A STAGED APPROACH TO CLASSIFICATION IN HIGH SPEED CONCEPT DRIFTING DATA STREAMS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisors

Assoc. Prof. Russel Pears

Dr. M. Asif Naeem

December 2018

By

Chamari I. Kithulgoda

School of Engineering, Computer and Mathematical Sciences

Abstract

Data stream classification task needs to address challenges of enormous volume, continuous rapid flow, and concept drift of data in the presence of limited computer resources. A successful classifier is expected to result in higher accuracy and throughput under constrained memory conditions. This thesis aims at the problem of increasing throughput without sacrificing the accuracy of the classifier in concept drifting and recurring data streams. The solution introduces a novel stage learning framework that senses the context of data to determine the level of volatility in the stream.

Two learning stages namely, Stage 1 and Stage 2 are defined in accordance with stream volatility. Stage 1 is the high volatility state where many new, previously unseen concepts appear. Stage 1 represents an intensive learning phase in the lifetime of a data stream and hence an ensemble of classifiers is used in this stage as previous research has shown that ensembles excel in such situations. In Stage 1, concepts are captured by the ensemble learner and then stored in an online repository for future use in the event that such concepts recur in the future. In contrast, Stage 2 represents a low volatility state and features a limited learning ability as it relies to a greater extent on concepts already captured in Stage 1. Our empirical results reveal that such a staged approach achieves a significant throughput advantage as a result of the deactivation of computationally expensive ensemble learner in Stage 2.

Two main versions of the stage learning paradigm, namely Staged Online Learner (SOL), and SOL with Incremental Fourier Classifier (SOL-IFC) is presented. The Stage

1 segment of both SOL and SOL-IFC is driven by a forest of Hoeffding decision trees, each of which has its own drift detector. The Stage 2 learning segment of SOL is carried out by Fourier spectral representations of learned decision trees and a derivative in the form of a single decision tree. Although Stage 2 processing overhead of SOL is much smaller than in Stage 1, peaks in processing time occur at concept drifts have to be prevented. Consequently, an extended version of SOL named as SOL-IFC introduces an innovative incrementally adaptive Fourier spectrum arrangement for Stage 2. The IFC comprises of novel noisy feature filtering and instance synopsis generation mechanisms that contribute to further performance enhancements. As per empirical evidence, SOL-IFC outperforms state-of-art algorithms on a combined metric of throughput and classification accuracy while avoiding the issue of peak processing times at drifts with a smoother incremental Fourier adaptation process.

The implementation independence of the staged learning approach is proven by the third version named as MLP-FC that replaces the decision tree forest with a neural network classifier during Stage 1.

Contents

List of Tables

List of Figures

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 nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of student

Acknowledgements

I am profoundly grateful to my primary supervisor Assoc. Prof. Russel Pears for supporting me wholeheartedly during this journey of PhD. His extensive knowledge, guidance, and patience motivated this study from the beginning to end. My sincere thanks also go to Dr. Asif Naeem, my secondary supervisor for his constant encouragement, and insightful comments.

My research would have been impossible without the aid and support I received from both Auckland University of Technology and the School of Engineering, Computer, and Mathematical Sciences specifically. I am grateful for the chance to work in excellent facilities, more importantly, for the immense support from my colleagues; ranging from administrative help to stimulating conversation over coffee, their contribution has made this thesis all the richer.

Looking further back, I am also highly grateful to all the teachers I have had along this journey, both in universities and schools. Their passion for their subjects and their skill in articulating it sparked a similar passion in me and their expertise has helped guide me down this intellectual journey.

The course of a PhD has many ups and downs and friends make celebrating the victories all the sweeter and moving on from the failures all the swifter. Many thanks to my friends in Auckland and back home in Sri Lanka. Your presence in my life has helped light up my doctoral experience.

Finally, a huge thanks to my parents, siblings, and extended siblings. Together, you have helped teach me the most important lessons of all: how to live a good life, how to pursue my passions, and how to look after the others in my life. Without you, this thesis simply would not exist.

Chapter 1

Introduction

1.1 Research problem

A variety of prevalent real-world data sources such as sensors, mobile phones, social networks, stock markets, automated teller machines, telephone communication networks, and the World Wide Web generate streams of data ceaselessly. As a result of the booming Internet of Things (IoT), the number of such data sources will grow in the future. These enormous, continuous, and rapid data flows are called data streams. In order to operate on these data streams, meaningful information extraction is essential. The process of pursuing knowledge from data streams in real time is known as data stream mining.

Data stream mining has been extensively researched over the last two decades due to its increasing importance in mining data from real-world applications. Stream mining can be categorised as clustering, classification, frequent itemset mining, and forecasting (Aggarwal, 2007a; Gama, 2010; Kholghi, Hassanzadeh & Keyvanpour, 2010). Among these, the major focus of this thesis is the task of data stream classification. Solving the data stream classification problem, that is identifying an outcome based on the

given input, is of high interest as timely decision making is paramount in many real-world applications. Identifying the credit risk level of loan applicants, identifying spam emails, recognising malware, sentiment analysis, predicting the level of electricity demand, weather prediction, and recognising stock market fluctuations represent a few examples for such applications. For its criticality, classification has been one of the most frequently discussed topics in the data stream mining literature (Lemaire, Salperwyck & Bondu, 2015; Nguyen, Woon & Ng, 2015).

Different from traditional data mining where the datasets are static, relatively small in volume, and available in their entirety, stream mining is challenging due to the inherently challenging nature of stream data. The "store and then process" approach is not practical because of the unbounded nature. In addition, multiple scans per record are not possible as it is necessary to cope with the quick arrival rate. On account of its volume and speed, stream mining requires a single-pass, learning and then discard strategy.

Furthermore, the underlying distribution of streaming data is not guaranteed to remain the same over time, thus introducing a further complication. This property of being non-stationary will hereafter be referred to as concept drift in this work. Concept drift is the property that is mainly responsible for distinguishing data stream mining from mining of fixed data size data repositories. In real-world scenarios, the reasons behind concept drifts are unforeseen, and neither the frequency nor the exact time of their occurrence is certain. As a result, research in the field of data stream mining has challenged conventional thinking and forced the research community to extend solutions that were developed for environments where the statistical properties of data remain static over time. In other words, models produced by stream classifiers need to evolve with time, in response to the drifts in the data distribution.

The surveys and reviews done in Domingos and Hulten (2001); Gaber, Zaslavsky and Krishnaswamy (2005); Aggarwal (2007b); Gama (2010), and Kholghi et al. (2010)

underscored the significance of data stream mining, recognised the aforementioned challenges and recommended evolving knowledge representations, which are feasible under strict time and space constraints for learning in data streams. According to Domingos and Hulten (2001); Lemaire et al. (2015), and Bifet, Read, Holmes and Pfahringer (2018), a data stream learning algorithm is needed to adhere to certain qualities such as:

- Process instances in real time
- Learn in a single inspect operation
- Operate under a limited amount of memory
- Adapt the models produced by them in the presence of concept drift
- Produce predictions to be used at any time in the progression of the stream

The overall implication of the aforementioned requirements is that a data stream classifier needs to be capable of providing higher accuracy in real time while consuming a limited amount of memory (Aggarwal, 2007b; Gama & Kosina, 2009; Kholghi et al., 2010). The accuracy of a stream mining solution is reliant on the classifier's ability to capture knowledge from data, perform timely drift detection, possess robustness to noise, differentiate drift from noise, and the quality of the consequent model adaptation (Gama, Žliobaitė, Bifet, Pechenizkiy & Bouchachia, 2014). Time spent on classification is dependent on the model building time, the duration for the model to update in response to drift, and the time to process input towards the relevant category. The quicker the learning, the higher the data instance processing rate, which is called throughput. In general, memory consumption is the space consumed for processing and holding models. As distributed systems that generate distributed data streams exist in settings of limited memory and battery power, the work of Kholghi et al. (2010); Aggarwal (2007b), and Kargupta and Park (2004) emphasised the value of decreasing the memory consumed by classifiers, the unit computational cost of processing data, and the cost of model

update operations.

With the objective of meeting these requirements, researchers have developed one-pass, memory bounded, evolving classifiers which are distinct from traditional data mining algorithms such as C4.5 (Quinlan, 1993) and SLIQ (Mehta, Agrawal & Rissanen, 1996) which require to have an entire dataset resident in memory, as well as multiple access to data instances.

Such evolving classifiers use single incremental approaches, such as decision tree (Hulten, Spencer & Domingos, 2001; Gama, Fernandes & Rocha, 2006; Hoeglinger, Pears & Koh, 2009; Yang & Fong, 2011) , Naive Bayes (Oza, 2005; J. Gomes, Menasalvas & Sousa, 2010; Alippi, Boracchi & Roveri, 2013), rule-based system (Widmer & Kubat, 1996; Ferrer-Troyano, Aguilar-Ruiz & Santos, 2006), neural network (Gomide, 2009), and support vector machine (Domeniconi & Gunopulos, 2001; Zheng, Shen, Fan & Zhao, 2013). Alternatively, groups of models can be produced, as exemplified by an ensemble of classifiers that reach a final classification decision by combining individual decisions according to a well defined group decision policy (Dietterich, 2000). Some examples are AWE (H. Wang, Fan, Yu & Han, 2003), CBDT (Hoeglinger et al., 2009), LeveragingBag (Bifet, Holmes & Pfahringer, 2010), EP (Sakthithasan, Pears, Bifet & Pfahringer, 2015), and ARF (H. Gomes, Bifet et al., 2017). Importantly, H. Wang et al. (2003); Kuncheva (2004); Elwell and Polikar (2009), and Zliobaite (2010) have shown that ensembles are the better choice for concept drifting streams, in contrast to single model classifiers.

Meanwhile, researchers who focused on the challenge of concept drift have been working on change detection techniques (Tsymbal, 2004; Gama et al., 2014; Khamassi, Mouchaweh, Hammami & Ghedira, 2018). A number of widely used methods have emerged: sequential analysis based Page-Hinkley test (Page, 1954; Mouss, Mouss, Mouss & Sefouhi, 2004), statistical process control related EWMA (Ross, Adams,

Tasoulis & Hand, 2012), time-window based DDM (Gama, Medas, Castillo & Rodrigues, 2004), EDDM (Baena-Garcia et al., 2006), ADWIN (Bifet & Gavalda, 2007) and SeqDrift (Pears, Sakthithasan & Koh, 2014). These drift detectors trigger the drift points when the difference between two data distributions is deemed to be statistically significant. As a result, a classifier could adapt itself to the current distribution when it is informed by the signal of drift detection strategy rather than change the model at a constant rate despite the presence or absence of drift in data. Such stream data sensitive concept drift adaptation techniques are known by different terms: informed adaptation or explicit detection (Gama et al., 2014), and active approach (Ditzler, Roveri, Alippi & Polikar, 2015). This thesis uses the term explicit drift detection henceforth.

Moreover, drifts can be categorised into two main types: abrupt, when the new concept suddenly replaces the old one, and gradual, where the old concept is slowly converted into a new one (Tsymbal, 2004; Khamassi et al., 2018). The situation where a concept observed once is likely to happen again is referred to as a recurring concept and has also been considered as a type of concept drift in previous work (Gama et al., 2014). This suggests an approach of classifier systems that store previously learned concepts for reuse in the future, without the need for relearning them from scratch. Some examples of such systems are Widmer and Kubat (1996); Nishida, Yamauchi and Omori (2005); Alippi et al. (2013); Jaber, Cornuéjols and Tarroux (2013a), and Sakthithasan et al. (2015).

In this setting, cutting-edge studies (Bifet, Holmes & Pfahringer, 2010; Bifet, Frank, Holmes & Pfahringer, 2010; Sakthithasan et al., 2015; H. Gomes, Bifet et al., 2017) have suggested incrementally adaptive classifier ensembles operating under the control of a concept change detection mechanism. According to Ditzler et al. (2015), a merger of incremental learners with explicit drift detectors enhances a stream learner's ability to cope with any kind of time changing data stream. In addition to learners and drift detectors, research done in J. Gomes et al. (2010); Sripirakas and Pears (2014), and

Sakthithasan et al. (2015) take the advantage of concept recurrences through storage of already learned concepts in a repository.

All these advancements in the stream mining literature were trying to fulfil the requirements listed for a decent stream mining algorithm as described above. Even though the above-listed qualities guarantee an effective classifier, building a system that simultaneously achieves all such objectives is certainly a difficult task in practice. Among many such difficulties, the critical trade-off in a successful stream classification solution is between the accuracy and efficiency in terms of both time and space (Aggarwal, 2007b).

In this background, the research carried out in this thesis concentrates on increasing the throughput of a data stream classification task without compromising the accuracy of the classifier. At the same time, reducing storage requirements is also desirable, especially in memory constrained devices such as personal devices and sensors. Even though there have been a few previous attempts at scaling up throughput while preserving accuracy, this research seeks to extend the performance boundary further by achieving a better trade-off between accuracy and throughput in relation to previous work.

1.2 Motivation for research

In the data stream mining literature, some studies have focused on increasing throughput while others have aimed at pursuing better accuracy. As a consequence of the well-known throughput and accuracy trade-off in stream data classification, the problem of optimising both accuracy and throughput has been addressed by only a few researchers (J. Gomes et al., 2010; Sakthithasan et al., 2015; H. Gomes, Bifet et al., 2017).

In the interest of meeting the processing speed demand of massive, continuous data flows, research has followed several different approaches: one-pass learning, load

shedding, learning through the use of synopses, learning with Fourier model repositories, and parallel data stream processing.

Avoidance of multiple pass learning was one of the viable remedies attempted in dealing with high-speed data streams initially. As claimed by Domingos and Hulten (2000) in the benchmark study of the one-pass decision tree learner VFDT, the Hoeffding tree method could learn instances in a very short time. That speed advantage enables the classifier to meet the required data read/write rate while learning without storing instances in the memory. The main shortcoming of VFDT was its stationary data distribution assumption. In contrast, later Hoeffding tree based algorithms such as CVFTD (Hulten et al., 2001), VFDTc(Gama et al., 2006), VFDTt (T. Wang, Li, Hu, Yan & Chen, 2007), and optimised VFDT (Yang & Fong, 2011) have been designed to work in evolving data stream environments. However, having a single incremental classifier was not a strong solution for non-stationary environments where concepts keep evolving over time.

Reducing the workload via a load shedding strategy has been practiced in Babcock, Datar and Motwani (2004); Tatbul, Cetintemel, Zdonik, Cherniack and Stonebraker (2003); Chi, Yu, Wang and Muntz (2005); Babcock, Datar and Motwani (2007), and Ning, Wang, Shu and Yeh (2016). With the aim of lesser workload, those studies discarded subsets of arrival data in the stream. Unknown dataset size and unawareness of the influence of dropped data can be cited as potential drawbacks of the load shedding (Gaber et al., 2005) approach.

Synopses of data are obtained from a data stream by applying a variety of data summarisation techniques such as wavelets and sketches. Such structures have been used as a method for managing the enormous data volume and the speed of data streams. According to the surveys of Aggarwal and Yu (2007), and Cormode, Garofalakis, Haas and Jermaine (2012), synopses are mostly applied to query processing of stream

data, space-constrained distributed data stream applications, and frequent itemset problems. As identified by Gaber et al. (2005) accuracy is one of the major issues of data summarisation in streams.

The challenge of meeting decent accuracy in difficult concept drifting scenarios has been addressed by strategies such as having a group of incremental classifiers in the place of a single classifier, prompt drift detection technique that is robust to noise and followed by model adaptation at drift, learning from a repository of previously learned concepts, and noise robust Fourier classifiers.

According to Dietterich (2000), an ensemble of classifiers is more accurate than any of its member classifiers for statistical, computational and representational reasons. In addition, ensembles are considered as more stable solutions for concept drifting data (H. Wang et al., 2003; Aggarwal, 2007b; Scholz & Klinkenberg, 2007; Kolter & Maloof, 2007). For these causes, an ensemble of classifiers has become a popular choice for non-stationary data stream mining as exemplified by surveys of H. Gomes, Barddal, Enembreck and Bifet (2017), and Krawczyk, Minku, Gama, Stefanowski and Wozniak (2017). Unfortunately, these accuracy and stability advantages come at a price of greater computational overhead since statistics for every classifier in an ensemble need to be updated on a per-instance basis.

Furthermore, classifiers such as OzaBagADWIN (Bifet, Holmes, Pfahringer, Kirkby & Gavaldà, 2009), LimAttClassifier (Bifet, Frank et al., 2010), and LeveragingBag (Bifet, Holmes & Pfahringer, 2010) accompany ADWIN (Bifet & Gavalda, 2007) change detector with group of incremental Hoeffding trees. These solutions report better accuracies while coping with drifts effectively. However, almost all these ensemble learners are recommended for applications where processing speed and memory consumption are not priorities.

The work discussed so far have either achieved throughput at the cost of accuracy or accuracy at the cost of throughput. As both throughput and accuracy are crucial

aspects of data stream mining, a few studies have sought to achieve both requirements simultaneously through strategies such as the maintenance of a repository for learned models, optimised Fourier representation of stored models, and parallel processing.

The stream learning framework proposed in J. Gomes et al. (2010) highlights the importance of storing learned models as a response to the likelihood of concepts recurrence. Further to an ensemble of classifiers and the drift detection mechanism termed as DDM (Gama et al., 2004), a repository of learned models were used in J. Gomes et al. (2010). Results showed that using model repositories reduced the effort demanded for drift adaptation when compared to the scenarios that learn from scratch. Study of Sakthithasan et al. (2015) also focused on the recurring concepts and therefore maintained a repository of learned models other than the ensemble of trees accompanied by drift detector instances of SeqDrift (Pears et al., 2014). By contrast with J. Gomes et al. (2010), Sakthithasan et al. (2015) stored concepts in the form of Fourier spectra by applying discrete Fourier transform on decision trees as described in Park (2001), and Sripirakas and Pears (2014). Proving the effectiveness of recurrence capture and the advantage of noise robust Fourier models, Sakthithasan et al. (2015) claimed better accuracy and competitive processing speed in stream classification.

A recent study of H. Gomes, Bifet et al. (2017) implemented a parallel processing capability on top of the extended random forest solution integrated with drift detector ADWIN. Hence, trees in the forest can be operated in separate threads independently. In their comparative study of serial and parallel versions of algorithms, a parallel implementation achieved 3 times faster performance than the serial version while maintaining accuracy.

What is striking in all previous data stream classification approaches is the maintenance of computationally expensive incremental learners throughout the lifetime of a data stream. Upon the arrival of each instance, incremental ensemble approaches are needed to process that instance through all individual members and update the sufficient

statistics stored in each classifier. While this approach is effective from the viewpoint of accuracy, it might not be the most efficient solution in terms of computing resource utilisation.

Having access to previously seen concepts that either reappear in exactly the same form or with slight deviations, a continuous learning strategy may be unnecessary and wasteful of system resources. This suggests approaches that temporarily suspend their continuous learning strategy in stream segments when recurrences of past concepts manifest in the stream. Some examples of such systems are Ramamurthy and Bhatn (2007); Gama and Kosina (2009), and Pears et al. (2014), although none of them are sensitive to the rate of appearance of new concepts within the stream. These approaches improve efficiency by removing the need for concept re-learning, but they still maintain expensive ensemble learners, nevertheless. Thus none of these systems have taken optimum advantage of periods with a low rate of appearance of previously unseen concepts.

Recognising that maintaining an expensive, resource-consuming ensemble of classifiers in low volatility stream segments is inherently an inefficient approach, the major motivation of this study is to initiate a novel framework of staged learning. In staged learning, each learning stage is determined in accordance with the context. More specifically, the context that is characterised by the rate of appearance of previously unseen concepts. In light of this, the following research objectives were formulated.

1.3 Research objectives

This research is intended for a data stream classification solution that increases throughput without compromising on accuracy. As indicated previously in section 1.2, the solution is based on a new staged learning framework driven by the rate of appearance of previously unseen concepts. With that in mind, a solution is accomplished by achieving

the four objectives listed below:

1. Objective 1: To design a context sensitive staged learning framework.

   In this study, being context sensitive means the ability to differentiate between segments that exhibit concept recurrences, from those that do not. A contiguous collection of stream segments that contain a high degree of new concept occurrence is termed Stage 1. On the other hand, a collection of segments that embed a high rate of reappearance of concepts, as opposed to new concepts, is termed Stage 2. On this account, it is of critical importance to establish a theoretical perspective and relevant operational measures that define the transition between the stages.

2. Objective 2: To implement and evaluate the framework.

   The framework utilises the Ensemble Pool (EP) classifier produced in Sakthithasan et al. (2015) which in turn utilised the CBDT classifier discussed in Hoeglinger et al. (2009). In EP, SeqDrift (Pears et al., 2014) drift detection instances monitor each classifier for drifts. Once the current best performing classifier signals a drift, the tree is converted into a Fourier model (Park, 2001) and stored in the repository for future use (Sripirakas & Pears, 2014).

   The feasibility of the implemented novel staged learning framework needs to be evaluated against the throughput, memory usage, and accuracy measures over several different datasets. The new design needs to outperform or be competitive with comparative alternatives.

3. Objective 3: To design, implement and evaluate an advanced version of the framework.

   The initial design is upgraded to an optimal design which aims at further performance improvements. The implementation of this advanced design uses an

adaptive version of a Fourier classifier developed for the initial implementation. This optimum version is compared against state-of-art algorithms for two-fold measures of throughput and accuracy.

4. Objective 4: To identify the generalisability of the framework.

   Finally, the generalisability of the staged learning framework is verified. Implementation independence of the type of classifier used for learning in the proposed framework needs to be investigated. Basically, the ability of generating Fourier model through base classifiers other than decision trees needs to be explored.

1.4 Overview of research solution

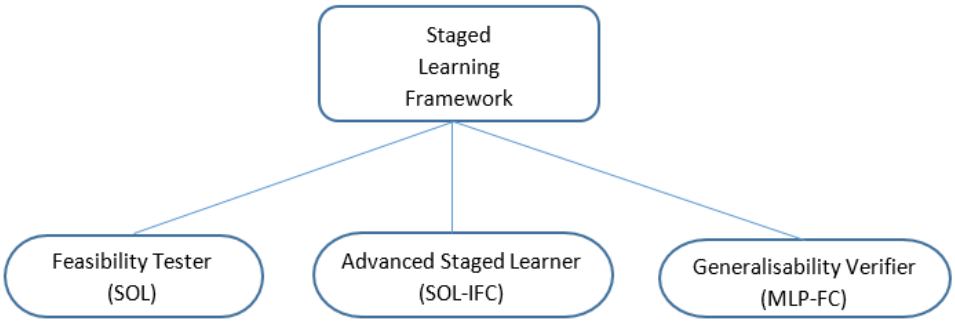


Figure 1.1: Overview of research solution

The overview of the complete solution is outlined in Fig. 1.1 which comprises the proposed framework, and three corresponding products: the framework's feasibility tester, its advanced version, and the generalisability verifier.

The core of Fig. 1.1 is the architecture of the framework which consists of components that include the incremental learner, model repository and drift detection technique. These inter-related components contribute significantly to the stream classification process as per the context of the stream. Chapter 3 presents the framework named staged learning approach.

The feasibility tester product represents the initial implementation of the proposed framework. This is supposed to confirm the appropriateness of the staged learning framework, and its ability to gain a significant throughput improvement whilst not reducing accuracy. This initial product that is named as Staged Online Learner (SOL) is evaluated in chapter 4.

The enhanced version of the staged learning framework is represented by the advanced staged learner. While enabling smoother functionality compared to its initial product, further performance optimisations are ensured by the use of an innovative incremental Fourier classifier. Theories and design concepts behind this advanced learner are discussed in chapter 5. The product termed as Staged Online Learner with Incremental Fourier Classifier (SOL-IFC) is assessed in chapter 6.

The last product is the proof of concept which illustrates the implementation independence of the staged learning paradigm. Decision tree forest is replaced by an artificial neural network and the pattern gathered from the neural network is converted into a Fourier model in a creative method. This version of the product is identified as Multi-Layer Perceptron based Fourier Classifier (MLP-FC). Chapter 7 introduces the design and experiments the product MLP-FC.

1.5 Research contributions

1. Kithulgoda, C.I., Pears, R., & Naeem, M.A. (2018). The Incremental Fourier Classifier: Leveraging the Discrete Fourier Transform for Classifying High Speed Data Streams. Journal of Expert Systems with Applications, 97, 1– 17.
   Chapter 5 and 6 of this thesis are based on this publication.

2. Kithulgoda, C. I. & Pears, R. (2016, July). Staged Online Learning: A New Approach to Classification in High Speed Data Streams. Paper presented at

the IEEE 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada.

Chapter 3 and 4 of this thesis are based on this publication.

3. Kithulgoda, C. I. & Pears, R. (2019). A Context Sensitive Framework for Mining Concept Drifting Data Streams. In E. Lughofer & M.S. Mouchaweh (Eds.), Predictive Maintenance. (Accepted)

   Chapter 2, 3 and 4 of this thesis are based on this publication.

4. Kithulgoda, C. I. (2015, August). Data Stream System Classification Framework for Concept Recurring Situation. Abstract presented at the Postgraduate Research Symposium of Auckland University of Technology, Auckland, New Zealand.

   Chapter 1 is based on this publication.

1.6 Thesis structure

Chapter 2 describes the background information relevant to this research, including classification, concept drift, Discrete Fourier Transformation, and other relevant literature. The novel staged learning framework is introduced in chapter 3 while the subsequent chapter (chapter 4) evaluates its feasibility and effectiveness in terms of accuracy, throughput, and memory. In chapter 5, the theories, definitions, and basic concepts of enhanced staged learning system that consists of innovative incremental Fourier classifier is discussed. Experimental evaluation of the enhanced staged learner against state-of-art classifiers is given in chapter 6. Chapter 7 discusses a proof of concept whereby the feasibility of extending base classifiers from Heoffding trees to neural network classifiers is tested. The thesis concludes in chapter 8 by discussing research achievements, limitations, and future directions.

Chapter 2

Background

2.1 Introduction

This chapter summarises the background knowledge related to the research presented in this thesis. The discussion starts with a formal definition of the data classification problem. Then, the crucial challenge of the data stream classification problem is explored. The challenge, in summary, is to determine the actions which should be taken by a classifier when concept drift renders the model generated by the classifier redundant over time as a result of changes that take place in the underlying data distribution.

In order to address major issues caused by the potentially enormous, and continuous flow of data that arrives at a high speed and with concept drift, researchers have taken various approaches. These strategies are categorised and presented based on their characteristics.

Since this research utilises decision trees and Fourier spectra as classifiers, along with drift detectors, background knowledge with regard to the Discrete Fourier Transform (DFT) is provided. The foundations of the DFT, an explanation on the Fourier spectrum generation from decision tree and details related to the effective maintenance of the repository of Fourier spectra are discussed.

2.2 Data stream classification

In its literal meaning classification is the task of deciding the category of objects under consideration. The data classification problem in data mining related domains refers to the process of extracting a representative model (function) which is capable of differentiating data into categories (Han, Kamber & Pei, 2012). In formal terms, given a set of instances of the form $(\vec{X}, y)$, the classification problem constructs a model $y = f(\vec{X})$, where $\vec{X} = X_1, ..., X_k$ is a vector of attribute values and y is a discrete class from a set of C different classes (Bifet et al., 2018). This model can be built in various forms such as decision trees, rule-based methods, and neural networks.

The data classification process has two basic steps to be followed in order: (1) model building and (2) model testing (Gaber, Zaslavsky & Krishnaswamy, 2007; Lemaire et al., 2015). The model building phase is the period where it is being learned by taking a representative sample of available data. In traditional data mining, the model building act is done in several inspects over the stored data. Thereafter, the built model $y = f(\vec{X})$ takes the responsibility for deciding the category of objects arriving with unknown class labels in the test phase.

In the data stream environment which demands a time and memory efficient solution to deal with high speed, continuous and enormous flow of data, the aforementioned " store and process" approach is not acceptable. Hence, one-pass, incrementally learning algorithms are needed in place of the several passes, batch learning approach. Further to that, and more importantly, data in data streams is not stationary which renders a given classifier redundant over time. This dictates that models be synchronised with changes that occur periodically in the stream.

The realisation that concept drift detection plays a central role in data stream mining sparked off a flurry of research in this area. In order to get an in-depth understanding of concept drifting data streams, an understanding of the formal definition and forms of

concept drift is important.

2.3 Concept drift

In essence, the change in the relationship between an outcome variable and its observed features is called a concept drift. In real-world scenarios, the reasons behind these changes are unforeseen and neither the frequency nor the exact time of occurrence is certain.

In formal terms, Gama et al. (2014) defines concept drift as the dissimilarity of the joint probability distribution of input features and class label at two subsequent time points t_0 and t_1. The definition given by that study is adopted here:

$$\exists X : p_{t_0}(\vec{X}, y) \neq p_{t_1}(\vec{X}, y) \tag{2.1}$$

This dissimilarity of the likelihood of events $\vec{X}$ and y occurring can be caused by changes in components ((Kelly, Hand & Adams, 1999; Gao, Fan, Han & Yu, 2007) as cited by Gama et al. (2014)), namely the prior probabilities $p(y)$ of classes or the conditional probabilities of classes, that is $p(\vec{X}|y)$. These changes result in a change in the posterior probability $p(y|\vec{X})$. This change in $p(y|\vec{X})$ over time is the reason behind accuracy fluctuations.

Accordingly, any solution should have the capability of sensing the changes in $p(y|\vec{X})$ throughout the lifespan of the data stream in a time efficient manner in order to maintain classification accuracy. This is called the concept drift detection problem which has been widely studied (Baena-Garcia et al., 2006; Bifet & Gavalda, 2007; Ross et al., 2012; Pears et al., 2014). Another classification of concept drift is the speed with which it occurs. A sudden deviation in the $p(y|\vec{X})$ value is said to be an abrupt change whereas deviations that occur in an incremental and cumulative manner over time are

said to be gradual (Tsymbal, 2004; Khamassi et al., 2018). Abrupt changes require a fast response to the change in order to preserve accuracy. This, in turn, requires an alternative model that is better suited to the new concept be deployed as soon as the drift is detected. In order to achieve this goal, a pool of learners needs to be available at any given point in time. Deployment can be done through a switch from learner L_1 to another learner L_2 which may be better suited to the new concept whenever a drift is detected. If no alternative learner is available that matches to the new concept, then accuracy will be severely compromised until one or more of the learners adjust to the new concept.

On the other hand, gradual changes allow time for the system to adjust to the change and individual learners may have adapted sufficiently well to cope with the new concepts. Thus, in general, their effects may not be as severe as with abrupt drift.

Yet another categorisation of drift is whether the drift pattern reappears over a period of time. Such recurrences may follow a periodic pattern to a greater or lesser degree or be aperiodic and completely unpredictable in its recurrence pattern. In either case, the action that needs to be performed at detection time is a switch to a new learner, just as with the case of abrupt drift. However, unlike with the case of abrupt drift, if the recurrence pattern is strong, i.e. repeated appearances have statistical properties very similar to each other, then it could be profitable to store such concepts separately in an online repository which is separate from the pool of learners that adapt their models over time. The use of the repository will guarantee that models associated with recurring concepts are preserved in their original form in between successive recurrences. However, they may be subject to change at the next appearance and a new version of the recurring concept may then be stored in its place in the repository.

In summary, having an explicit drift detection technique, a set of learners rather than a single learner, and a repository of stored concepts can be considered as significant enhancements for coping with different types of concept drifts productively. These

findings have enriched the data stream classification literature and research is ongoing with a variety of different classifier designs.

2.4 Data stream classification strategies

As described in sections 2.2, the high speed, continuous nature of a data stream dictates the importance of time efficiency and memory management while concept drift implies the necessity for a time-evolving classifier. Further to this, embedding drift detectors, strategies of using a group of learners, and maintaining a model repository are recommended in section 2.3. This section reviews previously suggested solutions for meeting such data stream classification challenges. These solution strategies are grouped into four categories as per their characteristics.

2.4.1 Non ensemble incremental approaches

The data stream classification problem in concept drifting streams through incremental algorithms has been researched by many scholars. In one of the pioneering data stream studies, the Very Fast Decision Tree (VFDT) (Domingos & Hulten, 2000) introduced an incrementally learning Hoeffding decision tree classifier. VFDT differs from traditional decision tree learners such as ID3 (Quinlan, 1986), and C4.5 (Quinlan, 1993), as it is not necessary to keep training examples in main memory. Attributes were selected as decision nodes of the tree after observing a statistically sufficient number of instances as determined by the Hoeffding bound (Hoeffding, 1963). Importantly, the Hoeffding bound is independent from the distribution of observations. The Hoffeding bound selects a particular attribute as the best attribute to make the split after observing n instances and that decision was the same as observing an infinite number of instances with a given level of confidence $(1 - \delta)$, where δ is the level of uncertainty.

Figure 2.1 exemplifies the incremental learning process of a Hoeffding tree over a

given period. The classification task illustrated in this example categorises incoming instances with attributes $X_1, X_2, \ldots, X_n$ into two groups: Yes or No. Tree splits are determined by the Hoeffding tree algorithm proposed in Domingos and Hulten (2000). Given the difference between two highest split evaluation function values $G(.)$s is greater than the ϵ, a leaf node is split on the attribute with maximum $G(.)$. The ϵ is defined by $\sqrt{(R^2 ln(1/\delta))/2n}$, where n is the number of observation, $(1-\delta)$ is the confidence and R is 1. In this example, splits on X_1, X_3 and X_n occur respectively within the interval of observation.

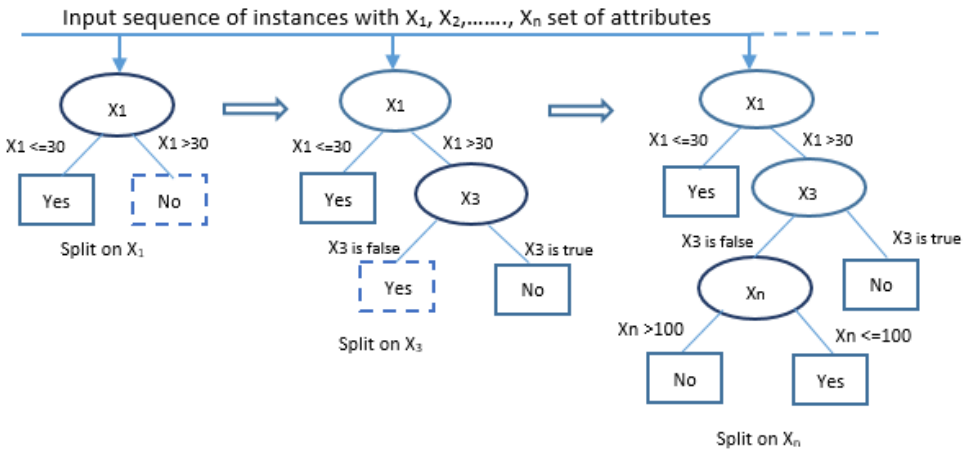


Figure 2.1: Incremental Decision tree

Even though the one-pass learning capability and classification speed of VFDT fit with the requirements of speed and continuous nature of data streams compared to conventional learners, the stationary nature of data assumed by VFDT was considered as a drawback. Their concept change sensitive extended study of Concept-adapting Very Fast Decision Tree (CVFDT) (Hulten et al., 2001) examined the suitability of the previous decision tree splits through continuous up-to-date Hoeffding tests. Whenever a split fails its validity test, an alternative tree is grown with the current optimum attribute, ensuring the validity of tree in a concept drifting environment.

The VFDTc (Gama et al., 2006) extended VFDT by proposing three additions: competency of handling numeric attributes, use of Naïve Bayes classifier which considers conditional class probability at tree leaves, and being drift sensitive through repeated comparison of two class-distribution examples. In contrast to VFDTc, VFDT used only the prior probability of classes at leaf nodes. It has been claimed that VFDTc is better even with small datasets due to its enhanced split frequency compared to conservative VFDT. A further extension, in the form of VFDTt (T. Wang et al., 2007), was based on both VFDT and VFDTc to improve the processing time.

Another variant of VFDT is optimised VFDT (Yang & Fong, 2011) that finds the best number of splits by an adaptive tie mechanism compared to VFDT's user-defined tie-breaking threshold. Furthermore, the classifier $\text{CVFDT}_{\text{NBC}}$ (Nishimura, Terabe, Hashimoto & Mihara, 2008) incorporated Naive Bayes classifiers at leaf nodes of well-known CVFDT in order to achieve a higher accuracy.

Hoeffding Option Tree (HOT) presented in Pfahringer, Holmes and Kirkby (2007) is another variant to the standard Hoeffding tree. In HOT, there are option nodes which allow instances to go along multiple paths and hence end up at multiple leaves. The class outcome of a particular instance is determined by the sum of individual leaf node probabilities. Tree growth has been restricted by defining an upper limit for the number of options a node can provide, and not allowing an attribute to split on a given node more than once. Tree pruning was also considered as it was necessary to deal with the memory consumption, yet concluded as not being effective.

The study of this thesis adopts an incremental decision tree learner Concept Based Decision Tree (CBDT) described in Hoeglinger et al. (2009). This is also a variation of Hoeffding tree studied in CVFDT (Hulten et al., 2001). Differently, in CBDT tree, each node maintains a counter C which shows the total number of instances seen by the time. In the case of a split, C value of the new decision node is assigned by the root node's C in an effort to indicate the relative age of this split in future.

Similar to VFDT and CVFDT, sufficient statistics are given in a 3-dimensional array $S_{i,j,k}$ where i is the class, j is the attribute and k is the value of that attribute. In addition, those C values are taken into consideration by CBDT when weighing the information gain of a subtree. The subtree with the minimum sum of weighted information gain across all leaf nodes is pruned as a method of reversing splits for the avoidance of overfitting.

In addition to decision trees, there are several other incremental classifiers namely, incremental support vector machine, incremental K-nearest neighbour classifier, rule learners, Naive Bayes, and regression trees. The incremental support vector machines were experimented in Syed, Liu and Sung (1999), Domeniconi and Gunopulos (2001), and Zheng et al. (2013). Studies such as STAGGER (Schlimmer & Granger, 1986), FLORA (Widmer & Kubat, 1996), AQ-11PM (Maloof & Michalski, 2004), and Ferrer Troyano, Aguilar-Ruiz and Riquelme (2005) used incremental rule learners. Naive Bayes algorithm has been taken as an incremental learner in several studies such as Oza (2005), J. Gomes et al. (2010) and Alippi et al. (2013). The study of Ikonomovska, Gama and Džeroski (2011) describes regression model trees.

2.4.2 Incremental ensemble classifiers

Another significant advancement in stream learning was the use of multiple classifiers in the place of a single decision model. Several studies including H. Wang et al. (2003); Kuncheva (2004); Hoeglinger et al. (2009); Elwell and Polikar (2009); Bifet et al. (2009), and Zliobaite (2010) recognised and empirically verified, that a group of classifiers, also known as ensemble learning (Dietterich, 2000) is better at addressing challenges in concept drifting environments. In the ensemble learning literature, researchers have used several different approaches to cope with concept drift, diverse decision making strategies to decide the class outcome for an instance, a variety of base classifiers,

various diversity measures, and different numbers of base classifiers.

Coping with concept drift can be done implicitly where the model is being updated continuously, or explicitly by feeding the binary classification outcome, "incorrect" or "correct" into the change detector units. Implicit drift detection approaches were used by CBDT (Hoeglinger et al., 2009), ASHT (Bifet & Gavaldà, 2009), AWE (H. Wang et al., 2003), AUE (Brzeziński & Stefanowski, 2011), AD (Jaber et al., 2013a). In those studies, models in the ensemble were updated after observing instances over fixed size windows or sliding windows that discard an instance at a time. Explicit drift detectors such as ADWIN (Bifet & Gavalda, 2007), SeqDrift (Pears et al., 2014) or Page-Hinkley test (Page, 1954) were embedded into classifiers in OzaBagADWIN (Bifet et al., 2009), LeveragingBag (Bifet, Holmes & Pfahringer, 2010), LimAttClassifier (Bifet, Frank et al., 2010), SOL (Kithulgoda & Pears, 2016), ARF (H. Gomes, Bifet et al., 2017), SOL-IFC (Kithulgoda, Pears & Naeem, 2018), and Ikonomovska et al. (2011). In either case, changes were done on ensemble in response to drifts via one or few of strategies among classifier removal, classifier insertion, model resets, or model statistic resets. Meanwhile some ensemble solutions such as H. Wang et al. (2003), Zhang, Zhu, Tan and Guo (2010) and Brzeziński and Stefanowski (2011) adjusted weights of individual ensemble members as the method of drift adaptation.

Among a variety of strategies to decide overall class label from a group of learners, three main trends can be found in the literature. One such trend is to assume homogeneity within a concept, and hence appoint the most accurate classifier (Hoeglinger et al., 2009; Sakthithasan et al., 2015; Kithulgoda et al., 2018) as the solo decision maker for that particular concept based on its performance compared to others in the ensemble. This thesis also pursues the assumption of intra-concept homogeneity. Whenever a drift is detected, the classifier with maximum accuracy is selected as the best decision maker for the recently emerged concept.

The other two approaches have followed voting mechanisms; either majority voting

or weighted majority voting when deciding the overall class outcome per incoming instance. In the simple majority voting method, a class label decided by the majority of classifiers is considered as the overall outcome (Oza, 2005; Bifet, Holmes & Pfahringer, 2010). Ties are avoided by taking an odd number of models. In contrast, the class outcome of each classifier is weighted relative to the performance of that classifier which can be increased or decreased over the time in weighted majority voting technique. This weighted majority voting technique has been applied in many studies including Brzeziński and Stefanowski (2011), Scholz and Klinkenberg (2007) and Kolter and Maloof (2007). Performance might be measured over constant size or variable size windows.

Many studies in literature consist of homogeneous base learners that consist of a single classifier type (e.g. all base learners are decision trees) within an ensemble, and a few have heterogeneous base learners (e.g. some base learners are decision trees while others are Naive Bayes). A popular class of incremental ensemble learners uses Hoeffding decision tree as the basic learning mechanism. Some examples of incremental classifiers using ensembles of decision trees include CBDT (Hoeglinger et al., 2009) which is used in this thesis, OzaBagADWIN (Bifet et al., 2009), LimAttClassifier (Bifet, Frank et al., 2010), LeveragingBag (Bifet, Holmes & Pfahringer, 2010), and ARF (H. Gomes, Bifet et al., 2017) with slight variations between each other. In addition, an ensemble of incrementally learning hypotheses are studied in Learn^{++} NSE (Elwell & Polikar, 2009) and ADAIN (He, Chen, Li & Xu, 2011). Incremental Naive Bayes learner is the base learner used in Katakis, Tsoumakas and Vlahavas (2008), J. Gomes et al. (2010), and Ren, Lian and Zou (2014). The studies of Nguyen, Woon, Ng and Wan (2012) and van Rijn, Holmes, Pfahringer and Vanschoren (2015) are some examples of heterogeneous base learners.

Even though there is a sensible belief that none of the group members can individually perform better than the group itself in terms of accuracy (Dietterich, 2000), the

extent to which those individual base learners should be diversified from each other is an interesting matter which remains as an open research question. Scholars have used distinct pair-wise diversity measures as summarised in H. Gomes, Barddal et al. (2017), including Yule's Q statistic (Yule, 1900), the correlation coefficient of a given pair of classifiers for all possible outcomes, and a disagreement measure that quantifies the ratio of disagreed instances on the classification outcome by a pair of classifiers under consideration. This study follows the method based on the degree of agreement calls as distance similarity between two classifiers as described in section 2.5.1.

The cardinality, i.e. the number of base learners of an ensemble can be dynamic or constant throughout the lifetime of the system. Even though a larger number of classifiers tend to produce better overall accuracy, it comes at the expense of more memory usage and less throughput. Studies such as H. Wang et al. (2003); Bifet et al. (2009); Bifet, Holmes and Pfahringer (2010); Brzeziński and Stefanowski (2011), and Jaber et al. (2013a) maintained a fixed number of models in their ensembles, whereas Nishida et al. (2005); Ramamurthy and Bhatn (2007); Sripirakas and Pears (2014), and Sakthithasan et al. (2015) classified with a dynamic number of classifiers subject to some maximum allowable number. This thesis follows the latter option, with the use of a predefined maximum number of classifiers, as described in section 2.5.1.

2.4.3 Pairing drift detectors with stream learners

A vast amount of studies have focused on the concept drift detection problem (Pears et al., 2014); in the context of stream classification, several studies have paired explicit drift detector modules with stream learners.

For instance, OzaBagADWIN (Bifet et al., 2009), LeveragingBag (Bifet, Holmes & Pfahringer, 2010), and LimAttClassifier (Bifet, Frank et al., 2010) have associated a change detector ADWIN (Bifet & Gavalda, 2007) in each tree for the purpose of

recognising drifts by accuracy degradation. ADWIN method maintains a variable length sliding window which can be shrunk when the concept is drifted or can be grown when the concept is stabilised. In response to drift, both OzaBagADWIN and LeveragingBag algorithms remove the worst learner and add a new learner. The LimAttClassifier responds to drift by resetting its learning rate and replacing poorly performing trees by their root nodes.

The Regression model tree stream learner presented in Ikonomovska et al. (2011) embeds Page-Hinkley test (Page, 1954; Mouss et al., 2004) at each tree node. This work modifies only the affected sub-tree at drift and builds an alternate tree rooted by the node that the change is detected. The decision of replacing an old tree with an alternative tree is taken with the aid of a relative performance measure.

The study in this thesis uses drift detector SeqDrift2 (Pears et al., 2014) on account of its low false positive rate and optimised drift detection delay. Two reservoirs, namely left and right are populated with a sequence of binary 1s and 0s for incorrect and correct classifications respectively. The size of the left reservoir is adaptive in accordance with drift rate, whereas the right reservoir size is fixed. At each block size which is equal to right reservoir size, the algorithm tests whether there is a significant mean difference between left an right reservoirs in order to detect concept drifts. The SeqDrift2 adopts the reservoir sampling method suggested in Vitter (1985) as the strategy of ensuring randomness when selecting a data point to be replaced by the new value given the left reservoir is already filled. Instead of using the conservative Hoeffding bound (Hoeffding, 1963), the Bernstein Bound (Bernstein, 1946) is taken when calculating mean difference threshold ϵ for the left and right reservoirs. The detector makes use of the sample means $\hat{h}_l$, $\hat{h}_r$, and a threshold ϵ to determine if the condition for drift $(\hat{h}_r - \hat{h}_l) > \epsilon$ is satisfied. Then drift is concluded with probability $(1 - \delta)$ where δ is the drift significance level. The ϵ threshold is given by: $\frac{1}{3(1-k)n_r}(p + \sqrt{p^2 + 18\sigma_s^2 n_r p})$ where $p = \ln \frac{4}{\delta}$, $k = \frac{n_r}{n_r+n_l}$, σ_s= sample variance, n_r = size of the right reservoir, and n_l = size of the left reservoir.

In response to drift, best performing model selection, insertion of the model to the repository for reuse in future, and model removal, as required have been done.

Further to those drift detection strategies, Drift Detection Method (DDM) by Gama et al. (2004), Early Drift Detection Methodology (EDDM) by Baena-Garcia et al. (2006), and Exponentially Weighted Moving Average (EWMA) by Ross et al. (2012) can be found as statistical process control based approaches.

2.4.4 Re-use of models when concepts recur

Moreover, scholars have recognised the likelihood of concept recurrence in data streams and have suggested storing learned models for re-use rather than relearning from scratch.

One of the initial, seminal adaptive learners which took advantage of concept recurrences was FLORA3 (Widmer & Kubat, 1996). The FLORA3 algorithm stored hypotheses of stable concepts for later use, with the belief of the reappearance of that concept in future. Stability of a concept was decided after considering that concept's accuracy, the trend of accuracy, and the instance count that agrees with the hypothesis under consideration. The level of concept stability also adjusted the window size of the observations. When the system suspected a concept drift, the best matching stored hypothesis was considered as a potential hypothesis for the current concept. The study of Lazarescu (2005) also stored learned concepts in a repository and searched the repository for matching concept at concept drift. In contrast to the FLORA framework, this study had two windows, small and large, in order to deal with different types of drifts.

The popularity of ensemble online learning in concept drifting streams has also had an influence on the idea of model re-usability when concepts recur. The study done by Nishida et al. (2005) has proposed a system consisting of an online classifier that is k-nearest neighbour, many batch learners which are C4.5 decision trees (Quinlan,

1993), and drift detection mechanism. The online learner updates the currently learning hypothesis with each incoming instance and the instance is stored in a buffer until the buffer is full or concept has drifted. Given the situation where either the buffer became full or the concept drifted, the system generates a new hypothesis based on the instances stored in the buffer. When making a final classification decision, a weighted majority vote approach is applied on the set of hypotheses. Weight of a classifier was determined by its suitability measure, interpreted as a function of the accuracy over recent instances. Detecting drifts were done by comparing the suitability measure with the both upper and lower endpoints' suitability measures of $100(1 - \alpha)\%$ confidence. The degree of reuse of a classifier was determined by its weight which is a function of accuracy. Compared to that, the work of Elwell and Polikar (2009) avoided irrelevant classifiers through weights that are derived by averaging errors over time. Both of these studies generated classifiers for batches of data.

In Ramamurthy and Bhatn (2007) also, classifiers were learned from data chunks. Those were stored in a repository called global set for reuse when concepts recur. When neither a single model in the global set nor the ensemble of relevant models are capable of classifying new data chunk with an error less than the user-defined permitted error, a new classifier is built in response to the new concept. An ensemble of ID3 decision trees (Quinlan, 1986) was used as the classifier.

The practice of using two learning layers can be found in the studies of Gama and Kosina (2009) and J. Gomes et al. (2010). These studies suggested the use of two learning layers consisting of a base classifier and a corresponding meta-classifier while maintaining a pool of learned classifiers. In the work of Gama and Kosina (2009), the first layer learned from labelled instances while the second acts as a referee who keeps an account of the base learner's performance with respect to regions of feature space. Each learned classifier is stored in a pool together with its referee. When the error rate of the current learner is at the warning level, the system checks the performance of stored

models through their referees proactively. A model is reused providing its applicability is greater than some given threshold. If not, a new model and its meta-learner are learned and are added to the pool. In J. Gomes et al. (2010), the base learner learns new concepts and the meta-learner learns the context-concept relation. In addition, the meta-learner is responsible for detecting concept drifts and selecting an appropriate model from the repository if the concept recurs. Learned models are stored in the form of conceptual vectors together with its context information based on the assumption of associativity between concept and context in every recurrence.

The Just-In-Time (JIT) classifier (Alippi et al., 2013) also stored concept representations of previously seen concepts when a drift is detected. JIT continuously updates concept details by making use of supervised input data. Such details consist of three properties: a collection of the supervised data instance (x_t,y_t), a set of statistical characteristics of the concept, and statistical features which signal the change of concept, together with classification errors. When drift is detected, details collected for the recent concept is subjected to a pair-wise comparison against stored concept details. If a match is found then an enhanced set of supervised instances is created. Then a classifier is trained by using the newly created set of supervised instances. The JIT classification framework has been tested with different base learners namely, k-nearest neighbour, Naive Bayes, and Support Vector Machine. This study of Alippi et al. (2013) demonstrated the effectiveness of drift-adaptive, incremental ensemble learning in the face of concept recurrence in the stream.

The Anticipative Dynamic Adaptation to Concept Change (ADACC) approach presented in Jaber et al. (2013a) is another study that stored models for reuse in the future. Ensemble classifiers were used as the classifier with Naive Bayes as the base classifier. Every base learner continuously adapts to new incoming data unless it has been removed due to poor performance over time. The decision of storing a model is taken on the basis of the stability of the ensemble over a sequence of examples. The

stability is measured by the difference between the sum of pair-wise agreements and the sum of errors measured over the best performing half of the ensemble over a sufficient period of time. If the stability measure is higher than some predefined threshold, and sufficiently varied from the models already in memory, the model is cached for re-use in the future.

The ensemble classifier used in this thesis is initialised by a forest of decision trees (Hoeglinger et al., 2009) which learns continuously from incoming data until the forest learning is suspended, as described in chapter 3. While learning from trees is in progress, in common with Sripirakas and Pears (2014), the system transforms a tree into a Fourier spectrum whenever that tree signals a drift. This model conversion is illustrated in detail in section 2.5. Over a period of time, a collection of Fourier spectra are built up in an online repository. For the purpose of memory management and model generalisation, those spectra are aggregated as described in Sakthithasan et al. (2015). More information on repository management can be found in section 2.5.1. Spectra are reused whenever the underlying concept that gave rise to its creation reappears. The recognition of correspondence between an emerging concept and its stored version (in the form of a spectrum) is done on the basis of the best accuracy measure over a recently observed block of instances in SeqDrfit2 (Pears et al., 2014) that was described in section 2.4.3.

Accordingly, the classifier used in this study contains both an ensemble of Fourier spectra as well as a forest of Hoeffding decision trees. The model conversion from tree to Fourier spectrum via DFT and maintenance of spectra are discussed below.

2.5 The use of the Discrete Fourier Transform in classification and concept encoding

The use of DFT in data mining has been of recent origin and has been focused on deriving a Fourier spectrum from Decision trees. Firstly, a basic overview of the derivation of the multivariate DFT from a decision tree is presented. Then the setting of Fourier encoding and classification scheme is described.

Before the presentation of the mathematical foundations of the DFT, fundamental ideas underpinning the Fourier transform are mapped to their meanings in Table 2.1 in order to communicate their roles in an intuitive manner.

Table 2.1: Mapping of Fourier concepts to their intuitive meanings

Symbol	Meaning
x	A schema consists of a vector of feature values drawn from features that comprise the dataset. A schema is a compact way of defining a set of data instances, all of which share the same set of feature values.
X	The schema set which contains the set of all possible schema for a given dataset.
j	This is a partition of the feature space. Essentially, it is also a vector of feature values, just as with a schema. The only (conceptual) difference is that a schema refers to the data whereas a partition indexes a Fourier spectrum.
J	The partition set that defines the number of coefficients in the spectrum and its size.
$w_{\vec{j}}$	A coefficient in the Fourier spectrum.
$\phi^{\vec{\lambda}}_{\vec{j}}(\vec{x})$	This is the Fourier basis function that takes as input a feature vector and a partition vector and produces an integer for a dataset with binary-valued features or a complex number for a dataset with non-binary feature values.

A Fourier spectrum is derived from a Fourier basis set which consists of a set of orthogonal functions that are used to represent a discrete function. Consider the set of all d-dimensional feature vectors where the l^{th} feature can take λ_l different discrete values, $\{0, 1, \cdots, \lambda_l - 1\}$. The Fourier basis set that spans this space consists of $\prod_{l=1}^{d} \lambda_l$

basis functions. Each Fourier basis function is defined as:

$$\psi_{\vec{j}}^{\vec{\lambda}}(\vec{x}) = \frac{1}{\sqrt[d]{\prod_{l=1}^{d} \lambda_l}} \prod_{m=1}^{d} \exp\left(\frac{2\pi l x_m j_m}{\lambda_m}\right) \tag{2.2}$$

where $\vec{j}$ and $\vec{x}$ are vectors of length d; $x(m)$, $j(m)$ are the m^{th} attribute values in $\vec{x}$ and $\vec{j}$, respectively. The vector $\vec{j}$ is called a partition and its order is the number of nonzero feature values it contains.

A function f: $X^d \rightarrow \mathcal{R}$ that maps a d-dimensional discrete domain to a real-valued range can be represented using the Fourier basis functions:

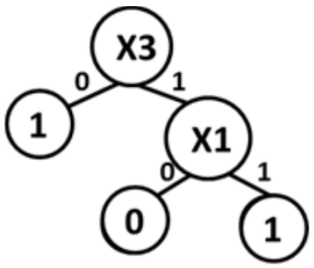
$$f(\vec{x}) = \sum_{j \in X} \overline{\psi_j}^{\vec{\lambda}}(\vec{x}) w_{\vec{j}} \tag{2.3}$$

, where $w_{\vec{j}}$ is the Fourier coefficient corresponding to the partition $\vec{j}$ and $\overline{\psi_j}^{\vec{\lambda}}(\vec{x})$ is the complex conjugate of $\psi_{\vec{j}}^{\vec{\lambda}}(\vec{x})$. Henceforth the superscript λ shall be dropped from the ψ_j function formulation to simplify the presentation. The Fourier coefficient $w_{\vec{j}}$ can be viewed as the relative contribution of the partition $\vec{j}$ to the function value of $f(x)$ and is computed from:

$$w_{\vec{j}} = \prod_{i=1}^{l} \frac{1}{\lambda_i} \sum_{x \in X} \psi_{\vec{j}}(\vec{x}) f(\vec{x}) \tag{2.4}$$

In a data mining context, $f(\vec{x})$ represents the classification outcome of a given data instance $\vec{x} \in X$. Each data $\vec{x}$ must conform to a schema and many data instances in the stream may map to the same schema. For example, in Fig. 2.2, many data instances for schema $(0, 0, 1)$ may occur at different points in the stream. Henceforth schema instances are referred rather than data instances as the Fourier classifier operates at the schema, rather than at the data instance level. Thus the notation $\vec{x}$ shall be adopted to denote a schema instance, rather than a data instance. The set X is the set of all possible schema, and for the simple example in Fig. 2.2 it is of size 8.

The absolute value of $w_{\vec{j}}$ can be used as the "significance" of the corresponding partition $\vec{j}$. If the magnitude of some $w_{\vec{j}}$ is very small compared to other coefficients, it is considered that the j^{th} partition to be insignificant and neglect its contribution. The order of a Fourier coefficient is simply the order of its corresponding partition. Terms like high order or low order coefficients will be used to refer to a set of Fourier coefficients whose orders are relatively large or small, respectively.



x_1	x_2	x_3	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

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

Figure 2.2: A Decision tree and its equivalent Fourier spectrum.

The Fourier spectrum of a Decision Tree can be computed using the class outcomes predicted by its leaf nodes. As an example, consider the decision tree in Fig. 2.2 defined on a binary valued domain consisting of 3 features. Its truth table derived from the predictions made by the tree and the corresponding Fourier spectrum that results appears in Fig. 2.2. Below the computation of j^{th} Fourier coefficient w_j is illustrated for a data with d binary valued features which is given by the Boolean domain version (Park, 2001) of Eq. 2.4:

$$w_j = \frac{1}{2^d} \sum_X \psi_j(x) f(x) \tag{2.5}$$

where f(x) is the class outcome predicted by the leaf node with path vector x and $\psi_j(x)$, the Fourier basis function given by the simplified version of the Eq. 2.2:

$$\psi_j(x) = (-1)^{(j.x)} \tag{2.6}$$

Considering three binary valued features X_1, X_2 and X_3 given in Fig. 2.2, only X_1

and X_3 are appeared in tree and hence contributed to calculation. The study of Park (2001) guaranteed that coefficients for paths which are defined by attributes need to be computed since other coefficients are zero in value. Thus coefficients $w_{010}, w_{011}, w_{110}$ and w_{111} are zero. Computation for non-zero coefficients w_{000} and w_{001} are as follows.

$$w_{000} = \frac{1}{2^3}\sum_X \psi_j(000)f(x) = \frac{1+0+1+0+1+1+1+1}{8} = 3/4$$
$$w_{001} = \frac{1}{2^3}\sum_X \psi_j(001)f(x) = \frac{1+0+1+0+1+(-1)+1+(-1)}{8} = 1/4$$

The Fourier spectrum derived from a decision tree is compact due to the two following properties:

1. The number of non-zero coefficients is polynomial in the number of features represented in the tree (Kargupta & Park, 2004).

2. The magnitude of the coefficients w_j decreases exponentially with the order of the partition j (Kargupta & Park, 2004; Kargupta, Park & Dutta, 2006).

These two properties collectively make a spectrum derived from a tree very attractive. Firstly, the tree provides a natural filtering mechanism as typically only a fraction of the features have sufficient information gain to be represented in the tree. Once the tree is in place, only the set of low order coefficients defined from partitions appearing in the tree make a significant contribution to the classification outcomes.

Kargupta and Park (2004) and Kargupta et al. (2006) made use of spectral energy to derive a cut-off point for coefficient order. Given a spectrum s, its energy E is defined by: $E = \sum_{j \epsilon J} |w_j^2|$ where J is the partition set of s. For a given energy threshold T, the subset of J (in ascending spectral order) whereby $E \geq T$ is retained; all other coefficients are deemed to be zero and removed from the array. Thus for example in the spectrum defined in Fig. 2.2, the first order coefficients contain $\frac{9+1+0+1}{9+1+0+1+0+0+1+0} = 91.7\%$

of the total energy and so with a threshold of 90%, only coefficients $w_{000}, w_{001}, and w_{100}$ should be retained, thus reducing the size of the spectrum that needs to be maintained.

Once a Fourier spectrum is derived from a decision tree, it can fully replace the latter since the classification of a newly arriving schema instance x can be computed by applying the inverse transform given in Eq. 2.3 over the set J that contains the reduced set of coefficients that survive the energy thresholding process.

Computing of the classification outcome for a given schema through the Fourier spectrum is illustrated below.

$$
\begin{aligned}
f(x) &= \sum_{j} \overline{\psi_j}(x) w_j \\
f(010) &= \sum_{j} (-1)^{(j.010)} w_j \\
f(010) &= (-1)^{(000.010)} w_{000} + (-1)^{(001.010)} w_{001} + (-1)^{(010.010)} w_{010} \\
&\quad + (-1)^{(011.010)} w_{011} + (-1)^{(100.010)} w_{100} + (-1)^{(101.010)} w_{101} \\
&\quad + (-1)^{(110.010)} w_{110} + (-1)^{(111.010)} w_{111} \\
&= \frac{3}{4} + \frac{1}{4} - \frac{1}{4} + \frac{1}{4} = \frac{7}{4} = 1
\end{aligned}
$$

2.5.1 Repository management

As spectra in the repository may accumulate in number, it will be necessary to implement a memory management strategy to ensure that memory does not overflow in the repository. A simple strategy would be to delete the oldest spectrum when memory is not available to store a newly created spectrum. Instead, this thesis aggregates newly created spectra with existing spectra as a memory saving measure.

Aggregation of spectra (Sakthithasan et al., 2015) was implemented via a pair-wise algebraic summation of the spectra involved as given in Eq. 2.7:

$$
\begin{aligned}
S_c(x) &= \sum_k A_k \sum_j S_k(x) \\
&= \sum_k A_k \sum_{k \epsilon Q_k} {\omega_j}^{(k)} \overline{\psi}_{j(x)}
\end{aligned} \tag{2.7}
$$

where $S_c(x)$ denotes the aggregated spectrum produced from the individual spectra $S_k(x)$ produced at different points k in the stream; A_k is the classification accuracy of its corresponding spectrum and Q_k is the set of partitions for non zero coefficients in spectrum S_k.

Before proceeding with spectral aggregation, the implementation done in this study checks whether the new spectrum is significantly different from already existing spectra in the repository. Given the new spectrum S_{new} is different from existing spectra, and no more space is available in the repository, then S_{new}'s classifications on a test data segment of a certain size N is assessed against the classifications produced by spectra that exist in the repository. Suppose that C is the number of classes. Then the winner spectrum S_{new} is aggregated with the spectrum S_a determined by Eq. 2.11 if the distance similarity (E) is greater than the given threshold for similarity, as done in Sakthithasan et al. (2015).

In Equations 2.8 and 2.9, the class labels $c(S_{new}(i))$ and $c(S_p(i))$ of the i^{th} data instance is determined for spectra S_{new} and S_p respectively by applying the Inverse Fourier Transform (IFT) on the respective spectra to reconstruct a numeric approximation of the class value which is converted to a class label by multiplying by the number of classes C and then taking the ceiling of the resulting numeric value that is returned. The same operation is performed on all spectra which are already in the repository.

$$c(S_{new}(i)) = \lceil f(S_{new}(i)) \times C \rceil \tag{2.8}$$

$$c(S_p(i)) = \lceil f(S_p(i) \times C) \rceil \tag{2.9}$$

$$E = N - \sum_{p=1}^{P} \sum_{i=1}^{N} I(c(S_{new}(i)) \neq c(S_p(i))) \tag{2.10}$$

where P is the number of spectra in the repository.

$$S_a = \underset{p}{\operatorname{argmax}}(\mathrm{E}) \tag{2.11}$$

For example, if the number of classes C=3 and if the inverse Fourier value returned for data instance i with spectrum S_{new} is 0.61, then clearly instance i should be labelled with class value 2 as the class boundaries are [0.0..0.33], [0.34..0.66], [0.67..1.0]. This label of 2 is recovered by multiplying 0.61*3 and then taking the ceiling of 1.83, giving a class label of 2.

Equation 2.10 computes the distance similarity E between S_{new} and a spectrum S_p in the repository by counting the number of instances that return the same class labels over the test segment of size N and then subtracting the total count by N to get the similarity score. The identity function I where $I(b) = 1$ if b is true, 0 otherwise, is used to determine if S_{new}, S_p agree or not on class outcomes. In Eq. 2.11, the spectrum S_a that has the maximum similarity with S_{new} is returned.

If this maximum distance similarity is smaller than the given threshold, aggregation does not take place. In the case when the repository is full and insufficient distance similarity exists to meet aggregation requirements, the least accurate spectrum is removed

to make way for the newly generated spectrum.

Aggregation of spectra brings with it two major benefits. Firstly, a reduction in space as coefficients common to spectra being aggregated need to be stored only once. Secondly, aggregation performs a similar role to an ensemble of models and leads to better generalisability to new data arriving in the stream.

In a nutshell, Fourier spectrum of a bounded depth decision tree has several interesting properties including the exponential decay of energy with a coefficient order, ability to derive class label through the use of the IFT, and the possibility of deriving ensemble classifiers by aggregating Fourier spectra through an algebraic operation. In addition, Park (2001) presented an algorithm for reconstructing a decision tree from a Fourier spectrum. The recent study of Sakthithasan et al. (2015) observed that the robustness to noise of Fourier classifiers is significant when compared to the application of decision trees on their own. Taken together, the ease of algebraic Fourier model aggregation in contrast to trees, higher accuracy due to its noise robustness, and the space efficiency of ensemble models make a strong case for the use of Fourier ensembles in concept drifting, concept recurring data streams.

2.6 Conclusion

In summary, the high speed data stream classification research landscape has been enriched over the last two decades in order to address its inherited challenges of sheer volume, continuous high speed flow, and concept drift in the presence of limited computer resources. Amongst a variety of solutions, the use of incremental classifiers instead of conventional static learners, incremental ensembles of classifiers versus a single incremental classifier, incremental classifier ensembles with embedded drift detectors in contrast to ones without drift detectors, all have contributed to promising results.

More specific to concept recurring streams, the use of two classifier types, namely online learners and repository of previously learned models have highlighted the advantage of reuse, in contrast to relearning. Having a more compressed version of a Decision tree, made possible by the use of the DFT helps in lifting processing speed. The capability of simple algebraic aggregation of spectra has made it practical to maintain a repository of spectra which helps to lift accuracy.

The major contributions of this thesis revolve around classifying concept drifting and recurring data streams. The research has been inspired by recent improvements achieved with adaptive/incremental algorithms, ensemble classifiers, drift detectors, and recurrence capture mechanism through a repository of Fourier spectra.

Chapter 3

Staged Learning Approach

3.1 Introduction

In this chapter, a new framework and associated algorithms to classification in non-stationary streams of data are presented. Different from closely related previous work (Gama & Kosina, 2009; J. Gomes et al., 2010; Sakthithasan et al., 2015), this approach detects volatility in a stream and then matches the learning paradigm to the degree of volatility. In high volatility stream segments a decision forest is used as the learning mechanism, whereas in low volatility segments, an approach driven by the use of stored Fourier spectra are used for learning.

The framework is generic in the sense that it is able to cope with all of the drift types identified in chapter 2 and experimented with in chapter 4. Furthermore, the framework is modular in design as each component can have different implementations corresponding to different methods that have been proposed to solve a particular issue in a learning environment accompanied by concept drift.

3.1.1 Basic components

Firstly, the necessity of each component in the proposed framework is identified. An incremental learner that restructures models by synchronizing changes in data patterns to models is indispensable to cope with high data arrival rates in a data stream. The synchronisation of changes in data patterns is accomplished through the use of a concept drift detector. Without a drift detector a learner will experience severe drops in accuracy from time to time and hence it is also a mandatory component. As described in chapter 2, it is useful to maintain a repository of past concepts in cases where a recurring drift pattern is present. Thus the basic components required to support online learning in non-stationary environments are:

- An incremental classifier
- A concept drift detector
- An online repository of past concepts

3.1.2 Optimizing for stream volatility and speed

The above components are basic in the sense that they support core functionality but other supporting elements such as memory management and support for high speed streams are also essential. In streams that are highly volatile, many different concepts can manifest and it may not be feasible to store all concepts in the repository even if compression were to be applied. This calls for a memory management scheme that goes beyond a simple first-in-first-out strategy of populating spectra in the repository.

At the same time, the framework should be able to take advantage of periods of low volatility to speed up processing by reusing already learned models coupled with a minimal amount of learning that is needed to reflect changes in the recurring concepts. This would result in speeding up the learning process and would require a mechanism to sense the level of volatility in the stream. The volatility detector could then adjust the

mode of learning from an intensive learning mode to a less intensive one or vice versa, as the case may be. The level of volatility could be estimated by monitoring whether the probability p_r of usage of past concepts in the repository is significantly higher than the probability p_n of usage of concepts that are evolving or new. If this is the case then it indicates that the system is operating in a less volatile state and learning can then be adjusted accordingly. Learning in a less volatile state can rely to a large extent on classifiers stored in the repository with minor adjustments if needed, and hence should be more efficient than learning in a highly volatile state where new concepts need to be learned. The staged learning framework that will implement this key notion of sensitivity to stream volatility is presented in section 3.2 below.

3.2 The context sensitive Staged Learning framework

The framework has been referred to as being context sensitive as it recognises system behaviour and tailors the learning strategy accordingly. Thus it is able to recognise periods of stability, stages in which concepts are in a state of change, periods of concept reoccurrence, and finally, system states with different levels of volatility.

Starting with choices available for each of the basic components, this section first explores the design choices needed to achieve context sensitivity. The discussion on staged learning approach and volatility detection follows thereafter.

3.2.1 Implementation choices

Each of the components listed in section 3.1.1 can be implemented in several different ways. As discussed in section 2.4, a large number of incremental classifiers have been proposed for data stream mining, including the decision tree group of classifiers, the Bayesian family of classifiers, and others. A popular class of incremental learners uses ensembles of decision trees with the Hoeffding tree (Domingos & Hulten, 2000) as

the basic learning mechanism. Decision trees have proved to be a popular choice in data stream mining on account of their learning efficiency and their ability to cope with interdependence between features, unlike the Naive Bayes classifier. Some examples of incremental classifiers using ensembles of decision trees including CBDT (Hoeglinger et al., 2009), OzaBagADWIN (Bifet et al., 2009), LeveragingBag (Bifet, Holmes & Pfahringer, 2010), and ARF (H. Gomes, Bifet et al., 2017).

It is clear that any of the decision tree ensembles can be used as the incremental classifier component and this study has chosen CBDT as the implementation choice for the incremental classifier component. A number of previous studies (Hoeglinger et al., 2009; Sripirakas & Pears, 2014; Sakthithasan et al., 2015; Kithulgoda & Pears, 2016; Kithulgoda et al., 2018) have shown that CBDT provides a good foundation for learning in a concept drifting environment and this influenced our choice. However, we note that the methods developed in this chapter and other chapters can equally be applied to other types of decision tree ensembles.

In terms of concept drift detectors a large choice of drift detectors exist, including EDDM (Baena-Garcia et al., 2006), ADWIN (Bifet & Gavalda, 2007), and SeqDrift2 (Pears et al., 2014), among others. All of these detectors require the same input, which is a binary stream of the truth value of classification decisions, while all of them produce an output which is a binary variable, indicating whether or not the classification decision is correct or not by reference to ground truth data. Hence the drift detector component is completely interchangeable amongst the drift detectors that are currently available. The implementation choice for this work was SeqDrift2 on account of its low false positive rate and optimized drift detection delay in relation to other drift detectors (Pears et al., 2014).

With respect to an online repository, there have been two different approaches so far proposed in the literature. The first by Ramamurthy and Bhatn (2007) stores decision trees in their original form in the repository. The second approach used in Sripirakas and

Pears (2014); Sakthithasan et al. (2015); Kithulgoda and Pears (2016) and Kithulgoda et al. (2018) is to compress decision trees into Fourier spectra by applying the Discrete Fourier Transform (DFT), and then storing the resulting spectra in the repository (as illustrated in section 2.5). Classification can be performed directly on the spectra by applying the Inverse Fourier Transform (IFT) without having to recover the original tree, thus making such a solution attractive on account of the compression achieved. The time to classify new data could also reduce due to the compact nature of the spectrum. The trade-off with better memory utilisation is the transformation cost but this is a one time cost. When concept drift is signalled by the drift detector, all classifiers (including the spectra in the repository) are polled to determine which one has the best classification accuracy on the new concept and this particular classifier is then used on the current concept. This process and its results are independent of whether or not Fourier based compression is used and hence the repository component is also interchangeable. This work's implementation uses Fourier spectra as classifiers for the repository.

3.2.2 The Staged Online Learning (SOL) approach

So far emphasis in this chapter has been on maintaining classification accuracy in the presence of concept drift and there was no attempt to improve performance, apart from a possible improvement in classification time resulting from having more compact classifiers in the repository. As mentioned briefly in section 3.1.2, one optimization that could result in significant improvements to system throughput would be to model a data stream as a state machine that models interactions between two states. The first state can be thought of as a "learning state" (henceforth referred to as a Stage 1) where new concepts appear and these concepts are learned and stored as classifiers in the form of decision trees, Fourier spectra or other types of models.

The second state is a "deployment state" (henceforth referred to as Stage 2) in which

concept drifts still appear but the vast majority of drifts take place between concepts already learned in the first state. If such concepts are stored online in the repository then classifiers representing these concepts could be deployed as they were without the need for relearning them when concepts recur. Whilst this could yield significant gains in throughput, concepts may undergo some change when they reappear, and in practice, some level of learning may also need to take place in Stage 2.

In keeping with the low volatile nature of Stage 2, the decision tree forest is suspended and Fourier spectra are used as classifiers. When a concept drift occurs in Stage 2 processing, the spectrum S that reports the best accuracy is chosen as the classifier for the current concept. However, it is possible that some concepts may undergo a change after reappearance, and at the end point of a concept's progression, as signalled by concept drift, it may happen that the stored version of S may no longer be a precise representation of the concept. In order to synchronise S with the changed state of the concept, a single decision tree is used to learn any changes that take place in the current concept. This tree is induced from spectrum S at the start of the concept and thereafter learns any changes that may take place after that point onwards. The tree induction algorithm proposed in Park (2001) has been used for this purpose. At the end of the concept, the tree is transformed into a new spectrum S_{new} which is placed into the repository if space permits, otherwise it is aggregated with its closest matching spectrum in the repository using the process described in section 2.5.1.

An alternative strategy would have been a simple replacement of S with S_{new} but it is believed that aggregation would enable historical properties of the concept to be preserved, thus offering a better generalization capability. Overall, the processing overhead in Stage 2 is much lesser than in Stage 1 as only a single tree needs to be maintained in Stage 2 in contrast to a forest of trees in Stage 1. In addition, as mentioned earlier, Fourier spectra are more compact than their decision tree counterparts and hence classification can also be expected to be more efficient. Experimental results on

throughput presented in section 4.2 show clearly that this is the case.

The staged approach is a generalization of the "learn then deploy" paradigm used in classical machine learning on a stationary data environment. The difference here is that many cycles of learning and deployment may occur within a data stream, unlike a stationary environment which involves just one (unless the data miner retrains a classifier periodically).

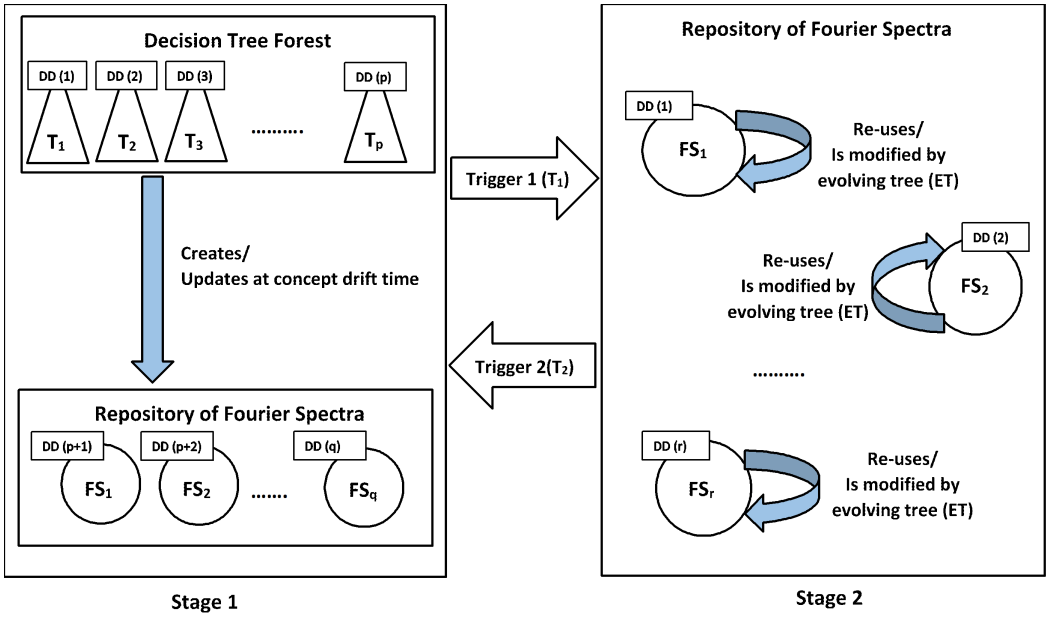


Figure 3.1: Staged Learning framework for context sensitive learning

Figure 3.1 shows the interactions between the major components of the staged learning framework. The incremental classifier component consists of a forest of decision trees. Each tree in the decision tree forest operates under the control of a drift detector.

The system starts off in Stage 1 with the repository in an empty state. Classification is initially done in a grace period G with a randomly selected tree from the forest. This tree is designated as the "winner" classifier, meaning that it is solely responsible for

classifying data arriving in the stream until a concept drift occurs. Within the span of the grace period the drift detection buffer of each drift detector associated with a tree is populated with its own classification decision, irrespective of whether or not it is the designated winner classifier. At the expiration of the grace period the tree that returns the highest average accuracy is chosen as the new winner tree and this tree is chosen to classify new data arriving in the stream beyond the grace period. This process continues until a drift signal is produced. At drift point, the classification accuracy of each tree in the forest is assessed and a new winner is selected which will be responsible for classification until the next drift occurs. At each drift point, the winner tree is compressed by applying the DFT and the resulting spectrum is stored in the repository. Once spectra appear in the repository they can be used for classification, just as with trees in the forest. As spectra are classifiers in their own right they too operate under the control of drift detectors.

In Stage 2, each Fourier spectrum is paired with its own Evolving Tree (ET) when that spectrum becomes the winner. As described before, the tree ET is used to synchronise the current state of a concept with the spectrum that it is paired with.

The staged approach requires a mechanism for determining the stream state and for transiting between states. Transition from Stage 1 to 2 is governed by the firing of a trigger T_1 when a shift from high stream volatility to low volatility is identified by the volatility detector. On the other hand, the reverse shift from low stream volatility to high volatility is triggered by T_2. Details of how these triggers function appear in the subsequent section.

3.2.3 Transition between Stages

Volatility shift is captured through the application of rigorous statistical methods. Firstly, a formal definition of volatility is presented and then the discussion proceeds to illustrate

how shifts in volatility are detected by framing the volatility shift problem in terms of a concept drift problem.

Definition 1 Volatility is defined as the rate of appearance of new concepts in the stream with respect to time. In any given stream segment of length l, if n new concepts appear, then volatility is defined as the probability of appearance of a new concept and is estimated by $\frac{n}{l}$.

Note that the definition is based on the appearance of new concepts and not on the probability of concept drift taking place. Concept drift can occur as a result of concept changing over to a new, previously unseen concept or reverting to a previously seen concept. If s switches in concept take place in a stream segment of size l, then in general only $n(\leq s)$ of them will be new and hence the volatility rate as defined above as $\frac{n}{l}$ will be less than or equal to the rate of concept drift, $\frac{s}{l}$.

Although Definition 1 characterizes volatility, its utility in practice is limited unless a method can be found to measure the rate at which new concepts appear in a given data stream. With this in mind, it is interesting to consider the role that the repository plays in classification.

With the staged transition learning framework in place, as long as a concept exists in the online repository that matches the newly emerging concept in the stream that is signalled by the drift detector, then no re-learning is required. In such cases classification is performed with the concept stored in the repository. This suggests that the rate of re-use of objects in the repository can be taken as a proxy for the rate of appearance of new concepts. The higher the rate of re-use, the lower is the rate of appearance of new concepts in the stream, and lower is the volatility. It is possible to provide an operational definition for volatility in this position.

Definition 2 At any given point in time during the operation of Stage 1 with the occurrence of s concept drifts, volatility is estimated as: $1 - \frac{\sum_{i=1}^{s} B(R)}{s}$, where $B(R)$ is a Boolean-valued function that returns "1" if the newly emerging concept i is found in the repository, otherwise it returns value "0".

Definition 2 quantifies volatility in terms of the empirical hit (success) rate of the repository in finding emerging concepts, which is given by: $h = \frac{\sum_{i=1}^{s} B(R)}{s}$. The higher the hit rate, the lower is the volatility. It is noted that with a drift detector in place that has high sensitivity and low false positive rate, the hit rate, and hence volatility can be determined. Now it is possible to determine the transition point between Stages 1 and 2.

In order to determine the transition point, a window of size w is maintained that contains samples drawn from values returned by function $B(R)$ defined above. A check for a transition detection point is made after the arrival of every s concept drifts. The window is divided into a left sub-window of size $(w - s)$ with the right sub-window containing the last s samples.

Definition 3 A transition from Stage 1 to Stage 2 occurs if at a concept drift point i the repository hit ratio satisfies:

- Condition 1: $h_r > h_l$
- Condition 2: $h_r > \alpha$

where h_l, h_r are the hit ratios across the left and right sub-windows respectively and α is a user defined threshold on hit ratio. Definition 3 establishes that the transition point i is reached only when an upward shift in the hit ratio takes place in the window prior to the hit ratio exceeding α.

In practice the hit ratio is a random variable and ensuring conditions 1 and 2 require statistical significance tests to be made. To check validity of condition 1, a one-tailed

statistical hypothesis test $H_0 : h_l \geq h_r$ versus $H_1 : h_l < h_r$ is formulated. Here h_l, h_r represent the population means of the data across the left and right sub-windows respectively.

Thus it can be seen that volatility detection is essentially a second order determination of concept drift. Amongst the set of drifts recorded in the stream if the rate of change of appearance of new concepts, as signalled by the repository hit ratio, is on a statistically significant decreasing trend, then sufficient evidence exists that the system has transited to a low volatile state (Stage 2). The implication is that the volatility detection problem can then be framed in terms of a concept drift problem and the SeqDrift2 (Pears et al., 2014) drift detector is used for volatility detection as well.

The SeqDrift2 detector makes use of the sample means $\hat{h}_l$, $\hat{h}_r$ and a threshold ϵ_1 to determine if condition 1 is satisfied. If $(\hat{h}_r - \hat{h}_l) > \epsilon_1$, then H_0 is rejected with probability $(1 - \delta)$, else H_1 is rejected. The ϵ_1 threshold is given by: $\frac{1}{3(1-k)n_r}(p + \sqrt{p^2 + 18\sigma_s^2 n_r p})$, where $p = \ln \frac{4}{\delta}$, $k = \frac{n_r}{n_r+n_l}$, σ_s= sample varience, n_r = size of the right sub window, n_l = size of the left sub window and δ= drift significance level.

If H_1 is rejected, then it is possible to conclude that no significant increase in hit ratio has occurred in the current window and hence it is recommended to proceed and update the left sub-window with samples from the right sub-window before proceeding to gather a new set of s samples in a new right sub-window for re-testing H_0 versus H_1.

If there is evidence to reject H_0 at the δ significance level, then the implication is that the stream is moving towards Stage 2 since classification is relying increasingly on the repository that contains previously captured concepts, in preference to the forest. However, as yet there is no definitive evidence to transit to Stage 2 as the recent activity may still not be high enough to justify suspending the operation of the forest of trees. Thus a further hypothesis test for condition 2 is carried out to ascertain whether $\hat{h}_r$ is greater than some acceptable threshold value α.

To check validity of condition 2, hypothesis H_2 tests whether $h_r \leq \alpha$ versus H_3

which corresponds to: $h_r > \alpha$. If H_2 is rejected in favor of H_3, then the stream is considered to have transited to Stage 2. If not, the stream is still in Stage 1 and at the arrival of the next s samples condition 2 is re-evaluated. As with the test for condition 1 above the sample hit ratio $\hat{h}_r$ is used and a threshold ϵ_2 to execute the test. If $(\hat{h}_r - \alpha) > \epsilon_2$, then H_2 is rejected in favor of H_3 with probability $(1-\delta)$. The threshold ϵ_2 is computed using the Hoeffding bound and is given by: $\epsilon_2 = \sqrt{\frac{\ln \frac{1}{\delta}}{2n_r}}$.

Here the attention is trigger T_2. The rationale behind T_2 is based on tracking how good the spectra in the repository are in classifying concepts that are evolving. To the extent that spectra return high classification accuracy, Stage 2 processing should continue. High classification accuracy can result when concepts produced by the stream are similar to those produced in the past or there is a collection of trees that is capable of reacting quickly to the arrival of several new concepts that are dissimilar to those seen in the past. Thus when the spectra produced through the growth of a regenerated tree starts to deviate sharply from those already in the repository in terms of structural similarity, then there is an indication that the concepts appearing in the stream are novel in the sense that they have not been captured previously in the stream. Having a definition for structural similarity in place, now it is possible to implement T_2.

Definition 4 Structural similarity sim_C between the evolving tree ET and the Repository R is given by: $sim_C = \left(\frac{\max_{S \epsilon R} \sum_i (B[S(i)=ET(i)])}{m} \right)$, where C is the current concept; B is a Boolean valued function that returns binary "1" for data instance i if the classification outcome from spectrum S matches with the classification outcome for tree ET; if no match is produced, then binary "0" is returned in the window; m is the length of the current concept drift point - the number of data instances in the concept; and i is an index that ranges over the data instances in the current concept.

As illustrated in Definition 4, the similarity score sim_C returns the structural similarity between the tree ET and its best matching spectrum S in the repository. It is

measured at each concept drift point C. If $sim_C < \beta$, then binary "1" is written to the change detection window, otherwise "0" is recorded.

As with trigger T_1, SeqDrift2 is used as the change detector. SeqDrift2's window is split into left and right sub-windows and with the arrival of every s concept drift points the null hypothesis $H_4 : \mu_l \geq \mu_r$ is tested against $H_5 : \mu_l < \mu_r$. If H_4 is rejected with probability $(1 - \delta)$, then the system transits back to Stage 1 as the right sub-window shows a significant increase in the occurrence of structural dissimilarity, otherwise Stage 2 processing continues. The intuition behind triggering T_2 is that a state change back to Stage 1 needs to occur when the concepts stored in the form of spectra in the repository becomes sufficiently dissimilar to the currently emerging concepts in the stream.

3.2.4 Algorithm

The staged data classification approach of SOL is illustrated by algorithms 1 and 2. Those two algorithms summarise the sequence of steps to be followed in Stage 1 and Stage 2 respectively.

At the beginning of classification, the system is in Stage 1. Incoming data instances are classified by using the best performing classifier chosen from one of the trees in the forest or from one of the spectra in the repository as per step 3. According to step 6, the stage change detector is fed at every concept drift point by "1" or "0" as an indication of a repository hit or not, respectively. As given in step 7 of algorithm 1, after observing s number of concept drift points, hypotheses H_0 and H_2 are tested (step 8). When both hypotheses are rejected, the current best performing spectrum is transformed to its equivalent tree (step 9), which is to be used as the ET learner for the initial concept in Stage 2. The decision tree forest is suspended in step 10, and the system then transits to Stage 2 in step 15.

As illustrated by Algorithm 2, subsequent incoming instances are processed by the

Algorithm 1 Algorithm Process In Stage One

Input: Sample size *s*, T_1 firing threshold α

repeat
  Read next instance ▷ Stage 1 Processing
  Classify with best classifier chosen from CBDT forest or repository rep
  if concept drift is detected **then**
    Increment *Concept Drift Count*
    Feed stage 1 detector with 0 if $accuracy(forest) > accuracy(rep)$, else 1
    if *Concept Drift Count* $\bmod s = 0$ **then**
      if H_0 and H_2 are both rejected **then**
        Evolving tree (ET) from a copy of current best spectrum
        Suspend the forest and free its memory
      end if
    end if
  end if
until H_0 and H_2 are rejected
Call PROCESS IN STAGE TWO(*s*)

Algorithm 2 Algorithm Process In Stage Two

Input: T_2 firing threshold β

repeat
  Read next instance ▷ Stage 2 Processing
  Classify instance with best classifier chosen from ET or a spectrum S chosen from the Repository rep
  if concept drift is detected **then**
    Increment *Concept Drift Count*
    if accuracy (ET)> accuracy(rep) **then**
      Transform ET into its spectrum and update the rep
    else
      Generate ET from a copy of current best spectrum
    end if
    Feed stage 2 detector with 1 if $structuralsimilarity < \beta$, else 0
    if *Concept Drift Count* $\bmod s = 0$ **then**
      if H_4 is rejected **then**
        Create a new instance of CBDT forest and add ET to it
      end if
    end if
  end if
until H_4 is rejected
Call PROCESS IN STAGE ONE()

best performer chosen from either the ET or the spectra in the repository (step 3). At each concept drift point in Stage 2, the repository is modified by either adding a new spectrum or by aggregating with an existing spectrum when the best performer happens to be a tree chosen from the decision tree forest as given in step 7. A tree representation of the best performing spectrum is also constructed in step 9 to classify incoming data instances in Stage 2 and to adapt to small-scale concept changes in the data.

The degree of agreement (i.e. structural similarity) of the tree ET with spectra in the repository is examined in step 11 and the stage change detector window is fed with binary "1" or binary"0" by comparing the similarity score with the user-defined parameter β. When s number of drifts occur, hypothesis 4 is tested; in the event that it is rejected a new instance of the CBDT forest is started in step 14 as a result of triggering T_2.

3.3 Time and space complexity of spectral learning

Space and throughput advantages of SOL results from spectral learning. This section illustrates the space and time complexity for the classification based on spectra in the repository.

An upper bound for the space consumption of spectra is determined by the total number of coefficients M taken over all spectra in the repository. Assuming that there are P spectra and that Q_p is the size of the coefficient array for the p^{th} spectrum, the space complexity is given by Eq 3.1

$$M = \sum_{p=1}^{P} Q_p \tag{3.1}$$

: The spectral size Q_p is determined by the order O_p for the given energy threshold, as

described in section 2.5. Hence we have:

$$Q_p = \sum_{r=0}^{O_P} {}^{d}C_r \tag{3.2}$$

where d is the number of data features and ${}^{d}C_r$ is the number of combinations of selecting r features from a total of d. The memory complexity M is then given by:

$$M = \sum_{p=1}^{P} \sum_{r=0}^{O_P} {}^{d}C_r \tag{3.3}$$

The Fourier classifier does not rely on a deep hierarchical tree structure but instead uses a self-indexing hashing scheme to store its schema values. This is compact due to the reasons mentioned in section 2.5.

Now that an upper bound for the number of coefficients in a given spectrum has been formulated, the time complexity of the IFT operations defined by Eq. 2.3 can be expressed as:

$$O(Md^2) \tag{3.4}$$

as d^2 multiplications are needed for the computation of the vector product between $\vec{x}$ and $\vec{j}$ for each of the coefficients in the array.

3.4 Conclusion

The staged learning paradigm represents a major shift in the way that data streams are mined and was motivated by the need to scale classifiers to high speed data environments. The proposed system suspends the operation of the ensemble of learners upon detection of a sufficiently large rate of recurrences in the data stream. This was quantified and implemented through the use of trigger T_1 that shifts the system from Stage 1 to Stage 2.

Even though ensemble approaches, in general, perform better than single classifiers in terms of accuracy and stability, the model statistics update overhead is greater as statistics for every classifier in an ensemble needs to be updated on a per-instance basis. For that reason, the proposed framework is expected to result in much higher throughput through suspension of computationally expensive ensemble learning.

While scaling classifiers to high speed, it is necessary to guarantee that the accuracy is not compromised. This has been ensured through timely decisions on stage transition that are driven by sound statistical thresholds. In the low volatility stage where there is no significant appearance of previously unseen concepts, the framework depends on a set of stored Fourier spectra and a single derived learner instead of an entire ensemble. This learner is responsible for capturing small-scale changes in between recurrences.

The framework will be evaluated empirically for its timely stage transition, throughput, accuracy, and memory advantages over diverse datasets in the next chapter.

Chapter 4

Experimental Study on Staged Online Learning

4.1 Introduction

This chapter validates the feasibility of the proposed staged learning framework in chapter 3 through a comprehensive empirical study. The datasets were chosen so as to represent all of the drift types in order to test whether the framework covers all of the drift types that can be encountered in practice.

The first set of experiments studied the effectiveness of triggers and stages with respect to the inherent characteristics of various datasets. Several different evaluation criteria such as accuracy, throughput, and memory consumption of Staged Online Learning (SOL) approach were analysed against four other different classifier systems in the second set of experiments. The robustness of different classifiers was compared according to their average accuracy rank over all datasets. With the belief of the importance of both performance measures, namely accuracy and throughput of a stream classifier, an accuracy versus throughput analysis was also performed. Finally, the sensitivity analysis assessed the dependency of various parameter settings on the SOL.

4.2 Empirical study

The empirical study consists of four basic sections. Firstly, the effectiveness of the SOL approach was tested with two key performance measures, namely per-stage classification accuracy and per-stage processing speed. The study compares SOL against Ensemble Pool (EP) (Sakthithasan et al., 2015) and Recurrent Classifier (RC) described in section 4.2.1.

Then, overall accuracy and throughput of SOL were examined in comparison to several well-known algorithms. In this connection, a comparative study is conducted against two more algorithms that have been proposed for concept drifting data streams. These algorithms are state of the art meta learning algorithms featured in MOA[1], namely Adaptive Random Forest (H. Gomes, Bifet et al., 2017) and LeveragingBag (Bifet, Holmes & Pfahringer, 2010). A brief explanation of the operation of these classifiers can be found in section 4.2.1. Thirdly, the memory consumption was evaluated against EP. Finally, the effects of different parameter settings on the performance of SOL was analysed.

4.2.1 Algorithms used for the study

In this section, an overview of the operation of the four comparators is given.

1. Adaptive Random Forest (ARF):

   This recently proposed classifier ensemble (H. Gomes, Bifet et al., 2017) is a reworking of the widely cited Random Forest algorithm presented in Breiman (2001) together with a revised version of earlier efforts to adapt Random Forest in Abdulsalam, Skillicorn and Martin (2007, 2008). In contrast to those previous studies, one of the newly added features of ARF is its resampling strategy which is followed by Online bagging (Oza, 2005). Secondly, ARF uses a drift detector

[1] from *http://moa.cms.waikato.ac.nz/*

per tree in order to detect drift warning point and the drift point of the stream. Individual trees used here are Hoeffding trees, as described in Domingos and Hulten (2000), except for the policy of pruning of poor attributes which is not implemented in ARF.

Once a warning of drift is detected, the training of new trees is started and replacement is done at actual drift time. The study of ARF highlights the positive aspect of initiating training of new trees at the warning point rather than waiting until drift point to reset learners as the majority of studies do. Authors claim the ability to use any drift detector with ARF and their experimental study has been demonstrated with both ADWIN (Bifet & Gavalda, 2007) and Page Hinckley Test (Page, 1954) drift detectors. In our experiments, ADWIN was selected as the ARF's drift detector.

With the aim of dealing with the high execution overhead usually associated with ensemble learning, ARF has implemented multithreading to parallelize processing of trees in the ensemble. Their experimentation revealed that speedups of up to 3 were achievable over the serial version. In the comparative experimentation of this chapter, the serial version of ARF was considered in fairness to other algorithms that do not use a multithreaded implementation.

2. LeveragingBag (LB):

   This algorithm that is given in Bifet, Holmes and Pfahringer (2010) claims better performance than the well-known Bagging (Oza, 2005) algorithm through the use of two techniques. First, diversity of instance weights is increased by using a larger λ for the Poisson distribution as compared to the use of Poisson with $\lambda = 1$ in the original Bagging algorithm. The second modification is having distinct predictive functions for each classifier in the ensemble in order to reduce correlations between classifiers. Both variations are done with the objective of

increasing diversity in the ensemble.

Drifts are detected with ADWIN (Bifet & Gavalda, 2007) change detector. At drift, a new model replaces the worst performing model in the ensemble. According to their empirical study, better accuracy was achieved over several existing Bagging algorithms. However, the results showed that this accuracy comes at the price of efficiency.

3. Staged Online Learning approach (SOL):

   This is the approach presented in the previous chapter and published in Kithulgoda and Pears (2016). The system follows the staged learning approach as explained in chapter 3. Drift detector instances of SeqDrift2 (Pears et al., 2014) are embedded into each classifier.

4. Ensemble Pool (EP):

   The EP is the classifier system presented in Sakthithasan et al. (2015). The CBDT tree forest (Hoeglinger et al., 2009) and the Fourier classifier repository (Sripirakas & Pears, 2014) integrated with the concept drift detectors SeqDrift2 (Pears et al., 2014) are the main components of EP. Compared to SOL, the EP approach employ neither the staged learning approach nor an evolving tree and hence it is one of the ideal methods to compare against. This algorithm was selected as it would provide interesting contrasts and insights into the performance of unique features that were introduced into the SOL framework.

5. Recurrent Classifier (RC):

   This is another implementation of the staged approach which simply uses Fourier spectra generated in Stage 1 to classify data arriving in Stage 2 without any form of learning. This classifier thus assumes recurrence of concepts in Stage 2 and is thus termed Recurrent Classifier. This version provides a useful contrast with the

SOL as it enables us to assess the benefits of adapting spectra in Stage 2 through an evolving tree. Similar to SOL and EP, drift detector instances of SeqDrift2 (Pears et al., 2014) are embedded into each classifier in repository and forest.

4.2.2 Datasets used for the study

All experiments were carried out with the use of two synthetic datasets generated by MOA's stream generators and four real-world datasets.

Synthetic data

For synthetic datasets, several distinct concepts were generated, each of length 10,000 instances. Drift signals were applied at two levels on the concepts: firstly, abrupt drift was injected to produce a set of distinctive concepts, and then at the second level, recurrences of concepts generated at the first level were produced, as depicted in Fig. 4.1. Each concept recurred with a varying degree of change from its first appearance, depending on its cycle of repetition.

Two different types of changes were introduced at level 2. Firstly, in 'Synthetic data recurring with noise' a given amount of noise was superimposed on the recurring concepts to differentiate them from their previous appearance.

Secondly, instead of noise, drift patterns were used to generate the data. In 'Synthetic data recurring with a progressively increasing pattern of drift', a progressively increasing drift pattern was used, whereas in 'Synthetic data recurring with oscillating drift pattern' an oscillating pattern was superimposed on the recurrence signal. To the best of our knowledge, this is the first experimental study of its kind that embeds several different drift patterns simultaneously in its data. More details can be found below.

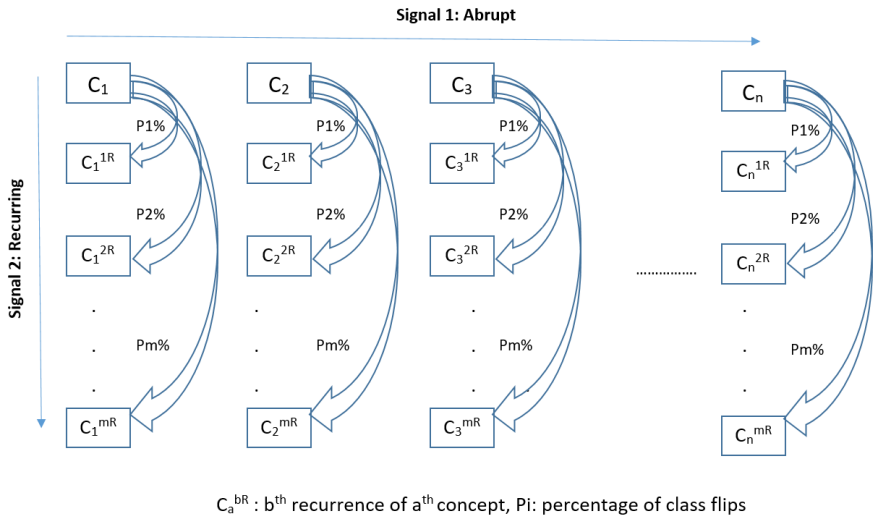


Figure 4.1: Preparation of synthetic datasets with two levels of drift signals

Synthetic data recurring with noise

In following two synthetic datasets, a 5% noise level was introduced for each concept recurrence by inverting the binary class label of randomly selected data instances with the belief that 'concepts do not repeat in exactly the same form' in reality. In this case, class flips P1= P2=....Pm= 5%. This is just the noise which has no any meaningful pattern.

1. Rotating Hyperplane dataset (Noisy RH): This dataset has a total of 10 attributes and six distinct concepts which were created by adjusting the magnitude of change in the range [0.03, 0.04, 0.05, 0.07, 0.08, 0.09]. The first three concepts were repeated 20 times more, with each concept being distorted by a noise level of 5% at each cycle over its base representation (i.e. its first generated state). Three previously unseen concepts were injected at the end of datasets with the intention of examining whether the staged learners would opt for adapting to the new concepts in Stage 2 or for triggering T_2 to transit back to Stage 1. The size of the dataset was 660,000 data instances.

2. RBF dataset (Noisy RBF): This 10-dimensional dataset generated concepts by changing the number of centroids. Here 12 different concepts were produced. The first 5 concepts were repeated 9 more cycles with noise as per the description for RH. The remaining 7 concepts were appended in order to simulate the appearance of completely new concepts in the stream. The size of the dataset was 570,000 data instances.

   The objective of including several concept repetitions was to evaluate the capability of triggering T_1 which should transit SOL system to Stage 2 when recurring concepts present in the stream. Further to that, the sensitivity of trigger T_2 was assessed by injecting new concepts to test whether trigger T_2 would reactivate Stage 1 operation.

Synthetic data recurring with a progressively increasing pattern of drift

With the objective of evaluating the robustness of the proposed framework to any given scenario, two more RH datasets were created by injecting two different monotonically increasing drift intensities on the first 5 concepts of the data stream. Whenever an instance belonging to any one of these concepts recurred with a certain attribute value, the class label for that instance was inverted to the other class with a given probability. In both of these datasets, the drifts between two consecutive concepts were abrupt while the recurrences have gradually deviated from their last occurrence. Experimented two drift intensities are given below.

1. RH progressive increase of 10% in flip probability over cycles (10% progressive RH): The original 5 concepts reappeared in 10 more cycles of repetitions, each of which had 10% (Where P2= P1+10% ,P3= P2+10%...Pm= P(m-1)+10% as per Fig. 4.1) greater flip probability of class label than in the previous cycle. The

process of pattern recognition was challenged by introducing 30% flips of class labels in the first repetition and thereafter 40%, 50%, and so on up to 100%.

2. RH progressive increase of 20% in flip probability over cycles (20% progressive RH): The original 5 concepts reappeared in 5 more cycles of repetitions, each of which had 20% (Where P2= P1+20% ,P3= P2+20%...Pm= P(m-1)+20% as per Fig. 4.1) greater flip probability of class label than in the previous cycle. The process of pattern recognition was challenged by introducing 20% flips of class labels in the first repetition and thereafter 40%, 60%, and so on up to 100%.

Synthetic data recurring with oscillating drift pattern

Further to above, the learning capability of the classifiers was tested when the pattern in between recurrences are tended to oscillate, rather than being monotonic in nature.

1. RH Oscillating flips (Oscillating RH): In this dataset, repetition cycles were appended by interleaving flip probability: P1 = 30%, P2= 70%, P3= 40%, P4 = 80%, P5= 50%, P6= 90%, P7= 60%, P8= 100% as per Fig. 4.1.

Real world data

1. Electricity (Elec) dataset: NSW Electricity dataset is used in its original form[2]. There are two classes *Up* or *Down* that indicate the change of price with respect to the moving average of electricity prices in the last 24 hours.

2. Flight dataset: This dataset[3] was generated by NASA's FLTz flight simulator which was designed to simulate flight conditions experienced with commercial flights. Each flight has four different concepts, corresponding to four flight scenarios: take off, climb, cruise and landing. The 'Velocity' feature was discretized

[2]from *http://moa.cms.waikato.ac.nz/datasets/*
[3]from *https://c3.nasa.gov/dashlink/resources/*

into two binary outcomes *Up* or *Down* depending on the directional change of the moving average in a window of size 10 data instances.

3. Covertype dataset: The original version of this dataset is available at Lichman (2013). The data was collected from Roosevelt national forest of Northern Colorado for the task of predicting forest cover type from 54 attributes derived from 12 cartographic variables. the initial 10% of instances were extracted from the two most frequent forest types, namely *Spruce-Fi* and *Lodgepole Pine*.

4. Occupancy detection dataset: This dataset was also obtained from Lichman (2013) and used by Candanedo and Feldheim (2016). The dataset consists of measurements of temperature, humidity, light, and CO_2 levels in a given room and was collected with the purpose of deciding the suitability of the room for human occupancy. Ground-truth occupancy label outcomes were determined on the basis of pictures taken at intervals of one minute.

4.2.3 Parameter values

The default parameter values used in the experimentation are as follows:
Maximum number of nodes in decision tree forest: 5000, SeqDrift drift significance value (δ): 0.01, Maximum number of Fourier spectra in repository: 40, Repository hit ratio threshold (α) for T_1 is 0.5, Similarity threshold (β) for T_2 is 0.7, and the Smaple size (s) is 200.

System configuration

All experiments were conducted on a Windows 10 Enterprise 64-bit machine featuring Intel Core i5 processor running at 3.2 GHz and having 16 GB of RAM. The framework was implemented using C# 5.0 in .NET Framework 4.5 runtime environment.

4.2.4 Effectiveness of Staged Learning approach

In order to test whether a significant difference in performance exists between proposed SOL classifier, EP, and RC both throughput and accuracy profiles of these algorithms were examined. Tables 4.1 to 4.5 show average values for throughput and accuracy for five datasets. The remaining four datasets follow the same trends and were omitted in the interest of avoiding information overload. The repository hit ratio of Stage 1 which represents the usage of stored classifiers was also reported for SOL. In addition, the averaged structural similarity (given in the definition 4 in section 3.2.3) was tracked to gain insights into the extent of change in the stream during Stage 2 processing for SOL.

Accuracy was observed in batches of size 1,000 and overall accuracy was obtained by averaging across the entire set of batches. The same sampling scheme was used to trace other measures such as throughput, repository hit ratio, and structural similarity. Measures were stable across multiple runs for all classifiers across all datasets with a standard deviation less than 0.05 and hence were not included.

The objective of experimenting with the Noisy RH and Noisy RBF recurring datasets was to assess the sensitivity of trigger T_1 in detecting recurring concepts by transiting from Stage 1 to Stage 2. Also the sensitivity of trigger T_2 was tracked in transiting back to Stage 1 when newly injected concepts appeared in the stream. The results presented in Tables 4.1 and 4.2 depict the potency of triggers T_1 and T_2 through necessary transitions through the stages. In the case of RH noisy dataset, T_2 was not observed even though three previously unseen concepts were injected at the end. This illustrates the ability to handle new concepts without dropping accuracy while working in Stage 2.

Table 4.1: Stage-wise throughput and accuracy profiles for the Noisy RH dataset

		Stage 1 (start-160000)	Stage 2 (160000-end)	Overall
SOL	Accuracy	72.2	73.8	73.4
	Throughput	7973	18304	13930
	Repository hit ratio	0.36	–	–
	Structural similarity	–	90.0	–
EP	Accuracy	72.2	72.8	72.6
	Throughput	–	–	6403
RC	Accuracy	72.2	63.6	65.7
	Throughput	–	–	16824

Table 4.2: Stage-wise throughput and accuracy profiles for the Noisy RBF dataset

		Stage 1 (start-80000)	Stage 2 (80000-560000)	Stage 1 (560000-end)	Overall
SOL	Accuracy	80.9	76.0	71.3	76.6
	Throughput	4357	4514	2976	4451
	Repository hit ratio	0.48	–	–	–
	Structural similarity	–	88.8	–	–
EP	Accuracy	80.9	76.0	70.2	76.6
	Throughput	–	–	–	1435
RC	Accuracy	80.9	60.4	45.0	63.0
	Throughput	–	–	–	4777

The results presented in Tables 4.3 to 4.5 provide clear evidence of the presence of recurrence patterns in real-world datasets (not just for RH and RBF for which they were injected) as SOL fired trigger T_1 on all of them. The fact that SOL triggered it on RH and RBF where there were known recurrences indicates that SOL is sensitive to state changes in the system. Furthermore, the results demonstrate the effectiveness of trigger T_2 in real-world scenarios as well.

Table 4.3: Stage-wise throughput and accuracy profiles for the Flight dataset

		Stage 1 (start-4000)	Stage 2 (4000-end)	Overall
SOL	Accuracy	77.1	82.7	81.8
	Throughput	1399	7201	4331
	Repository hit ratio	0.14	–	–
	Structural similarity	–	97.6	–
EP	Accuracy	77.1	80.7	80.1
	Throughput	–	–	982
RC	Accuracy	77.1	52.9	56.8
	Throughput	–	–	4553

Table 4.4: Stage-wise throughput and accuracy profiles for the Elec dataset

		Stage 1 (start-8000)	Stage 2 (8000-44000)	Stage 1 (44000-end)	Overall
SOL	Accuracy	67.0	66.7	83.5	67.1
	Throughput	14220	32447	13998	25661
	Repository hit ratio	0.38	–	–	–
	Structural similarity	–	85.7	–	–
EP	Accuracy	67.0	65.8	83.5	66.4
	Throughput	–	–	–	11559
RC	Accuracy	67.0	64.9	83.0	65.7
	Throughput	–	–	–	34589

In summary, it is noted that SOL had significant improvements in throughput over EP, specifically 117.6% for noisy RH, 210.2% for noisy RBF, 341.0% for Flight, 122.0% for Elec and 150.2% for the Covertype dataset. As expected, the reduction in overheads caused by replacing a forest of learners with a mechanism of refining already stored concepts yielded significant gains in throughput. Obviously, the throughput gains for the RC classifier were even higher than that of SOL but they came at a heavy price in accuracy.

The competitive accuracy returned by SOL is supported by the high structural similarity score registered by SOL in Stage 2 processing for all datasets. A high level of similarity between the concepts being formed and those already present in the repository

Table 4.5: Stage-wise throughput and accuracy profiles for the Covertype dataset

		Stage 1 (start-12000)	Stage 2 (12000-20000)	Stage 1 (20000-36000)	Stage 2 (36000-48000)	Stage 1 (48000-end)	Overall
SOL	Accuracy	79.3	82.6	88.7	87.0	77.3	84.8
	Throughput	492	7314	389	2510	461	663
	Repository hit ratio	0.25	–	0.23	–	0.07	–
	Structural similarity	-	93.7	-	95.2	–	–
EP	Accuracy	79.3	82.4	87.4	93.3	87.5	86.0
	Throughput			–	–	–	265
RC	Accuracy	79.3	56.8	57.8	56.8	60.0	62.7
	Throughput	-		-	–	–	1626

implies that spectra already generated during Stage 1 are effective in classifying newly arriving data during Stage 2.

Finally, it is observed that the Flight dataset contains strong episodes of concept recurrence as T_2 did not fire until the endpoint was reached. This type of behavior is expected to be exhibited in a number of real-world datasets. Given that Covertype does not have an explicit time dimension but rather a spatial one, yet another insight which can gain from the Covertype results is that the staged approach is effective at capturing recurrences defined on a spatial dimension. Interestingly, a relatively poor performance of RC in Stage 2 was observed. This means that while recurrences are present, they are not in exact form, once again underscoring the need for a limited learning capability in Stage 2 to learn small-scale changes in concepts.

Thus we note that relying totally on spectra stored during Stage 1 is not a viable strategy. This is illustrated by the difference in classification accuracy between SOL and RC. The SOL classifier significantly outperforms RC, thus demonstrating the need for learning and refinement of the spectra in Stage 2.

4.2.5 Accuracy evaluation

In this section, the overall accuracy of SOL against ARF, LB, EP, and RC was compared. All classifiers were run with the Hoeffding tree as the base learner. The leaf prediction method was set to the majority class. The split confidence and the tie confidence parameters were both set at 0.01 for all classifier ensembles in order to maintain fairness. Each meta-learner was run with default settings for its internal parameters.

In each case, the winner's accuracy (with accuracies rounded to 1 significant place) was bolded for easy identification. The LB couldn't complete the classification task for datasets symbolized by "–" due to a heap space error. Ranks for each algorithm per dataset were included in parentheses. The algorithm that reported the highest accuracy obtained a rank of 1, rank 2 was assigned for the runner-up and so forth. The last row of the table contains the average of ranks over datasets together with the overall rank of that algorithm accordingly.

Table 4.6: Classification accuracy with ranking

Dataset	ARF	LB	SOL	EP	RC
RH Noisy	73.9(2)	**76.5(1)**	73.4(3)	72.(4)	65.7(5)
RBF Noisy	78.8(2)	**78.9(1)**	76.6(3)	76.6(3)	63.0(4)
10% progressive RH	76.8(2)	–	**77.0(1)**	76.0(3)	73.4(4)
20% progressive RH	**76.2(1)**	–	**76.2(1)**	**76.2(1)**	75.3(2)
Oscillating RH	76.5(2)	–	**76.8(1)**	74.4(3)	73.1(4)
Flight	78.7(4)	79.2(3)	**81.8(1)**	80.1(2)	56.8(5)
Elec	65.7(3)	65.7(4)	**67.1(1)**	66.4(2)	65.7(3)
Covertype	82.8(4)	**86.3(1)**	84.8(3)	86.0(2)	62.7(5)
Occupancy	**95.9(1)**	85.2(3)	91.4(2)	91.4(2)	82.8(4)
Average Rank	2.3(3)	2.2(2)	**1.8(1)**	2.4(4)	4.0(5)

Rank of an algorithm per dataset followed by the average rank over all datasets was the algorithm of choice in determining the best performing algorithm out of multiple algorithms over multiple datasets. As explained in 4.2.2, datasets are fairly diverse and

this study intends to provide a suitable solution regardless of the domain. Therefore, the ranking method was considered to be more appropriate compared to average accuracy that was less meaningful when datasets were from considerably different domains. Moreover, average accuracy has the potential of being influenced by outliers Demšar (2006).

As shown in Table 4.6 algorithms can be divided into two groups, comprising more accurate classifiers and less accurate classifiers. The first group consists of SOL, LB, ARF, and EP whereas the second group contains only RC. This is evident by overall average ranks of algorithms. It recapitulates that RC's approach of relying totally on spectra stored during Stage 1 is not a viable strategy.

The proposed algorithm SOL emerges as the winner in 5 out of 9 datasets while it becomes the joint winner with ARF and EP for dataset 20% progressive RH. The runner-up algorithm LB results in highest accuracies for RH noisy, RBF Noisy, and Covertype. The Occupancy dataset reports the highest accuracy with ARF. Being the winner for the majority of datasets while being the overall winner over all nine datasets verifies the robustness of proposed stage online learning approach. More precisely, SOL is the best with three challenging synthetic datasets and two real- world datasets. This observation validates the applicability of the framework in various types of recurring and drifting scenarios including real-world datasets.

Even though algorithms were ranked based on the accuracy value, it is also apparent that the difference between best and the second best is negligible (less than 1%) in the majority of datasets except for RH Noisy, Flight, and Occupancy. Interestingly, LB wins RH noisy by 2.6% accuracy difference compared to ARF whereas SOL becomes the winner for Flight by 1.7% compared to EP, and ARF defeats SOL for Occupancy by 4.5%.

The analysis which has been done above was based on the average accuracy ranks on datasets given in Table 4.6. The grouping that has been observed was confirmed by

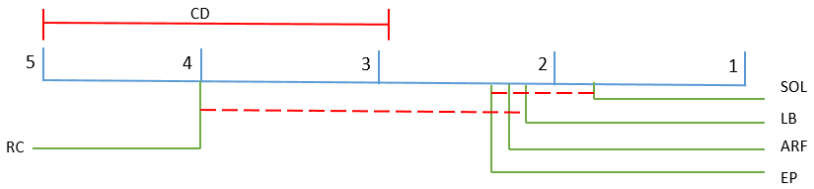


Figure 4.2: Statistical comparison of algorithms by accuracy. Subsets of classifiers that are not significantly different are connected with dashed lines.

non-parametric statistical test named Friedman test. This test has been recognised by Demšar (2006) as one of the best tests to use when it is necessary to compare multiple classifiers against multiple different datasets. The null hypothesis H_0 was that the average accuracy ranks across the 5 classifiers were the same.

Since Friedman test statistic is greater than the critical value, null hypothesis H_0 was rejected at the 95% confidence level, thus indicating that statistically significant differences exist between the classifiers. Then the classifiers were subjected to the post hoc Nemenyi test to identify exactly where those differences lay. The Nemenyi test yielded a Critical Difference (CD) value of 2.09 at the 95% confidence level. Fig. 4.2 graphically illustrates that the top group consisted of SOL, LB, ARF, and EP as none of the members in this group had significant differences with any of the other members within the group. One and only member of the other group is RC. However, the structural of RC is not significant from the subset EP, ARF, and LB.

ARF vs. SOL accuracy of a concept

Further to the above analysis, the accuracy of one particular concept over three consecutive repetitions that manifested in the 10% progressive RH dataset was contrasted. This gives us an in-depth understanding of the rationale behind SOL's performance advantage when compared to ARF.

Figure 4.3 shows that SOL is significantly better at capturing recurrences of past concepts in comparison to ARF that takes a long time to re-learn the concept during

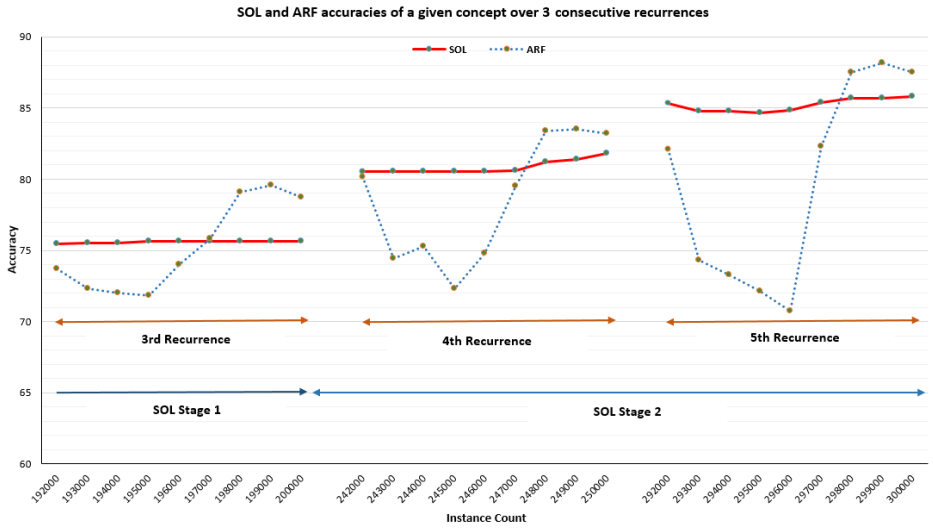


Figure 4.3: Accuracy of a concept

which time its accuracy suffers. Once ARF learns the concept, it eventually manages to capture the concept by adjusting its forest and is able to acquire a slightly higher accuracy than SOL. However, with shorter concepts, ARF would be at a disadvantage.

In addition, the accuracy of SOL is increasing with each repetition due to its property of applying modifications on previously captured patterns rather than a simplistic 'use a previously stored pattern that is the best match' policy or a more expensive 're-learning the currently appearing pattern from scratch strategy'.

The discussion is continued on analysis of throughput in section 4.2.6, and the trade-off between accuracy and throughput in section 4.2.7.

4.2.6 Throughput evaluation

In this section overall throughput of SOL against ARF, LB, EP, and RC were compared. Similar to section 4.2.5, algorithms are ranked so as the highest throughput classifier is assigned a rank of 1, rank 2 for the runner-up and so forth. The last row of the Table 4.7 also contains the average of ranks over datasets together with the overall rank of that algorithm accordingly.

Table 4.7: Throughput with ranking

	ARF	LB	SOL	EP	RC
RH Noisy	5537(4)	659(5)	13930(2)	6403(3)	**16824(1)**
RBF Noisy	4309(3)	221(5)	4451(2)	1435(4)	**4777(1)**
10% progressive RH	1432(3)	–	1991(2)	1046(4)	**2839(1)**
20% progressive RH	1386(3)	–	1469(2)	1176(4)	**1879(1)**
Oscillating RH	1621(3)	–	2917(2)	1150(4)	**4255(1)**
Flight	1903(3)	427(5)	4331(2)	982(4)	**4553(1)**
Elec	5388(5)	7527(4)	25661(2)	11559(3)	**34589(1)**
Covertype	1016(2)	327(4)	663(3)	265(5)	**1626(1)**
Occupancy	9057(5)	13526(4)	19934(2)	14378(3)	**22844(1)**
Average Algorithm Rank	3.4(3)	4.5(5)	2.1(2)	3.8(4)	**1.0(1)**

From the results, it is clear that SOL is the runner-up algorithm in terms of throughput whereas RC reported the highest throughput over all datasets. ARF and EP were at the 3rd and 4th place respectively while LB was the weakest in terms of speed.

In order to gain a statistically sound indication of how these five algorithms differ from each other, the ranks presented in Table 4.7 were subjected to the Friedman test. As displayed in Fig. 4.4 there are 2 distinct groups. The first group consists of RC and SOL as these two classifiers have a significant difference with remaining 3 classifiers. The other group comprises ARF, EP, and LB. On the other hand, the subset of SOL, ARF, and EP are not significantly different from each other in terms of speed as it formed another group (critical difference among these three is not significant). These insights are in accordance with the observations presented in Table 4.7.

4.2.7 Accuracy versus Throughput trade-off

Here the trade-off between accuracy and speed of each algorithm was studied. The following Fig. 4.5 clearly shows that classifiers that tend to be more accurate (e.g. LB, ARF) tend to be more time consuming and vice versa. The SOL is an exception

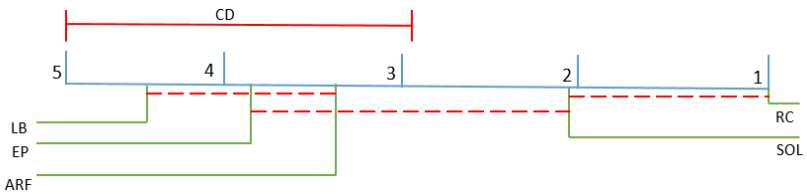


Figure 4.4: Statistical comparison of algorithms by throughput. Subsets of classifiers that are not significantly different are connected with dashed lines.

although it was not the fastest. For that reason, it is clear that SOL has provided a good balance between the two opposing characteristics of accuracy and speed. The nearest classifiers to SOL are ARF and EP as they achieved the next best compromise between accuracy and speed.

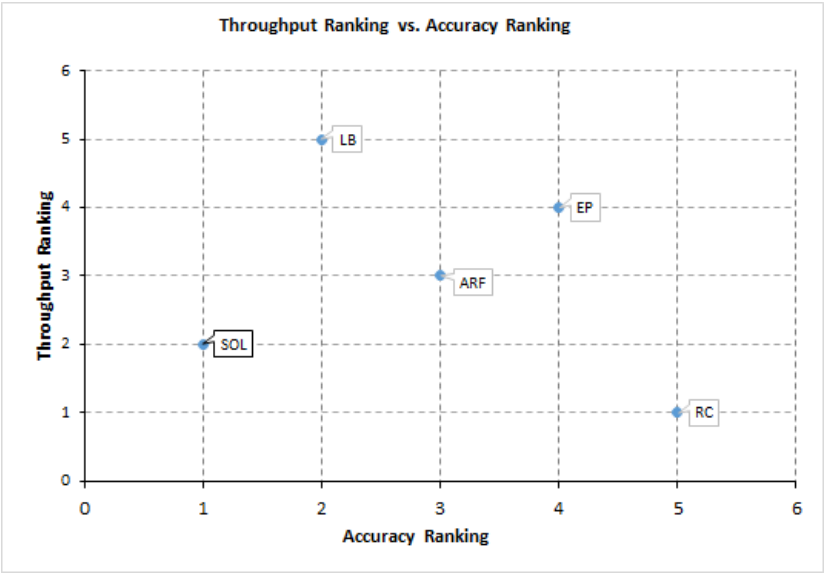


Figure 4.5: Accuracy vs. Throughput trade-off

4.2.8 Memory consumption evaluation

Memory consumption was sampled at the same intervals used in collecting other performance measures presented in previous sections. Table 4.8 presents average number of kilobytes for 10% progressive RH , Elec, and Flight datasets. Table 4.8

shows the gains in memory usage when SOL and EP operate with a repository that can accommodate a maximum of 10 spectra.

The memory gains mirror those of throughput, with improvements ranging from 38.9% for 10% progressive RH, and 57.4% for Flight to 59.7% for Elec. The memory profiles for the other datasets follow the same trends and were omitted in the interest of avoiding information overload. Once again this is due to the reduction of overheads in Stage 2 since SOL's memory overheads are exactly the same as that of EP in Stage 1 as they share the same learning mechanism. As SOL suspends the operation of its forest in Stage 2, its memory is released and its only memory overheads are that of the single tree that it grows and the repository which occupies a small fraction of the space taken up by a forest of trees. While it is also true that EP's repository is also very compact, it suffers by maintaining its forest unnecessarily in Stage 2 where volatility is low.

Table 4.8: Average memory consumption (in kBs)

		10% prog. RH	Elec	Flight
SOL	Stage 1 Forest	1514.80	58.59	585.29
	Stage 1 Repository	315.36	9.01	32.82
	Stage 1 Total	1831.55	67.60	618.10
	Stage 2 Total	571.79	27.16	126.55
	Weighted average total	1243.66	37.27	342.46
	Spectra count at the end of stream	10	9	10
EP	Total	2037.66	92.64	803.38
	Spectra count at the end of stream	10	10	10

4.3 Sensitivity analysis

The sensitivity analysis was done in two phases. Firstly, the effects of the threshold firing parameters α and β were investigated on all datasets and then, results were

presented for two representative datasets, namely noisy RBF and Elec. The second phase is focused on analysing the effect of the permitted maximum for the number of spectra, or repository size. The results are presented for noisy RH, 10% progressive RH, Flight, and Covertype.

Table 4.9 shows that the noisy RBF dataset throughput is sensitive to the cut off value used for α. As α is increased from a low value of 0.5 to 0.7 there was a 49.0% loss in throughput while not having a significant difference in accuracy. This throughput loss is to be expected as it delayed the T_1 for further 40,000 instances. This exemplifies the negative effects of too high cut-off value for α. The same effects of the α parameter were also seen for the Elec dataset with a cut-off value of 0.7 that inhibited the firing of trigger T_1, resulting in lower throughput. The effects of α on accuracy were very marginal, as illustrated by Table 4.9.

Table 4.9 shows that throughput was also sensitive to the value of β, albeit to a lesser extent than with the α parameter. With RBF, a β setting of 0.5 caused throughput to slight increase by 3.1% from its default value with the 0.7 setting. The lower setting for RBF allowed it to stay in Stage 2 for a little longer (as trigger T_2 was not activated), thus resulting in better throughput. On the other hand, a 0.5 setting of β for the Elec dataset did not cause any difference in triggers compared to its default setting. This difference in behavior is due to the difference in the dynamics of the two datasets - the lower setting for Elec had no effect on T_2 as the concept recurrence level and concept similarity were little lessor than with RBF, thus enabling it to remain in Stage 2 for the same length of time as with the higher setting for β. Table 4.9 shows that the effects of β on accuracy were also marginal, just as with α.

As depicted in Table 4.10, 10% progressive RH and Flight datasets needed a certain amount of spectra to stay in Stage 2; with size 10 the throughput increased by 68.7%

Table 4.9: Effect of Alpha and Beta on the Noisy RBF dataset and Elec dataset

Noisy RBF dataset		T_1	T_2	Throughput	Accuracy
α	0.6	same as default			
	0.7	120000	–	2268	76.1
β	0.5	80000	–	4590	76.5
	0.8	80000	520000	4015	76.5
Default setting α = 0.5 , β = 0.7		80000	560000	4451	76.6
Elec dataset					
α	0.6	same as default			
	0.7	24000	32000	12887	67.6
β	0.5	same as default			
	0.9	8000	36000	22654	67.2
Default setting α = 0.5 , β = 0.7		8000	44000	25661	67.1

and 4.4% respectively over the throughput obtained with size 5. However with the Covertype dataset having 5 spectra was better in terms of speed while not losing significant accuracy. Having more spectra, especially in the case of datasets with more attributes also results in speed disadvantages, which should be avoided. As with the other two parameters, the effects of repository size on accuracy were marginal.

Table 4.10: Effect of repository size on the 10% progressive RH, Flight, and Covertype datasets

Dataset	Repository Size	T_1	T_2	Throughput	Accuracy
10% progressive RH	5	160000 440000	320000	2656	76.4
	10	80000	–	4480	74.0
	40 (Default)	160000	–	1991	77.0
Flight	5	4000 18000	16000	4016	82.5
	10	4000	–	4195	81.8
	40 (Default)	4000	–	4331	81.8
Covertype	5	12000 28000	20000	1081	84.4
	10	12000 36000	20000	663	84.8
	40 (Default)	12000 36000	20000 48000	663	84.8

4.4 Conclusion

This chapter presented an in-depth examination of the staged learning paradigm that uses a two staged approach to learning in a data stream environment. The results demonstrated the effectiveness of the staged learning approach in terms of processing speed and memory usage without compromising on classification accuracy. The gain is significant for data streams that exhibit periods of low volatility and hence the detection of such periods of low volatility is of critical importance to the staged approach. Classification within those low volatility periods can be effectively dealt with through exploitation of concept recurrence.

The volatility detection strategy proved to be effective in identifying low volatility states that enabled the computationally expensive ensemble learning component to be suspended, thus directly contributing to the performance gains that were obtained. Likewise, precise recognition of the high volatility stage avoided potential accuracy drops by ensuring the timely reactivation of Stage 1 which is indispensable to learning a new batch of concepts unseen in the past.

Moreover, empirical results confirmed the robustness of the staged learning platform under a variety of challenging recurrence scenarios such as patterns repeating with noise, patterns repeating with monotonically increasing drift intensity, as well as oscillating patterns. It also demonstrates the capability of capturing recurrences across a spatial dimension.

The sensitivity analysis conducted on three critical system parameters: α, β, and repository size demonstrated the influence of these parameters on performance.

In summary, the empirical study has shown that the staged learning paradigm of tailoring the learning strategy to the level of volatility in the stream has significant

performance benefits in terms of throughput and memory savings while resulting in a better accuracy.

Chapter 5

Incremental Fourier Classifier

5.1 Introduction

In the previous chapter, the Staged Online Learning (SOL) framework which operates in two stages based on the volatility of data stream was presented. The major difference between classical data stream mining frameworks and SOL is that the latter takes advantage of periods with a lesser number of previously unseen concepts. When the majority of concepts have been seen previously, a computationally expensive group of experts approach is redundant. For this reason, the forest of decision trees could be replaced with a single tree in such periods. Even though the results presented in chapter 4 were promising, the repeated two-way transition between decision tree and best performing spectrum at each and every concept drift point in Stage 2 is sub-optimal due to the spikes in processing time at drift points. Therefore, this chapter proposes an extension to the naive version of SOL introduced in chapter 3.

The extension, named Staged Online Learning with Incremental Fourier Classifier (SOL-IFC), presents a novel incrementally adaptive Fourier spectrum scheme that operates in low volatility stream segments. To the best of our knowledge, an incremental, adaptive Fourier spectrum strategy for data classification has not been proposed in

previous research. Along with the incremental approach, this chapter also proposes schemes for pruning noisy features and synopsis generation that enable the coefficient array to be refreshed efficiently on a periodic basis.

In order to present the rationale behind Incremental Fourier Classifier (IFC) in SOL-IFC, the chapter starts with a discussion of the use of Discrete Fourier Transform (DFT) in SOL and SOL-IFC contexts, followed by a comparison between Fourier classification and decision tree classification.

5.2 Application of DFT in SOL vs its application in SOL-IFC

Several attractive properties of the DFT in data stream environment have been exploited in mining data streams (Sakthithasan et al., 2015; Kithulgoda & Pears, 2016), but their use comes at a price. The application of the DFT on multivariate data to produce a spectrum is a non-trivial operation and has time complexity $O(|X|^2)$, where $|X|$ is the size of the feature space (Kargupta & Park, 2004). The size of the feature space $|X|$ grows exponentially with the dimensionality of the data. In a highly dynamic data stream environment where there are frequent concept drifts, the time spent on repeated application of the DFT at each drift can quickly become prohibitive as shown in section 6.5. A much more effective strategy would be to incrementally maintain a spectrum in a fashion analogous to the incremental maintenance of a conventional classifier such as a decision tree.

SOL-IFC draw on the SOL approach proposed in chapter 3 that uses a two-staged approach to data stream classification. The SOL approach divides data streams into two types of segments based on the level of volatility in the stream, which is measured by the rate of appearance of new concepts in the stream. Stage 1 represents a high

volatility phase while Stage 2 operates in a low volatility phase. The IFC is deployed in the low volatility phase for two reasons. Firstly, refining an already established spectrum by making small incremental changes to it, is far more attractive than having to generate a new spectrum from a decision tree. The experimental results in chapter 6 supports this premise very strongly. Secondly, the underlying philosophy of the staged approach is that Stage 1 provides a basis for Stage 2 by capturing new patterns as they arrive in a high volatility stage. These patterns, stored as spectra can then be refined in an incremental manner in the low volatility Stage 2 phase without compromising on accuracy.

5.3 Advantages of the Fourier Classifier over the Decision Tree Classifier

Two major performance bottlenecks with decision tree based classifiers in a data stream environment are the depth of the tree and the update overhead of maintaining leaf node statistics on an instance-wise basis to ensure that classification is consistent with the current state of the data stream. The proposed classifier has been chosen to contrast with the decision tree classifier. The decision tree classifier has been shown in a number of empirical studies to be one popular choice for a data stream environment (Gama, Rocha & Medas, 2003; Kargupta & Park, 2004; Bifet et al., 2009). This is due to two important reasons. Firstly, decision tree induction is efficient, and when used in conjunction with the Hoeffding bound (Hoeffding, 1963), can adapt to new patterns in a data stream with a single pass through the data. Secondly, they capture dependencies between features without the need for time consuming graph traversals, such as in neural networks or Bayesian networks.

5.3.1 Decision Tree overhead

Although decision trees are efficient when compared with other types of conventional classifiers, they still have shortcomings that negatively impact on performance. The depth of the tree directly impacts classification performance as the label for a data instance can only be determined by a traversal from the root node to the node that matches with the schema of the given data instance. The deeper the tree, the greater is the number of intermediate nodes that need to be traversed to extract the required class label.

Furthermore, once the true label for a data instance is known, the sufficient statistics, n_{fvc} (Domingos & Hulten, 2000) which record the number of data instances for each feature f taking value v that belong to class outcome c, need to be updated. These statistics are required to determine whether a leaf node should split into two or more decision nodes. The average time complexity of this update is:

$$O(dvc) \tag{5.1}$$

where d is the dimensionality (number of features in the data), v is the average cardinality taken over all features and c is the number of classes. This update needs to be performed on a per-instance basis even though splits may only occur in a small percentage of cases, thus exposing a fundamental performance weakness of the decision tree method when operating in a data stream environment. In many implementations of decision tree classifiers for a data stream environment, a forest of trees is used and this will further increase overheads by a factor proportional to the size of the forest. Thus overall, the appearance of each data instance involves an overhead of $O(dvc)$ in the case of a decision tree.

5.3.2 Fourier Classification overhead

The Fourier classifier does not rely on a deep hierarchical tree structure but instead uses a self indexing hashing scheme to store its schema values. Access to schemas is provided via a shallow two level extendible hashing scheme. The biggest advantage of such a two level structure is that classification is very efficient when the schema for the data instance to be classified is cached in the reservoir described in section 5.5.2. However, when the spectrum is updated on a periodic basis, classification needs to be performed once again for data arriving after the update point. In this case, classification is performed via the Inverse Fourier Transform (IFT) operation.

If s is the size of the coefficient array (number of coefficients) then the time complexity of IFT from Eq. 2.3 is:

$$O(sd^2) \tag{5.2}$$

as d^2 multiplication are needed for the computation of the vector product between $\vec{x}$ and $\vec{j}$ for each of the s coefficient in the array.

Although a direct comparison between the decision tree and the Fourier classifier is not possible in terms of time complexity, it is observed that there is no control over any of the factors governing Eq. 5.1, whereas in Eq. 5.2, the time complexity can be controlled by limiting the size s of the array as well as the dimensionality of the data d. Pruning of the coefficient array can be accomplished effectively by energy thresholding. In addition to energy thresholding discussed in section 2.5, this chapter proposes a schema pruning scheme in section 5.5.1 that complements on the energy thresholding scheme proposed by Kargupta and Park (2004).

In SOL-IFC, two types of pruning are used. Firstly, by reducing the size s of its coefficient array it reduces the time complexity of the IFT, leading to smaller classification times after a spectral update is performed. The second type of benefit is that it reduces overhead by excluding some of the d features from the computation of

the vector product between $\vec{x}$ and $\vec{j}$. This is done by eliminating features that do not contribute to the classification process. The details are given in section 5.5.1.

5.3.3 Fourier Coefficient Array update overhead

Just as with the decision tree the Fourier classifier needs to update its model by refreshing the coefficient array. Each coefficient of the array plays a role similar to the leaf node of a decision tree classifier but there is no need to store sufficient statistics for each coefficient apart from its numeric value. Thus the cost of accessing a large 3-dimensional array is avoided during the model update operation.

From Eq. 2.4, we observe that the time complexity for the update operation is $O(nd^2|X|)$ where n is the number of coefficients affected by the update and X is the schema set. Each coefficient to be updated requires $d^2 \times |X|$ multiplications to compute the new value of the coefficient that is being updated.

Our strategy for reducing the update overhead is two-fold. Firstly, as noted in section 5.3.2, feature selection enables us to eliminate some of the d features from the vector product operation. The reduction factor R_1 obtained through feature selection is:

$$R_1 = \frac{|X|}{|F|} \tag{5.3}$$

, where $|X| = \prod_{i \epsilon X} \lambda_i$ and $|F| = \prod_{i \epsilon F} \lambda_i$, where F denotes the subset of features (drawn from feature set X) that survive the feature selection process described in section 5.5.1.

Secondly, Theorem 1 in section 5.4.1 enables us to prune the schema set X. It does this by identifying a subset C of X consisting of those schemas that change their class labels from the last observed interval. The Theorem ensures that the coefficient array can be updated efficiently without compromising on classification accuracy. Theorem 2 further constrains the size of C when certain conditions are met.

If C is the subset of X identified by Theorem 1 (which we present in section 5.4.1)

over which the update operation must be performed, then the reduction factor obtained through schema pruning is:

$$R_2 = \frac{|X|}{|C|} \tag{5.4}$$

In stream segments that exhibit a low level of change, we expect that only a small fraction of schema will change their class labels, i.e. $|C| \ll |X|$ and so $|R_2| \gg 1$. The reduction factors R_1 and R_2 are independent of each other as the class label change rate for a given schema (from one interval to another) is an inherent property of the data and is not influenced by the information content of individual features. Thus the cost of the update operation reduces by a factor $R_1 \times R_2$ with the help of the two optimisations. Our experimentation in chapter 6 will quantify the performance gains from feature selection and the incremental coefficient update strategy.

With these optimisations, SOL is enhanced with IFC to enable a trade-off between accuracy and efficiency to be made in a concept changing data environment. A detailed presentation of the novel incremental Fourier classifier approach follows.

5.4 Incremental approach to Fourier Spectrum maintenance

Real-time classifier update techniques have been extensively researched in the domain of data stream classification as discussed in section 2.4. In incremental learning, small-scale changes are made on the decision model over time on a continuous basis rather than recreating models from scratch.

Even though the SOL proposed in chapter 3 yielded good savings in time and memory, several inefficiencies remain. Firstly, in the low volatility state, the spectrum (S) that most closely fits is used to induce a decision tree (T). This induction process

can involve high computational overhead, depending on the actual size of the spectrum.

Secondly, when concept drift takes place from an existing concept C to a new concept C', the tree T that has evolved to a new version T^c that embodies changes from the former concept C will need to be converted to spectral form S^c, so that it can be stored in the repository for future use if C recurs. At the same time, a conversion from S' to tree T' is needed in order to learn future changes in the new concept C'. Such conversions from spectrum to tree and tree to spectrum can be avoided by tracking changes in the raw data and translating such changes to the spectrum without the need for transformations to or from the tree to spectral domains.

There has been very little research done on optimizing the learning process within the Fourier domain. None of those studies (Park, 2001; Kargupta & Park, 2004; Kargupta et al., 2006; Sakthithasan et al., 2015; Kithulgoda & Pears, 2016) have worked on an incremental adaptation of the Fourier spectrum. As Park (2001) says, learning directly in the Fourier domain is challenging in a high dimensional data environment and optimization strategies such as gradient descent were used to update the coefficient array. This work shows that there is no need for using such computationally complex methods.

5.4.1 Incremental maintenance of Spectra

Spectra in the repository can be maintained in one of two ways. The first approach taken by Sripirakas and Pears (2014); Sakthithasan et al. (2015), and Kithulgoda and Pears (2016) is to select the best performing tree from a decision tree forest at each concept drift point and apply the DFT to produce a spectrum. This spectrum is then aggregated with the most similar spectrum already resident in the repository through the use of a Euclidean distance measure discussed in section 2.5.1. There are two major issues with this approach. The first is the high overhead in applying the DFT. Secondly,

the maintenance operation is not incremental as an existing spectrum is updated in its entirety with a newly generated spectrum from the best performing tree.

The second approach to maintenance exploits the fact that in any given stream segment only a subset of schema instances will experience a change in their class values over the previous segment. Thus, instead of treating the spectrum that covers the data in the newly arriving data segment as being a new spectrum in its own right, this work treats it as an extension of the spectrum that was generated from the previous segment.

For that reason, this study proposes to deploy these extended spectra which are considered as incremental Fourier classifiers in low volatility segments of a data stream in SOL. The proposed method adapts SOL framework described in section 3.2.2 as depicted in Fig. 5.1.

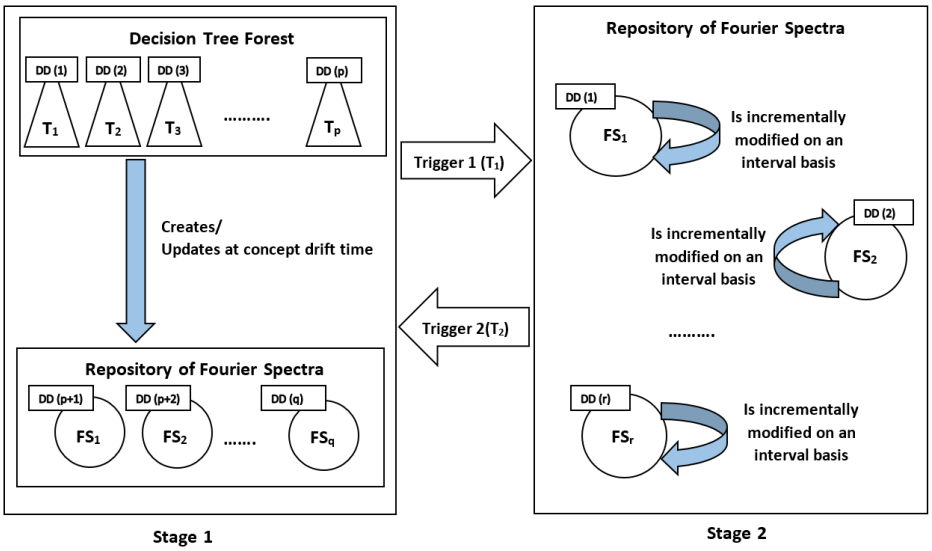


Figure 5.1: Staged Transition Learning framework adapted from chapter 3

Figure 5.2 illustrates the incremental update process that is carried out periodically in intervals. Schema instances x that have changed their class labels $f^{current}(x)$ in the current interval are used in conjunction with $f^{previous}(x)$ from the previous interval, together with the current version of the spectrum array $w_{\vec{j}}^{current}$ to compute the new version $w_{\vec{j}}^{new}$ that will be used in the next interval. Theorem 1 quantifies the update

computation and provides a proof of its correctness.

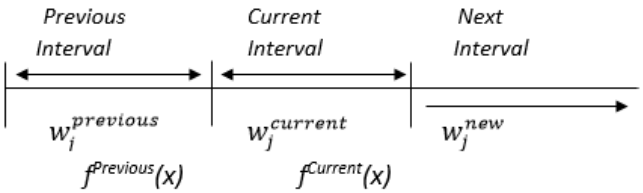


Figure 5.2: Periodic update of spectra

Theorem 1: Let C be the change set containing all schema instances that have had their class labels changed since the previous update point. The new version of the coefficient array $w_{\vec{j}}^{new}$ can be derived from its current version $w_{\vec{j}}^{current}$ as follows:

$$w_{\vec{j}}^{new} = w_{\vec{j}}^{current} + \frac{1}{|X|}\sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \tag{5.5}$$

,where $f^{current}(x)$, $f^{previous}(x)$ are the class labels in the current and previous intervals respectively for a given schema x.

Proof

$$\begin{aligned}
w_{\vec{j}}^{new} &= \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)f^{current}(x) \\
&= \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&+ \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)f^{previous}(x) \\
&= \frac{1}{|X|}\sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&+ \frac{1}{|X|}\sum_{x\in X\setminus C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&+ w_{\vec{j}}^{current}, \text{ as } w_{\vec{j}}^{current} = \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)f^{previous}(x) \\
&= w_{\vec{j}}^{current} + \sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&\text{as } f^{current}(x) = f^{previous}(x)\forall x \in X \setminus C
\end{aligned}$$

In certain cases, it is possible to optimize further by restricting the refresh operation

to only a subset of coefficients in the coefficient array. Definition below establishes when this optimization applies and how this subset can be identified.

Definition (Pivot) *A feature p is said to be a pivot on the schema set X if* $\forall x \in C$ $\exists y \in C$ *such that*

1. x and y differ only in index position p
2. $f(x) = f(y)$
3. $\bigcup_{x(p) \in C} = \{0, 1, \cdots, \lambda_p - 1\}$, where $x(p)$ is the p^{th} position of a schema instance x

Suppose a pivot feature p exists with change set C. Theorem 2 below identifies the spectrum coefficient subset that is unaffected by the class label changes amongst schema recorded in the change set C.

Theorem 2: If a feature p is a pivot in the previous interval, and remains as a pivot in the current interval then the set of coefficients U that is unchanged is given by $w_{\bar{j}}$ with $j(i) = * \ \forall i \neq p$ and $j(i) \in \{1, 2, \cdots, \lambda_p - 1\}$ when $i = p$.

Proof

Consider an arbitrary $\vec{j} \in U$

$$\psi_{\vec{j}}(x) = \sum_{\vec{x} \in C} \prod_{l=0}^{d-1} \cos\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big)$$
$$+ i \sum_{\vec{x} \in C} \prod_{l=0}^{d-1} \sin\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big)$$
Define $F = \{0, 1, \cdots, d-1\}$
$$\text{Let } t_1 = \sum_{\vec{x} \in C} \prod_{l \in F} \cos\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big) \tag{5.6}$$
$$\text{and } t_2 = \sum_{\vec{x} \in C} \prod_{l \in F} \sin\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big)$$
$$\text{Let } m = \prod_{l \in F \setminus \{p\}} \cos\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big)$$
$$\text{Then } t_1 = \sum_{x(p) | \vec{x} \in C} m \cos\Big(\frac{2\pi j(p)x(p)}{\lambda_p}\Big)$$

We are interested in coefficients $j \in U$ such that $j(p) \neq 0$ Now with $k = j(p)x(p)$ and $N = \lambda_p$, Eq. 5.6 can be written in the form:

$t_1 = m \sum_{k=0}^{N-1} \cos(\frac{2\pi k}{N}) = 0$ for all integers k and N (Knapp, 2009).

Likewise, $t_2 = m\Big(\sin\Big(\frac{2\pi j(p) \times 0}{\lambda_p}\Big) + \sin\Big(\frac{2\pi j(p) \times 1}{\lambda_p}\Big)$
$+ \cdots + \sin\Big(\frac{2\pi(j(p) \times (\lambda_p - 1))}{\lambda_p}\Big)$
$= m \times 0$ as $\sum_{k=0}^{N-1} \sin(\frac{2\pi k}{N}) = 0$

for all integers k and N, also from Knapp (2009).

Thus $\psi_{\vec{j}}(x) = 0 \ \forall x \in C$

From Theorem 1 above, we have $\forall j \in U$,

$$\begin{aligned} w_{\vec{j}}^{new} &= w_{\vec{j}}^{current} \\ &+ \sum_{x \in C} \psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\ &= w_{\vec{j}}^{current} \text{ as } \psi_{\vec{j}}(x) = 0 \ \forall x \in C \text{ and } j \in U \end{aligned}$$

The utility of Theorem 2 is illustrated in the following example which uses binary data to maintain simplicity.

Suppose there is a 3-dimensional dataset with X_2 as a pivot in the previous interval. The Theorem then deals with scenarios such as given below.

Class labels in previous interval: $f(**1) = 0, f(**0) = 1$.

As feature X_2 was a pivot in the previous interval $f(*0*) = f(*1*)$.

Now suppose that the class labels in the current interval are: $f(**0) = 1, f(0*1) = 0, f(1*1) = 1$.

Hence change set $C_{PreviousToCurrent} = \{101, 111\}$.

It has been observed that feature X_2 remains as a pivot in the current interval.

Now from Theorem 2, we can conclude that coefficients given by $w_{\vec{(*1*)}}$ do not need to be updated.

Efficient and effective recognition of the change set is facilitated by hashing and reservoir management process as explained below.

5.5 Hashing and Reservoir management

This section describes two further optimisations applied on IFC as introduced in section 5.3.3: features selection based schema hashing, and capturing change set with the aid of hashing and synopsis generation.

5.5.1 Schema Hashing through Feature Selection

Feature selection strategy is based on identifying the subset of features that play a critical role in the classification process. In the context of classification via a Fourier spectrum, feature selection means the selection of features which contribute to the IFT operation, as given in Eq. 2.3. Kargupta et al. (2006) proved that when a feature X_k does not appear in a decision tree then all coefficients $w_{\vec{j}} = 0$, whenever $j(k)$ take a non zero value, irrespective of the values taken by other elements of $\vec{j}$. With respect to coefficients with $j(k) \neq 0$ their contribution to the IFT is hence zero.

Further to this, consider the remaining set of Fourier coefficients where $j(k) = 0$ at feature X_i's position. Even though the Fourier coefficients w_j take non zero values, the Fourier basis function has no contribution from such coefficients as shown below. On this basis, it is concluded that features that do not appear in decision trees in Stage 1 play no part in the classification process. This property is exploited when obtaining the hash function as described below.

Eq. 2.2 is applied on the subset of coefficients where $j(k) = 0$. If $k \neq 1$, $\vec{j}$ can be reordered to make $k = 1$ then without loss of generality.

$$\phi^{\vec{\lambda}}_{\vec{j}}(\vec{x}) = \frac{1}{\sqrt[d]{\prod_{l=1}^{d} \lambda_l}} \prod_{m=1}^{d} \exp\left(\frac{2\pi l x(m) j(m)}{\lambda_m}\right)$$

Let $r(m) = \dfrac{2\pi l}{\lambda_m}$

$$\text{Now} \quad \phi^{\vec{\lambda}}_{\vec{j}}(\vec{x}) = \exp\left(r(1)x(1) \times 0\right) \frac{1}{\sqrt[d]{\prod_{l=2}^{d} \lambda_l}} \prod_{m=2}^{d} \exp\left(r(m)x(m)j(m)\right)$$

$$= 1 \times \frac{1}{\sqrt[d]{\prod_{l=2}^{d} \lambda_l}} \prod_{m=2}^{d} \left(\exp\left(r(m)x(m)j(m)\right)\right)$$

Thus the Fourier basis function's contribution to coefficients where $j(i) = 0$ has not been affected by the X_l^{th} feature.

This feature selection is done, besides the feature selection done in Stage 1 prior to transformation of a decision tree to its corresponding spectrum. In Stage 1, all features that appear in a tree are stored along with its corresponding spectrum. When one spectrum is aggregated with another, a union of the feature sets of the two spectra that are aggregated is stored with the new aggregated spectrum.

Knowing the set of influential features per spectrum, a hash scheme for efficiently storing schema instances can be devised. Given a schema instance, a hash function for the i^{th} spectrum S_i is the string $\bigcup_{j=1}^{n} Y(j)$, where $Y(j) = X(j)$ if feature j survives the feature selection process, otherwise $Y(j) = 0$. The rationale for this is that all schema instances will take the same class value regardless of the non contributing feature values. Hence only one instance indexed by a position of 0 for such feature will suffice. For instance, if we consider the binary tree given in Fig. 2.2, X_1 and X_3 are influential features but not X_2 as it does not appear in the tree and hence does not play a part in the classification process. Thus it follows that the hash function for the spectrum is $H(X_1(x)X_2(x)X_3(x)) = (X_1(x)0X_3(x))$.

5.5.2 Reservoir organization

A critical element of incrementally adapting the Fourier spectrum to changes in the data stream is a dynamic data structure that captures changes in the schema over a period of time. Fig. 5.3 shows a dynamic reservoir structure that keeps track of the evolution of schema over time based on an extendible hash implementation. Each and every spectrum in the repository has its own reservoir built using the hashing scheme described earlier.

The reservoir is compact for two reasons. First of all, it only stores schema and

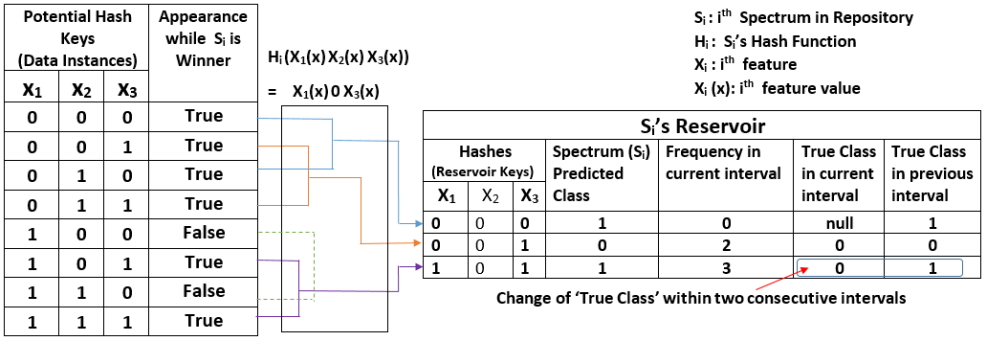


Figure 5.3: Hashing and Reservoir structure

not data instances, in a manner analogous to a decision tree that only stores decision paths and not actual data instances. Secondly, feature selection ensures that the worst case space complexity of the reservoir is $O(|X|)$. Thus for the example shown in Fig. 5.3, the appeared schema reduces in size from 6 to 3 as feature 2 plays no part in classification and hence both 0 and 1 occurrences of this feature maps to the same class, thus requiring only one representation for this feature, which we choose to be 0.

Although the schema set can be sparse, the reservoir is compact as only a subset of schema instances may actually manifest in the stream. For this example schema set, although there are a total of 8 possible schema instances, only 6 may actually appear in the stream, as illustrated in Fig. 5.3. This calls for a dynamic data structure that only utilises space as and when needed, and hence our use of a hashing scheme. The extendible hashing scheme also ensures that an entry in the reservoir can be extracted with no more 2 memory fetches.

Each spectrum in the repository will be stored its own reservoir according to the structure defined in Fig. 5.3. At any given point in time, only the winner spectrum undergoes a change; all other spectra in the repository remain static until one of them emerges as the winner. Each entry in a reservoir is indexed by its hash value, the class value of the schema at its first appearance, its frequency of appearance, the true class label within the current interval if it appeared, and finally the true class label within its

previous appearance if any.

The reservoir serves two main purposes. Firstly, it helps to determine when spectra should be refined, and secondly, it provides a cache for extracting the class label whenever several data instances recur within the same interval with the same schema, or when schema only differ in one or more index positions involving non contributing features. In this respect the higher the number of collisions in the hash table, the better the efficiency. For any given schema, the class value returned by the IFT will remain the same and will represent the predicted class value as long as the spectrum remains unchanged.

5.6 Algorithm

The discusion on SOL-IFC approach in sections 5.4 and 5.5 are presented in two algorithms 3 and 4. The pseudocode given in algorithm 3 presents the reservoir management and classification process.

As given in steps 6-10 of algorithm 3, if the hash value returned by the corresponding spectrum's hash function already exists in that spectrum's reservoir, instead of reclassifying it class value is simply extracted from its first appearance. If no entry exists, then it is necessary to classify and insert a new record. If the winner is not in a drift state and a threshold I on the number of instances arrived (steps 12-14), then the Refine Winner procedure which is given in algorithm 4 is activated.

In algorithm 4, the requirements for refinement are first tested in step 8. In case the cumulative change rate is higher than the tolerable change rate T, the coefficients of the winner spectrum will be modified by utilizing an incremental factor, as shown in step 17 of Algorithm 4. At this point in time, the reservoir is reset by clearing its contents (step 17 of algorithm 3).

If it fails to meet the conditions for refinement, subsequent tests for refinement will

Algorithm 3 Incremental Fourier Classifier

Input: Interval Length I, Spectrum Repository, $InstanceCount$ = 0, $EffectiveChangeRate$ = 0

```
1: repeat InStage2
2:     Read Instance
3:     Increment InstanceCount by 1
4:     for all Spectrum S_i in Repository do
5:         Apply S_i's Hash Function on Instance
6:         if Hash Value found in S_i's Reservoir then
7:             Extract predicted class value, update frequency and True class columns
8:         else
9:             Classify by IFT and insert into Reservoir
10:        end if
11:    end for
12:    if Winner is not in drift then
13:        if Instance Count MOD Interval Length = 0 then
14:            EffectiveChangeRate= RefineWinner (EffectiveChangeRate)
15:            if WinnerRefined then
16:                Reset Instance Count = 0
17:                Clear Winner's Reservoir
18:            else
19:                for all Winner Reservoir Record do
20:                    Previous True Class = Current True Class
21:                    Current True Class = null, Frequency = 0
22:                end for
23:            end if
24:        end if
25:    else
26:        Define spectrum with the highest accuracy as New Winner
27:        Aggregate Previous Winner with most similar Spectrum if such exist
28:        Reset InstanceCount =0, Reset EffectiveChangeRate=0
29:        Clear New Winner Spectrum's Reservoir
30:    end if
31: until InStage2
```

be done using the accumulated change rate in intervals of size I.

Algorithm 4 Refine Winner

Input: Tolerable Change Rate, $WinnerRefined = False, NumberofChanges = 0$

for all Winner Reservoir Hash Value appeared in Current Interval **do**
  if Current True Class is not equal to Previous True Class **then**
    Increment $NumberOfChanges$ by Frequency
  end if
end for
$ChangeRate = NumberOfChanges/$ Interval Length
$CumChangeRate = EffectiveChangeRate + ChangeRate$
if CumChangeRate > Tolerable Change Rate **then**
  for all Coefficient Index j that belongs to Winner Spectrum **do**
    $IncrementalFactor = 0$
    for all Reservoir Hash Value x where Classes are changed **do**
      Calculate $ClassDifference = (f_{(x)}^{current} - f_{(x)}^{previous})$
      Compute $F = \psi_j(x) \times ClassDifference$ where $\psi_j(x)$ is from Eq. 2.2
      ▷ Update Incremental Factor in Eq. 5.5 as $\sum_{x \in C} \psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x))$
      $IncrementalFactor = IncrementalFactor +$ F
    end for
    Compute $AverageIncrementalFactor$ as $\frac{1}{|X|} IncrementalFactor$
    ▷ where $|X|$ is the current reservoir size
    $w_j^{new} = w_j^{current} + AverageIncrementalFactor$
    ▷ as in in Eq. 5.5
  end for
  Set $WinnerRefined = True$, Reset $CumChangeRate = 0$
else
  Set $WinnerRefined = False$
end if
  return $CumChangeRate$

5.7 Conclusion

This chapter presented a novel approach to classification that exploits the DFT called SOL-IFC. This approach believes that most data streams either exhibit recurrence of previous concepts, closely resemble, or are derivatives of previously seen concepts. In stream segments exhibiting recurrence of previous concepts, SOL-IFC re-uses previously stored spectra that most closely matched with the recurring concept.

On the other hand, when the current stream segment does not closely resemble any

of its stored concepts from the past, SOL-IFC refines the spectrum that it currently operates on in order to adapt to concept change in the stream. In cases where completely new concepts appear in the stream, SOL-IFC suspends its operation via a transit to Stage 1, in order to learn concepts with the help of the decision tree forest.

The chapter also discussed how the spectrum refinement process was optimised by introducing a self indexing hashing scheme that efficiently references a reservoir containing key statistics that quantify changes occurred in a given time window. In order to guarantee fast access to the reservoir, a schema pruning strategy which ensures the removal of noisy features was also implemented.

As a positive consequence of the above mentioned enhancements, namely, self indexing hashing scheme and schema pruning strategy, the time efficiency of the IFT process was increased in two ways. Firstly, through the use of the hash function, the number of different schemas is reduced and hence the likelihood of finding the result of the IFT in the reservoir was increased. Also, the removal of noisy features reduces the effective schema length involved in the IFT calculation and thereby reduces classification time.

The effect of these three novel conceptual contributions will be examined through an extensive experimental study reported in the subsequent chapter.

Chapter 6

An Empirical Study of the Incremental Fourier Classifier

6.1 Introduction

This chapter presents an empirical study that examines the effectiveness of SOL-IFC in terms of classification accuracy and throughput. An examination of SOL-IFC's accuracy against eleven other adaptive classifiers that have been previously proposed for concept drifting data streams is undertaken. Nine of those algorithms were chosen from state of the art meta learning algorithms featured in MOA[1]. The SOL, predecessor of SOL-IFC introduced in chapter 3, and an algorithm named Recurrent Classifier (RC) were also used as comparators. The experiments were carried out on two synthetic datasets and five real-world datasets described in section 6.1.2.

In addition to the accuracy study, the trade-off between throughput and accuracy of the best performing algorithms (defined in terms of accuracy) was also examined. Finally, section 6.5 presents an in-depth comparison of SOL-IFC over SOL with the focus on execution time and CPU utilisation.

[1] from *http://moa.cms.waikato.ac.nz/*

6.1.1 Algorithms used in study

A brief explanation of the twelve algorithms that we experimented with is given below.

1. Accuracy Weighted Ensemble (AWE):

   This is one of the highly cited ensemble algorithms presented by H. Wang et al. (2003) for concept drifting data stream processing. This study also presents a theoretical proof for the capability of reducing classification errors with an ensemble classifier.

   The models are built sequentially on equal sized partitions of the stream. Each classifier's mean square error determines the weight of its contribution to the final class label. In order to limit the number of classifiers in the ensemble, AWE tests how confident it is to make the same classification decision with and without that classifier under consideration of removal. A decision on a class outcome of a given instance was taken by conferring classifiers one after another in descending order of their weights until it reached a certain user-defined confidence about the outcome. The authors adopted a pruning algorithm from Fan, Wang, Yu, Lo and Stolfo (2002).

   Even though an explicit concept drift detector has not been used, authors claim good adaptability via combining models based on weights. In their original experimental study, C4.5 (Quinlan, 1993) and a few other base classifiers were used. In the comparative study given in sections 6.2 and onwards, Hoeffding tree was used as the base classifier in order to be consistent with all other approaches.

2. OzaBagASHT (AS):

   The study of Bifet et al. (2009) has presented an algorithm named Adaptive Size Hoeffding Tree (ASHT) which features two variations when compared to the original Hoeffding tree (Domingos & Hulten, 2000). The ASHT has a maximum number of split nodes as a parameter, that is, its 'size'. If the size exceeds its

maximum setting after node splits, some nodes are then deleted. Two deletion choices are recommended: deletion of all nodes except for the latest split node or the removal of the entire tree, followed by regrowth from the new root.

The individual ASHTs are combined by the use of a novel bagging strategy which ensures a pool of trees with higher diversity than could be obtained through standard bagging methods (Oza, 2005). A higher diversity is obtained by having trees of different size. The size of the initial tree in the ensemble is two and thereafter the size of every subsequent tree is doubled. The trees are weighted according to the inverse of the square of their errors obtained after applying an exponential weighted moving average on error. In this algorithm, no drift detector was used.

3. OzaBagADWIN (OB):

   The OzaBagADWIN in Bifet et al. (2009) is an integration of the well-known online bagging algorithm of Oza (2005) and the ADWIN change detector presented in Bifet and Gavalda (2007). The worst performing learner is removed and a new learner is added at each drift point in the stream.

   The study suggests the use of variations on the basic Bagging algorithm for situations where the focus is not on processing speed or memory consumption.

4. LeveragingBag (LB):

   The algorithm description is given in section 4.2.1.

5. LimAttClassifier (LA):

   This ensemble learner presented in Bifet, Frank et al. (2010) consists of Hoeffding trees (Domingos & Hulten, 2000) built on all possible attribute subsets of some user-defined size that is small. Per-instance basis class predictions are collected from each tree in the ensemble to train perceptron classifiers. This training uses stochastic gradient descent with a learning rate $\frac{2}{2+m+n}$, where m is the number of attributes and n is the observed training instances by the time. In order to avoid

excessive delay in convergence for the perceptron, the algorithm resets n at each drift.

The ADWIN (Bifet & Gavalda, 2007) drift detector is the change detection mechanism. Other than resetting the learning rate n parameter, poorly performing trees are also reset by replacing the tree with its root node at drifts.

The subset size of '2' is recommended as a practical value for relatively higher dimensional datasets while larger subset sizes are recommended for datasets with small dimensionality. The authors of this algorithm also conclude the importance of using an apt relevance measure for combining individual tree classification result when deciding classification output for the ensemble rather than the use of a simple averaging method.

6. Accuracy Updated Ensemble (AUE):

   The algorithm presented in Brzeziński and Stefanowski (2011) is an extension of the above given AWE (H. Wang et al., 2003). In contrast to AWE's adjustable weight driven adaptability, AUE maintains an online classifier that adapts itself if changes are observed between consecutive instance blocks. In addition, AUE only updates the weights of selected classifiers that reports adequate accuracy. Their experimental study showed accuracy improvements compared to AWE while consuming similar amounts of processing time and memory.

7. Dynamic Adaptation to Concept Changes (DA):

   The study of Jaber, Cornuéjols and Tarroux (2013b) explores various forgetting mechanisms that decide which ensemble leaners be removed while learning takes place. Highlighting the limitation of removing the worst learner, the study introduces arbitrarily selection of one model from the subset of poorly performing learners for the removal.

   Experimentation on different subset sizes revealed '1' as the best subset size for stationary environments while half size of learners was optimal for concept

drifting streams. The deletion of one of the poor learners and the insertion of a new learner was used as the method of coping with drift.

8. Anticipative Dynamic Adaptation to Concept Change (AD):

   The Anticipative Dynamic Adaptation to Concept Change (ADACC) algorithm presented in Jaber et al. (2013a) is an extension of the work DA (Jaber et al., 2013b) described above. This study integrates an approach of storing critical past concepts and recognising reappearance of those concepts in the stream.

   The decision of storing a given model is taken on the basis of the difference between the Kappa statistic (Carletta, 1996) and the error of that model over a period of time. If the difference is higher than some predefined threshold and sufficiently varied from the models already in memory, the model is cached for re-use in the future.

9. Adaptive Random Forest (ARF):

   The algorithm description is given in section 4.2.1.

10. SOL-IFC (IFC):

    This is the algorithm proposed in the previous chapter and published in Kithulgoda et al. (2018). Drift detector instances of SeqDrift2 (Pears et al., 2014) are embedded into each classifier in the spectrum repository and decision tree forest.

11. Recurrent Classifier (RC):

    The algorithm description is given in section 4.2.1.

12. Staged Online Learning approach (SOL):

    This is the predecessor of SOL-IFC which was presented in chapter 3 and briefly described in section 4.2.1. This algorithm was selected as it would provide an interesting contrast between SOL-IFC and its more naive version, SOL.

6.1.2 Datasets used for the empirical study

All experiments were carried out with the use of two synthetic datasets generated by MOA's stream generators and five-real world datasets.

Synthetic data recurring with noise

For synthetic datasets, several distinct concepts were generated, each of length 10,000 instances. With the belief that 'concepts do not repeat in exactly the same form' in reality, a 5% noise level for each concept recurrence was introduced by flipping the class label of randomly selected data instances.

1. Rotating Hyperplane dataset (RH): The dataset description is given in section 4.2.2.

2. RBF dataset (RBF): This 10-dimensional dataset generated concepts by changing the number of centroids. The size of this dataset is 1,300,000 data instances with 5 different concepts which were repeated 25 more times with noise as per the description for RH.

Real world data

The real word datasets that this study experimented with have been widely used as benchmarks in a number of studies on data stream mining. The accuracy profiles of the most accurate classifiers show that these datasets have widely different levels of concept drift and concept recurrence.

1. Sensor (Sensor) dataset: The Sensor stream dataset extracted from Zhu (2010) contains a total of 130,073 instances. It consists of temperature, humidity, light, and sensor voltage measures collected from sensors installed in Intel Berkeley

Research Lab. The task is to recognise the sensor ID. Instances of two most frequently appeared sensor IDs *29* and *31* were filtered.

2. Electricity (Elec) dataset: The dataset description is given in section 4.2.2.
3. Flight dataset: The dataset description is given in section 4.2.2.
4. Covertype (Cover) dataset: The dataset description is given in section 4.2.2.
5. Occupancy (Occ) dataset: The dataset description is given in section 4.2.2.

6.1.3 Parameter values

The default parameter values used in the experimentation are as follows:
Maximum number of nodes in decision tree forest: 5000, SeqDrift drift significance value (δ): 0.01, Maximum number of Fourier spectra in repository: 20, Repository hit ratio threshold (α) for T_1 is 0.5, Similarity threshold (β) for T_2 is 0.7, Sample size (s) is 200, Tolerable change rate for refinement is 0.05, and Interval length (I) is 100.

System configuration

System configuration is given in section 4.2.3.

6.2 Accuracy evaluation

In order to maintain fairness, all classifiers were run with the Hoeffding tree as the base learner. The split confidence and tie confidence parameters were both set at 0.01 for all classifier ensembles. Each meta learner was run with default settings for its internal parameters. Accuracies for each classifier was obtained in intervals of size 1,000 and an overall accuracy was then computed by averaging over all intervals for a given dataset. Accuracies were stable across multiple runs for all classifiers across all datasets with a standard deviation less than 0.05 and hence were not included. The outright and

joint winners (with accuracies rounded to 1 significant place) were bolded for easy identification of winner. Ranks for each algorithm were included in parentheses in order to facilitate the analysis of the trade-off between accuracy and throughput conduct in section 6.3.1.

Table 6.1: Classification accuracy with ranking

	RBF	RH	Flight	Elec	Cover	Occ	Sensor	Algo. Rank
ARF	81.8(2)	73.9(4)	80.8(2)	**68.5**(1)	84.5(3)	**95.9**(1)	65.2(6)	**2**
OB	75.3(10)	73.7(5)	73.0(6)	65.4(4)	83.2(5)	82.3(9)	66.5(3)	5
AS	80.3(4)	74.2(3)	73.1(5)	64.4(5)	75.6(9)	84.0(8)	62.2(9)	6
LA	74.8(11)	75.3(2)	71.4(9)	61.3(11)	84.5(3)	89.1(5)	62.9(8)	7
LB	**82.4**(1)	**76.3**(1)	77.2(3)	66.0(3)	**86.1**(1)	86.4(6)	66.2(4)	**2**
AWE	72.1(12)	69.3(9)	49.3(10)	63.7(9)	78.5(8)	73.4(12)	65.8(5)	10
AUE	76.4(7)	71.1(7)	72.5(7)	64.2(7)	82.8(7)	81.8(10)	65.1(7)	8
AD	76.2(8)	68.9(10)	76.9(6)	64.3(6)	84.3(4)	95.3(3)	65.8(5)	5
DA	75.9(9)	68.9(10)	77.2(3)	64.0(8)	84.5(3)	95.8(2)	65.8(5)	4
IFC	80.7(3)	74.2(3)	**81.8**(1)	**68.5**(1)	84.6(2)	93.0(4)	**70.3**(1)	**1**
RC	79.0(5)	70.6(8)	71.6(10)	62.4(10)	73.8(10)	75.6(11)	54.4(10)	9
SOL	78.7(6)	72.7(6)	80.8(2)	67.2(2)	82.9(6)	85.4(7)	69.5(2)	**3**

Table 6.1 shows that the algorithms can be divided into 3 groups with respect to accuracy. The first group of the most accurate classifiers consisted of ARF, LB, IFC, and SOL. Interestingly, the accuracy ranks (shown in parentheses) of three out of four of these classifiers indicate that they collectively accounted for winners across the entire range of datasets. The accuracy profiles of LB (Bifet, Holmes & Pfahringer, 2010) and ARF (H. Gomes, Bifet et al., 2017) reinforce results from previous studies where they excelled in performance. The strong performance of ARF is to be expected as the basic Random Forest algorithm has proved its superior performance in numerous studies and ARF builds on this success by adapting the forest to concept drift in a data stream environment.

The middle group comprised of AD, DA, and OB while the rest formed the bottom group. The performance of the algorithms are further analysed using RC as a benchmark. Since RC does not adapt its model upon entering its low volatile state (Stage 2 of its execution), it is an ideal algorithm to assess the impact of the scale of concept change on the performance of the algorithms. Using this benchmark, it is observed that the Sensor, Occ, Cover, and Flight datasets gave rise to the highest accuracy improvements with respect to RC.

The IFC execution log with the Sensor dataset revealed moderate rates of drift and refinement when compared to the other datasets. IFC gracefully copes with moderate intra-concept variation by incremental refinements on its initially captured spectra while inter-concept switches trigger concept drift and re-use of previously captured spectra. In contrast, LB had to re-learn concepts from scratch rather than making slight adjustments or reuse of previous concepts. The effects of this on LB's accuracy profile is apparent from Fig. 6.1, as there is a noticeable time lag between its peaks and those of IFC over much of the stream. IFC is able to capitalise on concept recurrence by self adaptation or by replacing its currently used spectrum with another in its pool that more closely matches the concept that has recurred. Also, the 29% of accuracy advancement with respect to RC highlights the importance of being self adaptive through periodic refinements of initially learned patterns.

Similarly, the Occ dataset also revealed significant improvement (23%) of accuracy over RC and 8% improvement over LB. The execution log of IFC revealed that Occ has a high rate of inter-concept variation, resulting in switches among a few distinct concepts which were captured by only 5 different spectra in the repository. At the same time, it demanded a high rate of adjustments on recurring concepts thus strengthening the need for refinements to be made on models stored from the past.

On the other hand, LB excelled on the Cover dataset. The accuracy profiles of these classifiers over time show no regular pattern in the peaks and troughs, suggesting a

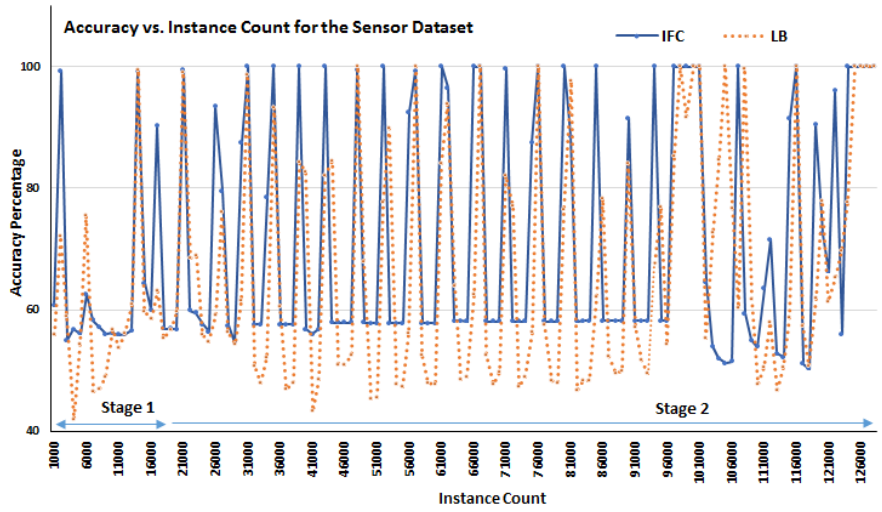


Figure 6.1: Sensor accuracy chart

lack of significant recurrence of concepts over time for this particular dataset. The LB adapts by adjusting the trees in its ensemble to new concepts as they occur. The IFC classifier experienced its highest rate of refinement amongst all datasets, resulting in significantly different spectra through the refinement process in Stage 2. This scenario exemplifies high inter-concept variation between concepts which are interspersed with abrupt intra-concept changes within a given concept. Even though the performance of IFC is somewhat lower compared to LB, it is noted that IFC is second best with a 15% of improvement over RC. This shows IFC's ability to provide good accuracy even in non-recurring situations by generating models that are derivatives of previously stored ones.

IFC with the Flight dataset experienced its highest inter-concept drift rate. The number of concepts in its repository is lesser when compared to Cover but higher when compared to Occ. IFC obtains the best accuracy for this dataset by taking advantage of the reuse of concepts but also by subjecting them to continuous refinement. The accuracy advancements are 14% and 6% over RC and LB respectively. As shown in Fig. 6.2, IFC remains the clear winner in Stage 2 of its execution cycle as a result of

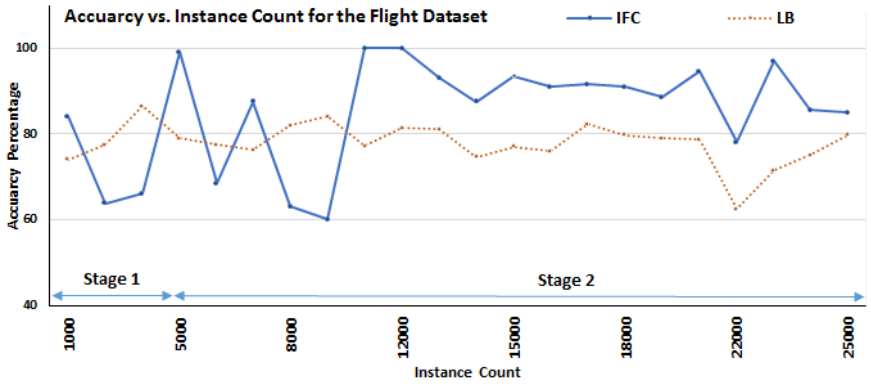


Figure 6.2: Flight accuracy chart

reuse and refine of concepts, whereas with LB concepts were relearned.

In terms of RBF, LB was the winner with marginally better accuracy than IFC. LB adapted its model well to concept drift in this dataset. A close analysis of the models produced showed that LB produced the deepest trees in its ensemble, with comparatively higher concept lengths when compared to other datasets (with the exception of RH). Such deep trees when grown over a significant time duration do better than IFC and other algorithms, especially when drift rate is low. With IFC the comparison was harder in terms of tree depth as it relies exclusively on Fourier spectra in Stage 2, but its average tree height in Stage 1 was much shorter than that of LB.

Finally, it is noted that the triad of algorithms, ARF, IFC and LB performed well across the full range of datasets, in general. The SOL classifier which is at the third place in overall performance also remains competitive with IFC on Flight, Elec, and Sensor datasets. The two classifiers share the use of the DFT although IFC in Stage 2 relies entirely on Fourier spectra whereas SOL uses Hoeffding trees induced from spectra as the learning mechanism. The essential difference between the two is that a Hoeffding tree induced from a spectrum is constrained by its initial structure. The tree is free to grow by splitting its leaf nodes and forming new decision nodes but changes in higher level nodes are not possible. In contrast, Fourier spectra are refined

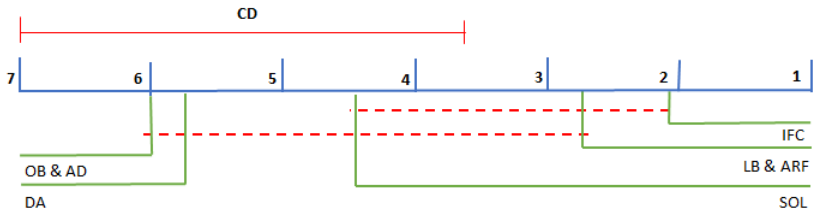


Figure 6.3: Statistical comparison of top 7 ranked algorithms by accuracy. Subsets of classifiers that are not significantly different are connected with dashed lines.

when changes occur in the stream and such changes could reflect more fundamental changes in its decision model. Conceptually, such changes could correspond to better adaptability to changes that take place over time in the stream. In other words, spectra have better flexibility and are able to prune away redundant decision paths and replace them with new ones that are a better match with currently arriving trends in the stream.

The above analysis of the behaviour and performance of classifiers was based on the groupings formed by considering average accuracy rank on the datasets as shown in Table 6.1. This grouping was confirmed through a non- parametric Friedman test that established statistically significant differences between certain sets of classifiers. For this exercise, we subjected the top 7 algorithms by average accuracy rank to the Friedman test. As discussed in Demšar (2006) the Friedman test is recognised as one of the best tests to use when multiple classifiers are to be compared simultaneously on multiple different datasets. The null hypothesis H_0 that we used was that the average accuracy ranks across the 7 classifiers were the same.

Figure 6.3 shows that the null hypothesis H_0 was rejected at the 95% confidence level, thus indicating that statistically significant differences exist between the classifiers. Then those classifiers were subjected to the post hoc Nemenyi test to identify exactly where those differences lay. The Nemenyi test yielded a Critical Difference (CD) value of 3.4 at the 95% confidence level. Fig. 6.3 graphically illustrates that the top group consisted of IFC, LB, ARF, and SOL as none of the members in this group had

significant differences with any of the other members within the group. All members of the other group (i.e. OB, DA, and DA) are also not significant from other members of their group. However, it is also noted from Fig. 6.3, that the IFC classifier stands out as it is the only classifier that was significantly different from all other members of the other group. This was not the case with ARF, LB, and SOL. For instance, LB, ARF, and SOL are not significantly different from OB, AD, and DA.

6.3 Throughput evaluation

This section turns the attention to the processing speed of the algorithms, and in particular, it examines how IFC fares in relation to the other 11 algorithms. The processing speed is measured over the same interval size (1,000) just as the section 6.2 did with accuracy. Processing speed is evaluated by throughout which is the number of data instances processed in an interval for any given dataset. An average throughput measure was then computed over the entire dataset. Multiple runs were conducted for each dataset and negligible variance in timing was observed between runs.

For the purpose of simulating real-world data streams, the size of the smaller datasets was increased, namely Elec, Flight, Cover, and Occ by concatenating each of them with multiple copies of themselves. This also served to produce more reliable estimates of execution time and throughput. The other 3 datasets were large enough to be used in their original form.

Although IFC performed well in terms of accuracy, it was designed specifically to scale up to high speed data streams and the performance aspect is crucial in its evaluation. Table 6.2 shows that IFC excels on throughput, obtaining the highest average rank when taken over all datasets and emerging the outright winner in 3 out of the 7 datasets.

Table 6.2: Throughput with ranking

	RBF	RH	Flight	Elec	Cover	Occ	Sensor	Algo. Rank
ARF	25811(8)	26481(7)	18866(3)	33038(7)	11837(4)	45729(9)	50342(7)	6
OB	30729(4)	30634(5)	4975(7)	37313(6)	4398(6)	51692(8)	50263(8)	5
AS	30056(5)	39910(4)	6353(6)	53882(5)	3371(8)	71425(3)	86447(2)	4
LA	3195(12)	2827(12)	1686(11)	25869(10)	117(12)	64000(4)	64000(3)	8
LB	9810(9)	10838(11)	1082(12)	21667(12)	1413(11)	33641(12)	36525(11)	11
AW	**59468**(1)	47099(3)	19722(2)	54428(4)	14122(3)	60618(6)	62345(4)	2
AU	50577(2)	59704(2)	10777(4)	55755(3)	6811(5)	63289(5)	62080(5)	3
AD	26446(7)	24933(8)	3517(9)	23110(11)	2788(10)	39282(11)	43313(10)	10
DA	29028(6)	24138(9)	3720(8)	30501(9)	2883(9)	45630(10)	53818(6)	7
IFC	35599(3)	**73335**(1)	**70990**(1)	89338(2)	35466(2)	155560(2)	**137466**(1)	1
RC	9468(10)	26553(6)	7975(5)	**415660**(1)	**83856**(1)	**464493**(1)	48889(9)	4
SOL	5140(11)	17076(10)	3044(10)	31106(8)	4102(7)	53031(7)	30869(12)	9

To gain insights into why IFC was the overall winner, its throughput is compared to that of the RC, SOL, and the MOA group of algorithms. Firstly, IFC is compared with SOL and RC, as they all use the staged approach. It is observed that IFC significantly outperforms RC on RBF, RH, Flight, and Sensor datasets. Despite the fact that RC does not update its spectra during Stage 2, IFC is much faster due to the lesser number of distinct schema when compared to RC. RC does not use feature selection and therefore number of influential features in dot product calculations are much higher than that of IFC. Since it uses the Inverse Fourier Transform (IFT) in Stage 2 which has complexity $O(|X|^2)$, its Stage 2 processing time is much longer than that of IFC.

In contrast, SOL uses trees induced from spectra during Stage 2. Even though it maintains a single tree at a time instead of an ensemble it still has higher classification overhead than IFC. The IFC classifier simply fetches the class label of an instance to be classified from its reservoir if present, otherwise it performs the IFT, but unlike RC its schema set is much smaller and hence the overall classification time is much smaller.

In comparison to IFC, all of MOA's classifiers use an ensemble of trees during Stage 2, and depending on average tree size their performance is impacted. In particular, LB that was ranked alongside ARF and IFC on accuracy suffers the most when compared

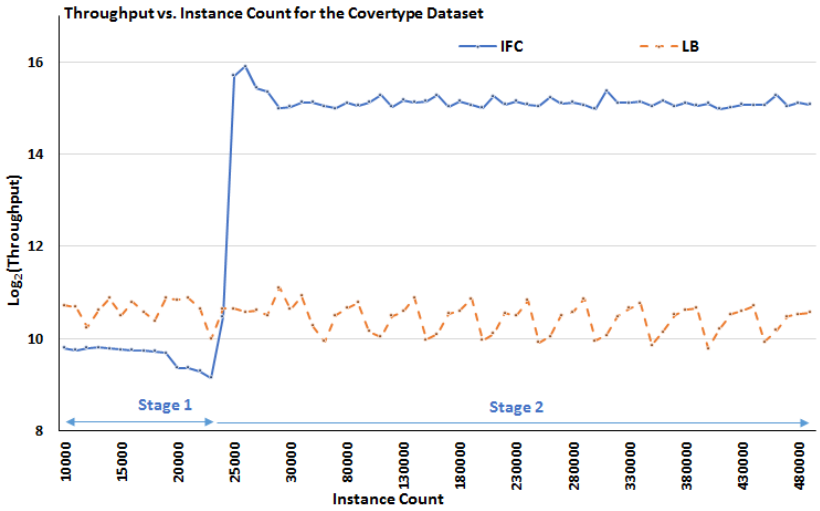


Figure 6.4: Covertype throughput chart

to IFC as it had the highest tree size out of all of MOA's group of classifiers. Overall, the ARF classifier had a higher throughput than LB even without the use of a large multi-core machine for which it was designed. It was reported in H. Gomes, Bifet et al. (2017) that its speed increased by a factor of around 3 with the use of a machine that supported a high degree of parallelism with 40 cores.

Figures 6.4 and 6.5 show the contrast between the IFC and LB classifiers. In Fig. 6.5, the granularity of the horizontal axis in the range 1,000 to 10,000 was scaled up in order to highlight the presence of the relatively short lived Stage 1. These charts clearly show that IFC's speed advantage was gained in Stage 2 of its processing when classification was performed exclusively with the use of Fourier spectra.

In order to gain an understanding of how the most accurate classifiers perform with respect to throughput, we subjected the same set of 7 classifiers that we selected for accuracy analysis to the Friedman test. Fig. 6.6 shows that there are three distinct groups. The first group comprises of IFC on its own as it has statistically significant differences with all other 6 classifiers that form the other two groups. The second group consists of ARF, OB, DA, SOL, and AD while LB makes up another group together

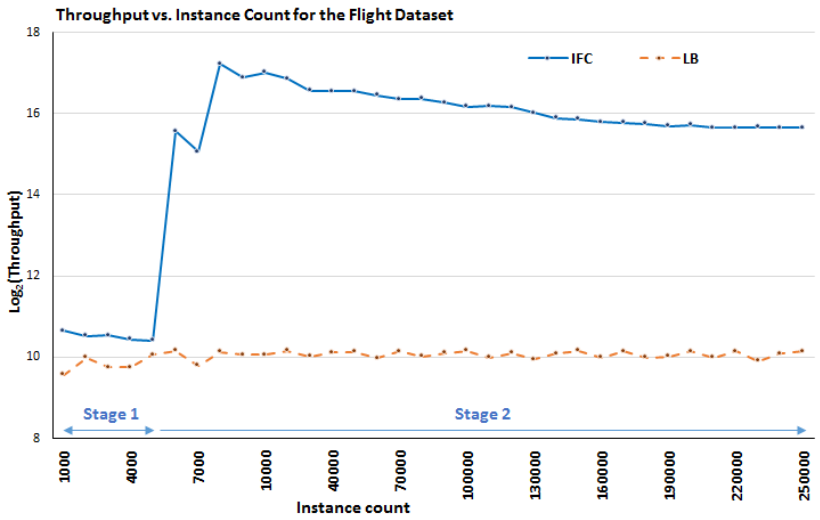


Figure 6.5: Flight throughput chart

with AD, SOL, and DA. This statistical analysis thus yields additional insights into the relative throughput performance of the classifiers over those given in Table 6.2.

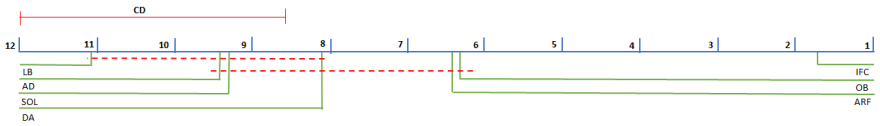


Figure 6.6: Statistical comparison of the throughput performance of the top 7 ranked algorithms that were ranked on accuracy. Subsets of classifiers that are not significantly different are connected with dashed lines.

6.3.1 Accuracy versus Throughput trade-off

This section visualises the trade-off between accuracy and speed. Fig. 6.7 clearly shows that IFC and LB are at opposite sides of the speed spectrum. Classifiers that tend to be more accurate (e.g. LB, SOL) tend to be more time consuming and vice versa; the exception being IFC although it was not always the most accurate, nor was it always the fastest. However, it is clear that IFC has achieved its design goals as it is by far the fastest amongst the set of the most accurate algorithms that we experimented with.

The closest neighbours to IFC are ARF and OB as they achieved the next best balance between accuracy and speed.

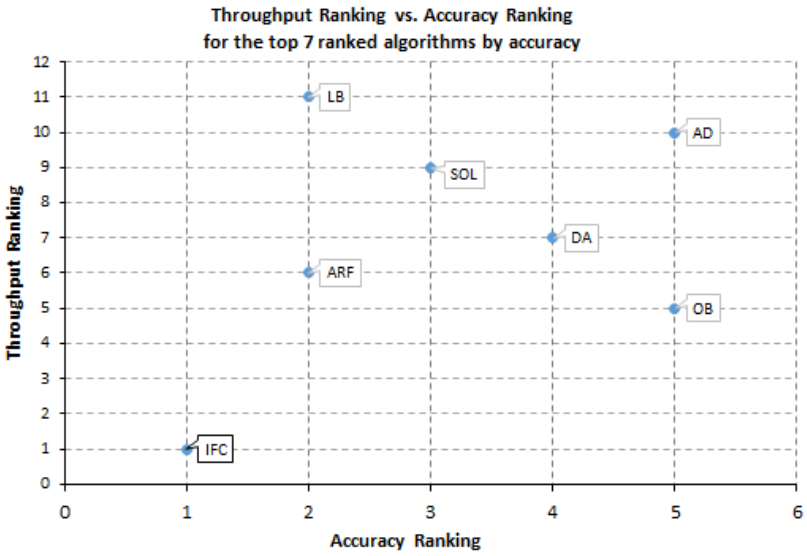


Figure 6.7: Accuracy vs. Throughput tradeoff

6.4 Load Shedding

High speed data streams have a greater likelihood of causing computing resource allocation problems. In this respect, load shedding is an attractive solution for dealing with such resource problems. In the majority of studies done in the domain of load shedding in data streams, the approach taken is discarding of some subset of the data in order to decrease the workload.

The approach given in Babcock et al. (2007) decides whether to process or drop a data instance by performing a Bernoulli trial. The data instance is processed with a given probability p, while the probability of neglecting it is $(1 - p)$. In the scenario of multiple streams processing studied by Chi et al. (2005), it is suggested to shed some streams' data with the objective of reducing data pre-processing effort with the

assumption that pre-processing is more resource demanding than classification. In the next step of their setup, a corrective action is taken by inferring those dropped data features based on their historic values.

The recent study of Ning et al. (2016) proposes not to use common loss ratio-based random dropping techniques for stream processing. Alternatively, it recommends a (m, k) scheduling algorithm that guarantees to avoid dropping of successive instances. Here m is the number of processed tuples whereas the k is the number of total arrivals per stream. In this context, at most $(k–m)/k$ number of instances are neglected for reducing the workload.

The simulation given in section 6.4.1 has taken a different approach to load shedding compared to the above studies. Rather than dropping data to ensure lesser resource consumption, here we reduce the number of classifiers in the ensemble or encourage lesser decision node splits as devices to reduce the workload.

6.4.1 Load Shedding exercise

The IFC and LB classifiers were subjected to a load shedding exercise that assessed their sensitivity to accuracy by simulating a high speed data stream environment.

Firstly, the speed achieved by IFC was used as the benchmark and then manipulated parameters for LB until it matched IFC's throughput. The matching was achieved by decreasing the number of classifiers in LB's ensemble and/or setting the split confidence parameter value for the Hoeffding tree base classifier to low settings (e.g. 10^{-7}). The load shedding simulation was performed for the RBF, Flight, and Cover datasets. These datasets were chosen on the basis of their accuracy performance with the LB and IFC classifiers and the fact that they represented the greatest diversity in dimensionality across the datasets that we experimented with.

Table 6.3: Load Shedded LB accuracy

Dataset	Ensemble Size	Split Confidence	Accuracy
RBF	7	1E(-4)	80.0
Flight	3	1E(-7)	73.6
Cover	3	1E(-7)	82.4

Table 6.4: Load Shedded throughput and accuracy for IFC

Dataset	Scaled Throughput	Throughput Gain (%)	Accuracy
RBF	45156	26.8	81.1
Flight	80408	13.3	77.2
Cover	55155	55.6	81.4

Table 6.3 when read in conjunction with Table 6.1 shows clearly that IFC now achieves the highest accuracy over each of these datasets. This results show that an algorithm's accuracy needs to be judged by its load handling capability, while it may be more accurate than another algorithm on a stream with a certain speed, its accuracy advantage may disappear when it runs on a higher speed stream, and is forced to perform load shedding to cope with the speed of the stream.

This work also tracked IFC's ability to scale up to data streams in excess of its capacity reported in Table 6.2. Load shedding with IFC was simulated by increasing its interval length I by a factor of 5. Table 6.4 shows that IFC's throughput gain ranged from a modest 13.3% for the Flight dataset to 55.6% for the Cover dataset. Interestingly, the accuracy for the RBF dataset has been increased marginally by 0.5%. This is consistent with the moderate drift rate detected in IFC's execution log for this dataset. The decrease was 3.8% for Cover and 5.6% for Flight. The Flight dataset recorded the highest decrease as it exhibited the highest drift rate and the highest rate of refinements amongst all datasets.

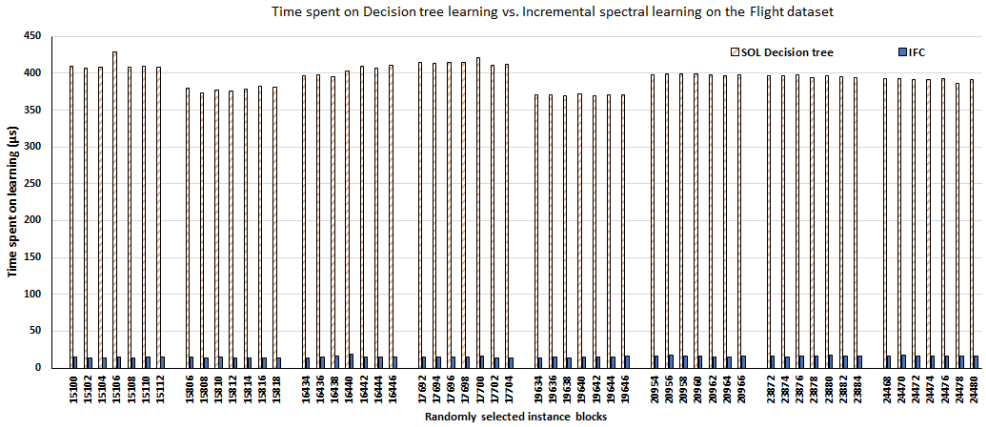


Figure 6.8: Throughput advantage of Incremental spectral learning over Decision tree learning

6.5 Overheads of Decision Tree Learning vs Incremental Fourier Classification

In this section, the overheads of classification with a decision tree are contrasted to an incremental learning approach using Fourier spectra. The SOL presented in chapter 3 was featured in this experimental study since it uses the staged learning approach just as IFC does, except that it classifies using a decision tree instead of a Fourier spectrum in the low volatility segments of the stream. The SOL classifier takes advantage of low volatility to dispense with a forest of trees and uses a single tree instead. This provides a good platform for comparison as IFC which uses an ensemble of models (in the form of spectra) would need to compete with a single model.

Figure 6.8 depicts the time spent by IFC and SOL in randomly selected stream blocks. The blocks were selected from the execution logs of both classifiers to exclude blocks where either SOL and/or IFC experienced concept drifts. This was done to ensure that the time reported reflected the time spent in classification. The overheads of processing concept drift are presented in section 6.5.1.

Figure 6.8 shows clearly that classification using Fourier spectra is much faster than

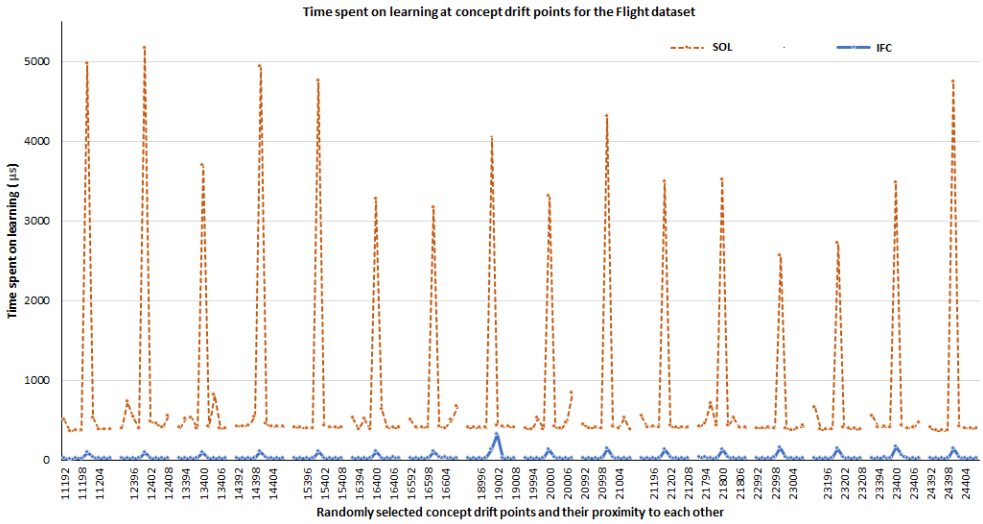


Figure 6.9: Time spent when learning in Non-Incremental and Incremental modes of operation

with a decision tree. This is due to the replacement of a tree traversal operation with a simple lookup operation on the hash based reservoir structure that extracts class labels in the event of a cache hit. In the event of a cache miss, the IFT would be used to retrieve the class label of the instance.

6.5.1 Incremental versus Non Incremental approach to Fourier Classification

In this section, the effects of incrementally maintaining a Fourier spectrum are studied and contrasted with its naive version (SOL) which generates a new spectrum from a decision tree whenever a concept drift is signalled in the stream.

Two new experiments were conducted. The first was on the Flight dataset and shows the difference in timing between the two versions at concept drift points. It is clear from Fig. 6.9 that the incremental version (IFC) is far more efficient and is able to avoid spikes in processing time that are experienced by a conventional DFT application in SOL.

The second experiment was a controlled experiment run on a RBF dataset whereby the dimensionality was varied from 10 to 60 in intervals of 10. The intention was to examine the CPU utilisation as a function of dimensionality (D), drift rate (R) and stream speed (S). For each value of dimensionality, the average processing time spent on classifying data instances (C) as well as the time spent on the application of the DFT was recorded (F). Likewise, the average time spent in classification as well as in reorganising the spectrum with our incremental Fourier version was recorded. With these average values in place, CPU utilisation (U) was modelled using Eq. 6.1.

$$U = (1 - R) \times S \times C + R \times S \times F \tag{6.1}$$

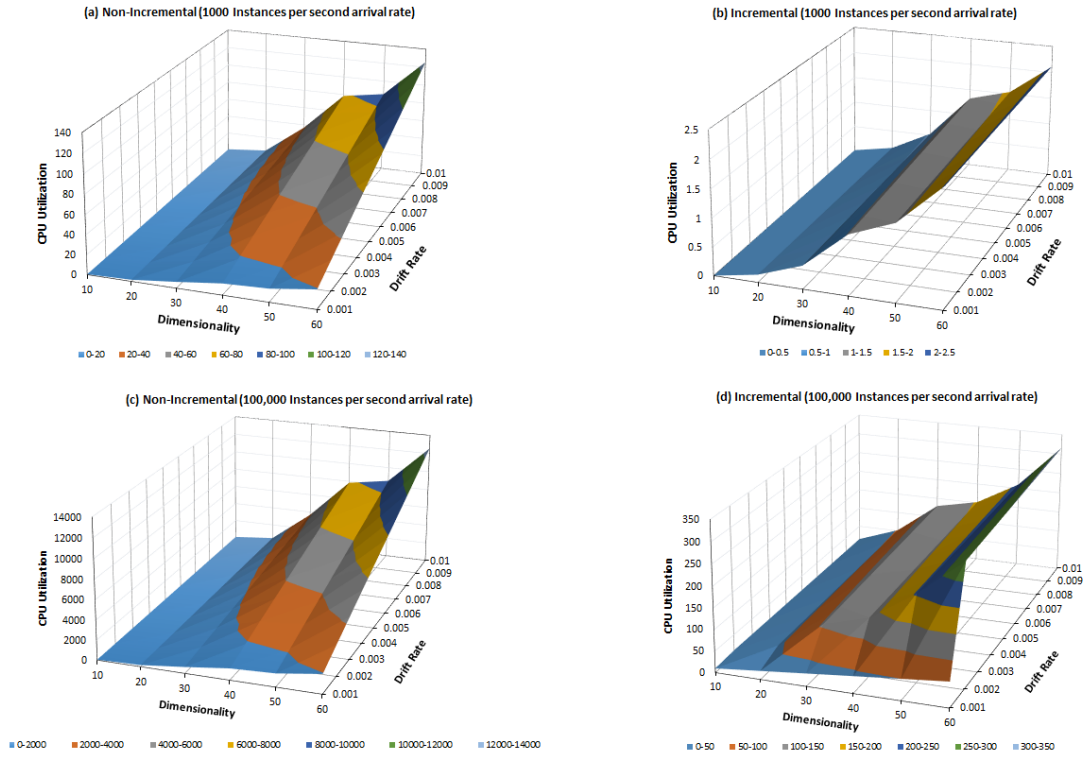


Figure 6.10: Resource utilisation of Incremental and Non-Incremental DFT approaches

A stream with a sample arrival rate of 1,000 instances per second is modelled first.

The top panel of Fig. 6.10 shows clearly that CPU utilisation is an increasing function of both drift rate and dimensionality for both the incremental and non incremental versions. For the non-incremental version (SOL) in the top left panel of Fig. 6.10, the utilisation varies from a low bound in the range 0-20 to an upper bound of 120-140 when both dimensionality and drift rate are their maximum levels. At the high end of the CPU range (120-140), 2 resources with own processor and local memory would be needed to ensure that the system functions without any form of disruption.

In sharp contrast we note from the top right panel (IFC) of Fig. 6.10, that the upper bound for CPU utilisation is very much smaller at 2.5%, thus managing comfortably with a single resource.

A high speed data stream with a speed setting of 100,000 instances per second is modelled as the next. The bottom left and right panels of Fig. 6.10 shows the same trends as with a stream speed of 1,000 but at a much higher scale. With the non-incremental version, the left panel of Fig. 6.10 shows that the CPU utilisation reaches a maximum value of 12,000-14,000, thus requiring up to 140 resources to maintain operation without disruption. If lesser than 140 resources are available, then the cost of the non-incremental version becomes prohibitive. In contrast, the peak CPU utilisation for the incremental version was just 300-350; thus requiring far fewer resources (up to 4) to maintain normal operation.

These experiments show clearly the resource utilisation advantage of the incremental approach, and explain the massive gains in throughput of IFC over the non-incremental SOL classifier that was tabulated in Table 6.2.

6.6 Conclusion

This empirical study reveals that SOL-IFC is able to adapt its behaviour successfully to different types of situations that manifest in data streams. This novel algorithm exploits

the staged learning framework and outperforms most of the algorithms, both in terms of accuracy and throughput.

In terms of speed, SOL-IFC proved to be superior on account of its compact data structure used for classification. The coefficient pruning strategy implemented via energy thresholding ensured that the coefficient array was as compact as possible, thus reducing classification overhead. In addition, the schema pruning strategy, when used in conjunction with the incremental Fourier coefficient update strategy ensured that update overheads were minimised.

Amongst the set of best performing algorithms on accuracy, SOL-IFC ranked in the first position when evaluated on a combined metric of throughput and classification accuracy. Moreover, SOL-IFC successfully addresses the issue of peak processing times that was observed at drift points in SOL, by replacing the computationally expensive tree-spectrum and spectrum-tree transformations with a smooth incremental adaptation process that takes place on a continuous basis.

Chapter 7

Verifying the Potential of Using Different Classifiers in SOL

7.1 Introduction

This chapter demonstrates the staged learning framework's ability to be independent from a particular type of incremental learner. Chapter 3 of this thesis listed 3 main components of the staged learning framework: an incremental learner, a concept drift detector, and a repository for capturing past concepts. The same chapter claimed that each of those components is replaceable, that is can accommodate different implementations in the place of any of those. Hence, this chapter can be viewed as a proof of concept of a neural network successfully replacing a decision tree ensemble that was previously being used.

More specifically, a feed-forward Multi-Layer Perceptron (MLP) Neural Network (NN) with a Backpropagation training algorithm takes over the Stage 1 learning in SOL. To put it briefly, this is an effort to verify two premises: (1) the effectiveness of the Discrete Fourier Transformation (DFT) on classifiers other than decision trees, that happens to be the neural network in this exercise and, (2) the suitability of such a

classifier in SOL.

With those two main premises in mind, several advanced capabilities integrated to SOL in Chapter 3 and Chapter 5 are kept out of scope in this chapter for the sake of simplicity. For instance, concept drift detection has not been implemented. Furthermore, volatility sensitive stage transition has been implemented through a trial and error process. After observing a certain fraction of data in Stage 1, a learned MLP model was condensed into a Fourier spectrum via application of the DFT.

Sections 7.2 and 7.3 outline related domain knowledge and implementation choices made in this particular implementation. Generation of the Fourier model for an obtained MLP was addressed with novel instance clustering strategy described in section 7.4. Then, the performance of the derived MLP based Fourier model has been explored in section 7.5. Lastly, the section 7.6 concludes the chapter.

7.2 Neural Network

Artificial Neural Network (ANN) classifier is a collection of connected layers each of which comprises a set of nodes called artificial neurons. ANNs are initially inspired by biological neural networks' architecture of animal brains.

Architecture based grouping done in Jain, Mao and Mohiuddin (1996) recognised feed-forward networks and recurrent networks as two main categories of ANN. Feed-forward networks are static in the sense of producing only one set of output per given input. The output is independent of the previous state of the network and hence called memoryless. In contrast, recurrent networks are dynamic due to its feedback paths which update inputs according to outputs of the previous state. Aiming to achieve higher performance, this study adopts the feed-forward architecture that is commonly used.

Furthermore, single-layer perceptron, MLP and Radial Basis function networks are identified as sub-groups (Jain et al., 1996) of the feed-forward category. Among those,

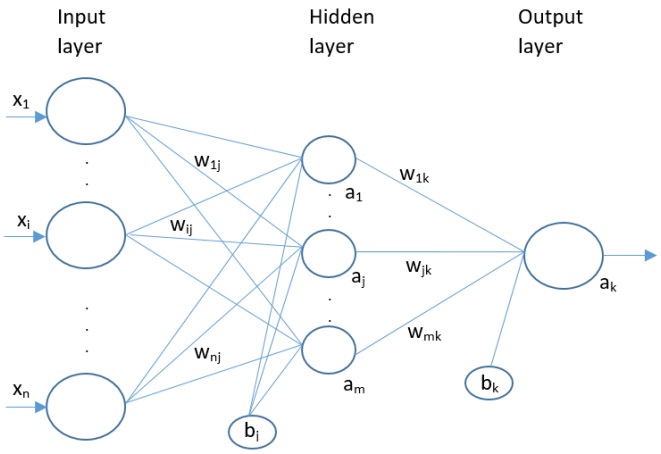


Figure 7.1: Multilayer perceptron with a single hidden layer

multilayer feed-forward networks are more accurate and robust for even non-linear functions (Gardner & Dorling, 1998; Gudise & Venayagamoorthy, 2003), and hence are regarded as universal approximators. The multilayer perceptron consists of one or more hidden layers in addition to basic input and output layers that exist in the single layer perceptron version. However, the number of hidden nodes, the effectiveness of the learning strategy used, and the nature of the relationship between input and output all affect the performance of MLP (Hornik, Stinchcombe & White, 1989). This chapter uses the feed-forward MLP with one hidden layer.

7.2.1 Feed-forward MLP

Figure 7.1 illustrates the design of this type of neural network. Accordingly, nodes of the input layer are fed by input value $\vec{X} = (X_1, .., X_i, .., X_n)$ and then weighted input values are sent through the hidden layer. Weight w_{ij} is associated with the connection between input node i and successor node j. The hidden layer outputs a_js are produced after applying hidden layer's transfer function (activation function) g_j on the weighted sum of inputs X_is plus bias value b_j. More details about activation function can be found in the section 7.2.2. Likewise, output layer is fed by a_js. The activation function

of output layer g_k is applied on the weighted sum of a_js plus bias value b_k.

Accordingly, output a_k for a given input instance $\vec{X} = (X_1, .., X_i, .., X_n)$ can be given below:

$$a_k = g_k(b_k + \sum_j g_j(b_j + \sum_i X_i w_{ij}) w_{jk}) \tag{7.1}$$

7.2.2 Activation function

Activation function facilitates the capability of dealing with non-linear complex arbitrary relationships between inputs and outputs. These functions need to be differentiable for the purpose of minimising error as described in 7.2.4. Some of the popular activation functions are Sigmoid, Tangent Hyperbolic (Tanh), SoftMax and Rectified linear unit (ReLu). This study uses Tanh as the activation function for the hidden layer and SoftMax as activation function for the output layer. The conceptual design of this work does not rely on any of the specific activation functions, hence they can be altered as necessary.

Tanh activation function

Tanh activation function given in Eq.7.2 outputs values between -1 and 1.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{7.2}$$

SoftMax function

SoftMax function results in probabilities of each target class as expressed in the Eq. 7.3 below. The values are in the range of 0 and 1 for each target class while the sum of all the probabilities of output nodes is equal to 1. These probabilities can be interpreted as the estimate of the class distribution for a given input.

$$P(y = c_m|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{m} e^{x^T w_{kc_m}}} \tag{7.3}$$

7.2.3 Error function

Error or the cost over L number of training examples is quantified using the averaged squared difference between the output a_k and the target value (true class label) t_k of each example as shown below in Eq. 7.4.

$$E = \frac{1}{2L} \sum_{k} ||(a_k - t_k)||^2 \tag{7.4}$$

7.2.4 Backpropagation algorithm

The backpropagation learning algorithm is used by the gradient descent optimisation algorithm in order to find optimum weights and biases of neurons by calculating the gradient of the prediction error. This iterative weight readjustment process was proposed in Werbos (1974).

The error of a neural network is the difference between predicted output and expected output. Given labeled input instances for some period of time, the system calculates the error, adjust weights and bias values so as to minimise the error.

For this, it is necessary to compute partial derivatives of cost function E given in Eq. 7.4 with respect to bias values and weights which are called gradients for each layer. Lastly, weights and biases of each level are updated iteratively so that the gradient of the cost function is decreased.

7.3 Learning from Neural Network in Stage 1

In addition to classification, Stage 1 learning involves iterative adjustment of network parameters. Weights and biases need to be updated so as to minimise the error of the network. The error minimisation is implemented by the gradient descent procedure. The gradient (derivative) of the error function is calculated with respect to each of the

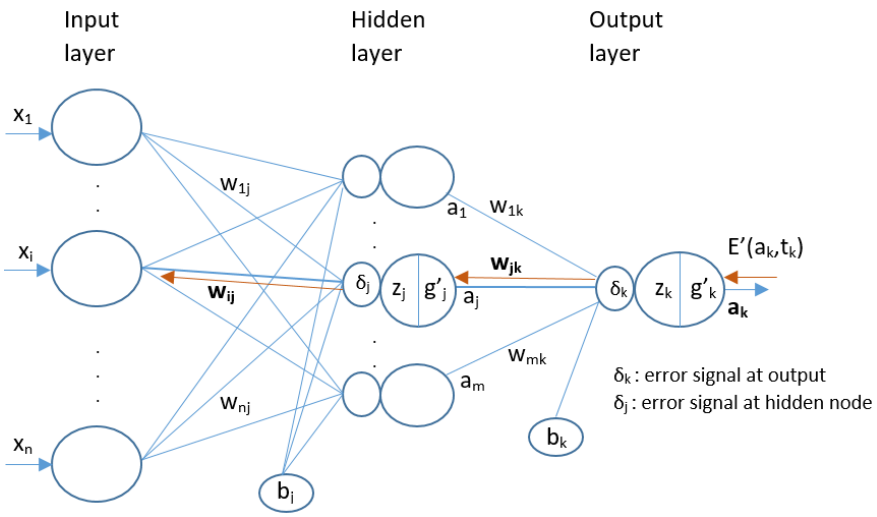


Figure 7.2: Backpropagation of error signals

model parameters. The calculated gradient information is needed to be sent across the layers for parameter updates. The process done in this study is depicted Fig. 7.2 and can be summarised as below.

1. Forward-propagation towards the outputs.

   The output is produced through the Eq. 7.1 given in section 7.2.1. Initially, all weights w_{ij}s and w_{jk}s are assigned randomly for each connection in network. Tanh function and SofMax functions are taken as hidden layer's activation function g_j and output layer's activation function g_k respectively.

2. Backpropagation of error signals to the inputs.

   Error signals at output node a_k and hidden layer node a_j are calculated as follows:

$$\delta_k = g_k^{'}(z_k)E^{'}(a_k, t_k) \tag{7.5}$$

$$\delta_j = g_j^{'}(z_j)\sum_k \delta_k w_{ij} \tag{7.6}$$

   ,where $z_k = b_k + \sum_j g_j(b_j + \sum_i X_i w_{ij})w_{jk}$ and $z_j = b_j + \sum_i X_i w_{ij}$.

3. Evaluate gradients to lower the error.

   The gradients of the error function are calculated with respect to the model parameters at each layer using the forward signals (X_i, a_j) and backward error signals (δ_j, δ_k) as shown below:

$$\frac{\partial E}{\partial W_{ij}} = X_i \delta_j \tag{7.7}$$

$$\frac{\partial E}{\partial W_{jk}} = a_j \delta_k \tag{7.8}$$

4. Update the weights.

   At the learning rate of η, weights can be updated as below:

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial W_{ij}} \tag{7.9}$$

$$w_{jk} = w_{jk} - \eta \frac{\partial E}{\partial W_{jk}} \tag{7.10}$$

These steps are repeated until the maximum number of epochs is reached.

7.4 Generating Fourier model for Stage 2 Learning

For the purpose of generating a Fourier model, it is necessary to represent the dataset in the form of a schema set (X) that is defined in section 2.5. These schemas and their corresponding class labels are gathered through the novel cluster pool algorithm explained below in section 7.4.1. This is followed by the section 7.4.2 which summarises steps of Fourier coefficients calculation.

7.4.1 Extracting Schema Set

Following algorithm 5 is implemented on a set of distinct data instances appeared in Stage 1. A collection called HashSet introduced in .NET 3.5 environment is used for this purpose. The HashSet is an ordered collection of unique elements, and hence guarantees to avoid duplicates from being inserted.

Algorithm 5 Cluster Pool for Schema Set Extraction

Input: Distinct data instances seen in Stage 1

```
1: repeat
2:     Classify instance by trained MLP
3:     if No cluster pool for the class then
4:         Create a new cluster pool for new class
5:         Go to 7
6:     else
7:         if No clusters in the cluster pool for the class then
8:             Create a cluster with the instance pattern, Go To 2
9:         else
10:            for all Clusters in the cluster pool do
11:                Generate qualifying pattern: put "*" if values are mismatched
12:                                    ▷ Null pattern (pattern with all "*"s) is not allowed
13:                if No valid qualified pattern then
14:                    Create a cluster with the instance pattern, Go To 2
15:                else
16:                    if Pattern not found in any cluster in other cluster pools then
17:                        Remove previously compared cluster          ▷ follow LIFO
18:                        Create a new cluster for the qualified pattern
19:                    end if
20:                end if
21:            end for
22:        end if
23:    end if
24: until End of input
```

Each Stage 1 instance is accommodated by a cluster pool associated with its class label. In the absence of a corresponding cluster pool, a new cluster pool is to be created for that particular class (line 4 in the algorithm 5). This ensures a separate cluster

pool for distinct class labels identified in the data. For example, in the case of binary data, each cluster in a cluster pool can be represented with a label (e.g. 0*1, **0, 1*1) that symbolises a schema pattern as shown in Fig. 7.3. At the arrival of each new instance, iterating through all clusters (line 10) in a cluster pool is done on the basis of Last-In-First-Out (LIFO). The algorithm 5 and the Fig. 7.3 are further illustrated through the use of a worked example below.

Assume that 001 and 011 instances arrived with class label 0. Currently, no any clusters exist in the cluster pool for class 1. Firstly, a new cluster is created with 001 inside the cluster pool for class 0, and then the next instance 011 is considered. A qualifying pattern is generated by replacing mismatched attribute values with the '*' character. Therefore, this example produces 0*1 as the qualifying pattern after observing 001 and 011. As a result, cluster 001 is removed and cluster 0*1 is formed.

Assuming 000, 010, 100, 110, 101 and 111 all arrived with class label 1, the algorithm proceeds as follows. Note that, the cluster pool for class 0 consists of cluster 0*1 at this point in time. A new cluster is created for 000, and then it starts to process the next instance 010. As described above, the qualifying pattern 0*0 is formed. Since the cluster pool for class 0 is not empty, the algorithm verifies that the pattern 0*0 is not

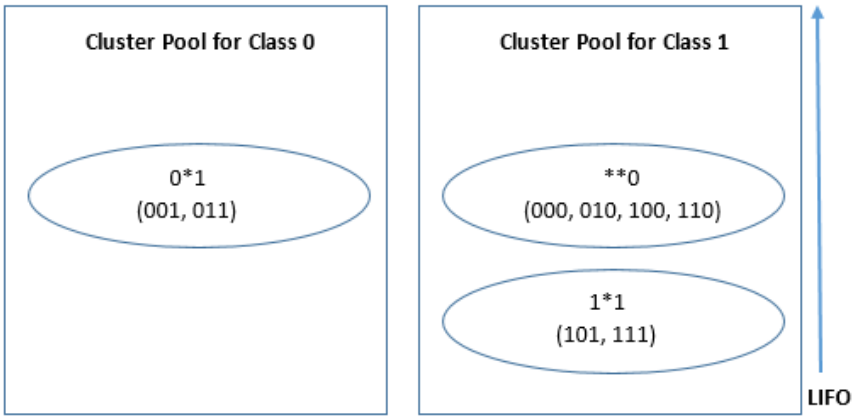


Figure 7.3: Schema Patterns (clusters) generated after clustering algorithm

found in the other cluster pool for class 0. Given it cannot be found in the other cluster pool, the previously created 000 cluster is removed and then a new cluster is created for 0*0. The algorithm continues with the next instance, 100. The qualifying pattern **0 is generated after comparing 100 with 0*0. Provided that the pattern **0 is not found in the other cluster pool, the previously created 0*0 pattern is removed before the creation of **0. The instance 110 is evaluated, the pattern is matched with the cluster **0. Thereafter, the instance 101 is evaluated. Since the null pattern *** is not allowed, a new cluster is created. The next instance 111 produces 1*1 pattern with 101 cluster. The 101 instance is now represented by the pattern 1*1. Finally, a 0*1 cluster is in class 0 pool while **0 and 1*1 clusters result in the class 1 pool.

These clusters, i.e. x vectors (elements of the schema set X) and corresponding classes gathered at the end of Stage 1 are used in computing spectrum coefficients.

7.4.2 Fourier Coefficient calculation for Stage 2 Learning

Firstly, a set of important coefficient indexes are filtered out after energy thresholding is applied, as described in section 2.5. Secondly, the coefficient value for every important index is calculated by Eq. 2.4 over x schema vectors gathered in the preceding section.

The derived Fourier spectrum is deployed for Stage 2 learning upon the suspension of neural network learning. Predicted class labels are taken as given in the Eq. 2.3 in section 2.5.

Figure 7.4 presents the complete system design of MLP to Fourier model transformation discussing so far.

7.5 Experimental study

As introduced in the section 7.1, experimenting the suitability of MLP based Fourier models within the SOL framework was one of the chapter objectives. This section

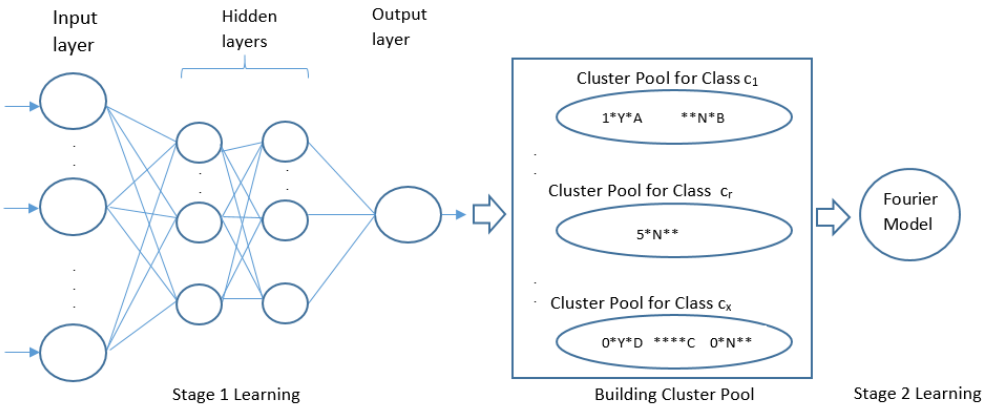


Figure 7.4: Learning in Stages with MLP

analyses the performance of 3 classifier systems in the staged learning context. The Stage 1 learner was MLP while Stage 2 classification was done using one of three options. The MLP based Fourier Classifier "MLP-FC" learns during Stage 2, by applying the Fourier model derived from the trained neural network. One of the other approaches were "MLP-MLP" where the classification continues with MLP. The approach named as "MLP-ClusterPool" does Stage 2 classification via the cluster pool developed according to section 7.4.1. The "MLP-FC" was compared against "MLP-MLP" and "MLP-ClusterPool" in order to see whether it is beneficial to have a MLP based Fourier model in Stage 2.

Furthermore, this experimental study compares several different data partition lengths for Stage 1 as the derivation of a suitable statistical trigger was out of scope for this study. Different lengths tested for Stage 1 were 10%, 30%, 50%, and 70% of each dataset.

7.5.1 Datasets used for the experimental study

Following five datasets were subjected to experiments.

1. Elec : The description of this dataset is given in section 4.2.2.

2. Sensor: The description of this dataset is given in section 6.1.2.
3. Occupancy: The description of this dataset is given in section 4.2.2.
4. Flight: The description of this dataset is given in section 4.2.2.

7.5.2 Parameter values

Below are the MLP configuration related and Fourier model generation related parameters.

MLP configuration related parameters

Maximum epochs: maximum number of iterations the algorithm sees the entire data set. This was set to 100.
Learning rate: this is the step-size which influences the rate of reaching ANN's minimum error. This was set to 0.3.
Momentum: this is the increment of step-size which helps the network not to converge to local minima. This was set to 0.2.
Number of Input Nodes: this corresponds to the number of attributes. This was set according to the dataset.
Number of Hidden Nodes: this was set to 8.

Fourier model derivation related parameters

Energy threshold: this is used to determine the maximum coefficient order used by the energy thresholding process as described in section 2.5. This was set to 95%. Given the threshold is met, no higher order coefficients are considered.

System configuration

System configuration is given in section 4.2.3.

7.5.3 Effectiveness of SOL with MLP based Fourier model

Accuracy and throughput values per stage are tabulated below for the three above-mentioned approaches over 4 different Stage 1 lengths. Overall accuracy was calculated by weighting stage-wise accuracy according to the percentage of data that appeared in each stage. Overall throughput (number of instances processed within a second) is calculated by dividing the total number of instances by the total time spent for classification, despite the stage.

Stage-wise performance measures for the Elec dataset

The Elec dataset shows a clear pattern between performance measures and Stage 1 duration for MLP-FC. Longer the Stage 1 duration, the higher the overall accuracy of MLP-FC, yet it sacrifices throughput as depicted in Tables 7.1 to 7.4. An accuracy increase of 419.7% was accompanied by a decrease in throughput of 85.9% from the shortest Stage 1 to the longest. An accuracy improvement is expected in most cases due to the longer learning period of MLP which justifies the lower throughput as well. Error minimisation of MLP is a time consuming process in general, although most of the time the model becomes more accurate.

Table 7.1: Elec: 10% data for Stage 1

		Stage 1 (start-4531)	Stage 2 (4531-end)	Overall
MLP-FC	Accuracy	60.5	8.5	13.7
	Throughput	1354.7	10958215.8	13532.2
MLP-MLP	Accuracy	60.5	57.2	57.5
	Throughput	–	–	356255.9
MLP-ClusterPool	Accuracy	60.5	57.2	57.5
	Throughput	–	583419170.2	13546.9

Table 7.2: Elec: 30% data for Stage 1

		Stage 1 (start-13593)	Stage 2 (13593-end)	Overall
MLP-FC	Accuracy	61.0	63.8	63.0
	Throughput	1371.4	11396184.4	4570.1
MLP-MLP	Accuracy	61.0	65.0	63.8
	Throughput	–	342440.9	4529.1
MLP-ClusterPool	Accuracy	61.0	8.5	24.3
	Throughput	–	613520309.5	4571.4

Table 7.3: Elec: 50% data for Stage 1

		Stage 1 (start-22656)	Stage 2 (22656-end)	Overall
MLP-FC	Accuracy	64.0	72.3	68.2
	Throughput	1394.8	14023273.1	2789.3
MLP-MLP	Accuracy	64.0	63.3	63.6
	Throughput	–	266223.3	2775.1
MLP-ClusterPool	Accuracy	64.0	34.0	49.0
	Throughput	–	362496000.0	2789.6

Table 7.4: Elec: 70% data for Stage 1

		Stage 1 (start-31718)	Stage 2 (31718-end)	Overall
MLP-FC	Accuracy	60.5	96.3	71.2
	Throughput	1333.9	26324554.6	1905.6
MLP-MLP	Accuracy	60.5	96.3	71.2
	Throughput	–	358089.3	1902.6
MLP-ClusterPool	Accuracy	60.5	63.0	61.2
	Throughput	–	241028368.8	1905.6

MLP-MLP also increases overall accuracy and decreases throughput when Stage 1 gets longer, except the 50% Stage 1 length. In this case, overall accuracy was affected by the drop in Stage 2 accuracy. This could be caused by the Stage 1 MLP model overfitting the data, which could not be overcome by the MLP-MLP configuration, but

nevertheless was successfully overcome by the MLP-FC version with the aid of Fourier coefficient energy thresholding. MLP-ClusterPool approach depicts almost the same performance trend along various Stage 1 sizes, except the abnormally better accuracy given when 10% data in Stage 1 compared to other partition sizes. As per detailed experimental logs, this happens due to the imbalance of data that appears in the initial 10% segment of data belonging to Stage 1.

Analysis of the staged learning based results reveals a sharp increase of throughput in Stage 2 for all three methods, especially for the MLP-FC and MLP-ClusterPool versions. This obvious throughput advantage occurs as a consequence of the suspension of the real-time stream learner as proposed in the stage learning framework. Clearly, the gain of MLP-ClusterPool's throughput costs its accuracy in all 4 Stage 1 size configurations, unlike with MLP-FC. This ubiquitous Stage 2 dilemma of accuracy versus throughput has been gracefully managed by MLP-FC in all cases except for the 10% data length segment for Stage 1. This shorter Stage 1 duration highly likely resulted in model overfitting which can be even worse if data is imbalanced within that particular partition as observed via result logs in Elec 10%. All 3 methods result in significant Stage 2 accuracy growth for 70% Stage 1 segment length, thus indicating strong episodes of concept recurrence.

Stage-wise performance measures for the Sensor dataset

Tables 7.5 to 7.8 show continuous growth in overall accuracy, yet exhibit a decline in throughput with MLP-FC when the length of Stage 1 increases, in the same way as with the Elec dataset above. The percentage of accuracy increase is relatively lesser; to be precise it is only 23.5% while it was 419.7% for Elec. This gives an indication of the degree of homogeneity between Elec and Sensor. It appears that sensor record partitions are more homogeneous compared to Elec and therefore the correlation between Stage 1 observation count and the accuracy is not as strong as with Elec, although it does

manifest. The decline of throughput along the 10% Stage 1 to 70% Stage 1 size segment range is 86.1%, which is much similar to Elec. The rationale behind this approximately similar throughput decline would be the minor difference in the number of attributes in two datasets.

Table 7.5: Sensor: 10% data for Stage 1

		Stage 1 (start-13007)	Stage 2 (13007-end)	Overall
MLP-FC	Accuracy	51.8	50.0	50.6
	Throughput	1563.9	418990694.3	15639.0
MLP-MLP	Accuracy	51.8	58.7	56.6
	Throughput	–	369778.1	15066.0
MLP-ClusterPool	Accuracy	51.8	40.0	43.6
	Throughput	–	4739514170.0	15639.5

Table 7.6: Sensor: 30% data for Stage 1

		Stage 1 (start-39021)	Stage 2 (39021-end)	Overall
MLP-FC	Accuracy	59.9	50.0	53.0
	Throughput	1524.6	362179793.2	5082.0
MLP-MLP	Accuracy	59.9	53.3	55.3
	Throughput	–	376124.7	5034.4
MLP-ClusterPool	Accuracy	59.9	40.0	46.0
	Throughput	–	1424913928.0	5082.0

Significant throughput advantage in Stage 2 is common with all three approaches for the same reason of the absence of an online classifier. In this Sensor dataset also,

Table 7.7: Sensor: 50% data for Stage 1

		Stage 1 (start-65036)	Stage 2 (65036-end)	Overall
MLP-FC	Accuracy	56.9	50.0	53.5
	Throughput	1543.0	293488267.1	3085.9
MLP-MLP	Accuracy	56.9	53.6	55.3
	Throughput	–	341998.9	3072.1
MLP-ClusterPool	Accuracy	56.9	40.0	48.5
	Throughput	–	1725119363.4	3085.9

Table 7.8: Sensor: 70% data for Stage 1

		Stage 1 (start-91051)	Stage 2 (91051-end)	Overall
MLP-FC	Accuracy	59.6	69.2	62.5
	Throughput	1518.3	345633303.8	2169.0
MLP-MLP	Accuracy	59.6	66.5	61.7
	Throughput	–	359775.4	2165.1
MLP-ClusterPool	Accuracy	59.6	38.5	53.3
	Throughput	–	2097956989.2	2169.0

MLP-ClusterPool approach has failed to maintain accuracy in Stage 2 for all 4 different stage lengths. MLP-FC also dropped its accuracy in Stage 2, until it observes 70% of data in Stage 1 where it reports significant accuracy improvement in Stage 2. Observing more data appears to be helpful when recognising miniature variations. MLP-MLP classifier reports a 6.9% increase in accuracy for both 10% and 70% Stage 1 size segments.

Stage-wise performance measures for the Occupancy dataset

Occupancy dataset follows the same trend of overall accuracy increase while the size of Stage 1 is growing except for the case of longest Stage 1 segment which is 70% for MLP-FC. The drop in throughput is 86%, similar to the above two datasets. Even though Stage 1 with 70% of data could observe 20% more data compared to 50% data in Stage 1 case, there is no any positive effect on the accuracy of Stage 1 for any of the classifiers. In contrast, MP-MLP and MLP-ClusterPool Stage 2 accuracies were

boosted as a result of more evidence observed in Stage 1.

Table 7.9: Occupancy: 10% data for Stage 1

		Stage 1 (start-2056)	Stage 2 (2056-end)	Overall
MLP-FC	Accuracy	84.4	69.2	73.8
	Throughput	1682.8	19297105.7	16825.2
MLP-MLP	Accuracy	84.4	70.6	74.7
	Throughput	–	313706.9	16052.6
MLP-ClusterPool	Accuracy	84.4	30.8	46.9
	Throughput	–	722812500.0	16827.2

Table 7.10: Occupancy: 30% data for Stage 1

		Stage 1 (start-6168)	Stage 2 (6168-end)	Overall
MLP-FC	Accuracy	84.8	75.0	77.9
	Throughput	1631.2	76962566.8	5437.1
MLP-MLP	Accuracy	84.8	60.2	67.6
	Throughput	–	369625.6	5381.9
MLP-ClusterPool	Accuracy	84.8	41.7	54.6
	Throughput	–	515842293.9	5437.3

Table 7.11: Occupancy: 50% data for Stage 1

		Stage 1 (start-10280)	Stage 2 (10280-end)	Overall
MLP-FC	Accuracy	85.2	100.0	92.6
	Throughput	1315.1	102492522.4	2630.1
MLP-MLP	Accuracy	85.2	76.6	80.9
	Throughput	–	429564.8	2622.1
MLP-ClusterPool	Accuracy	85.2	75.0	80.1
	Throughput	–	441201716.7	2630.1

Table 7.12: Occupancy: 70% data for Stage 1

		Stage 1 (start-14392)	Stage 2 (14392-end)	Overall
MLP-FC	Accuracy	85.0	100.0	89.5
	Throughput	1607.8	70011350.7	2296.8
MLP-MLP	Accuracy	85.0	79.3	83.3
	Throughput	–	374437.7	2292.6
MLP-ClusterPool	Accuracy	85.0	100.0	89.5
	Throughput	–	270526315.8	2296.8

In the case of 10% and 30% Stage 1 sizes, this dataset compromises its Stage 1 accuracy in Stage 2 with all three approaches due to the immaturity of its learned patterns. When MLP learned over 50% or more data in Stage 1, Stage 2 classification accuracy is increased with MLP-FC. Stage 2 throughput benefitted in all approaches, as usual.

Stage-wise performance measures for the Flight dataset

MLP-FC's overall accuracy for the Flight dataset does not improve any further after the 30% Stage 1 segment length. Common with the other datasets, throughput is decreasing, yet the difference between best and worst is only 29% for MLP-FC. This illustrates a lesser dependability of classification performance on the period it spent in Stage 1 learning when compared to the other datasets. Precisely, lengthier the Stage 1 segment, the lesser is the Stage 1 accuracy.

MLP-MLP overall accuracy kept decreasing when Stage 1 is lengthier while MLP-ClusterPool overall accuracy fluctuates.

When it comes to stages, MLP-FC and MLP-MLP experience significant accuracy decline in Stage 2 regardless of the length in Stage 1. Interestingly, MLP-ClusterPool approach performs better in Stage 2, except when the Stage 1 length is only 10%. Stage 2 accuracy loss of MLP-FC and MLP-MLP is occurred because of Stage 1 MLP model

overfitting. In contrast to previous datasets, this dataset has more features (30) and therefore the chance of having irrelevant features is even higher. It appears that the MLP model learns in Stage 1 gets confused when only a few features are influential, which is not the case with this dataset.

Table 7.13: Flight: 10% data for Stage 1

		Stage 1 (start-2504)	Stage 2 (2504-end)	Overall
MLP-FC	Accuracy	82.1	63.8	69.3
	Throughput	568.8	837.7	799.9
MLP-MLP	Accuracy	82.1	73.4	76.0
	Throughput	–	182850.4	5533.7
MLP-ClusterPool	Accuracy	82.1	79.2	80.1
	Throughput	–	3917986.3	5681.2

Table 7.14: Flight: 30% data for Stage 1

		Stage 1 (start-7512)	Stage 2 (7512-end)	Overall
MLP-FC	Accuracy	81.3	70.4	73.7
	Throughput	565.8	769.4	694.4
MLP-MLP	Accuracy	81.3	70.9	74.0
	Throughput	–	183685.1	1872.7
MLP-ClusterPool	Accuracy	81.3	85.5	84.3
	Throughput	–	2003817.7	1884.9

To sum up, Stage 2 constantly magnifies throughput for all 3 classifiers for the reason of using a computationally effective offline learning strategy. The risk of not being able

Table 7.15: Flight: 50% data for Stage 1

		Stage 1 (start-12521)	Stage 2 (12521-end)	Overall
MLP-FC	Accuracy	77.1	65.3	71.2
	Throughput	565.7	644.4	602.5
MLP-MLP	Accuracy	77.1	67.9	72.5
	Throughput	–	184282.0	1128.0
MLP-ClusterPool	Accuracy	77.1	84.0	80.5
	Throughput	–	1067255.3	1130.9

Table 7.16: Flight: 70% data for Stage 1

		Stage 1 (start-17530)	Stage 2 (17530-end)	Overall
MLP-FC	Accuracy	72.6	52.8	66.7
	Throughput	567.1	563.6	566.1
MLP-MLP	Accuracy	72.6	66.4	70.8
	Throughput	–	186927.3	809.1
MLP-ClusterPool	Accuracy	72.6	86.8	76.9
	Throughput	–	1090405.1	810.0

to provide better or same accuracy as Stage 1 has been fruitfully achieved by MLP-FC in Elec (30%,50%,70%), Sensor (70%), and Occupancy (50%, 70%), though not in Flight. MLP-MLP provides better Stage 2 accuracy in Elec (30%, 70%), and Sensor (10%, 70%), yet not with any of the Occupancy and Flight settings. MLP-ClusterPool surpasses Stage 1 accuracy in Stage 2 with Elec (70%), Occupancy (70%), and Flight (30%,50%,70%), though not with any of the Sensor data experiments.

Out of these success cases, MLP-FC becomes the winner in Elec (70%), Sensor(70%), and Occupancy (70%), while reporting best throughput as well. MLP-MLP accuracy is slightly better than MLP-FC in Elec (30%), but throughput is slightly lower. MLP-ClusterPool wins flight (30%,50%,70%), while it ties with MLP-FC in Occupancy (70%) in both performance measures.

7.6 Conclusion

This chapter firstly presented an approach to derive a Fourier spectrum from the neural network learned in Stage 1. The application of the novel cluster pool method which extracts a set of compressed schema and their corresponding class labels derived through MLP deployed at the end of Stage 1 are analogous to truth tables taken from a decision tree classifier.

Secondly, the applicability of derived Fourier spectra is verified over 4 different real-world datasets. Performance were compared and contrasted against two other options MLP classifier and Cluster pool obtained at the end of Stage 1. Out of these MLP-FC, MLP-MLP, and MLP-ClusterPool classifier systems, MLP-FC were relatively better in terms of both accuracy and throughput over a few more datasets than the runner-up, MLP-ClusterPool. Even though it seems MLP-FC is suitable for SOL in principle, several more potential advancements can be suggested.

For the intention of developing consistently robust solution, three main recommendations are listed: (1) integration of some form of real-time learning capability in Stage 2, (2) timely, yet firm evidence driven stage transition is necessary to incorporate rather than having trial and error data partitions for stage changes, (3) a concept drift sensitive strategy for learning in both Stages 1 and 2.

Chapter 8

Conclusion

8.1 Research achievements

This study concentrated on the maximisation of throughput in a stream classification context while retaining or bettering accuracy compared to state of the art data stream classifiers. In order to address this research problem, four objectives were formulated. A summary of accomplishments attained in achieving each of these objectives is detailed below.

8.1.1 Designing a context sensitive Staged Learning framework

A novel context sensitive staged learning framework is presented for concept drifting and recurring data streams. Two stages, namely Stage 1 and Stage 2, are responsible for learning in accordance with the level of volatility present in the data stream. High volatility stream segments, where the rate of appearance of previously unseen concepts is higher, are to be learned in Stage 1. Capturing patterns in the highly dynamic Stage 1 was facilitated by an ensemble of incremental learners. In Stage 2, the learning was carried out by a repository of previously learned concepts from Stage 1, together with a relatively lesser level of real-time ongoing learning capability.

The transition between stages was triggered in line with the statistically significant volatility trends. Two different stage change detectors were proposed; each fed with context information gathered for triggers T_1 and T_2. The detector of trigger T_1 continuously senses the rate of recurrences while the system is operating in Stage 1. Two hypotheses sets are tested prior to the decision to activate Stage 2. On the other hand, the trigger T_2 that is accountable for reactivation of Stage 1 collects evidence for the appearance of unseen concepts whilst working on Stage 2. The decision of triggering T_2 is also endorsed by a hypothesis test.

Since Stage 2 of this framework is operated under a limited real-time learning capability and mostly relies on stored concepts, this design was supposed to provide a significant reduction in computational overhead in contrast with other systems. Volatility driven triggers were expected to guarantee the timely transition between stages so that accuracy was not compromised.

8.1.2 Implementation and evaluation of the proposed framework

The staged learning framework was illustrated using two different algorithms and those were implemented with an ensemble of Hoeffding decision trees as the base classifier. At concept drift points, the tree with the highest accuracy was transformed into a Fourier spectrum and stored in an online repository in the event that a similar spectrum was not already resident in the repository. This version of the implementation was named as Staged Online Learner (SOL). The Fourier spectrum also acted as classifiers in the event that concepts recurred from the past. Each classifier was equipped with a concept drift detector to detect when new concepts occurred whereas stage detectors were responsible for transiting between stages.

While learning in Stage 1, the system kept feeding the Trigger 1 detector until a change was signalled. Whenever the statistical tests for T_1 were satisfied, the system

suspended Stage 1 operation which included the maintenance of the expensive ensemble learner. Thereafter, learning in Stage 2 was facilitated by the repository of spectra and a single decision tree obtained from the current best spectrum selected at each concept drift point.

The implemented framework was assessed for its feasibility against the criteria: effectiveness of triggers T_1 and T_2, throughput, memory usage, and accuracy. The detailed, comparative experimental study done over a variety of datasets revealed favourable outcomes. The triggers were capable of sensing the volatility and signalled stage transitions as appropriate. Given stream segments that were less volatile, classification could cope through stored patterns with little adjustments as evident by the absence of significant accuracy drops. The robustness of the staged learning platform was recognised on account of its consistent performance measures on a range of challenging recurrence scenarios. As expected, significant performance benefits of throughput and memory were gained without loss of accuracy in Stage 2, and consequently on an overall basis.

Though the results were encouraging, bidirectional conversion of tree and spectrum that happened at each drift point in Stage 2 appeared to be problematic. A sudden increase in processing time observed at drift points detracts from a smooth processing rate which is a highly desirable goal. With the aim of optimising the design of the framework to avoid such time peaks, the next level of this research proposed an extension to SOL.

8.1.3 Design, implementation, and evaluation of advanced version of the framework

The extension named Staged Online Learner with Incremental Fourier Classifier (SOL-IFC) contributed three novel aspects: an incrementally adaptive Fourier Classifier as

the learner for low volatile data segments (Stage 2), removal of noisy features through schema pruning, and instance schema synopsis with the help of a self-indexing hashing scheme. Two theorems and proof were given in relation to the incremental update of Fourier coefficients. Schema instances that changed their class labels since the last update of the spectrum contributed to the refinement process. If the rate of class label changes is greater than the user-defined tolerable change rate, the current coefficient set is modified as given in Theorems 1 and 2 of chapter 5.

The subset of influential features for a particular concept was recognised on the basis of the winner spectrum. Decision nodes of the corresponding tree that spectrum was originated from and coefficients which actively contributed to the Fourier basis function under user-specified energy threshold were retained as influential.

Synopses of instance schema were populated in the spectrum reservoir through the use of a hash schema devised from knowledge of the set of influential features that were retained after the feature selection process. These synopses served two purposes: firstly, determination of timing for spectrum refinement; and secondly, classification from cached class labels in the case of schema recurrences. The entire process of spectrum refinement with the use of the feature selection approach and reservoir management strategy was illustrated in two algorithms.

Performance of SOL-IFC was empirically tested against several comparative algorithms including its predecessor SOL, and nine more meta-learners given in MOA. In most cases, this novel algorithm outperformed other algorithms in terms of throughput as well as accuracy. These results confirmed the improvement obtained by the use of feature selection, hash reservoir, and the incremental Fourier classifier. Furthermore, because of the smooth incremental adaptation of SOL-IFC, it could avoid the processing time spike issue observed in the naïve version of SOL.

In summary, while both SOL and SOL-IFC met the research requirement of throughput versus accuracy trade-off, SOL-IFC enhanced the staged learning approach further.

It ranked at first place when evaluated on a combined metric of throughput and accuracy compared to existing state-of-art algorithms in data stream mining domain.

8.1.4 Examination of the generalisability of the proposed framework

Lastly, this study explored the potential of using classifiers other than decision trees in Stage 1 of the staged learning framework. As a proof of concept, the feed-forward Multi-Layer Perceptron (MLP) Neural Network (NN) was taken as the Stage 1 learner. The major challenge of obtaining Fourier spectra from NN was successfully addressed through the proposed cluster pool structure.

A brief experimental study done with this implementation of MLP NN for Stage 1 and the Fourier model generated from NN for Stage 2 classifier illustrated its applicability in data streams. Results were positive for the implementation independence of staged learning framework, although further advancements were recommended for obtaining even better performance.

8.2 Limitations of this research

This research has accomplished its objectives by contributing a novel staged learning paradigm, and three different product versions were produced, namely naïve SOL, SOL-IFC, and MP-FC. Moreover, the SOL-IFC version itself contains three novel conceptual contributions. Most importantly, the well-known dilemma of designing more accurate classifiers or classifiers with higher processing speed was gracefully met through SOL-IFC which outperformed comparable stream mining solutions. Hence, the research objectives set out at the commencement of the study have been successfully addressed.

Notwithstanding the success of the research work, there have been some constraints and limitations attached to some of the approaches in this research. A brief discussion of such limitations is given below.

- Scaling generation of Fourier spectra to high dimensional data

  The relationship between the number of features in a dataset and the total number of Fourier coefficients in a spectrum is exponential in general. Even though most of those coefficients are not significant, initial extraction of coefficients is a computationally expensive task. This potential drawback was mitigated in this work by applying several techniques such as coefficient calculation optimisation, focusing on features that only appeared in a decision tree rather than the number that appeared in the original raw data, and the IFC approach that eliminated the need for the generation of new Fourier spectra during Stage 2. Furthermore, the overhead of inverse Fourier computation was reduced in several ways through the use of energy thresholding based coefficient selection, feature selection, classification using cached class labels, and concentrating only on the set of schemas that changed class labels during the course of spectrum refinement in Stage 2. In the worst-case scenario where all features appear in a decision tree, the DFT process in Stage 1 will considerably slow down the mining process.

- Stage 2 adaptation limitations of SOL-IFC

  The Stage 2 noisy feature pruning assumes that there is no chance for a feature to be influential once it was declared as not influential at the first recurrence of a concept in Stage 2. The hash function identifies the subset of features as non-influential with respect to a given spectrum and then it continues its operation with the same set while the system is operating in that cycle of Stage 2. This limitation of adaptability in Stage 2 of SOL-IFC is not specifically addressed in the current solution. SOL-IFC makes the optimistic assumption that the option

of Stage 1 reactivation will mitigate against the loss of features removed during the course of Stage 2. This is due to the fact that Stage 1 effectively performs a system reset as it generates a new batch of decision trees that are grown that could incorporate these very same features that were removed in the previous system cycle of Stage 2.

- Delay in receiving true class label

  The performance, especially the accuracy of the solution presented in this study relies on prompt concept drift detection followed by the best model selection for the next concept. Drift detectors are fed with binary 0 or 1 after receiving the true class label of the instance. Therefore, a delay in the arrival of the true class label causes a delay in drift detection, and hence a delay in the selection of the most accurate classifier of the ensemble as the best classifier. The system suggested in this study solely depends on the accuracy of the best classifier for a given concept as it is based on the recurrence capture concept. For this reason, a deferral in detection and delayed response to a drift can result in an accuracy drop in the system. At the same time, this situation causes a delay in populating the repository with spectra which results in a delay in a stage change detection. Staying in Stage 1 for a longer time period of time will negatively impact throughput.

- Limitations of change detectors

  The success of the staged learning framework highly depends on the quality of spectra in the repository. The quality of spectra is dependent on the accurate detection of drift signal. Errors that occur in concept capture can propagate to the spectrum aggregation and spectrum refinement processes. Therefore, selection of the concept drift detector with the lowest false positive and false negative rates is crucial. On the other hand, false positive and false negatives that occur with respect to stage change detection can have an even higher impact as it could direct

the entire system into a different state prematurely. From a practical point of view, activation or deactivation decisions of the real-time learner have to be made on the basis of the best information that is available, but a risk exists that such a decision is made erroneously. This reinforces the importance of stage change detector design considerations.

- User defined parameters

  In common with most other machine learning algorithms, SOL and SOL-IFC also suffer from the curse of reliance on parameter tuning. The staged learning approach solution features six major parameters: α, β, repository size, sample size, tolerable change rate, and refinement interval length. Parameters α, β, repository size, and sample size are involved in triggering T_1 and T_2 whereas the tolerable change rate and interval length are for refinements in Stage 2 of SOL-IFC. Among them, parameters α, β, and repository size were shown to have a negligible effect on system accuracy.

  The throughput of the SOL framework was negatively affected when the cut-off value for α was too high. Certainly, it is not practical to set α to a value less than 0.5 as repository utility should not be measured on the basis of random chance. Therefore, α is set to 0.5 throughout this work. The influence of repository size on throughput appeared to be dependent on the dynamics of the dataset. Considering its potential to generate new spectra continuously, we set the maximum allowable spectrum count to 40 with SOL whereas it was 20 for SOL-IFC since the later learns incrementally with available spectra. Tolerable change rate was set to 0.05 for SOL-IFC as it analogous to a statistical confidence. Interval length parameter was set to 100 which is statistically significant. It was observed that slight variations of tolerable change rate or interval length do not have a considerable effect on system performance.

The higher the β value, the higher the probability of triggering T_2 and hence throughput was reduced. On the contrary, smaller β results in a lengthier Stage 2 at the risk of accuracy loss in the long run even though there was no immediate effect on accuracy. For these set of experiments, we set β to a moderate value 0.7 so as to cope with accuracy vs throughput trade-off. However, more studies need to be done in order to recognise influential parameters and their influence when determining optimum β. The sample size parameter of stage change detection, analogous to window size in a drift detection problem, however, has a substantial influence on timely stage change detection and ultimately on system performance. Small sample size may results in false positives and consequently an accuracy drop, whereas large samples lead to false negatives and low throughput thereby. Our current implementation resolves this dilemma by picking a statistically representative value of 200 which is proven to be the best for the change detector SeqDrift. Nevertheless, we note that the stage change detector does not need to be set with a constant sample size as it can be optimised with knowledge of the level of volatility in the data stream. Further research needs to be done in setting a context sensitive and self-adaptive sample size parameter value.

- Streams with few or no recurrences

  Stage 1 of the design given in this study also follows a common incremental ensemble learning approach with embedded drift detectors while the repository is being populated. In data stream environments where there are few or no recurrences, SOL and SOL-IFC might also experience low throughput as the system has to continue in Stage 1 for a longer period of time. In such cases, if resources are running out and the mining process continuously fails to meet the data arrival rate, it is recommended to set an upper boundary for the duration of Stage 1 operation. Once the stream reaches its upper bound, load shedding or

forced Stage 2 transition could be done as remedies at the cost of accuracy rather than accumulating the delay for the long run. If none of the spectra in Stage 2 gives an acceptable accuracy in this scenario, a new spectrum can be generated directly from data without the intermediate step of generating it from a decision tree.

8.3 Future work

There are several directions that future research can take for improving the solution proposed in this work. Those are listed as follows.

- Declare drift warning signal

  Defining a warning signal ahead of the actual drift signal will provide an opportunity for more accurately assessing the best performing classifier. The best performing classifier within the warning period can be considered to be the dominant classifier for the newly emerging concept. Furthermore, solutions like this which solely depend on the best classifier's accuracy until the end of that particular concept will have an opportunity to suspend and reset other learners in the ensemble until the next warning signal for the next concept shift occurs. This will also lead to an increase in throughput as overhead is decreased.

- Parallel processing

  Maintenance of individual trees in the forest and spectra in the repository demands significant processing power in an ensemble classifier system. In addition to classification, updating their own statistics and feeding their own drift detectors also need to be done for each learner. Interestingly, those tasks are mutually exclusive and therefore good candidates for multithreading. Given hardware requirements, each classifier can be allocated to a separate thread for processing.

- DFT on other classifiers

  Other than decision trees and neural networks, exploring the ability to convert other incremental learners such as Naïve Bayes and KNN to Fourier spectra might be important. In this way, staged learning approach with Incremental Fourier classifier can be applied to a greater diversity of application domains.

- Proactive stage transition for better resources utilisation

  It is evident that Stage 2 needs a significantly lesser amount of computational resources compared to Stage 1. Therefore, proactive resource utilisation in line with stage transition might support the concept of green computing. Rather than being reactive to the volatility of the stream, the generation of a stage change prediction model will be helpful for taking a proactive approach to this problem. Time series analysis on stage transition metadata will be useful in building such a prediction model.

- Fourier spectrum based feature drift detection

  In high volatility data segments, most of the concept switches resulted from the emergence of new concepts. In contrast, concept switches in low volatility stream segments are expected to see recurrences. Changes between consecutive recurrences of a particular concept might mostly be caused as a result of a shift in the importance of the features. This variation of feature importance which is called a feature drift might be an interesting topic to proceed with. In Stage 2 learning, the study of the deviation between the refined spectrum and the initial spectrum might provide a clue to recognise feature drifts. Agreement of classification outcome between the initial classifier and its refined version can be observed over the time. The binary output ‘agree’ or ‘disagree’ can be fed into the feature drift detector.

This research field of concept drifting data stream mining is challenging, yet essential due to the increasing number of sources which generate data streams. It is necessary to explore more enhancements and novelties in the future.

References

Abdulsalam, H., Skillicorn, D. B. & Martin, P. (2007). Streaming random forests. In *Proceedings of the 11th international database engineering and applications symposium* (pp. 225–232). Washington, DC, USA: IEEE Computer Society. Retrieved from https://doi.org/10.1109/IDEAS.2007.42 doi: 10.1109/IDEAS.2007.42

Abdulsalam, H., Skillicorn, D. B. & Martin, P. (2008). Classifying evolving data streams using dynamic streaming random forests. In *Proceedings of the 19th international conference on database and expert systems applications* (pp. 643–651). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dx.doi.org/10.1007/978-3-540-85654-2_54 doi: 10.1007/978-3-540-85654-2_54

Aggarwal, C. (2007a). Data streams: Models and algorithms. In C. Aggarwal (Ed.), (pp. 1–8). Springer US. doi: 10.1007/978-0-387-47534-9

Aggarwal, C. (2007b). *Data streams: Models and algorithms* (1st ed.; C. Aggarwal, Ed.). Springer US. doi: 10.1007/978-0-387-47534-9

Aggarwal, C. & Yu, P. (2007). Data streams: Models and algorithms. In C. Aggarwal (Ed.), *Data streams: Models and algorithms* (pp. 169–207). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/978-0-387-47534-9_9 doi: 10.1007/978-0-387-47534-9_9

Alippi, C., Boracchi, G. & Roveri, M. (2013, April). Just-in-time classifiers for recurrent concepts. *IEEE Transactions on Neural Networks and Learning Systems*, *24*(4), 620–634. doi: 10.1109/TNNLS.2013.2239309

Babcock, B., Datar, M. & Motwani, R. (2004, April). Load shedding for aggregation queries over data streams. In *Proceedings. 20th international conference on data engineering* (pp. 350–361). doi: 10.1109/ICDE.2004.1320010

Babcock, B., Datar, M. & Motwani, R. (2007). Load shedding in data stream systems. In C. Aggarwal (Ed.), *Data streams: Models and algorithms* (pp. 127–147). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/978-0-387-47534-9_7 doi: 10.1007/978-0-387-47534-9_7

Baena-Garcia, M., Campo-Avila, J., Fidalgo-Merino, R., Bifet, A., Gavald, R. & Bueno, R. (2006). Early drift detection method. In *In fourth international workshop on knowledge discovery from data streams* (pp. 77–86).

Bernstein, S. N. (1946). *The theory of probabilities* (4th ed.). Moscow, Leningrad: Gostekhizdat.

Bifet, A., Frank, E., Holmes, G. & Pfahringer, B. (2010). Accurate ensembles for

data streams: Combining restricted hoeffding trees using stacking. In M. Sugiyama & Q. Yang (Eds.), *Proceedings of 2nd asian conference on machine learning* (Vol. 13, pp. 225–240). Tokyo, Japan: PMLR. Retrieved from http://proceedings.mlr.press/v13/bifet10a.html

Bifet, A. & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 siam international conference on data mining* (pp. 443–448). Retrieved from http://epubs.siam.org/doi/abs/10.1137/1.9781611972771.42 doi: 10.1137/1.9781611972771.42

Bifet, A. & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Proceedings of the 8th international symposium on intelligent data analysis: Advances in intelligent data analysis viii* (pp. 249–260). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dx.doi.org/10.1007/978-3-642-03915-7_22 doi: 10.1007/978-3-642-03915-7_22

Bifet, A., Holmes, G. & Pfahringer, B. (2010). Leveraging bagging for evolving data streams. In J. L. Balcázar, F. Bonchi, A. Gionis & M. Sebag (Eds.), *Machine learning and knowledge discovery in databases: European conference, ecml pkdd 2010, barcelona, spain, september 20-24, 2010, proceedings, part i* (pp. 135–150). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-15880-3-15 doi: 10.1007/978-3-642-15880-3-15

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R. & Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th acm sigkdd international conference on knowledge discovery and data mining* (pp. 139–148). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1557019.1557041 doi: 10.1145/1557019.1557041

Bifet, A., Read, J., Holmes, G. & Pfahringer, B. (2018). Data mining in time series and streaming databases. In A. K. Mark Last Horst Bunke (Ed.), (pp. 1–25). World Scientific. doi: https://doi.org/10.1142/10655

Breiman, L. (2001, 01 Oct). Random forests. *Machine Learning*, *45*(1), 5–32. Retrieved from https://doi.org/10.1023/A:1010933404324 doi: 10.1023/A:1010933404324

Brzeziński, D. & Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. In E. Corchado, M. Kurzyński & M. Woźniak (Eds.), *Hybrid artificial intelligent systems: 6th international conference, hais 2011, wroclaw, poland, may 23-25, 2011, proceedings, part ii* (pp. 155–163). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-21222-2-19 doi: 10.1007/978-3-642-21222-2-19

Candanedo, L. M. & Feldheim, V. (2016). Accurate occupancy detection of an office room from light, temperature, humidity and {CO2} measurements using statistical learning models. *Energy and Buildings*, *112*, 28–39. Retrieved from http://www.sciencedirect.com/science/article/pii/S0378778815304357 doi: https://doi.org/10.1016/j.enbuild.2015.11.071

Carletta, J. (1996, June). Assessing agreement on classification tasks: The kappa statistic. *Comput. Linguist.*, *22*(2), 249–254. Retrieved from http://dl.acm.org/citation.cfm?id=230386.230390

Chi, Y., Yu, P. S., Wang, H. & Muntz, R. R. (2005). Loadstar: A load shedding scheme for classifying data streams. In *Proceedings of the 2005 siam international conference on data mining* (pp. 346–357). Retrieved from https://epubs.siam.org/doi/abs/10.1137/1.9781611972757.31 doi: 10.1137/1.9781611972757.31

Cormode, G., Garofalakis, M., Haas, P. J. & Jermaine, C. (2012, January). Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, *4*(1–3), 1–294. Retrieved from http://dx.doi.org/10.1561/1900000004 doi: 10.1561/1900000004

Demšar, J. (2006, December). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30. Retrieved from http://dl.acm.org/citation.cfm?id=1248547.1248548 doi: 2007-03550-001

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple classifier systems* (pp. 1–15). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ditzler, G., Roveri, M., Alippi, C. & Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, *10*(4), 12–25. doi: 10.1109/MCI.2015.2471196

Domeniconi, C. & Gunopulos, D. (2001). Incremental support vector machine construction. In *Proceedings 2001 ieee international conference on data mining* (pp. 589–592).

Domingos, P. & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth acm sigkdd international conference on knowledge discovery and data mining* (pp. 71–80). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/347090.347107 doi: 10.1145/347090.347107

Domingos, P. & Hulten, G. (2001). Catching up with the data: Research issues in mining data streams. In *In workshop on research issues in data mining and knowledge discovery.*

Elwell, R. & Polikar, R. (2009). Incremental learning of variable rate concept drift. In J. A. Benediktsson, J. Kittler & F. Roli (Eds.), *Multiple classifier systems* (pp. 142–151). Berlin, Heidelberg: Springer Berlin Heidelberg.

Fan, W., Wang, H., Yu, P. S., Lo, S.-h. & Stolfo, S. (2002). Progressive modeling. In *Proceedings of the 2002 ieee international conference on data mining.* IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=844380.844728

Ferrer Troyano, F., Aguilar-Ruiz, J. S. & Riquelme, J. C. (2005). Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005 acm symposium on applied computing* (pp. 568–572). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1066677.1066808 doi: 10.1145/1066677.1066808

Ferrer-Troyano, F. J., Aguilar-Ruiz, J. S. & Santos, J. C. R. (2006). Data streams

classification by incremental rule learning with parameterized generalization. In *Sac*.

Gaber, M. M., Zaslavsky, A. & Krishnaswamy, S. (2005, June). Mining data streams: A review. *SIGMOD Rec.*, *34*(2), 18–26. Retrieved from http://doi.acm.org/10.1145/1083784.1083789 doi: 10.1145/1083784.1083789

Gaber, M. M., Zaslavsky, A. & Krishnaswamy, S. (2007). Data streams: Models and algorithms. In A. C.C. (Ed.), (pp. 39–59). Springer US. doi: 10.1007/978-0-387-47534-9

Gama, J. (2010). *Knowledge discovery from data streams* (1st ed.). Chapman & Hall/CRC.

Gama, J., Fernandes, R. & Rocha, R. (2006, January). Decision trees for mining data streams. *Intell. Data Anal.*, *10*(1), 23–45. Retrieved from http://dl.acm.org/citation.cfm?id=1239076.1239079

Gama, J. & Kosina, P. (2009). Tracking recurring concepts with meta-learners. In *Progress in artificial intelligence* (pp. 423–434). Springer.

Gama, J., Medas, P., Castillo, G. & Rodrigues, P. (2004). Learning with drift detection. In A. L. C. Bazzan & S. Labidi (Eds.), *Advances in artificial intelligence – sbia 2004* (pp. 286–295). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gama, J., Rocha, R. & Medas, P. (2003). Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth acm sigkdd international conference on knowledge discovery and data mining* (pp. 523–528). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/956750.956813 doi: 10.1145/956750.956813

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. (2014, March). A survey on concept drift adaptation. *ACM Comput. Surv.*, *46*(4), 44:1–44:37. Retrieved from http://doi.acm.org/10.1145/2523813 doi: 10.1145/2523813

Gao, J., Fan, W., Han, J. & Yu, P. S. (2007). A general framework for mining concept-drifting data streams with skewed distributions. In *Proceedings of the 2007 siam international conference on data mining* (pp. 3–14). Retrieved from https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.1 doi: 10.1137/1.9781611972771.1

Gardner, M. & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. *Atmospheric Environment*, *32*(14), 2627–2636. Retrieved from http://www.sciencedirect.com/science/article/pii/S1352231097004470 doi: https://doi.org/10.1016/S1352-2310(97)00447-0

Gomes, H., Barddal, J., Enembreck, F. & Bifet, A. (2017, March). A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, *50*(2), 23:1–23:36. Retrieved from http://doi.acm.org/10.1145/3054925 doi: 10.1145/3054925

Gomes, H., Bifet, A., Read, J., Barddal, J., Enembreck, F., Pfharinger, B., ... Abdessalem, T. (2017, 01 Oct). Adaptive random forests for evolving data stream classification. *Machine Learning*, *106*(9), 1469–1495. Retrieved from

https://doi.org/10.1007/s10994-017-5642-8 doi: 10.1007/s10994-017-5642-8

Gomes, J., Menasalvas, E. & Sousa, P. (2010). Tracking recurrent concepts using context. In M. Szczuka, M. Kryszkiewicz, S. Ramanna, R. Jensen & Q. Hu (Eds.), *Rough sets and current trends in computing: 7th international conference, rsctc 2010, warsaw, poland, june 28-30,2010. proceedings* (pp. 168–177). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-13529-3-19 doi: 10.1007/978-3-642-13529-3-19t

Gomide, D. F. L. . P. C. . F. (2009). Evolving granular classification neural networks. IEEE. doi: 10.1109/IJCNN.2009.5178895

Gudise, V. G. & Venayagamoorthy, G. K. (2003, April). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 ieee swarm intelligence symposium.* (pp. 110–117). doi: 10.1109/SIS.2003.1202255

Han, J., Kamber, M. & Pei, J. (2012). *Data mining: Concepts and techniques* (Third Edition ed.; J. Han, M. Kamber & J. Pei, Eds.). Morgan Kaufmann.

He, H., Chen, S., Li, K. & Xu, X. (2011, 10). Incremental learning from stream data. , *22*, 1901-14.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58(301)*, 13–30. doi: 10.2307/2282952

Hoeglinger, S., Pears, R. & Koh, Y. S. (2009). Cbdt: A concept based approach to data stream mining. In T. Theeramunkong, B. Kijsirikul, N. Cercone & T.-B. Ho (Eds.), *Advances in knowledge discovery and data mining: 13th pacific-asia conference, pakdd 2009 bangkok, thailand, april 27-30, 2009 proceedings* (pp. 1006–1012). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-01307-2-107 doi: 10.1007/978-3-642-01307-2-107

Hornik, K., Stinchcombe, M. & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. Retrieved from http://www.sciencedirect.com/science/article/pii/0893608089900208 doi: https://doi.org/10.1016/0893-6080(89)90020-8

Hulten, G., Spencer, L. & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh acm sigkdd international conference on knowledge discovery and data mining* (pp. 97–106). New York, NY, USA: ACM. doi: 10.1145/502512.502529

Ikonomovska, E., Gama, J. & Džeroski, S. (2011). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, *23*(1), 128–168. Retrieved from http://dx.doi.org/10.1007/s10618-010-0201-y doi: 10.1007/s10618-010-0201-y

Jaber, G., Cornuéjols, A. & Tarroux, P. (2013a). A new on-line learning method for coping with recurring concepts: The adacc system. In M. Lee, A. Hirose,

Z.-G. Hou & R. M. Kil (Eds.), *Neural information processing.iconip 2013.* (pp. 595–604). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-42042-9-74 doi: 10.1007/978-3-642-42042-9-74

Jaber, G., Cornuéjols, A. & Tarroux, P. (2013b). Online learning: Searching for the best forgetting strategy under concept drift. In M. Lee, A. Hirose, Z.-G. Hou & R. M. Kil (Eds.), *Neural information processing. iconip 2013.* (pp. 400–408). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-42042-9-50 doi: 10.1007/978-3-642-42042-9-50

Jain, A. K., Mao, J. & Mohiuddin, K. (1996). Artificial neural networks: A tutorial. *IEEE Computer*, *29*, 31–44.

Kargupta, H. & Park, B.-H. (2004, Feb). A fourier spectrum-based approach to represent decision trees for mining data streams in mobile environments. *IEEE Transactions on Knowledge and Data Engineering*, *16*(2), 216–229. doi: 10.1109/TKDE.2004.1269599

Kargupta, H., Park, B. H. & Dutta, H. (2006). Orthogonal decision trees. *Knowledge and Data Engineering, IEEE Transactions on*, *18*(8), 1028–1042. doi: doi.ieeecomputersociety.org/10.1109/TKDE.2006.127

Katakis, I., Tsoumakas, G. & Vlahavas, I. (2008). An ensemble of classifiers for coping with recurring contexts in data streams. In (pp. 763–764).

Kelly, M. G., Hand, D. J. & Adams, N. M. (1999). The impact of changing populations on classifier performance. In *Proceedings of the fifth acm sigkdd international conference on knowledge discovery and data mining* (pp. 367–371). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/312129.312285 doi: 10.1145/312129.312285

Khamassi, I., Mouchaweh, M. S., Hammami, M. & Ghedira, K. (2018). Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*. doi: https://doi.org/10.1007/s12530-016-9168-2

Kholghi, M., Hassanzadeh, H. & Keyvanpour, M. (2010). Classification and evaluation of data mining techniques for data stream requirements. In *2010 international symposium on computer, communication, control and automation (3ca).* doi: 10.1109/3CA.2010.5533759

Kithulgoda, C. I. & Pears, R. (2016). Staged online learning: A new approach to classification in high speed data streams. In *2016 international joint conference on neural networks (ijcnn)* (pp. 1–8). doi: 10.1109/IJCNN.2016.7727173

Kithulgoda, C. I., Pears, R. & Naeem, M. A. (2018). The incremental fourier classifier: Leveraging the discrete fourier transform for classifying high speed data streams. *Expert Systems with Applications*, *97*, 1–17. Retrieved from http://www.sciencedirect.com/science/article/pii/S095741741730845X doi: https://doi.org/10.1016/j.eswa.2017.12.023

Knapp, M. P. (2009). Sines and cosines of angles in arithmetic progression. In *Mathematics magazine* (Vol. 82, pp. 371–372). Mathematical Association of

America. doi: https://doi.org/10.4169/002557009X478436

Kolter, J. Z. & Maloof, M. A. (2007, December). Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.*, *8*, 2755–2790. Retrieved from http://dl.acm.org/citation.cfm?id=1314498.1390333

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J. & Wozniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, *37*, 132–156. Retrieved from http://www.sciencedirect.com/science/article/pii/S1566253516302329 doi: https://doi.org/10.1016/j.inffus.2017.02.004

Kuncheva, L. I. (2004). Classifier ensembles for changing environments. In F. Roli, J. Kittler & T. Windeatt (Eds.), *Multiple classifier systems* (pp. 1–15). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lazarescu, M. (2005). A multi-resolution learning approach to tracking concept drift and recurrent concepts. In *Pris.*

Lemaire, V., Salperwyck, C. & Bondu, A. (2015). Business intelligence. In K. R. Zimanyi E. (Ed.), (pp. 88–125). Springer, Cham. doi: https://doi.org/10.1007/978-3-319-17551-5_4

Lichman, M. (2013). *UCI machine learning repository.* Retrieved from http://archive.ics.uci.edu/ml

Maloof, M. A. & Michalski, R. S. (2004). Incremental learning with partial instance memory. *Artificial Intelligence*, *154*(1), 95–126. Retrieved from http://www.sciencedirect.com/science/article/pii/S0004370203001498 doi: https://doi.org/10.1016/j.artint.2003.04.001

Mehta, M., Agrawal, R. & Rissanen, J. (1996). Advances in database technology edbt96. In G. G. Apers P. Bouzeghoub M. (Ed.), (pp. 18–32). Springer, Berlin, Heidelberg. doi: https://doi.org/10.1007/BFb0014141

Mouss, H., Mouss, D., Mouss, N. & Sefouhi, L. (2004, July). Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In *Ieee 2004 5th asian control conference* (Vol. 2, pp. 815–818).

Nguyen, H. L., Woon, Y. K. & Ng, W. K. (2015). A survey on data stream clustering and classification. *Knowledge and Information Systems*, 535–569. doi: https://doi.org/10.1007/s10115-014-0808-1

Nguyen, H.-L., Woon, Y.-K., Ng, W.-K. & Wan, L. (2012). Heterogeneous ensemble for feature drifts in data streams. In P.-N. Tan, S. Chawla, C. K. Ho & J. Bailey (Eds.), *Advances in knowledge discovery and data mining* (pp. 1–12). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ning, G., Wang, H., Shu, L. & Yeh, G. (2016, 01 May). Towards load shedding and scheduling schemes for data streams that maintain quality and timing requirements of query results. *Soft Computing*, *20*(5), 1961–1976. Retrieved from https://doi.org/10.1007/s00500-015-1617-5 doi: 10.1007/s00500-015-1617-5

Nishida, K., Yamauchi, K. & Omori, T. (2005). Ace: Adaptive classifiers-ensemble system for concept-drifting environments. In N. C. Oza, R. Polikar, J. Kittler

& F. Roli (Eds.), *Multiple classifier systems* (pp. 176–185). Berlin, Heidelberg: Springer Berlin Heidelberg.

Nishimura, S., Terabe, M., Hashimoto, K. & Mihara, K. (2008). Learning higher accuracy decision trees from concept drifting data streams. In N. T. Nguyen, L. Borzemski, A. Grzech & M. Ali (Eds.), *New frontiers in applied artificial intelligence* (pp. 179–188). Berlin, Heidelberg: Springer Berlin Heidelberg.

Oza, N. C. (2005, Oct). Online bagging and boosting. In *2005 ieee international conference on systems, man and cybernetics* (Vol. 3, p. 2340-2345). doi: 10.1109/ICSMC.2005.1571498

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*(1-2), 100–115. Retrieved from http://dx.doi.org/10.1093/biomet/41.1-2.100 doi: 10.1093/biomet/41.1-2.100

Park, B. H. (2001). *Knowledge discovery from heterogeneous data streams using fourier spectrum of decision trees* (Unpublished doctoral dissertation). Washington State University, Pullman, WA, USA.

Pears, R., Sakthithasan, S. & Koh, Y. S. (2014). Detecting concept change in dynamic data streams. *Machine Learning*, *97*(3), 259–293. Retrieved from http://dx.doi.org/10.1007/s10994-013-5433-9 doi: 10.1007/s10994-013-5433-9it

Pfahringer, B., Holmes, G. & Kirkby, R. (2007). New options for hoeffding trees. In M. A. Orgun & J. Thornton (Eds.), *Ai 2007: Advances in artificial intelligence* (pp. 90–99). Berlin, Heidelberg: Springer Berlin Heidelberg.

Quinlan, J. R. (1986, 01 Mar). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. Retrieved from https://doi.org/10.1007/BF00116251 doi: 10.1007/BF00116251

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Ramamurthy, S. & Bhatn, R. (2007, Dec). Tracking recurrent concept drift in streaming data using ensemble classifiers. In *Sixth international conference on machine learning and applications (icmla)* (pp. 404–409). doi: 10.1109/ICMLA.2007.80

Ren, S., Lian, Y. & Zou, X. (2014). Incremental naive bayesian learning algorithm based on classification contribution degree. *Journal of Computers*, *9*.

Ross, G. J., Adams, N. M., Tasoulis, D. K. & Hand, D. J. (2012). Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, *33*(2), 191–198. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167865511002704 doi: https://doi.org/10.1016/j.patrec.2011.08.019

Sakthithasan, S., Pears, R., Bifet, A. & Pfahringer, B. (2015, July). Use of ensembles of fourier spectra in capturing recurrent concepts in data streams. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8). doi: 10.1109/IJCNN.2015.7280583

Schlimmer, J. C. & Granger, R. H. (1986, 01 Sep). Incremental learning from noisy data. *Machine Learning*, *1*(3), 317–354. Retrieved from https://doi.org/10.1007/BF00116895 doi: 10.1007/BF00116895

Scholz, M. & Klinkenberg, R. (2007, January). Boosting classifiers for drifting concepts. *Intell. Data Anal.*, *11*(1), 3–28. Retrieved from http://dl.acm.org/citation.cfm?id=1367489.1367491

Sobhani, P. & Beigy, H. (2011). New drift detection method for data streams. In *Proceedings of the second international conference on adaptive and intelligent systems* (pp. 88–97). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=2045295.2045309

Sripirakas, S. & Pears, R. (2014). Mining recurrent concepts in data streams using the discrete fourier transform. In L. Bellatreche & M. K. Mohania (Eds.), *Data warehousing and knowledge discovery: 16th international conference, dawak 2014, munich, germany, september 2-4, 2014. proceedings* (pp. 439–451). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-10160-6-39 doi: 10.1007/978-3-319-10160-6-39

Syed, N. A., Liu, H. & Sung, K. K. (1999). Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the fifth acm sigkdd international conference on knowledge discovery and data mining* (pp. 317–321). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/312129.312267 doi: 10.1145/312129.312267

Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M. & Stonebraker, M. (2003). Load shedding in a data stream manager. In J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger & A. Heuer (Eds.), *Proceedings 2003 vldb conference* (pp. 309–320). San Francisco: Morgan Kaufmann. Retrieved from http://www.sciencedirect.com/science/article/pii/B9780127224428500355 doi: https://doi.org/10.1016/B978-012722442-8/50035-5

Tsymbal, A. (2004). The problem of concept drift : definitions and related work..

van Rijn, J. N., Holmes, G., Pfahringer, B. & Vanschoren, J. (2015, Nov). Having a blast: Meta-learning and heterogeneous ensembles for data streams. In *2015 ieee international conference on data mining* (pp. 1003–1008). doi: 10.1109/ICDM.2015.55

Vitter, J. S. (1985, March). Random sampling with a reservoir. *ACM Trans. Math. Softw.*, *11*(1), 37–57. Retrieved from http://doi.acm.org/10.1145/3147.3165 doi: 10.1145/3147.3165

Wang, H., Fan, W., Yu, P. S. & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth acm sigkdd international conference on knowledge discovery and data mining* (pp. 226–235). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/956750.956778 doi: 10.1145/956750.956778

Wang, T., Li, Z., Hu, X., Yan, Y. & Chen, H. (2007). A new decision tree classification method for mining high-speed data streams based on threaded binary search trees. In *Proceedings of the 2007 international conference on emerging technologies in knowledge discovery and data mining* (pp. 256–267). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=1780582.1780612

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences* (Unpublished doctoral dissertation). Harvard University.

Widmer, G. & Kubat, M. (1996, 01 Apr). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101. Retrieved from https://doi.org/10.1007/BF00116900 doi: 10.1007/BF00116900

Yang, H. & Fong, S. (2011, Oct). Optimized very fast decision tree with balanced classification accuracy and compact tree size. In *The 3rd international conference on data mining and intelligent information technology applications* (pp. 57–64).

Yule, G. U. (1900). On the association of attributes in statistics: With illustrations from the material of the childhood society. *Philosophical Transactions of the Royal Society of London*. Retrieved from http://www.jstor.org/stable/90759

Zhang, P., Zhu, X., Tan, J. & Guo, L. (2010). Classifier and cluster ensembles for mining concept drifting data streams. In *Proceedings of the 2010 ieee international conference on data mining* (pp. 1175–1180). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dx.doi.org/10.1109/ICDM.2010.125 doi: 10.1109/ICDM.2010.125

Zheng, J., Shen, F., Fan, H. & Zhao, J. (2013, 01 Apr). An online incremental learning support vector machine for large-scale data. *Neural Computing and Applications*, *22*(5), 1023–1035. Retrieved from https://doi.org/10.1007/s00521-011-0793-1 doi: 10.1007/s00521-011-0793-1

Zhu, X. (2010). *Stream data mining repository.* Retrieved from http://www.cse.fau.edu/~xqzhu/stream.html

Zliobaite, I. (2010). Learning under concept drift: an overview. *CoRR*. Retrieved from http://arxiv.org/abs/1010.4784

Appendix A

Glossary

Concept A joint probability distribution of input features and class label, i.e. the relationship between the output variable and the input.

Concept drift A dissimilarity of the joint probability distribution of input features and class label at two subsequent time points.

Noise Meaningless information with respect to the mining task.

Explicit drift detection Recognition of concept drift through a drift detection strategy.

Throughput The number of instances processed per time unit.

Concept recurrence A reappearance of a previously seen pattern.

Repository A place where the collection of learned models are stored.

Context sensitive The ability to differentiate between segments that exhibit concept recurrences, from those that do not.

Schema instance A compact representation of a set of data instances, all of which share the same set of feature values.

Schema set A set of all possible schema for a given dataset.

Fourier partition An index of a Fourier spectrum.

Fourier coefficient A value which represents the "significance" of the corresponding Fourier partition.

Fourier partition set A collection of Fourier indexes.

Fourier basis function The function which takes a schema instance vector and a partition vector as input and produces an integer for a dataset with binary-valued features or a complex number for a dataset with non-binary feature values.

Fourier spectrum A set of Fourier coefficients indexes and corresponding values.

Order of a Fourier partition The number of nonzero feature values it contains.

Order of a Fourier coefficient The order of its corresponding partition.

Aggregation of spectra The definition of the second term.

Volatility The rate of appearance of new concepts in the stream with respect to time.

Stage 1 The period where unseen concepts appear and these concepts are learned and stored, i.e. high volatile stage.

Stage 2 The period where a vast majority of concepts are already learned in Stage 1, i.e. low volatile stage.

Staged learning The framework which switches learning between two stages Stage 1 and Stage 2 according to the volatility.

Evolving tree The tree that is induced from spectrum at the start of the concept and thereafter learns any changes in the concept that may take place after that point onwards.

Trigger The signal which indicates a transition between stages.

Repository hit When a concept extracted from the repository offers more accuracy for the newly emerging concept rather than from the tree forest.

Hit ratio The proportion between the count of repository hits to the total of hits and misses (repository failures, i.e. best match comes from the forest of trees).

Degree of agreement between two classifiers The proportion between the number of data instances where the classification outcome of both classifiers are the same to the total number of instances under consideration.

Distance similarity The degree of agreement between two spectra to be aggregated.

Structural similarity The degree of agreement between a spectrum and evolving tree.

Highly dynamic data stream A data stream where a combination of sample arrival rate and concept drift rate is relatively higher. It could cause system CPU utilization to rise above a maximum tolerable user-defined threshold.

Reservoir A dynamic data structure associated with a spectrum in Stage 2 of SOL-IFC. This captures changes in the relationship between schema and class label over a period of time.

Coefficient pruning The process of reducing the size of a spectrum's coefficient array.

Schema pruning The process of eliminating features that do not contribute to the classification process from a given schema.

Spectrum refinement The process of incremental Fourier coefficients update.

Cluster pool A collection of distinct schema instances which are categorised by their class label.