1993] Michèle Basseville and Igor V. Nikiforov. Detection of abrupt changes: Theory and application. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. 35

[Bender 1999] Ralf Bender and Stefan Lange. *Multiple Test Procedures Other Than Bonferroni's Deserve Wider Use*. BMJ : British Medical Journal, vol. 318, no. 7183, pages 600–600, 1999. 50

[Bernstein 1946] Sergei N. Bernstein. The theory of probabilities. Moscow, Leningrad, 1946. 39, 41

[Bifet 2007] Albert Bifet and Ricard Gavaldà. *Learning from Time-Changing Data with Adaptive Windowing.* In Proceedings of the 7th SIAM International Conference on Data Mining, pages 443–448. SIAM, 2007. 5, 37, 38, 39, 41, 102, 130, 159

[Bifet 2009] Albert Bifet and Ricard GavaldÃ. *Adaptive Learning from Evolving Data Streams.* In Advances in Intelligent Data Analysis VIII, volume 5772 of *Lecture Notes in Computer Science*, pages 249–260. Springer Berlin Heidelberg, 2009. 37, 111

[Bifet 2010a] Albert Bifet. Adaptive stream mining: Pattern learning and mining from evolving data streams. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2010. 8, 22

[Bifet 2010b] Albert Bifet, Geoff Holmes, Richard Kirkby and Bernhard Pfahringer. *MOA: Massive Online Analysis.* The Journal of Machine Learning Research, vol. 11, pages 1601–1604, 2010. 13, 111, 114, 136, 171

[Borne 2007] Kirk D. Borne. *A Machine Learning Classification Broker for Petascale Mining of Large-scale Astronomy Sky Survey Databases.* In Next Generation of Data Mining and Cyber-Enabled Discovery for Innovation (NGDM07), Baltimore, Maryland, 2007. National Science Foundation. 83

[Brown 2010] Gavin Brown. *Ensemble Learning.* In Claude Sammut and Geoffrey I. Webb, editeurs, Encyclopedia of Machine Learning, pages 312–320. Springer US, 2010. 153

[Byung-Hoon 2001] Park Byung-Hoon. *Knowledge Discovery from Heterogeneous Data Streams Using Fourier Spectrum of Decision Trees.* PhD thesis, Pullman, WA, USA, 2001. 123, 126, 168, 226, 230

[Das 2009] Kamalika Das, Kanishka Bhaduri, Sugandha Arora, Wesley Griffin, Kirk Borne, Chris Giannella and Hillol Kargupta. *Scalable Distributed Change Detection from Astronomy Data Streams using Local, Asynchronous Eigen Monitoring Algorithms.* In 2009 SIAM International Conference on Data Mining, pages 245–156, Nevada, 2009. SIAM. 83

[Datar 2002] Mayur Datar, Aristides Gionis, Piotr Indyk and Rajeev Motwani. *Maintaining Stream Statistics over Sliding Windows.* SIAM Journal on Computing, vol. 31, no. 6, pages 1794–1813, 2002. 65

[Domingos 2000] 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. 129

[Fayyad 1996] Usama M. Fayyad, Gregory Piatetsky-Shapiro and Padhraic Smyth. *Advances in Knowledge Discovery and Data Mining.* chapitre From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996. 1

[Freund 1996] Yoav Freund and Robert E. Schapire. *Experiments with a New Boosting Algorithm.* In Lorenza Saitta, editeur, Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996), pages 148–156. Morgan Kaufmann, 1996. 153

[Gaber 2005] Mohamed Medhat Gaber, Arkady Zaslavsky and Shonali Krishnaswamy. *Mining Data Streams: A Review.* ACM SIGMOD Record, vol. 34, no. 2, pages 18–26, June 2005. 2, 17

[Gama 2004] João Gama, Pedro Medas, Gladys Castillo and Pedro Rodrigues. *Learning with Drift Detection.* In AnaL.C. Bazzan and Sofiane Labidi, editeurs, Advances in Artificial Intelligence - SBIA 2004, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer Berlin Heidelberg, 2004. 36

[Gama 2005] João Gama, Pedro Medas and Pedro Rodrigues. *Learning Decision Trees from Dynamic Data Streams.* In Proceedings of the 2005 ACM Symposium on Applied Computing, SAC '05, pages 573–577, New York, NY, USA, 2005. ACM. 129

[Gama 2010] João Gama. Knowledge discovery from data streams. Data Mining and Knowledge Discovery Series. CRC Press, Minneapolis, 2010. 5

[Gama 2011] João Gama and Petr Kosina. *Learning about the Learning Process.* In João Gama, Elizabeth Bradley and Jaakko Hollmén, editeurs, Advances in Intelligent Data Analysis X, volume 7014 of *Lecture Notes in Computer Science*, pages 162–172. Springer Berlin Heidelberg, 2011. 4, 119, 120, 121, 134, 136, 151, 227

[Gomes 2010] João Bártolo Gomes, Ernestina Menasalvas and Pedro A. C. Sousa. *Tracking Recurrent Concepts Using Context.* In Rough Sets and Current Trends in Computing, volume 6086, pages 168–177. Springer Berlin Heidelberg, 2010. 4, 119, 120, 121, 122

[Hardy 1988] G. H. Hardy, J. E. Littlewood and G. Polya. Inequalities. Cambridge University Press, Cambridge, England, 1988. 41

[Ho 2005] Shen-Shyang Ho. *A Martingale Framework for Concept Change Detection in Time-varying Data Streams.* In ICML '05 Proceedings of the 22nd International Conference on Machine Learning, ICML '05, pages 321–327, New York, NY, USA, 2005. ACM. 27, 36, 37

[Hoeglinger 2007] Stefan Hoeglinger and Russel Pears. *Use of Hoeffding trees in concept based data stream mining.* In 3rd International Conference on Information and Automation for Sustainability, ICIAFS 2007, pages 57–62. IEEE, 2007. 7, 159

[Hoeglinger 2009] Stefan Hoeglinger, Russel Pears and Yun Sing Koh. *CBDT: A Concept Based Approach to Data Stream Mining.* In Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD '09, pages 1006–1012, Berlin, Heidelberg, 2009. Springer-Verlag. 8, 129, 159

[Hosseini 2012] Mohammad J. Hosseini, Zahra Ahmadi and Hamid Beigy. *New Management Operations on Classifiers Pool to Track Recurring Concepts.* In Alfredo Cuzzocrea and Umeshwar Daya, editeurs, Data Warehousing and Knowledge Discovery, volume 7448 of *Lecture Notes in Computer Science*, pages 327–339. Springer Berlin Heidelberg, 2012. 119

[Hulten 2001] Geoff Hulten, Laurie Spencer and Pedro Domingos. *Mining Time-changing Data Streams.* In Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM. 8, 129, 135

[Inglada 2007] Jordi Inglada and Grégoire Mercier. *A New Statistical Similarity Measure for Change Detection in Multitemporal SAR Images and Its Extension to Multiscale Change Analysis.* IEEE Transactions

on Geoscience and Remote Sensing, vol. 45, no. 5, pages 1432–1445, May 2007. 39

[Iqbal 2012] Md.RidwanAl Iqbal. *Rule Extraction from Ensemble Methods Using Aggregated Decision Trees.* In Tingwen Huang, Zhigang Zeng, Chuandong Li and ChiSing Leung, editeurs, Neural Information Processing, volume 7664 of *Lecture Notes in Computer Science*, pages 599–607. Springer Berlin Heidelberg, 2012. 154

[Kargupta 2004] Hillol Kargupta and Byung-Hoon Park. *A Fourier spectrum-based approach to represent decision trees for mining data streams in mobile environments.* IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 2, pages 216–229, Feb 2004. 8

[Kargupta 2006] Hillol Kargupta, Byung-Hoon Park and Haimonti Dutta. *Orthogonal Decision Trees.* IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 8, pages 1028–1042, 2006. 8, 120, 123, 124, 127, 138, 156

[Katakis 2008] Ioannis Katakis, Grigorios Tsoumakas and Ioannis Vlahavas. *An Ensemble of Classifiers for Coping with Recurring Contexts in Data Streams.* In Proceedings of the 18th European Conference on Artificial Intelligence, pages 763–764, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. 4, 24, 121

[Kifer 2004] Daniel Kifer, Shai Ben-David and Johannes Gehrke. *Detecting Change in Data Streams.* In Proceedings of the 30th International Conference on Very Large Data Bases, volume 30 of *VLDB '04*, pages 180–191. VLDB Endowment, 2004. 36, 37, 38

[Klinkenberg 2000] Ralf Klinkenberg and Thorsten Joachims. *Detecting Concept Drift with Support Vector Machines.* In Proceedings of the 17th

International Conference on Machine Learning, ICML '00, pages 487–494, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. 36

[Krempl 2014] Georg Krempl, Indre Žliobaite, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, Sonja Sievi, Myra Spiliopoulou and Jerzy Stefanowski. *Open Challenges for Data Stream Mining Research.* SIGKDD Explorations Newsletter, vol. 16, no. 1, pages 1–10, 2014. 2, 17

[Kuncheva 2013a] Ludmila I. Kuncheva. *Change Detection in Streaming Multivariate Data Using Likelihood Detectors.* IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 5, pages 1175–1180, 2013. 36

[Kuncheva 2013b] Ludmila I. Kuncheva. *Change Detection in Streaming Multivariate Data Using Likelihood Detectors.* IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 5, pages 1175–1180, May 2013. 39

[Lazarescu 2005] Mihai Lazarescu. *A Multi-Resolution Learning Approach to Tracking Concept Drift and Recurrent Concepts.* In 5th international workshop on Pattern Recognition in Information Systems, page 52, 2005. 4, 120

[Linial 1993] Nathan Linial, Yishay Mansour and Noam Nisan. *Constant Depth Circuits, Fourier Transform, and Learnability.* Journal of the ACM, vol. 40, no. 3, pages 607–620, 1993. 127

[Maurer 2009] Andreas Maurer and Massimiliano Pontil. *Empirical Bernstein Bounds and Sample-Variance Penalization.* In The 22nd Conference on Learning Theory, COLT 2009, 2009. 42

[Mnih 2008] Volodymyr Mnih, Csaba Szepesvári and Jean-Yves Audibert. *Empirical Bernstein Stopping.* In Proceedings of the 25th International Conference on Machine Learning, ICML '08, pages 672–679, New York, NY, USA, 2008. ACM. 42

[Morshedlou 2009] Hossein Morshedlou and Ahmad Abdollahzade Barforoush. *A New History Based Method to Handle the Recurring Concept Shifts in Data Streams.* vol. 3, no. 10, pages 895–900, 2009. 119

[Mouss 2004] H. Mouss, D. Mouss, N. Mouss and L. Sefouhi. *Test of Page-Hinckley, an approach for fault detection in an agro-alimentary production system.* In 5th Asian Control Conference, 2004, volume 2, pages 815–818, Melbourne, Victoria, Australia, 2004. IEEE. 5

[Narum 2006] Shawn R. Narum. *Beyond Bonferroni: Less conservative analyses for conservation genetics.* Conservation Genetics, vol. 7, no. 5, pages 783–787, 2006. 50

[Nishida 2007] Kyosuke Nishida and Koichiro Yamauchi. *Detecting Concept Drift Using Statistical Testing.* In Proceedings of the 10th International Conference on Discovery Science, volume 4755 of *DS'07*, pages 264–269, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 36

[Page 1954] E. S. Page. *Continuous Inspection Schemes.* Biometrika, vol. 41, no. 1/2, pages 100–115, 1954. 5, 36, 39

[Pears 2014] Russel Pears, Sakthithasan Sripirakas and Yun Sing Koh. *Detecting concept change in dynamic data streams.* Machine Learning, vol. 97(3), pages 259–293, 2014. 5, 11, 120

[Ramamurthy 2007] S. Ramamurthy and R. Bhatnagar. *Tracking recurrent concept drift in streaming data using ensemble classifiers.* In 6th Inter-

national Conference on Machine Learning Applications, pages 404–409, Dec 2007. 4, 121

[Ross 2012] 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, vol. 33, no. 2, pages 191–198, 2012. 5, 38, 39

[Schlimmer 1986] Jeffrey C. Schlimmer and Jr. Richard H. Granger. *Incremental learning from noisy data.* Machine Learning, vol. 1, no. 3, pages 317–354, 1986. 121

[Sebastiao 2009] Raquel Sebastiao and João Gama. *A Study on Change Detection Methods.* In Proceedings of the 14th Portuguese Conference on Artificial Intelligence, EPIA 2009, pages 353–364, Berlin, Heidelberg, 2009. Springer-Verlag. 35

[Shivaswamy 2010] Pannagadatta K. Shivaswamy and Tony Jebara. *Empirical Bernstein Boosting.* In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), volume 9 of *JMLR Proceedings*, pages 733–740. JMLR.org, 2010. 42

[Song 2007] Xiuyao Song, Mingxi Wu, Christopher Jermaine and Sanjay Ranka. *Statistical Change Detection for Multi-dimensional Data.* In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07, pages 667–676, New York, NY, USA, 2007. ACM. 39

[Sripirakas 2013] Sakthithasan Sripirakas, 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. 11

[Sripirakas 2014a] Sakthithasan Sripirakas and Russel Pears. *Mining Recurrent Concepts in Data Streams Using the Discrete Fourier Transform.* In Ladjel Bellatreche and Mukesh K. Mohania, editeurs, Data Warehousing and Knowledge Discovery, volume 8646 of *Lecture Notes in Computer Science*, pages 439–451. Springer International Publishing, 2014. 12, 162, 163

[Sripirakas 2014b] Sakthithasan Sripirakas and Russel Pears. *Use of Ensembles of Fourier Spectra in Capturing Recurring Concepts in Data Streams.* Submitted for SIAM International Conference on Data Mining for review, 2014. 12

[Street 2001] Nick W. Street and YongSeog Kim. *A Streaming Ensemble Algorithm (SEA) for Large-scale Classification.* In Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '01, pages 3770–382, New York, NY, USA, 2001. ACM. 135, 171

[Vitter 1985] Jeffrey S. Vitter. *Random Sampling with a Reservoir.* ACM Transactions on Mathematical Software (TOMS), vol. 11, no. 1, pages 37–57, 1985. 64, 71

[Wang 2003a] Haixun Wang, Wei Fan, Philip S. Yu and Jiawei Han. *Mining concept-drifting data streams using ensemble classifiers.* In Proceedings of the 9th ACM SIGKDD, KDD '03, pages 226–235, 2003. 32

[Wang 2003b] Haixun Wang, Wei Fan, Philip S. Yu and Jiawei Han. *Mining Concept-drifting Data Streams Using Ensemble Classifiers.* In Proceedings of the Ninth ACM SIGKDD International Conference on Knowl-

edge Discovery and Data Mining, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM. 153

[Widmer 1996] Gerhard Widmer and Miroslav Kubat. *Learning in the presence of concept drift and hidden contexts.* Machine Learning, vol. 23, no. 1, pages 69–101, 1996. 121