

Diversity Measures as New Concept Drift Detection Methods in Data Stream Mining

Osamah Abdulsattar Mahdi

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Engineering and Mathematical Sciences Department
of Computer Science and Information Technology College of
Science, Health and Engineering

La Trobe University
Melbourne, Australia

9 September 2020

Dedication

This thesis is dedicated to my dearest, precious and first teachers,

my father and mother

I will always be grateful for your endless love, unlimited support and deep faith in me

&

my brother and sisters, Nawfal, Aseel and Zina, who are like candles that burn to provide others with light,

&

I would like to dedicate this thesis to my wife Rand, for all her love and support.

Thanks to Allah for sending these angels into my world.

Statement of Authorship

Except where reference is made in the text of the thesis, this thesis contains no material published elsewhere or extracted in whole or in part from a thesis accepted for the award of any other degree or diploma. No other person's work has been used without due acknowledgment in the main text of the thesis. This thesis has not been submitted for the award of any degree or diploma in any other tertiary institution.

Osamah A. Mahdi

09/09/2020

Acknowledgements

I would like to express my deepest gratitude and appreciation for the supervision, patience, and the kindness of Dr Eric Pardede over the past four years, which made this work possible. I extend my sincere thanks to Dr Jinli Cao for her more than generous guidance and support. It has been an honour to work with both of you.

Publications

We list our research accomplishments and deliverables, each of which made this thesis possible, as follows:

- **Journal Paper:** Mahdi, Osama A., Eric Pardede, Nawfal Ali, and Jinli Cao. “Diversity measure as a new drift detection method in data streaming.” *Knowledge-Based Systems* 191 (2020): 105227.
- **Journal Paper:** Mahdi, Osama A., Eric Pardede, Nawfal Ali, and Jinli Cao. “Fast Reaction to Sudden Concept Drift in the Absence of Class Labels.” *Applied Sciences* 10, no. 2 (2020): 606.
- **Journal Paper:** “*A Hybrid Block-Based Ensemble Framework for the Multi-Class Problem to React to Different Types of Drifts*”, Paper submitted to *Journal Cluster Computing* on July 30 2020.
- **Conference Proceeding:** Mahdi, Osama A., Eric Pardede, and Jinli Cao. “Combination of information entropy and ensemble classification for detecting concept drift in data stream.” In the *Proceedings of the Australasian Computer Science Week Multiconference*, pp. 1-5. 2018.
- **Conference Proceeding:** “KAPPA as Drift Detector in Data Stream Mining”, Paper under preparation.

Abstract

Continuous change / evolve is a vital issue in evolving environments and applications, which include aviation, self-driving cars, nuclear reactors, medicine, the military, smart cities, and aerospace. That is, the essential features of these kinds of environments could possibly change (evolve), resulting in harmful outcomes, e.g., placing people's lives at risk if no response is followed. Thus, learning methods have to use intelligent algorithms to keep track of the evolution in environments and update themselves effectively. Additionally, we might experience fluctuations in the functionality of learning algorithms as a result of the dynamics of incoming information as it constantly evolves. That is, today's efficient learning approach can be deprecated after a change in the environment or data. Hence, the question as to how to develop an effective learning algorithm as time passes in the presence of changing (evolving) data needs to be tackled.

In this thesis, we make three contributions to address the aforementioned issues. According to the existing data stream learning literature, the event of (class distribution) change / evolve in data streams is known as concept drift. The appearance of concept drift in data streams may shift decision boundaries and also result in a drop in accuracy. Learning algorithms are therefore required to identify concept drift in evolving data streams and update / replace their predictive models appropriately. In order to tackle this task, adaptive learners are devised which utilize drift detection techniques to track down the drift points in evolving environments. In this thesis, we introduce five algorithms in three chapters (Chapter 5 - Chapter 7), namely the diversity measure as a new drift detection method in data streaming (DMDDM), fast reaction to sudden concept drift in the absence of class labels (DMDDM-S), a hybrid block-based ensemble (HBBE) framework for the multi-class problem to react to different types of drifts, combination of information entropy and ensemble classification for detecting concept drift in data streams and KAPPA as a drift detector in data streams, to effectively handle/detect concept drift in a rapid and also

minimal computational resource environment. First, in a novel way, DMDDM is a drift detector which combines a diversity measure called the disagreement measure with the PH test and an algorithm is developed to detect drift. Rather than checking the error estimates, DMDDM monitors the diversity of a pair of classifiers using the fading factor. In this way, the PH test is triggered whenever the predictions of components (h_u and h_v) start to disagree in an unusual way.

Second, DMDDM-S extends the DMDDM algorithm to detect concept drift in a semi-supervised environment. The primary benefit of calculating diversity is that for binary classification, the genuine label of an example is not essential to determine whether components disagree. Thus, we implement the suggested drift detector to identify sudden drifts when the class labels of incoming data are not available. To the best of our knowledge, this is the first work to utilize such a technique to detect concept drift. We adopt k prototype clustering as a strategy to label the unlabeled data and utilize the newly labelled data along with the labelled data to retrain the model in line with the current concept. Third, the HBDE brings together the best of the online drift detectors and block-based weighting with a view to enhancing the reaction to sudden drifts and to respond to other types of concept drift. For online drift detectors, we propose a new way of calculating the diversity which has been designed for a K -class problem.

Lastly, information entropy and KAPPA are considered to be effective ways of measuring uncertainty and the level of agreement, respectively, and they may be suitable to detect concept drift in a reliable, fast, and computationally efficient way. Hence, the entropy-based ensemble (EBE) does not build a new classifier for every new block of data, but instead, builds a new classifier only when there is a drift. The new classifier will be trained on more recent instances and added to the ensemble. Therefore, this mechanism will result in a low computational cost. On the other hand, contrary to the disagreement measure that has been proposed, we define this problem by posing the following question: is measuring the level of agreement using KAPPA when different classifiers access data items suitable for detecting concept drift? Therefore, the task is to signal concept drift in less time and with less memory consumption, keeping the accuracy of the data stream models constant.

Contents

| | |
|--------------------------------------------------------------------------|-----------|
| Dedication | 2 |
| Statement of Authorship | 3 |
| Acknowledgements | 4 |
| Publications | 5 |
| Abstract | 6 |
| List of Figures | 13 |
| List of Tables | 16 |
| 1 Introduction | 1 |
| 1.1 Introduction | 1 |
| 1.2 Main Contributions | 3 |
| 1.3 Thesis Organization | 4 |
| 2 Research Problems and Motivations | 7 |
| 2.1 Research Problems | 7 |
| 2.1.1 Problem I: Concept Drift Detection for Adaptive Learning | 9 |

| | | |
|----------|---------------------------------------------------------------------------------|-----------|
| 2.1.2 | Problem II: Ensemble for Adaptive Learning | 10 |
| 2.1.3 | Problem III: Adapting Related Knowledge for Detecting Concept Drift. | 11 |
| 2.2 | Motivations | 13 |
| 3 | Research Methodology | 15 |
| 3.1 | Introduction | 15 |
| 3.2 | Methodology of the Research | 15 |
| 3.2.1 | Literature Review | 15 |
| 3.2.2 | Data Collection and /or Generation Task | 16 |
| 3.2.3 | Design and Development | 17 |
| 3.2.4 | Implementation | 17 |
| 3.2.5 | Adjustment of Parameters, Experiments, and Discussion | 18 |
| 3.2.6 | Analysis | 18 |
| 3.3 | Research Scope | 18 |
| 4 | Data Stream Classification: | |
| | Background and Related Works | 19 |
| 4.1 | Background: Machine Learning | 20 |
| 4.1.1 | Batch Setting | 20 |
| 4.1.2 | Stream Setting | 20 |
| 4.1.3 | Learning Modes | 21 |
| 4.2 | Online Classification | 23 |
| 4.2.1 | Data Stream Classification | 23 |

| | | |
|-------|---------------------------------------------------------------|----|
| 4.2.2 | Assumptions | 23 |
| 4.2.3 | Requirements | 24 |
| 4.2.4 | The Cycle of Online Classification | 25 |
| 4.2.5 | Online Classification Algorithms | 27 |
| 4.3 | Adaptive Classification | 33 |
| 4.3.1 | Concept Drift Phenomenon | 33 |
| 4.3.2 | Concept Drift Adaptation Approaches | 38 |
| 4.4 | Related Work: Concept Drift Detection Techniques | 39 |
| 4.4.1 | Fully Supervised Concept Drift Detection Techniques | 39 |
| 4.4.2 | Semi-Supervised Learning Methods | 48 |
| 4.5 | Experimental Evaluation | 49 |
| 4.5.1 | Evaluation Procedures for Data Streams | 50 |
| 4.5.2 | Data Stream Datasets | 52 |
| 4.6 | Experimental Setup and Evaluation | 55 |
| 4.7 | Applications | 57 |
| 4.7.1 | Observing and Management | 57 |
| 4.7.2 | Personal Assistance and Information Management | 59 |
| 4.7.3 | Decision Making | 60 |
| 4.7.4 | Artificial Intelligence (AI) | 61 |
| 4.8 | Summary | 61 |

| | | |
|----------|------------------------------------------------------------------------------------------------------------------|-----------|
| 5 | Diversity Measure as a New Drift Detection Method in Data Streaming for the Binary Classification Problem | 64 |
|----------|------------------------------------------------------------------------------------------------------------------|-----------|

| | | |
|----------|---------------------------------------------------------------------------------------------------------|------------|
| 5.1 | Problem Statement | 64 |
| 5.2 | Diversity Measure as a New Drift Detection Method (DMDDM) | 66 |
| 5.2.1 | Experimental Evaluation | 72 |
| 5.3 | Fast Reaction to Sudden Concept Drift in the Absence of Class Labels (DMDDM-S) | 81 |
| 5.3.1 | Experimental Evaluation | 84 |
| 5.4 | Summary | 87 |
| 6 | A Hybrid Block-Based Ensemble Framework | 90 |
| 6.1 | Problem Statement | 90 |
| 6.2 | Framework for the Multi-Class Classification Problem to react to different types of drifts | 92 |
| 6.2.1 | Online Drift Detector for the K-Class Problem (ODDK) | 92 |
| 6.2.2 | Hybrid Block-Based Ensemble (HBBE) | 95 |
| 6.2.3 | Experimental Evaluation | 98 |
| 6.3 | Summary | 103 |
| 7 | Adapting Related Knowledge for Detecting Concept Drift | 105 |
| 7.1 | Problem Statement | 105 |
| 7.2 | Entropy in Data Streams | 107 |
| 7.2.1 | Experimental Evaluation | 108 |
| 7.3 | Inter-rater Agreement, k , in Data Streams | 110 |
| 7.3.1 | Experimental Evaluation | 114 |
| 7.4 | Summary | 115 |

| | | |
|----------|------------------------------------------------------------|------------|
| 8 | Conclusion and Future Work | 117 |
| 8.1 | Conclusions | 117 |
| 8.2 | Future Work | 120 |
| A | Appendix: Pseudocodes of Online Learning Algorithms | 122 |
| A.1 | Naive Bayes | 123 |
| A.2 | Decision Stump | 124 |
| A.3 | Hoeffding Tree | 125 |
| A.4 | Perceptron | 126 |
| A.5 | K-Nearest Neighbours | 126 |
| B | Appendix: Samples of Generated Datasets | 127 |
| B.1 | SEA Generator (SEA) | 128 |
| B.2 | AGRAWAL Generator (AGR) | 129 |
| B.3 | Mixed | 130 |
| B.4 | Sine1 | 131 |
| B.5 | Sine2 | 132 |
| B.6 | Wave | 133 |
| B.7 | RBF GR | 134 |
| B.8 | Tree R | 135 |
| B.9 | Electricity (Elec) | 136 |
| B.10 | Airline | 137 |
| | References | 138 |

List of Figures

| | | |
|-----|--------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1 | Thesis Structure and Relationship Between Chapters. | 6 |
| 2.1 | A General Framework of Concept Drift Handling. | 8 |
| 3.1 | Thesis structure and relationship between chapters. | 16 |
| 4.1 | The Taxonomy of Learning Modes. | 22 |
| 4.2 | The Cycle of Online Classification. | 26 |
| 4.3 | Decision Stump Example. | 28 |
| 4.4 | Perceptron Structure. | 31 |
| 4.5 | Two types of drift: where instances are represented by circles and different classes are represented by different colors. | 36 |
| 4.6 | Concept Drift Patterns | 37 |
| 4.7 | Sliding window· The borders identify two different windows | 42 |
| 4.8 | Block workflow | 44 |
| 4.9 | Illustration of the Evaluation | 56 |
| 5.1 | Framework of DMDDM | 71 |
| 5.2 | Delay Detection Rate | 78 |
| 5.3 | False Alarm Rate | 79 |

| | | |
|-----|---------------------------------------------------------------------------------------------------------------------|-----|
| 5.4 | Accuracy Rate | 79 |
| 5.5 | Effect of Noise Rate | 79 |
| 5.6 | Effect of Noise Rate | 80 |
| 5.7 | Framework of DMDDM-S. | 84 |
| 5.8 | Changing DMDDM-S threshold (a,c,e) and the stability of each detector's accuracy (b,d,f). | 88 |
| 6.1 | The Framework of Hybrid Block-Based Ensemble | 97 |
| 6.2 | Memory Consumption | 99 |
| 6.3 | Average Accuracy and Delay Detection (Wave dataset) | 100 |
| 6.4 | Average Accuracy and Delay Detection (SEA Dataset) | 101 |
| 7.1 | Changing DMDDM-S threshold (a,c,e) and the stability of each detector's accuracy (b,d,f). | 115 |
| A.1 | Pseudocode of Naive Bayes | 123 |
| A.2 | Pseudocode of Decision Stump | 124 |
| A.3 | Pseudocode of Hoeffding Tree | 125 |
| A.4 | Pseudocode of Perceptron | 126 |
| A.5 | Pseudocode of K-Nearest Neighbours | 126 |
| B.1 | Sample of SEA Generator | 128 |
| B.2 | Sample of AGRAWAL Generator | 129 |
| B.3 | Sample of Mixed | 130 |
| B.4 | Sample of Sine1 | 131 |
| B.5 | Sample of Sine2 | 132 |

| | | |
|------|----------------------------------------|-----|
| B.6 | Sample of Wave | 133 |
| B.7 | Sample of RBF GR | 134 |
| B.8 | Sample of Tree R | 135 |
| B.9 | Sample of Electricity (Elec) | 136 |
| B.10 | Sample of Airlines | 137 |

List of Tables

| | | |
|------|------------------------------------------------------------------------------------------------------------------------|----|
| 4.1 | Batch data versus streaming data | 21 |
| 4.2 | Sample from the airlines dataset/ where Airline=A, Flight=F, Day of week= DOW, Time=T, Length=L, Delayed=D. | 36 |
| 4.3 | Concept Drift Terminology | 38 |
| 4.4 | Characteristics of Each Dataset. | 54 |
| 5.1 | The Correlation of a Pair of Classifiers (2×2) | 67 |
| 5.2 | Abbreviations of the Names of the Measures. | 72 |
| 5.3 | Results of Mixed dataset with 10% noise | 74 |
| 5.4 | Results of Sine1 dataset with 10% noise | 75 |
| 5.5 | Results of AGR dataset with 10% noise | 76 |
| 5.6 | Results of SEA dataset with 10% noise | 77 |
| 5.7 | The results of the Electricity dataset | 78 |
| 5.8 | Results of Sine1 dataset with 10% noise. | 85 |
| 5.9 | Results of Sine2 dataset with 10% noise | 86 |
| 5.10 | Results of Mixed dataset with 10% noise. | 87 |
| 6.1 | Output of a Pair of Classifiers (2×2) for the Binary Classification Problem. | 93 |

| | | |
|-----|-----------------------------------------------------------------------------------------|-----|
| 6.2 | Output of a Pair of Classifiers for the Multi-class Classification Problem . . | 94 |
| 6.3 | Results of Wave Dataset | 100 |
| 6.4 | Results of SEA Dataset | 102 |
| 6.5 | Classification accuracy of the different algorithms | 103 |
| 6.6 | Runtime of the different algorithms. | 103 |
| 6.7 | Memory usage of the different algorithms. | 104 |
| 7.1 | Comparison of EBE and prequential evaluation with and without concept drift. | 110 |
| 7.2 | The Correlation of a Pair of Classifiers (2×2) | 111 |

Chapter 1

Introduction

1.1 Introduction

As a result of the increasing number of uses of computer systems, huge volumes of digital data associated with almost all aspects of living are collected either for storage or processing reasons. From traffic management to stock indexes, from microblog articles to grocery store checkouts, contemporary societies record massive datasets which might contain hidden knowledge. However, as a result of the amount of the gathered information, this knowledge can't be extracted manually. This is why data mining techniques are recommended to routinely discover interesting, non-trivial patterns from huge datasets [1–4]. Typical data mining tasks are clustering, classification, and association mining, almost all of which have been mastered for more than two decades.

A data stream may be considered an unbounded sequence of instances (e.g., phone call records, site trips, sensor readings) that arrive continuously with time varying intensity. Because of the speed and size of data streams, it is frequently impossible to store instances or even process them a few times [5–8]. Examples of application domains where such data needs to be prepared in streams include: network monitoring [9], banking [10], traffic management [10], sensor networks [11, 12], disaster management [13], ecology [14], sentiment analysis [15], object tracking [5], and robot vision [16]. The existence of streaming information in this new category of uses has opened a fascinating line of investigation issues, which includes novel techniques for data mining, known as data stream mining.

There are three major obstacles to learning from data streams [7, 8, 17, 18]:

variability, *size*, and *speed*. The size and speed of data force algorithms to process them despite a restricted amount of memory and time, while examining each incoming instance only once [19–21]. Variability, on another hand, includes learning in environments that are dynamic with changing patterns. The most commonly studied explanation of variability in data streams is *concept drift*, i.e., changes in definitions and distributions of learned concepts over time [1]. Such unforeseen changes are mirrored in new learning instances which diminishes the accuracy of the algorithms trained from previous instances. For instance, take the example of examining a stream of microblog content regarding a movie in production. Upon changing the actor accountable for the key role, the stream of views regarding the film can rapidly be unfavorable. This problem is usually viewed as being concept drift in the sentiment of numerous groups of people. An algorithm trained on all the available content will recommend an overly optimistic regular opinion about the film [22, 23]. Thus, the data mining strategies which cope with concept drift are designed to execute forgetting, adaptation, or drift detection mechanisms to be able to adjust to changing environments. Additionally, based on the rate of these changes, concept drifts are generally split into sudden, gradual, incremental, and recurring, almost all of which need different responses [24, 25].

Generally speaking, supervised learning, unsupervised learning, and reinforcement learning are the three common machine learning problems [26, 27]. Supervised learning algorithms produce a predictive model working with labelled data which is widely known as training data [28]. Next, the predictive model is used for labelling new instances. Classification and regression are good examples of supervised learning where output labels are discrete and continuous, respectively [29, 30]. Unsupervised learning finds hidden patterns in unlabelled data. Clustering is a type of unsupervised learning that discloses hidden patterns in data by grouping equivalent data points into distinct clusters [27]. The discovered clusters might later be applied to identify labels for any classification task [29]. Reinforcement learning produces smart agents which are able to get in touch with their environments and consequently capture optimal actions. An agent learns to select the optimal action by obtaining feedback, i.e., penalty or reward, from its surroundings [31]. Although classification has been studied for many years in the fields of statistics, pattern recognition, machine learning, and data mining [27, 32–35], streaming apps require completely new, dedicated, learning strategies. This is due to the aforementioned speed, size, and variability of data streams, with variability needing special steps in the context

of classification. To handle these difficulties, classifiers for evolving data streams can make use of sliding windows, sampling techniques, drift detection strategies, and adaptive ensembles [1]. In this thesis, our focus is on the classification task, especially in relation to data streams in which instances continuously turn up over time. More information on the extent of the effort is provided in Section 3.1.

Detection techniques and adaptive ensembles are typical methods for dealing with concept drift in data streams and they also improve prediction accuracy. Ensemble algorithms are sets of individual classifiers whose predictions are aggregated to generate an ultimate decision [7, 36], while detection algorithms are individual classifiers in which its accuracy predictions are maintained for examination in case there is a drift. The characteristics, overall performance, and variations involving methods of ensemble detection are the primary subject of this thesis.

1.2 Main Contributions

The main contributions of the thesis in advancing the state-of-the-art in the classification of data stream mining are as follows:

1. The thesis advances novel methods for adaptive learning in concept drift detection. As a result, two algorithms, named *Diversity Measure As a New Drift Detection Method (DMDDM)* and *Fast Reaction to Sudden Concept Drift in the Absence of Class Labels (DMDDM-S)* were developed and experimentally validated. The proposed algorithms achieve higher than average predictive performance under sudden drift, the binary class problem, and fully and semi-supervised classification compared to competitive adaptive learning algorithms.
2. Based on the analysis of block-based ensembles and drift detections strategies, the thesis contributes to the understanding of adaptive block-based ensembles and online drift detection in general and the relations between their concept drift reaction mechanisms in particular. As a result, *A Hybrid Block-Based Ensemble Framework for the Multi-Class Problem to React to Different Types of Drifts (HBBE)* is developed and experimentally validated. The proposed algorithm achieves higher than average predictive performance under sudden, gradual, recurring, and multi-class problems

compared to competitive adaptive learning algorithms.

3. This thesis contributes to the understanding of adapting related knowledge through using information entropy and KAPPA in concept drift detection. As a result, *Combination of Information Entropy and Ensemble Classification for Detecting Concept Drift in Data Streams* and *KAPPA as a Drift Detector in Data Streams* were developed and experimentally validated.

Additionally, this thesis conducted large-scale comparisons of both concept drift detection techniques and ensembles for mining data streams impacted by concept drift. Particularly, 19 different configurations of concept drift detectors and ensemble approaches are compared in regard to their ultimate accuracy and the accuracy of their concept drift detection. Furthermore, each of these comparisons was conducted using a relatively large number of synthetic datasets, with different types of concept drift versions of numerous sizes, using two base classifiers, and each was run in the MOA framework.

1.3 Thesis Organization

The layout of this thesis (the chapters and also the corresponding research contributions) and the connections between the chapters are shown in Fig 1.1. The primary content of each chapter is summarised as follows:

- **Chapter 2** The primary problems which are tackled in relation to this thesis are presented in this chapter in addition to the general framework of concept drift. This chapter also details the background to this research area and the research objectives.
- **Chapter 3** This chapter discusses the seven steps in the research design and methodology and highlights the methodical research approach followed in carrying out this study. This includes the design and development of each approach proposed in this thesis, in addition to a brief discussion of the solution.
- **Chapter 4** This chapter 4 introduces the basic definitions and terminology. We define the notion of classification, data streams, online processing, concept drift and block-based ensemble. Moreover, we discuss the related work in the field of drift reaction strategies and data stream classification, in particular drift detection and ensemble classifiers for concept-drifting data streams.

- **Chapter 5** This chapter focuses on the drift detection processing of data streams and discusses the limitations of existing drift detection algorithms. We propose a new drift detection technique for fully supervised classification, called the diversity measure as a new drift detection method in data streaming (DMDDM), which aims at reacting to sudden drifts in binary classification. In addition, we propose a new drift detection for semi-supervised classification, called fast reaction to sudden concept drift in the absence of class labels (DMDDM-S), which aims at reacting to sudden drifts in binary classification. The proposed algorithms are experimentally compared with state-of-the-art streaming methods in different drift scenarios.
- **Chapter 6** This chapter focuses on the block-based processing of data streams and discusses the limitations of existing ensemble classification algorithms. We propose a new data stream classifier, called a hybrid block-based ensemble framework for the multi-class problem, which aims at reacting equally well to several types of drift. The proposed algorithm is experimentally compared with state-of-the-art streaming methods in different drift scenarios.
- **Chapter 7** This chapter focuses on understanding and adapting the related knowledge using information entropy and KAPPA in concept drift detection. As a result, first we propose an ensemble titled, combination of information entropy and ensemble classification for detecting concept drift in data streams (EBE), which aims at reacting to sudden and gradual drifts in labelled data. Second, by measuring the inter-rater agreement between the successful predictions and the statistical distribution of the data classes, a simple and efficient drift detector called KAPPA as a drift detector in data streams is developed and experimentally validated.
- **Chapter 8** This chapter summarizes the contributions of this thesis and concludes with a discussion on lines of future research in the field of data stream classification.

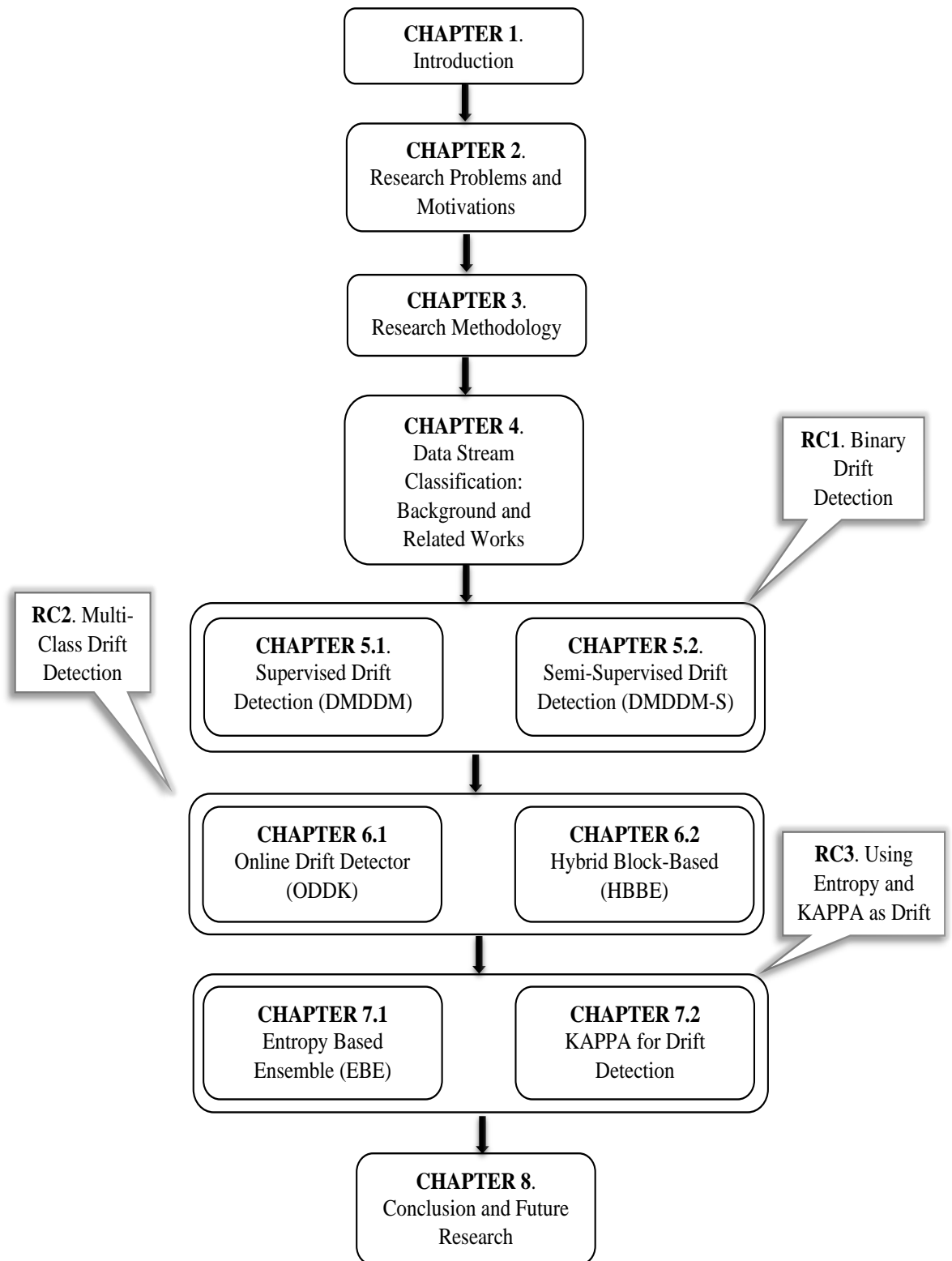


Figure 1.1: Thesis Structure and Relationship Between Chapters.

Chapter 2

Research Problems and Motivations

2.1 Research Problems

Conventional machine learning has two primary components: training/learning and prediction. Research on machine learning under concept drift comprises three innovative components: concept drift detection (whether or not drift happens), drift understanding (when, in which, and how) and drift adaptation (reaction to the presence of drift). The most commonly used techniques for learning from streaming data in the presence of concept drift are illustrated in Fig 2.1.

Numerous scientific studies in the area of concept drift have been conducted over the last ten years. Recent research targets more difficult problems, i.e., *how to accurately detect concept drift* [37–39]; *how to efficiently understand concept drift in ways that may be detected* [40, 41], and also *how to efficiently respond to concept drift by adapting related knowledge* [8, 42], thereby endowing prediction and decision making with the essential adaptability in a concept drift environment. Moreover, stream learning poses more challenges due to time and storage limitations. The trade-off between computational costs and learning accuracy is also an essential requirement which must be dealt with for real-world applications [43–45]. Since learning accuracy is not the sole evaluation metric to determine the efficiency of learning models, *how to efficiently tackle concept drift in a rapid and low computational resource environment* is vital.

These innovative results greatly enhance research in artificial intelligence and data science overall, and in pattern recognition and data stream mining particularly. Additionally, in a current technical report from Berkeley [46], acting in continual learning

and dynamic environments continues to be viewed as one of nine research opportunities which will help address the current AI research challenges.

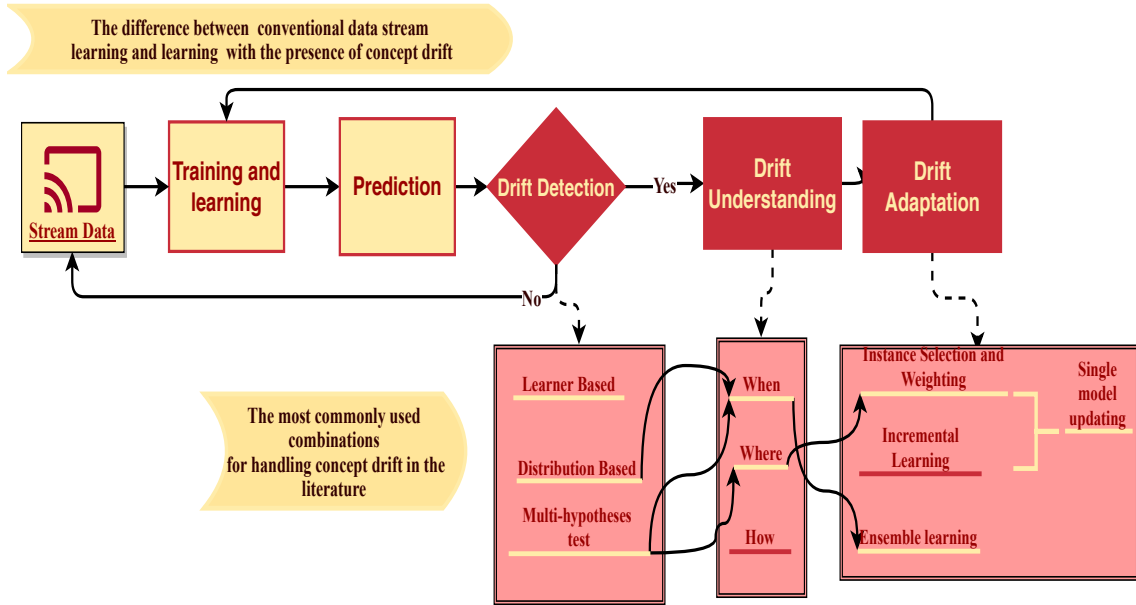


Figure 2.1: A General Framework of Concept Drift Handling.

Hence, this thesis focuses on handling the main research problems related to detection techniques and adaptive ensembles for evolving data streams. Firstly, to boost the quality of concept drift detection, we present novel drift detection methods for binary classification (fully supervised and semi-supervised) which reacts quickly and accurately to concept drift, consuming little time and memory compared to various other drift detectors. Second, to properly understand the different ways to detect concept drift, we present an adaptive ensemble learning algorithm for the multi- class problem which is able to respond to various kinds of concept drift. Thirdly, to successfully respond to drift by adapting related knowledge, we apply information entropy and KAPPA which is a highly effective method of computing uncertainty in data streams and is also able to discover concept drift in a reliable, fast and computationally effective way. The solutions to these three research problems will enable concept drift to be handled effectively in a rapid manner, using minimal computational resources.

We -discuss each research problem in the following subsections.

2.1.1 Problem I: Concept Drift Detection for Adaptive Learning

Learning from evolving data streams is difficult due to distributional changes, i.e., the concept drift event. Learning algorithms have to adjust to the latest distributions to try to keep the classification error rate low. Drift detection methods, as the key part of adaptive learning algorithms, are accountable for detecting concept drift with the least delay the minute they appear in data streams. According to [8], a predictive model in a non-stationary environment should be able to achieve the following four requirements: (i) diagnose concept drift in a short time, (ii) differentiate interference from drift and be adaptive to change but robust to noise, (iii) operate in less time than new data appears and (iv) make use of no more than a constant amount of memory. An excellent data stream model/classifier should have these four abilities.

An adaptive learning algorithm must obtain better accuracy compared to a non-adaptive algorithm to be seen as good for a learning undertaking from an evolving data stream. We provide the research problem and its objective as follows:

Research Problem 1.1: *Will the classification accuracy of data stream models reduce due to the concept drift phenomenon in evolving data ?* Therefore, the task is to signal for concept drift in less time and with less memory consumption, keeping the accuracy of the data stream models constant.

Research Objective 1.1: To propose a novel concept drift detection method which detects concept drift more accurately and with less time and memory consumption compared to the other drift detectors. To do this, we empirically compare our proposed drift detection method with the existing ones using synthetic and real-world data streams considering different performance measures, e.g., detection delay, true detection, memory and accuracy.

According to the authors of [47], most of the existing drift detection methods are based on fully supervised learning and assume that the entire incoming data stream is completely labelled and these labels can be used instantly. These methods have proven their effectiveness in their application/domain. However, in real-world scenarios, labelled data are not always available or are costly to obtain and time consuming. Therefore, in an environment where incoming data streams appear at high speed, it is not always possible to manually label all the data as soon as they arrive. Semi-supervised learning could solve this

problem by using labelled and unlabelled data together for the learning process. Detecting concept drift in a semi-supervised environment has received little attention from the research community [47]; thus, the main contributions of this part of the thesis can be summarized as follows:

Research Problem 1.2: *Can a lack of class labels aggravate the problem of concept drift detection in data streams?* Therefore, in the absence of class labels, the task is to signal concept drifts in less time and with less memory consumption, keeping the accuracy of the data stream models constant.

Research Objective 1.2: To propose a semi-supervised drift detection method which detects concept drift in the absence of class labels more accurately, compared to other fully supervised drift detectors.

We introduce our diversity measure as a new drift detection method in data streaming (DMDDM), and fast reaction to sudden concept drift in the absence of class labels (DMDDM-S) to address the aforesaid research problems in Chapters 5.

2.1.2 Problem II: Ensemble for Adaptive Learning

The problem with changing the definitions of classes over time is that it decreases the performance (accuracy) of a predictive model which has been trained using old instances [7, 48]. Also, processing the multi-class problem is computationally more expensive, particularly in the presence of concept drift in data streams where the data is changing over time, and this aggravates the problem of a loss of performance during the process of drift detection in data streams [49]. A solution for these problems is a mechanism which uses continuous diagnostics of concept drift. Then, upon detection of concept drift, a process of updating the model to maintain the classification performance is required.

Research Problem 2.1: *How can DMDDM detect concept drift in multi-class classification?* The task is to signal concept drifts in less time and with less memory consumption, keeping the accuracy of the data stream models constant.

Research Objective 2.1: To devise a new formalism that facilitates a way to detect concept drifts in the multi-class problem with more accuracy and less time memory consumption compared to other drift detectors. To do this, we empirically compare our

proposed drift detection method with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., detection delay, true detection, memory, and accuracy.

Research Problem 2.2: *Can we use the block-based ensemble approach to enhance the reaction to sudden drifts and respond to different types of concept drift?* Therefore, the task is to keep the accuracy of the data stream models constant.

Research Objective 2.2: To devise a hybrid block-based ensemble which is a framework for multi-class classification in evolving data streams. To do this, we empirically compare our proposed framework with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., memory, time and accuracy.

We introduce the proposed hybrid block-based ensemble framework for the multi-class problem to react to different types of drifts (HBBE) to address the aforesaid research problem in Chapter 6.

2.1.3 Problem III: Adapting Related Knowledge for Detecting Concept Drift.

In the ensemble environment, it is well known that the performance of each component classifier is evaluated by estimating its expected prediction error on the examples from the most recent data chunk. After substituting the poorest performing component, the remaining ensemble members are updated, i.e. incrementally trained, and their weights are adjusted according to their accuracy. Dynamic weighted majority (DWM) [50] is a method that is based on an ensemble classifier. In DWM, a set of incremental classifiers are weighted according to their accuracy after each incoming example. This method can add and remove classifiers according to the algorithm's global and local performance. For example, if the ensemble commits an error, classifiers are added. If one classifier commits an error, its weight is reduced. If after many examples, a classifier continues to achieve low accuracy, it is removed from the ensemble. Furthermore, after a period of predictions p the entire ensemble is evaluated and, if needed, a new classifier is added to the ensemble.

However, with its mechanism of creating a new classifier and training it for every new block of data, the approach has a high computational cost. In addition, one of the main

disadvantages of evolving the ensemble of classifiers is that it builds a new classifier for every batch of new data. This results in the high usage of memory.

- **Research Problem 3.1:** *Is measuring uncertainty in data streams instead of the classifier's error rate suitable for detecting concept drift in a reliable, fast, and computationally efficient way?* Therefore, the task is to keep the accuracy of the data stream models constant.
- **Research Objective 3.1:** To propose a model called entropy-based ensemble (EBE) which is based on incorporating entropy as a drift detector into the ensemble in a reliable, fast, and computationally efficient way. To do this, we empirically compare our proposed framework with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., memory, time and accuracy.

Additionally, since data class distributions may change through the progress of the stream, KAPPA provides better insight to detect any data class distribution changes. This is because KAPPA is a strict measure that quickly drops in the case of incorrect predictions, making it much more useful rather than using accuracy/ error-rate which only introduce small changes. Second, the main reason to address concept drift is due to changes/drifting in data class distribution as the stream progresses. Therefore, KAPPA is capable of capturing the competence of the components, reflecting the possibly varying data class distribution with time [51, 52]. In other words, different to the disagreement measure that is used in **Problem I**, measuring the level of agreement when different classifiers access data items is suitable to detect concept drifts.

- **Research Problem 3.2:** *Contrary to the disagreement measure that was used in Chapter 5, is measuring the level of agreement using KAPPA suitable to detect concept drift when different classifiers access data items?* The task is to signal concept drift in less time and with less memory consumption, keeping the accuracy of the data stream models constant.
- **Research Objective 3.2:** To propose a drift detector based on KAPPA calculations. To do this, we empirically compare our proposed drift detector with our proposed drift detector from **Problem I** using synthetic data streams, considering different performance measures, e.g., delay detection, true positives and the mean accuracy.

We introduce our combination of information entropy and ensemble classification for detecting concept drift in data streams (EBE) and KAPPA as the drift detector in data streams to address the aforesaid research problems in Chapters 7.

2.2 Motivations

After detailing the research problems and the analysis objectives, we now address our motivations for each research problem as follows:

- **The Need for Accurate Drift Detection Methods (Problem I)** - Learning adaption is essential in evolving uses, such as, but not limited to, medication, nuclear reactors, self-driving automobiles and smart cities, otherwise it will be unavoidable that a high price will be paid in regard to individual lives, horrific accidents and green threats. Hence, blindly retraining certain time factors isn't a suitable option because it doesn't ensure the accurate and immediate detection of concept drift, and subsequently, an essential reaction. Therefore, detection methods which are capable of responding promptly to concept drift are needed together with the minimal computational resource. The existing techniques show poor overall performance in our preliminary assessments and earlier research studies, and this observation motivated us to develop new drift detection techniques to detect concept drift successfully.
- **Ensemble approaches that process instances in blocks might not respond to abrupt changes sufficiently quickly (Problem II)** - To do this, we propose a hybrid block-based ensemble paradigm which is designed to put together the finest online drift detector and also block-based weighting in an effort to respond to various kinds of drift. Hence, we were motivated to propose a hybrid block-based ensemble (HBBE), which is a supervised multi-class classification framework for classifying evolving data streams. Moreover, the issue of changing the definitions of classes after a while reduces the performance (accuracy) associated with a predictive model which has been trained utilizing old instances. Likewise, processing the multi-class problem is computationally more costly, especially in the presence of concept drift in data streams in which the data is changing as time passes, which aggravates the issue of a loss in performance throughout the process of drift detection. This motivated us to suggest a drift detection method for the multi-class issue which satisfies the four

requirements.

- **The classifier's error rate and the ensemble are utilized in the majority of the prior works to deal with classification accuracy as a criterion to judge whether concept drift is occurring or not. (Problem III)** - Information entropy is an excellent method of computing anxiety and it is well suited to discover concept drift in a reliable, fast and computationally effective way. Therefore, this motivates us to propose the model known as the entropy-based ensemble (EBE) which is based on including the entropy as a drift detector in the evolving ensemble.

Chapter 3

Research Methodology

3.1 Introduction

We follow an incremental methodology in this thesis comprising seven tasks as depicted in Fig. 3.1. We begin with the literature review and the data collection and generation task. We then design our drift detection methods for adaptive learning and the ensemble combined with an online drift detection method. The next tasks were parameter tuning, conducting the experiments and presenting the discussion. The analysis task plays an influential role because we discuss the pros and cons of the existing methods which guides our developments. Finally, the process is not like a queue where tasks are completed one after the other, but rather, they are like a lattice where all the tasks are completed interactively.

3.2 Methodology of the Research

The methodology of this research work is outlined as follows:

3.2.1 Literature Review

In this stage, we review the background to this work and present a review of the literature related to this research. The review covers the following:

1. The fundamental concepts and background of machine learning and how this differs



Figure 3.1: Thesis structure and relationship between chapters.

from data stream mining are thoroughly reviewed. The literature review includes the different learning approaches and learning modes which are available for each setting. Furthermore, since the main focus of this thesis is classifying evolving data streams, we review the fundamental concepts regarding online classification with a focus on data stream classification which includes commonly used online classification algorithms such as naive Bayes, decision stump, Hoeffding trees, perceptron, and K-nearest neighbours.

2. Adaptive classification in terms of the concept drift challenge, the formal definition and patterns are thoroughly reviewed. This includes several approaches emphasizing handling the problem of concept drift in data streams by categorising drift detection methods as either statistical-based, window-based or ensemble-based.
3. The final step of the literature review is devising the research questions and objectives that will address the gap in the literature.

3.2.2 Data Collection and /or Generation Task

In this stage, we collect and generate synthetic and real-world data stream datasets that are widely used to evaluate the performance of the online and adaptive learners. For the synthetic dataset, we use the SEA Generator (SEA), AGRAWAL Generator (AGR), Mixed, Sine1, Wave, RBF_{GR} and $Tree_R$, while for the real-world datasets we use the Electricity

and Airlines datasets. We describe the characteristics of the synthetic and real-world data streams in Section 3.5.2.

3.2.3 Design and Development

After reviewing the literature and collecting/generating the required datasets, in this stage we start designing and developing approaches to handle concept drift based on the predefined research questions:

1. Based on the literature analysis of drift detections strategies, we design a novel drift detection method that reacts accurately and quickly to concept drift. As a solution for fully and semi-supervised environments, the disagreement measure as a diversity measure of a pair of classifiers is designed to detect concept drift in a binary classification problem and under sudden drift, as detailed in Chapter 5.
2. Based on the analysis of block-based ensembles and drift detections strategies, we design a hybrid framework of adaptive ensemble learning which brings together the best of the online drift detectors and block-based weighting with a view to enhancing the reaction to sudden drifts and responding to other types of concept drift, as detailed in Chapter 6.
3. We design an ensemble that adapts information entropy to detect concept drift. In addition, we also propose an approach using KAPPA as a drift detector, as detailed in Chapter 7.

To address the main challenges associated with processing a data stream, we ensure that in achieving the above aims, we devise effective methods to handle concept drift in a fast and low-computational resource environment which are better than the state-of-the-art.

3.2.4 Implementation

In this stage, the proposed drift detection methods and the ones used for comparison are implemented. MOA, a framework for data stream mining developed in Java, is used for implementation.

3.2.5 Adjustment of Parameters, Experiments, and Discussion

In this section, we discuss the adjustment of the parameters and the experimentation. The parameter adjustments and the experimental setup are discussed in Section 4.6.

3.2.6 Analysis

In this section, we analyse the existing methods in the literature and discuss their advantages and disadvantages.

3.3 Research Scope

- Thesis focus – The primary focus of this thesis is on classification and adaptation in evolving streaming data.
- Data characteristics – The data streams used in our experiment are essentially cross-sectional data. That is, we performed our experiments on data streams. Furthermore, the data streams are assumed to be fully labelled and partially labelled (50%). We suggest the challenges imposed by delayed labels, unlabelled instances, and time-series data are future work.
- Experiments – Our experiments are conducted on synthetic and real-world data streams which are frequently used in the literature [48, 53–57]. The data streams contain numerical or categorical attributes. Concept drift may appear abruptly or gradually in the synthetic data streams. We added noise to the synthetic data streams to confirm that our methods are robust against noisy data. Please note that drift detectors are beneficial if they distinguish concept drift from noise [8, 17, 58]. Finally, we process only one single stream per task.
- Evaluation – We consider drift detection delay, false positive and false negative rates, runtime, memory usage and classification accuracy to assess the drift detection methods for adaptive learning. As for ensemble learning, we study classification accuracy, memory usage and runtime.

Chapter 4

Data Stream Classification: Background and Related Works

Recently, a lot of research has focused on data streams and the problem of concept drift. Researchers have classified concept drift changes according to their frequency, speed, and severity, and many drift detection mechanisms have been proposed. Moreover, research on concept drift combined with effective stream processing methods has resulted in the improvement of numerous classification algorithms designed to deal with evolving data, for example: sliding window, online algorithms, drift detection strategies, and adaptive ensembles.

This chapter provides basic definitions and reviews the existing works related to the field of data stream classification. The subsequent sections are organized as follows. Section 4.1 introduces the background of machine learning in terms of batch settings and learning modes. The basic terminology concerning classification, data streams, and online processing is given in section 4.2. In Section 4.3, we formally define the problem of concept drift and discuss several concept drift adaptation approaches. In Section 4.4, we discuss state-of-the-art works in the field of drift reaction strategies and data stream classifiers. Section 4.5 details the experimental evaluation in terms of the classification measures, the drift detection measures, the resource consumption measures, and the datasets. Finally, the experimental setup and applications are presented in Sections 4.6 and 4.7, respectively.

4.1 Background: Machine Learning

In the field of artificial intelligence, machine learning involves research and the advancement of computational models to enhance their performance with the ability to acquire knowledge on their own [59]. Examples of activities that can be performed by machine learning techniques are prediction, classification and recognition to name a few. Machine learning may occur in either a *batch environment* or a *stream environment*. Conventional machine learning algorithms and strategies were developed depending on the batch environment in the late 1990s. When action in relation to online learning, which includes data stream mining, started as the outcome of rapidly growing data. Various learning techniques have been devised for every environment. We explain each setting in the next subsections.

4.1.1 Batch Setting

The batch setting assumes data need to be gathered and preprocessed which means data are stored, static, and reside within the memory. The actual size of data could be in the order of hundreds or perhaps much less. Therefore, to determine the most effective utilization of a dataset, it can be processed several times for the learning task. The user will often find there is no memory or runtime restrictions in relation to batch learning [60]. The concept of learning is stationary, i.e., the distribution of data doesn't change/transform. Ultimately, the learning process runs in an offline mode. In other words, the learning process is implemented after data collection. No process is conducted throughout the learning phase and also, once the learning phase is finished, the system cannot be enhanced or altered.

4.1.2 Stream Setting

Streaming data is a sequence of data samples which arrive very rapidly in large volumes. Every sample is handled only once and is then discarded. Learning algorithms might deal with this restriction by buffering samples for a short term for potential training or testing [8]. Accommodating all of the data in the primary memory isn't feasible, so the learning process must be undertaken quickly in a near real-time fashion [61, 62]. Furthermore, the

Table 4.1: Batch data versus streaming data

| Batch data | Stream data |
|-------------------------------------|------------------------------------|
| Offline | Real time |
| Slow data generation | Rapid data generation |
| Persistent data | Transient data |
| Process entire data | Process samples of data |
| Constant availability | Limited availability |
| Complex techniques used if required | Linear techniques widely used |
| Fixed size | Unbound in size |
| Random access | Sequential access |
| Known data characteristics | Unpredictable data characteristics |

concept of learning could change as time passes because of the evolving character of the environment. In this setting, learning occurs in an online or incremental way. We summarize the aforementioned settings in Table 4.1. Recall that the size of data is in the order of hundreds in the batch environment, whereas it is in the order of over a million in the data stream environment. Regarding the number of scans, data might be handled many times in the batch environment. On the other hand, data are usually handled in one round in the data stream environment because of the restricted memory bottleneck. Memory, time, and the concept of learning are the three essential dimensions of stream mining which a learning algorithm needs to address or else it might not be appropriate for the learning process. Lastly, the mode of learning is incremental for data stream mining.

In the following subsection, we compare in detail the offline and online learning modes.

4.1.3 Learning Modes

We detail the taxonomy of machine learning modes in Fig. 4.1. Machine learning has two common modes, namely batch and online learning, as follows:

- **Batch Learning / Offline Learning:** Batch learning algorithms are utilized for the batch setting, in which the data are finite, static, and already preprocessed for the data mining process [31]. A batch learning process comprises two phases, namely (1) learning, in which a classification model is constructed, and (2) classification, where the classification model is evaluated and utilized for labelling new instances. The learning stage is commonly known as the training stage in the literature. There is no

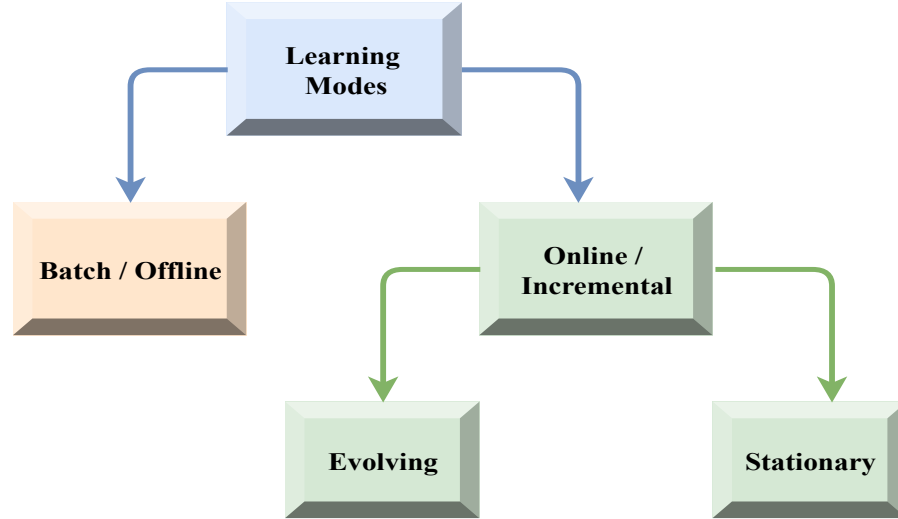


Figure 4.1: The Taxonomy of Learning Modes.

more learning, once a model has been trained [63].

- **Online/Incremental learning:** Online/incremental learning algorithms are applied in the data stream setting in which a large volume of data arrive quickly instance-by-instance. An incremental algorithm revises its classification model by processing instances one at a time. Instances may be buffered later. Because incremental learners should act in real-time as opposed to batch learning, the training and tests phases occur together continually [19, 48, 64]. Even though incremental learning and online learning are usually applied interchangeably in the literature, online learning is a type of incremental learning in which every instance is handled only once and is then thrown away [8]. Learning may take place in either stationary or evolving environments. In the former case, the classification models are kept up to date through completion, whereas in the latter case, they are updated by both completion and adaptation. Completion demonstrates the new instances, followed by the identical distribution of earlier ones into the model, while adaptation handles concept drift, generally, by retraining the model from zero. The reader must be aware that the online learning terminology is interchangeably employed for stationary learning in the literature when both concept drift and adaptation are missing.

Recall that the primary emphasis of this thesis is on the classification and adaptation of evolving data streams, as discussed with Section 1.3. We detail the basic concepts relating to online classification and adaption in Sections 4.2 and 4.3, respectively.

4.2 Online Classification

As previously discussed, we classify the learning tasks into two groups of batch/offline learning and incremental/online learning and then describe each in detail. We then compare two modes of learning, i.e., adaptive learning and stationary learning. Adaptive learning intrinsically shares essential concepts with (stationary) online learning, so we review the required notions for online classification in this section and describe adaptive learning in Section 4.3.

4.2.1 Data Stream Classification

Data stream classification is the process of constructing a model and utilizing the readily available data (i.e., the training data) to predict the labels of unseen examples. We provide the formal definition of data stream classification as follows:

Let stream S be a sequence of instances as $(x_1, y_1), (x_2, y_2), \dots, (x_t, y_t)$, arriving one after another over time. The pair (x_t, y_t) belongs to an instance at time t , in which x_t is a vector which has the values of k attributes as $x = (x_1, x_2, \dots, x_k)$, and y_t is a class label coming from a limited set of m class labels as $y_t \in \{c_1, c_2, \dots, c_m\}$. Suppose there is a target function $y_t = f(X_t)$ that maps an input vector to a class label. The learning process is usually to incrementally build a model \tilde{f} which approximates function f while instances are dealt with. An approximation which *maximizes* the classification accuracy is necessary.

For data stream classification, several assumptions are made about the dynamics of data streams. We discuss these assumptions in Section 4.2.2. Incremental learners must also comply with four fundamental requirements for all data stream classification tasks as discussed in Section 4.2.3. Finally, we detail the cycle of data stream classification in Section 4.2.4.

4.2.2 Assumptions

In general, there are six essential assumptions which are presently held for the task of data stream classification in the literature [61, 62]. These assumptions are as follows:

- A data stream possesses a fixed range of attributes. A large number of attributes might impede the learning process as well as increase memory utilization.
- The total number of instances or records is considerable compared to the number of attributes. In reality, learning algorithms claim to have the ability to deal with an infinite amount of data while not consuming memory resources.
- The number of class labels should be small. More class labels bring in additional statistics for inducing a classification model [61]. As a consequence, considering that the values of the statistics are continually updated over time, mining a data stream with a lot of classes is computationally costly, which grows linearly in computational complexity with the number of classes.
- The size of the data is usually bigger than the accessible memory. Thus, loading all of the data into memory is not doable
- The learning algorithms have to undergo both training and testing phases in close to real-time since instances of data streams arrive extremely quickly.
- The concept of learning is always assumed to be fixed or evolving. Concept drift occurs if the underlying distribution of data changes.

The first three assumptions focus on the dynamics of data streams, whereas the last three assumptions refer to what learning algorithms must look in relation to classification against data streams.

4.2.3 Requirements

The primary challenges for data stream classification are enforced through the restricted amount of computational resources and also the concept drift event. Classification algorithms have to meet the following four fundamental requirements to make the learning task from data streams feasible [61, 62, 65]:

- **Requirement One (R1): handle an instance at a time and examine it only once** – Recall which instances of a data stream arrive one after another and they are handled only once in the order of appearance. In other words, random access to the instances is

not doable. An instance is thrown away once it has been handled. While this is a vital necessity for data stream mining, a learning algorithm can internally store instances for a short time for additional usage without violating Requirement 2.

- **Requirement Two (R2): use a restricted amount of memory** – The reason for training classification models incrementally is because the size of the data is considerably bigger than the size of the available memory. That is, a large amount of data cannot be stored in a limited memory. Thus, an upper bound needs to be determined over the memory utilization to avoid potential memory exhaustion. A learning algorithm might use the primary memory to maintain the current model.
- **Requirement Three (R3): handle an instance in a restricted amount of time** – Learning algorithms must approach instances as quickly as they arrive. That is, the learners have to process instances more quickly than the speed of the arriving data. Any kind of breakdown unavoidably results in a loss in information. Hence, a top bound needs to be set in relation to how much time is allotted for processing an instance.
- **Requirement Four (R4): be prepared to predict at any point** – A recommended learning algorithm establishes a classification model which is in a position to predict the label of unseen instances in a short amount of time. Therefore, prediction at any point means a class label is accessible at any point of time, which is essential for data stream classification.

4.2.4 The Cycle of Online Classification

We present the data stream classification cycle in Fig. 4.2. It incorporates three stages, namely processing, learning, and utilizing [61], detailed as follows:

- **Processing:** This stage is fully compliant with Requirement 1 where instances of streaming data are prepared, after which, they are transferred to the learning stage.
- **Learning:** The learning algorithm updates its predictive model by training each new instance. It is additionally guaranteed that Requirements 2 and 3 are not violated by not exceeding the memory or runtime bounds.

- **Utilizing:** The model is utilized to predict the class labels of unseen instances. Requirement 4 can be met by ensuring the model is ready for prediction

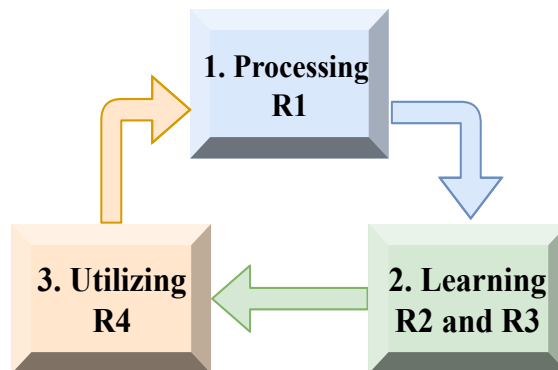


Figure 4.2: The Cycle of Online Classification.

4.2.5 Online Classification Algorithms

This section presents several of the widely used classifiers recommended for stationary data which fulfil standard stream mining requirements, i.e., they have the characteristics of an online learner as well as some kind of forgetting mechanism. Additionally, a few algorithms which are able to process data sequentially but don't adapt can be improved to respond to changes and are regularly applied as benchmarks in the data stream mining literature [48, 57, 66–68]. We address *Naive*, *Bayes*, *Decision*, *Stump*, *Hoeffding*, *Trees*, *Perceptron* and *K-Nearest Neighbours* algorithms. We explain these algorithms in the following subsections.

Naive Bayes

The Naive Bayes algorithm is based on Bayes' theorem and computes class-conditional probabilities for every new example [29]. Bayesian procedures are able to learn incrementally and also need constant memory. Nevertheless, Naive Bayes is a lossless classifier, which means it "produces a classifier functionally equivalent to the corresponding classifier trained on the batch data". To incorporate a forgetting mechanism, sliding windows are generally used to "unlearn" the earliest examples. An individual Naive Bayes model will usually not be as accurate as more complex models [29]. Nevertheless, Bayesian networks, which are usually much more advanced and offer much better outcomes, are usually suitable for the data stream setting; it is only necessary to dynamically learn their framework. Lastly, the Naive Bayes algorithm is sometimes a subcomponent of more complex strategies like decision trees for data streams. The pseudocode of the incremental Naive Bayes algorithm is presented in Algorithm A.1

Decision Stump

A decision stump models a single level decision tree [69], i.e., a decision stump is a decision tree together with the root node. A decision stump constitutes a prediction working with the values of an attribute which is given as the root. For instance, Fig. 4.3 presents a decision stump for predicting whether an individual plays a game outside according to the humidity level. Decision stumps are also called 1-rules algorithms [70].

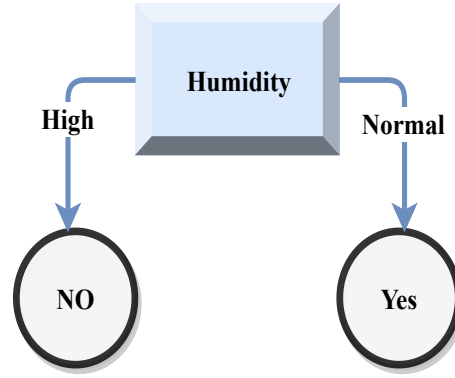


Figure 4.3: Decision Stump Example.

Taking into account the kind of the attribute given as the root, some cases are feasible: (1) for nominal attributes, one could create a stump that has a leaf for every likely value [71] or maybe a stump with two leaves, one which corresponds to a selected group, and the other which corresponds to each additional group. The missing value might also be handled as an additional group (2). For a continuous attribute, typically, a threshold is determined and also two leaves are held by the stump for values below and above the threshold. It is also possible to have multiple thresholds, and also as an outcome, the stump could contain 3 or more leaves. Fig. 4.3 presents a decision stump where the attribute ‘Humidity’ is selected as the stump.

Attribute selection procedures are used to discover an attribute as the stump. Attribute selection procedures are employed to rank the attributes depending on their informativeness to construct a decision tree in a top-down manner. That is, the most helpful attribute is selected as the root. Other attributes comprise the other levels of the tree [29]. In the same way, in a decision stump, the attribute with the highest ranking is selected as the stump.

In relation to an incremental learning scenario, when instances are handled, we have to upgrade the data necessary for the attribute choice methods. Assuming information gain as the attribute selection measure, complete entropy and the entropies of all attributes are kept up to date while instances are processed one-by-one. Lastly, the attribute together with the highest gain is considered as the stump, which is utilized to predict the label of a new example. We demonstrate this approach in Algorithm A.2.

Hoeffding Trees

The work in [19] introduced the Hoeffding tree for learning from extremely large, and possibly infinite, data streams. The Hoeffding Tree algorithm mines data streams incrementally without keeping instances in the main memory and builds potentially very complex decision tree models with an affordable computational cost.

The Hoeffding tree algorithm is founded on the observation made in [72], which states that a small subset of available instances, as a training set, may be sufficient to find the best attribute as a test node in a decision tree. That is, the Hoeffding tree considers the early instances for choosing the root test and once the root attribute is chosen, the succeeding instances are passed down and used for choosing the appropriate attributes as the test nodes in the next levels. The Hoeffding tree guarantees with a high probability that the attribute chosen as a node after processing n examples is the same as the one which would have been chosen after processing infinite examples. In this case, [19] applied the Hoeffding inequality [73] to find the best attribute as a test node.

Let function $G(X_i)$ be an attribute selection measure, e.g., information gain, which is used to select test attributes, i.e., the nodes of the tree. Assume G is to be maximized, and let X_a be the attribute with the highest observed G after seeing n examples, and X_b be the second-best attribute. Let $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b)$ be the difference between their observed values. Given a desired confidence level δ , the Hoeffding bound guarantees that X_a is the correct choice if $\Delta\bar{G} > \varepsilon$, where ε is calculated by:

$$\varepsilon = \sqrt{\frac{R^2}{2n} \ln \frac{1}{\delta}} \quad (4.1)$$

The range R is equal to 1 for probability, and equal to $\log(|C|)$ for information gain where $|C|$ is the number of classes. [19] proved that the Hoeffding tree algorithm generates trees that are asymptotically close to the ones produced by a batch learner. Note that when two or more attributes have very close G 's, many examples are potentially needed to decide between the attributes to find the best split. This is wasteful because it makes little difference to the quality of the tree. Hence, the Hoeffding tree determines that there is a tie between attributes, and they are split on the current best attribute if $\Delta\bar{G} < \varepsilon < \tau$, where τ is a user-specified threshold. The pseudocode of the Hoeffding Tree is available in Algorithm

A.3.

Perceptron

The classic perceptron algorithm is used for *binary* classification [74]. The algorithm compares a weighted sum of inputs with a threshold to determine the output. The output is 1 when the summation is greater than or equal to the threshold, and 0 otherwise. Fig. 4.4 presents the general structure of a Perceptron network, where an instance x with k attributes, i.e., $X = (x_1, x_2, x_3, \dots, x_k)$, is fed to the perceptron. Each attribute is associated with a weight. The weighted sum of inputs, i.e., $\sum_{n=1}^k W_i x_i$, is computed by the input function, and the result is sent to the activation function as input. The weighted sum may also be calculated in a vectorized way as $W^T X$ where W is a weight vector as $W = (w_1, w_2, w_3, \dots, w_k)$. Following [67], we use the Sigmoid function as the activation function in our perceptron algorithm. It is worth mentioning that perceptron can be used to model linearly separable functions [75]. The pseudocode of the perceptron is available in Algorithm A.4.

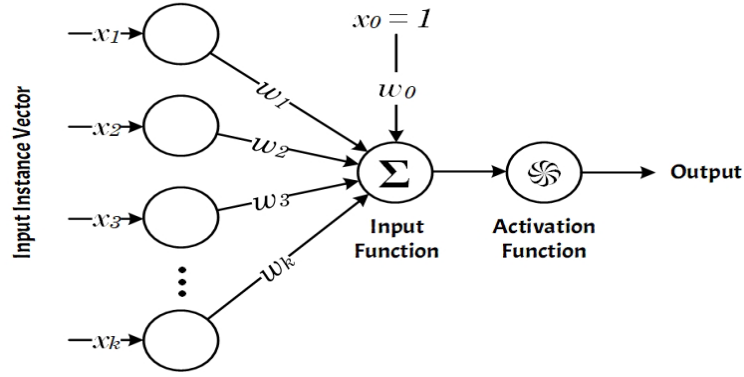


Figure 4.4: Perceptron Structure.

In perceptron, the weights should be randomly initialized first. In the training phase, the weights are updated to minimize the number of misclassified instances. That is, the weights are continuously updated to decrease the cost of misclassification until perceptron converges. In most studies, the mean squared error (MSE) is considered as a cost function and calculated by Equation (4.7), where n is the number of instances in the training set, y_j is the real class and $\tilde{f}_w(X_j)$ is the activation function that outputs the prediction.

$$J(W) = \frac{1}{2} \sum_j^n (y_i - f_w(X_j))^2 \quad (4.2)$$

To minimize the cost function $J(w)$, the gradient descent optimizer may be used to update the weights as instances and are processed until it converges to a minimum point. The update rule is defined as follows:

$$W = W - \eta \nabla j(W) \quad (4.3)$$

where w is the weight vector and $\Delta J(W)$ is the gradient of the cost function. The gradient is used to define the direction of the update. That is, if the gradient is positive, the weight should decrease to get closer to the minimum point. Otherwise, the weight should increase to get closer to the minimum point. Finally, η is the learning rate which controls the size of each update. The gradient of the cost function, i.e., $\Delta J(W)$, is measured by Equation (4.9).

$$\nabla j = - \sum_j^n (y_i - \tilde{f}_w(X_j)) \nabla \tilde{f}_w(X_j) \quad (4.4)$$

Assuming the Sigmoid function as the activation function, the gradient of the hypothesis function, i.e., $\nabla \tilde{f}_w(X_j)$, is calculated by Equation (4.10).

$$\nabla \tilde{f}_w(X_j) = -\tilde{f}_w(X_j)(1 - \tilde{f}_w(X_j))X_j \quad (4.5)$$

Using Equations (4.3) to (2.5), the final weight update rule is:

$$W = W + \eta \sum_j^n (y_i - \tilde{f}_w(X_j)) \tilde{f}_w(X_j)(1 - \tilde{f}_w(X_j))X_j \quad (4.6)$$

Equation (4.11) is used to update the weights in a batch learning mode, where there is a training set with n instances. In a data stream scenario, this update rule is would not practical. As an alternative, the stochastic gradient descent optimizer is used for updating the weight vector for every instance of the data stream [67]. Using the stochastic gradient descent optimizer, the weights are updated as follows:

$$W = W + \eta \cdot (y - \tilde{f}_w(X)) \tilde{f}_w(X)(1 - \tilde{f}_w(X))X \quad (4.7)$$

K-Nearest Neighbours

Nearest neighbor classifiers, also named lazy learners or instance-based learners [76], present a natural way of learning data incrementally. Every processed example is kept and can serve as a reference for new data. Classification is based on the labels of the closest historical examples. In this particular lossless version of the closest neighbor algorithm named IB1, the reference set grows with every example, increasing memory requirements and also classification time. Another technique from this family named IB3, restricts the number of stored historical data points only to only those that were useful for the classification procedure. Apart from decreasing the time period and also memory needs, the size limitation of the reference set offers a forgetting mechanism as it eliminates outdated examples from the model. A more recent example of utilizing the closest neighbor strategy to classify streaming data is the ANNCAD algorithm [77]. In ANNCAD, the authors suggest dividing the feature space many times to generate a multi-resolution data representation, where finer levels contain more training points than coarser levels. Predictions are designed

according to the majority of closest neighbors starting at finer levels. Once the finer levels present an inconclusive prediction, coarser levels are used. Concept drift is addressed by using a fading factor, which decreases the weight of older training examples. The pseudocode of a K-NN with a sliding window is presented in Algorithm A.5.

Discussion

We summarize the advantages and disadvantages of the learning algorithms. Naive Bayes and perceptron are effective in terms of runtime and memory usage when compared with other algorithms. Despite the class conditional independence presumption, naive Bayes demonstrates similar accuracy to other algorithms in the literature [29]. A decision stump might be prone to underfitting. Perceptron can't adequately model nonlinear problems. Models produced by decision stump and also Hoeffding trees can easily be interpreted. Hoeffding trees generally consume more memory compared to naïve Bayes and perceptron. Although the K- nearest neighbours algorithm is not difficult to execute and understand, it is costly in terms of memory usage and execution runtime. In addition, based on [78], the Hoeffding and perceptron are definitely more appropriate in our work.

4.3 Adaptive Classification

Once a drift is experienced, adaptive learning addresses concept drift by retraining the predictive model. In Section 4.3.1, we provide a formal definition for concept drift, and the adaptation approaches for handling concept drift are explained in Section 4.3.2, followed by the state-of-the-art methods for drift detection in Section 4.4.

4.3.1 Concept Drift Phenomenon

When it comes to non-stationary environments, the underlying distribution might change $D_i \neq D_j$, for any two time points i and j . Accordingly, the concept of the two points becomes unstable and the model will not be able to approximate the recent incoming data distribution [79]. Consequently, the essential task of streaming data analytics is finding any significant changes in incoming data [80]. Furthermore, the problem of changing definitions

and distributions of a learner of incoming data will influence the classification accuracy of a model that is trained on data used previously [81, 82].

We discuss the formal definition of concept drift in the following subsections.

Formal Definition

One of the most important properties of data streams is that they can change over time. Therefore, classifiers for data streams need to be capable of predicting, detecting, and adapting to concept changes. In order to do so, the nature of changes needs to be studied, including their rate, cause, predictability and severity [8].

According to Bayesian decision theory [83], a classification model can be described by the prior probabilities of classes $p(y)$ and class conditional probabilities $p(y|x)$, for all classes $y \in K_1, \dots, K_c$, where c is the number of predefined classes. The dynamic nature of data streams is reflected by changes in these probability distributions in an event called concept drift. In practical terms, concept drift means that the concept about which data is being collected may shift from time to time after a minimal stability period [1]. Depending on the research area, concept drift can sometimes be referred to as temporal evolution, population drift, covariate shift, or non-stationarity. Most studies assume that concept drift occurs unexpectedly and is unpredictable, in contrast to seasonal changes. However, concept drift adaptation mechanisms often entail solutions for cases where changes can be anticipated in correlation with environmental events. Formally, concept drift can be defined as follows [8]:

Definition 1. For a given data stream S , concept drift may occur between two points in time, t and $t + \Delta$, *iff* $\exists x: p^t(x, y) \neq p^{t+\Delta}(x, y)$ where p^t refers to the joint distribution at time t between the set of input attributes and the class label.

By considering this, any changes in incoming data can be characterized by changes in the components of Bayesian decision theory [84] [85] [86]:

- Prior probabilities $p(y)$ are prone to changes.
- Probabilities $p(X|y)$ of class conditional are also prone to changes.
- Consequently, posterior probabilities $p(y|X)$ may/may not change.

Based on the cause and effect of these changes, two types of drift are identified: real drift and virtual drift [8].

Real drift is defined as changes in $p(y|x)$. It is worth noting that such changes can occur with or without changes in $p(x)$, therefore, they may or may not be visible from the data distribution without knowing the true class labels. Such a distinction is crucial, as some methods attempt to detect concept drifts solely using attribute values [87]. Real drift has also been referred to as concept shift [88] and conditional change [89].

Virtual drift is usually defined as changes in the attribute-value $p(x)$ or class $p(y)$ distributions that do not affect $p(y|x)$ [24, 90, 91]. However, the source and therefore the interpretation of such changes differs among authors. Widmer and Kubat [91] attributed virtual drift to incomplete data representation rather than true changes in concepts. Tsymbal [24] on the other hand defined virtual drift as changes in the data distribution that change the decision boundary, while Delany [90] described it as a drift that does not affect the target concept. Furthermore, virtual drifts have also been called temporary drifts [92], sampling shifts [88], and feature changes [89].

To simplify the difference between real and virtual drift, let us take the example of the classification problem presented in Table 4.2. The task in the example is to determine whether a given flight will be delayed or not. If an airline company changes the flight time, but this does not result in a delay, this is regarded as virtual drift. Similarly, if due to a crisis, a company changes the frequency of certain flights but again, the flights leave without any delay, this is also regarded as virtual drift. However, if some flights are regularly delayed, even though they used to be on time, real drift is occurring. The difference between real and virtual drift is also illustrated in Fig.4.5. The plot shows that only real concept drifts change the class boundary making any previously created model obsolete. The illustrated real drift occurs without any changes in the attribute space, however, in practice, changes in prior probabilities may appear in combination with real drift.

One of the challenges in concept drift detection is recognizing real concept drift from outliers or noise in data. No adaptation is required if an outlier or noise is experienced in a data stream [8]. This is a critical observation since a decision model is discarded because of a wrong alarm for concept drift. Outlier detection methods might be used alongside drift detection methods to avoid false alarms for concept drift. [93] conducted a survey on outlier and anomaly detection in data.

Finally, from a predictive perspective, adaptation is required once a real concept drift occurs since the current decision boundary is outdated for the new incoming data [8]. Adaptation means updating the classification model for the new distribution to keep the classification accuracy high. We discuss adaptive learning methods in Section 2.3.2.

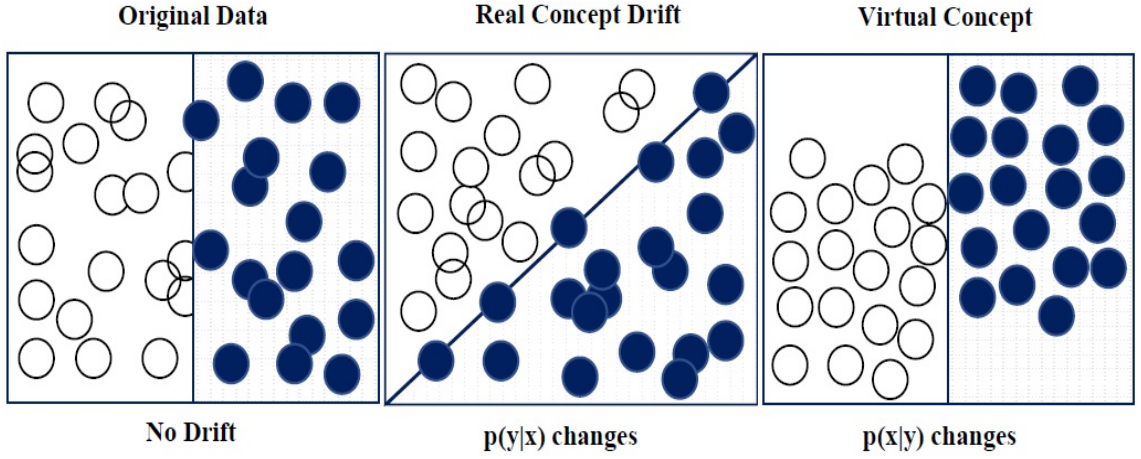


Figure 4.5: Two types of drift: where instances are represented by circles and different classes are represented by different colors.

Table 4.2: Sample from the airlines dataset/ where Airline=A, Flight=F, Day of week=DOW, Time=T, Length=L, Delayed=D.

| A | F | From | To | DOW | T | L | D? |
|----|------|------|-----|-----|----|-----|-----|
| CO | 269 | SFO | IAH | Wed | 15 | 205 | yes |
| US | 1558 | PHX | CLT | Wed | 15 | 222 | yes |
| AA | 2400 | LAX | DFW | Wed | 20 | 165 | yes |
| AA | 2466 | SFO | DFW | Wed | 20 | 195 | yes |
| AS | 108 | ANC | SEA | Wed | 30 | 202 | no |
| CO | 1094 | LAX | IAH | Wed | 30 | 181 | yes |

Concept Drift Patterns

In addition to differences in the cause and effect of concept changes, researchers have identified several ways in which such changes occur. In this regard, drifts can be further characterized, for example, by their permanence, severity, predictability, and frequency [92, 94, 95]. However, the most analyzed aspect of drifts is the way they manifest themselves over time [1, 21, 24, 82]. Fig 4.6 shows three basic structural types of changes that may occur over time.

A sudden/abrupt drift happens when suddenly the source distribution in S^t at a moment in time t is substituted by another distribution S^{t+1} . Once the new distribution has been used to train a generated classifier, a sudden drift reduces the classification abilities of a classifier, whereas gradual drift is connected with a slower rate of change and refers to a transition stage where examples of two different distributions P^j and P^{j+1} are mixed. With the passage of time, the likelihood of monitoring P^j examples decreases whereas the likelihood of monitoring P^{j+1} examples increases. Another kind of drift refers to recurrent concepts, i.e., after a period of time, previous concepts may reappear. The most recent approaches to address the three types of drifts are: abrupt drift [96] gradual drift [97] [98] and recurring drift [99–101]. We emphasize that the proposed concept drift detection method has been designed mainly for sudden/abrupt drift.

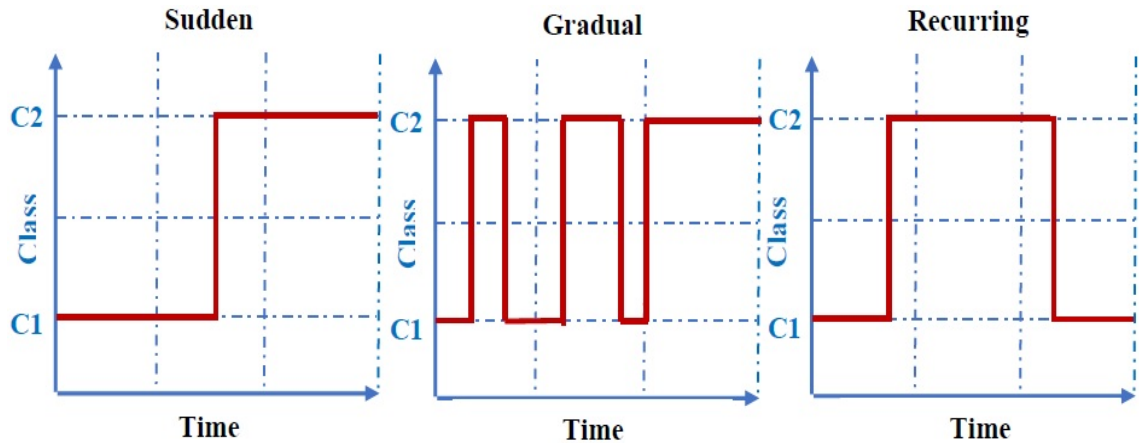


Figure 4.6: Concept Drift Patterns

Concept Drift Terminology

The concept drift phenomenon has been studied by various research communities including data mining, machine learning, statistics, and information retrieval [86]. This phenomenon, however, may be known by different terms in each community. Table 4.3 lists the terms that correspond with the concept drift phenomenon in different research areas.

Table 4.3: Concept Drift Terminology

| Domain | Term |
|--------------------------|--------------------------------|
| Data Mining | Concept Drift |
| Machine Learning | Concept Drift, Covariate Shift |
| Evolutionary Computation | Changing Environment |
| AI and Robotics | Dynamic Environment |
| Statistics, Time Series | Non-Stationary |
| Databases | Concept Drift, Load Shedding |
| Information Retrieval | Temporal Evolution |

4.3.2 Concept Drift Adaptation Approaches

Data stream mining with the existence of concept drift is mainly formed on the trigger/active or evolving/passive approaches [98, 102]. Trigger/active approaches are designed to update the model after detecting the drifts. Evolving/passive approaches continuously update the model whenever new data become available, regardless as to whether drift is occurring or not.

The two approaches share the same goal, which is to update the model according to the recent concept of incoming data. However, the mechanism of each is different. For example, the trigger/active approach observes the error rate of the algorithm; that is, when the incoming data distribution is constant, there is a decreasing error rate, otherwise, it rises. This happens by placing two levels of drifting and warning. Whenever there is an increase in the actual classifier error, this means there will be a drift in the current concept of incoming data, whereas when the error rate keeps increasing and then reaches the level of error, this means that a drift has been detected [48]. Conversely, the evolving/passive approach does not detect changes or drifts, rather it provides an updated model at all times [103–105]. The next section discusses approaches belonging to the aforementioned two categories and are related to this work.

4.4 Related Work: Concept Drift Detection Techniques

4.4.1 Fully Supervised Concept Drift Detection Techniques

This section critically reviews existing concept drift detection techniques by categorising drift detection methods as either statistical based, window based or ensemble based. Table 2 provides a full illustration of the categorisation and techniques covered by this paper.

Statistical Methods

Statistical parameters such as the mean and standard deviation associated with the predicted results are analysed in order to detect concept drifts. Various statistical methods are detailed as follows:

- **DDM: Drift Detection Method** Gama et al. [48] monitor the error rate of the classifier to detect concept drift. On the basis of the probably approximately correct (PAC) learning model (Mitchell, 1997), the classification error rate decreases or stays constant as the number of instances increases. Otherwise, it suggests the occurrence of a drift. Let p_t be the error-rate of the classifier with a standard deviation as shown by Equation 4.8, at time t .

$$s_i = \sqrt{\frac{p_i(1 - p_i)}{i}} \quad (4.8)$$

As instances are processed, DDM updates two variables p_{min} and s_{min} when $p_t + s_t < p_{min} + s_{min}$. DDM warns of a drift when $p_t + s_t \geq p_{min} + 2 * s_{min}$, and alarms of a drift when $p_t + s_t \geq p_{min} + 3 * s_{min}$. The variables p_{min} and s_{min} are reset when a drift occurs.

- **EDDM: Early Drift Detection Method** Baena-Garcia et al. [54] proposed a modification of DDM called EDDM. The authors use the same warning-alarm mechanism that was proposed by Gama, but instead of using the classifier's error rate, they propose the *distanceerrorrate*. They denote p'_i as the average distance between two consecutive misclassifications and s'_i as its standard deviation. Using

these values the new warning and alarm conditions are given by Equations 4.9 and 4.10.

$$\frac{p'_i + 2 \cdot s'_i}{p'_{max} + 2 \cdot s'_{max}} \leq \alpha \quad (4.9)$$

$$\frac{p'_i + 3 \cdot s'_i}{p'_{max} + 3 \cdot s'_{max}} \leq \beta \quad (4.10)$$

EDDM is designed to work better than DDM for slow gradual drift but is more sensitive to noise. Another drawback of this method is that it searches for concept drift when a minimum of 30 errors have occurred (as opposed to a minimum of 30 examples). This is necessary to approximate the binomial distribution by a normal distribution but this can cause a significant delay in change detection.

- **Page-Hinkley test (PH)** This allows the efficient detection of changes in the normal behavior of a process established by a model. The test variable m^t used in PH is defined as the cumulative difference between the observed values e^i and their mean up until the current moment in time:

$$m^t = \sum_{i=1}^t (e^i - e^t - \delta) \quad (4.11)$$

where $e^t = 1 / \sum_{i=1}^t e^i$ and α corresponds to the magnitude of changes that are allowed [69]. For drift detection, Gama et al. propose to treat the classifier's error rate as the observed value. Additionally, the minimal m^t is defined as $M^t = \min(m^i; i = 1 \dots t)$. The PH test calculates the difference between M^t and m^t ($PH^t = m^t - M^t$), and if this difference is higher than a user-specified threshold (λ), a change is flagged.

As an alternative to tracking the classifier's mean error over time, Gama et al. propose to perform the PH test with the ratio between two error estimates: a long-term error estimate (using a large window of examples or a weak fading factor) and a short-term error estimate (using a short window or a strong fading factor). If the short-term error estimator is significantly greater than the long-term error estimator, a drift is signaled. For sliding windows, the procedure involves two windows of different sizes

$W1 = e^i | i \in (t - d_1, t)$ and $W2 = e^i | i \in (t - d_2, t)$ (with $d_2 < d_1$) and computing the moving average w.r.t. both: $M_{w1}(t) = 1/d_1 \sum_{i=t-d_1}^t e^i$ and $M_{w2}(t) =$

$1/d_1 \sum_{i=t-d_2}^t e^i$. The PH test monitors the ratio $R(t)$ between both moving averages. It is worth noting that this test is designed to detect mainly abrupt drifts, with higher α entailing fewer false alarms, but possibly causing some changes to be missed [8].

- **STEPD** : STEPD is a method proposed by [55] over two windows, namely, recent and older, which calculates statistical tests with continuity correction. Within a concept, the accuracy of the learner using the two windows is expected to be stable. When there is a significant difference in the accuracy of the recent window, warning and drift are signalled.
- **MDDM-A, MDDM-G and MDDM-E** The work in [96] introduces the three McDiarmid Drift Detection Methods (MDDMs) which utilize the idea of McDiarmid's inequality [106] for detecting drifts. MDDMs use a window of size n over the prediction results. The idea is when the prediction results are correct, a 1 will be inserted into the window; and 0 otherwise. Inside the window, each element is associated with a weight, where $w_i < w_{i+1}$. When the inputs are treated, inside the window, the weighted average of the elements are calculated, i.e. μtw , and the average of the maximum weighted is monitored so far, i.e. μmw . Consequently, between the two weighted means, if a considerable variation has been observed, this indicates concept drift. Different weighting schemes are considered for the arithmetic (MDDM-A) and the geometric (MDDM-G) schemes. The arithmetic scheme is given by $w_i = 1(i - 1) \times d$, where $d \geq 0$ is the difference between two consecutive weights. The geometric scheme is given by $w_i = r^{(i-1)}$, where $r \geq 1$ is the ratio of two consecutive weights and the Euler scheme (MDDM-E) is defined by $r = e^\lambda$ where $\lambda \geq 0$.
- **SeqDrift2** This method [107] uses two repositories to store incoming data, the first holds a combination of new and older entries while the second contains new entries only. This is done through the application of the reservoir sampling strategy [108]. SegDrift2 measures the means of entries within both repositories i.e. μ^1 and μ^r and using the Bernstein inequality [109] the difference in the mean is measured. A difference between two means beyond an upper bound will indicate that concept drift has occurred.
- **RDDM**: RDDM: This method, developed in [110], solves the problem of a performance loss of DDM, which is caused by decreasing sensitivity which requires

many examples for drift detection. This method uses a mechanism to abandon older examples, regularly recalculating the statistics of RDDM which are responsible for detecting drifts. The authors found that RDDM delivers higher accuracy than DDM in most cases, by detecting drifts earlier, despite an increase in false positives and memory consumption.

Window-Based Methods

Usually, a fixed reference window is utilized to summarise past information and a sliding window is utilized to summarise the most recent information. A significant difference between the distributions of these two windows implies the occurrence of concept drift. Fig 4.7 illustrates the workflow of the window-based processing scheme. Various window-based methods are detailed as follows:

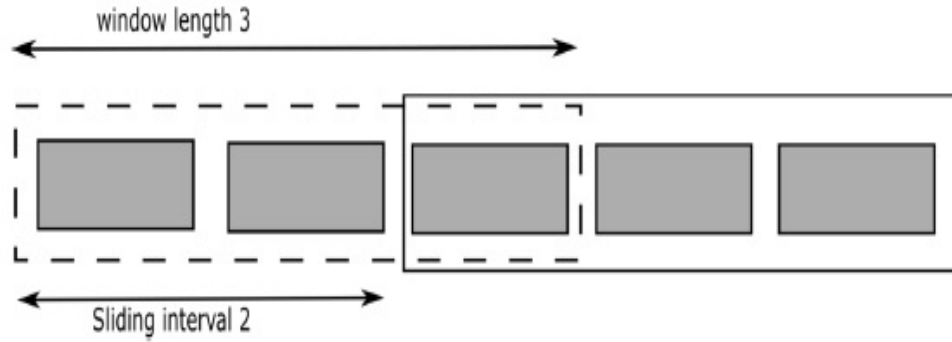


Figure 4.7: Sliding window. The borders identify two different windows

- **ADWIN: Adaptive Sliding Window** This method was developed in [66]. For drift detection, ADWIN examines two sub-windows by sliding a window w on the results of the predictions i.e. w_0 size n_0 and w_1 size n_1 , of w where $w_0 \cdot w_1 = w$. Thus, ADWIN indicates a concept drift when there is a considerable variation among the means of the two sub-windows. i.e. $\mu w_0 - \mu w_1 \geq \xi$ where $\xi = \sqrt{\frac{1}{2m} \ln(\frac{4}{\delta'})}$, m is the harmonic mean of n_0 and n_1 , $\delta' = \frac{\delta}{n}$, is the confidence level and n is the size of window w . After a drift has been detected, elements are dropped from the tail of the window until no significant difference is seen.
- **FHDDM** This method proposed in [111] implements a method for detecting drifts using Hoeffding's inequality through a window of size n . This window detects a drift if a considerable variation is observed between the current probabilities and the

maximum of correct predictions. By using the error's probability (δ , default 10-7), FHDDM finds the variations among the probabilities (ΔP) and the threshold (ϵ). Consequently, FHDDM will signal a drift if $\Delta P \geq \epsilon$.

- **SEED:** This method, proposed in [112], within a window W , compares two sub-windows. The older part of this window is dropped when there is a distinct average exhibited by the two sub-windows. Hoeffding Inequality with Bonferroni correction is used by SEED to calculate its test statistic, performing block compression to remove unnecessary cut points and then the blocks that are homogeneous in nature are merged.
- **HDDM_A test and HDDM_W test:** These two methods are proposed in [113] and use Hoeffding's bounds for drift detection. The HDDM_A test compares moving averages to detect drifts, whereas HDDM_W compares the weight of moving averages for drift detection, where the EMWA forgetting scheme [114] is used for weighting. The authors of this work noted that the HDDM_A test is more suitable for sudden drifts while HDDMW is best suited to gradual drift.

Algorithm 1: Generic Scheme of Drift Detector

Require: S : data stream of examples

Drift Test (DT): using statistical tests or mathematical inequalities.

C : classifier

Result: Drift $\in \{\text{TRUE}, \text{FALSE}\}$

```

1 Initialize (Parameters)
2 for each example  $x^t \in S$  do
3   | Measure DT;
4   | if drift detected then
5   |   | Return TRUE
6   | else
7   |   | Return FALSE
8   | end
9 end
```

Finally, regarding the existing drift detection methods, for instance, the Page-Hinkle (PH Test) measures the difference in incoming data using the mean and identifies a drift when the difference is larger than a user predefined threshold. Other methods determine sets of variables to monitor concept drift in data streams. DDM only maintains a few variables and hence requires less execution runtime. HDDM utilizes Hoeffding's inequality for drift detection. On the other hand, ADWIN requires more memory when detecting drift because these methods store prediction results within the sliding windows and utilize a sub-window compressor or reservoir sampling procedures. Seqdrift2 employs the Bernstein inequality to detect concept drift. It also uses sample variance and assumes that the sampled data will follow a normal distribution, which can be unrealistic in real situations. The use of parameter variance instead of a real instance also leads to delay in drift detection. FHDDM employs the Hoeffding's inequality. FHDDM differs from HDDMs by sliding a window on the prediction results to detect concept drift, which results in less runtime and memory consumption. Consequently, we believe there is still a need for a drift detector which is able to detect concept drift in less time and consumes less memory compared to the aforementioned methods.

Block - Based Ensemble Methods

In block-based approaches, instances arrive in portions, called blocks. Most block-based ensembles evaluate their components periodically and replace the weakest ensemble member with a new classifier. This approach reacts very well to gradual concept drift rather than sudden drift and ensures accuracy. Fig 4.8 presents the workflow of the block-based processing scheme. Various block-based ensemble methods are detailed as follows:

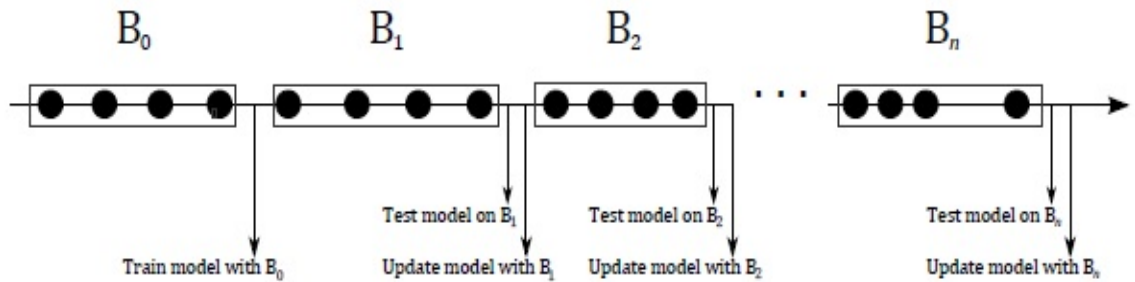


Figure 4.8: Block workflow

- **Accuracy Weighted Ensemble (AWE)** [115] For each new incoming data block, the authors train a new classifier using a typical static learning algorithm, such as naive

Bayes, C4.5, or RIPPER. After the new classifier is trained, all the existing trained classifiers in the ensemble are evaluated using the most recent block. Then, after each evaluation using mean square error, the algorithm selects the n best classifiers in order to update the ensemble.

- **Accuracy Updated Ensemble (AUE2)** [116] This uses an online classifier to allow individual learning models to be updated directly rather than only adjusting the weights as per AWE. If no drift is expected to occur, the classifiers improve as if they were trained on one large chunk. In other words, the size of the block can be lowered without affecting the performance accuracy of the ensemble.
- **Dynamic Weighted Majority (DWM)** Another popular online ensemble is an algorithm called Dynamic Weighted Majority (DWM) [50]. In DWM a set of incremental classifiers is weighted according to their accuracy after each incoming example. With each mistake made by one of DWM's component classifiers, its weight is decreased by a user-specified factor β . Furthermore, after a period of predictions p the entire ensemble is evaluated and, if needed, a new classifier is added to the ensemble. If learned on a large stream, DWM can potentially generate an extensive number of components, therefore, ensemble pruning is often considered as an extension [1]. DWM is an extension of the weighted majority algorithm [117] known from the field of online learning. However, DWM takes into account the dynamic nature of data streams and is designed to track concept drift. In contrast to its predecessor, DWM adds and removes component classifiers in response to the global performance of the entire ensemble and the local performance of individual components.
- **Adaptive Classifier Ensemble (ACE)** Both SEA and AWE are highly dependent on data block size. Larger blocks promote more accurate ensemble members but extend the period in which these algorithms cannot respond to sudden concept drifts. Small blocks, however, worsen the performance of each component classifier and as a result, the entire ensemble. To overcome these drawbacks, Nishida et al. proposed an online learning system, called the Adaptive Classifier Ensemble (ACE) [118], which uses an online learner alongside an ensemble of batch classifiers.

ACE consists of one online classifier, many batch classifiers, and a drift detector. With each incoming example, the online classifier is incrementally trained and the block is extended. Furthermore, the drift detector checks the average accuracy of each batch

classifier (calculated on the current block), and if the best performing component falls outside a $100(1-\alpha)\%$ confidence interval, where α is a user-specified parameter, a change is signaled. When concept drift is detected or the number of buffered examples exceeds the block size, a new batch classifier is created and the online learner is reset. ACE forms its final hypothesis by aggregating the predictions of the online learner and batch learners using a weighted majority vote, with each classifier C_i receiving at time t a weight W_i^t defined as:

$$w_i^t = \left(\frac{1}{1 - A_i^t} \right) \mu \quad (4.12)$$

where A_i^t is the accuracy of the i -th classifier calculated on the current block of examples and μ is a normalization factor. One of the characteristic features of ACE is that it does not limit the number of ensemble members. This property allows the algorithm to accurately react to recurring concepts by reusing previously trained classifiers. Furthermore, the addition of an online learner and drift detector offer quicker reactions to sudden concept changes compared to most block-based ensembles. Finally, it is worth noting that ACE detects changes based on the performance of a single batch classifier instead of the entire ensemble and uses a custom drift detection method. It is worth noting that the problem of quicker drift detection in block-based ensembles was also tackled in the Batch Weighted Ensemble algorithm, proposed by Deckert [119].

- **Learn++.NSE (.NSE)** Learn++.NSE [120] is a block-based ensemble inspired by human learning theory. Several components of this algorithm relate to schema theory [59], which is a psychological model that describes the process of human knowledge acquisition and memory organization. For example, Learn++.NSE retains, constructs, or temporarily discards knowledge depending on the nature of changes in the stream. Furthermore, the algorithm weights examples depending on their difficulty measured in terms of ensemble performance.

The training of Learn++.NSE starts with evaluating the ensemble on a block of new examples. Next, the algorithm identifies which examples are correctly predicted by the existing ensemble and gives lower weights to these examples, as they are less difficult. Using the block of examples with updated weights, a new classifier is created and added to the ensemble. Later, all of the ensemble members are evaluated

and their weights are calculated as log-normalized multiplicative inverses of their weighted errors. The weighting function is designed to temporarily block votes from component classifiers that do not match the current environment. Learn++.NSE has an interesting psychological inspiration, which is apparent mainly in the training and weighting of component classifiers. First of all, the algorithm tries to detect new concepts by analyzing the performance of the ensemble on new data, which can be connected to the process of problematizing known from tutoring theory [120]. Furthermore, the algorithm weights ensemble members using a sigmoid-based function which takes into account the recent performance of a given component classifier and draws inspiration from the human process of memory tuning. Finally, it is worth mentioning that Learn++.NSE does not permanently discard any component classifiers and is therefore particularly suitable for streams with recurring drifts.

Algorithm 2: Generic Scheme of Block-Based Ensemble

Require: S : Examples of Data stream split into blocks;

d : size of block;

k : number of the components in an ensemble;

$Q()$: measure of classifier quality

Result: E : k weighted classifiers (ensemble)

```

1 for each blocks  $B^i \in S$  do
2   using  $B^i$  and  $Q()$  to build and weight each nominee classifier  $C'$ ;
3   weight  $C^i$  using  $B^i$  and  $Q()$ ;
4   if  $|E| < k$  then
5      $E \leftarrow E \cup \{C'\}$ 
6   else
7     substitute the weakest component in ensemble with  $C$ ;
8   end
9 end

```

4.4.2 Semi-Supervised Learning Methods

To elaborate on the work presented in Section 5.3, under the assumption of limited access to class labels due to the high labelling cost and time consumption, it is worth mentioning some semi-supervised learning methods. Some of these are appropriate for processing data streams incrementally, namely co-training, tri-training, self-training and K-prototype clustering.

Co-Training The work in [121] assumes that features can be divided into two groups and each sub-group is sufficient to train a classifier. Firstly, two separate classifiers are trained on the two sub- group sets respectively. Each classifier is re-trained with the additional training samples confidently labelled by the other classifier. Unfortunately, the requirements can hardly be met.

Tri-Training The work in [122] extends the idea of co-training, which does not require sufficient and redundant views. This method generates three classifiers from the original labelled instance set using bootstrap sampling. These classifiers are then refined using unlabelled instances in the tri-training process. In each round of tri-training, an unlabelled instance is labelled for a classifier if the other two classifiers agree on the labelling. For tri-training, though it does not require sufficient and redundant views, it also does not use the attribute values of one sample sufficiently as it uses bootstrap sampling from the original sample.

Self-Training The work in [123] only trains one initial classifier. Then the classifier is used to classify the unlabeled samples in the data stream. The most confidently unlabeled points are used to re-train the classifier. It avoids the aforementioned shortcomings of co-training and tri-training, but it is difficult to measure the confidence of labeling.

K-prototype clustering The work in [124] proposes an algorithm based on the k-means paradigm but removes the numeric data limitation whilst preserving its efficiency. The K-prototype algorithm integrates the k-means and k-mode algorithms to deal with mixed data types. Therefore, the k-prototype algorithm is more useful practically because data collected in the real world are a mixed data type. Assume a set of n objects, $X = \{X_1, X_2, \dots, X_n\}$. $X_i = \{X_{i1}, X_{i2}, \dots, X_{im}\}$ consists of m attributes. The goal of clustering is to partition n objects into k disjoint clusters $C = \{C_1, C_2, \dots, C_k\}$, where C_i is an i -th

cluster center. The distance $d(X_i, C_j)$ between X_i and C_j can be calculated as follows ...

$$d(X_i, C_j) = d_r(X_i, C_j) + \gamma d_c(X_i, C_j) \quad (4.13)$$

where $d_r(X_i, C_j)$ is the distance between numerical attributes, $d_c(X_i, C_j)$ is the distance between categorical attributes, γ is a weight for categorical attributes.

$$d_r(X_i, C_j) = \sum_{i=1}^p |x_{il} - C_{jl}|^2 \quad (4.14)$$

$$d_c(X_i, C_j) = \sum_{i=p+1}^m \delta(x_{il} - C_{jl}) \quad (4.15)$$

In Equation 4.14, $d_r(X_i, C_j)$ is the squared Euclidean distance measure between cluster centers and an object on the numerical attributes. $d_c(X_i, C_j)$ is the simple matching dissimilarity measure on the categorical attributes, where $\delta(x_{il} - C_{jl}) = 0$ for $x_{il} = C_{jl}$ and $\delta(x_{il} - C_{jl}) = 1$ for $x_{il} \neq C_{jl}$.

Finally, in the case of mining an incremental data stream, clustering on a small sample set is rather efficient. In addition, clustering can also identify the classification boundary and the distributions. Therefore, it is valuable to perform clustering on unlabelled samples of incoming data stream. More importantly, considering that the incoming sample attributes have both numeric and categorical values, we propose to use K-prototype as a solution to our semi-supervised drift detection in data streams.

4.5 Experimental Evaluation

We study the analysis procedures, the classification methods, the drift detection methods, and also the resource consumption measures in the subsequent subsections.

4.5.1 Evaluation Procedures for Data Streams

In experimental configurations, evaluation methods determine which instances are employed for training and testing a learning algorithm [61].

In relation to the batch learning scenario, the size of data plays an important part in developing an evaluation process. Learning algorithms are frequently assessed by a (ten-fold) cross-validation procedure when datasets are tiny, e.g., with under a thousand instances, since it tends to make the highest use of data by several sub-evaluations. A four-fold cross-validation partitions the data into four subsets or even folds of the same size. Subsequently, there are four iterations for both training and testing. The fold which is employed for testing is defined as a validation set. In each iteration, a model is trained and its accuracy also is calculated, making use of the validation fold. The ultimate accuracy is the average of the accuracies of most iterations. The k-fold cross-validation process has the possibility of being extremely slow against large datasets. Thus, it is essential to decrease the statistics of repetitions or even folds to complete the process within a reasonable time.

As a substitute, the holdout procedure might be regarded as a solution for decreasing the computation effort. This particular procedure randomly selects, for instance, one third of instances as an exam set, also called a holdout set, in addition to making use of the remainder of instances as a training set. The model built using the training set is tested on the test set. Bootstrapping is a method which samples the training instances uniformly with replacement originating from a dataset. In sampling with a substitute, as soon as one example is selected for training, it is still a nominee for selection, and it also might be selected all over again. Finally, examples which were not selected for training are regarded as candidates for testing.

Although cross-validation is commonly employed for batch setting, it is impractical for data streams due to its computational expense (in relation to either memory or run time). Both the holdout and bootstrapping procedures need all the data to be readily available for partitioning which contradicts the nature of a data stream. In addition, in data stream mining, ensuring the accuracy of classification models over time without violating the four requirements discussed in Section 2.2.3 is a crucial issue.

To deal with these difficulties, the Incremental Holdout and Predictive Sequential (Prequential) methods are devised for the assessment of learning algorithms against data

streams [8, 61]. We explain and compare these methods as follows:

Holdout

When conventional batch learning gets to a scale in which cross-validation is overly time-consuming, it is a common approach to measure overall performance during a single holdout set. This is very helpful if the division between training and test sets has been pre-defined, hence the outcomes from various studies can be directly compared.

Interleaved Test-Then-Train or Prequential

Every individual example can be utilized to test the model just before it is utilized for training, and also as a result of this, the accuracy can be incrementally kept up to date. When intentionally carried out within this order, the model is definitely being tested on examples it has not observed. This scheme provides the advantage which absolutely no holdout set is required for testing, making maximum usage of available data. Additionally, it guarantees a smooth plot of accuracy over time, as every individual example is going to become increasingly much less significant to the overall average.

Comparison

Holdout evaluation provides a far more accurate estimation of the accuracy of the classifier on newer data. Nevertheless, it takes the latest test data that it is hard to obtain for real datasets. The work in [65] suggest to using a forgetting mechanism for estimating holdout accuracy using prequential accuracy: a sliding window of size w with the most recent observations, or fading factors which weigh observations using a decay factor α . The result of the two mechanisms is extremely comparable (every window of size w_0 might be approximated by some decay factor α_0). As data stream classification is a relatively new area, such evaluation practices are not nearly as well researched and established as they are in the conventional batch setting.

4.5.2 Data Stream Datasets

Synthetic and real-world data streams are widely used to evaluate the performance of online and adaptive learners. Synthetic data streams are also known as artificial data streams in the literature.

As for the real-world data streams, it is not recorded nor identified when concept drift occurs [68, 125]. That is, we do not have the ground truth regarding the drift points for the evaluation of the drift detectors. Alternatively, we use real-world static datasets to create data streams experiencing concept drift at specific points for our experiments. We describe the synthetic and real-world data streams as follows:

Synthetic Data Streams

SEA Generator (SEA), AGRAWAL Generator (AGR), Mixed, Sine1, Sine2, Wave, RBF_{GR} : $RBF_{Gradual}$, and $Tree_R$: $Tree_{Recurrent}$ are synthetic data streams that are frequently applied in the literature [53, 55, 57, 66, 126–128]. We describe these as follows:

- **SEA Generator (SEA)** This generator has been used to generate 100,000 instances of incoming data, where it generates 3 numerical attributes, that vary from 0 to 10. To simulate a single abrupt concept drift, SEA uses *ConceptDriftStream* as the main class, with the following parameters: (i) SEAGenerator -f 3 refers to the current concept of the incoming data, (ii) SEAGenerator -f 2 refers to the new concept of the incoming data, (iii) position of the drift within the incoming data, (iv) the width of the drift. Finally, the generated drift will be located at location 10k, 20k and 50k.
- **AGRAWAL Generator (AGR)** We use the AGR generator to generate 100,000 instances with multiple sudden drifts. We use the AGR generator to generate three drifts every 25,000 instances. Agarwal contains the hypothetical data of people applying for a loan, where class A representing the approval of loan and class B representing rejection of loan. AGR generator produces a stream, each instance of which consists of nine attributes: six numeric and three categorical.
- **Mixed** This dataset contains sudden drift and two types of data attributes. The first is a numeric data type and it has two attributes (x and y) which are uniformly distributed

in $[0, 1]$. The second is a Boolean data type and it has two attributes (v and w). From the three following cases, if two of them are placed: $v, w, y < 0.5 + 0.3 \times \sin(2\pi x)$, the instances are classified as positive. After a drift has occurred, the classification is reversed. At every 20,000 instances, a drift will occur.

- **Sine1** We generate the Sine1 dataset with sudden drift. It has with two attributes (x and y) which are uniformly distributed in $[0, 1]$. In addition, this dataset uses the following function $y = \sin(x)$ for classification. Therefore, any instances below the curve are classified as positive while the others are negative till the first drift occurs. At every 20,000 instances, a drift will occur and then the classification is reversed.
- **Sine2**: This dataset has two attributes (x and y) which are uniformly distributed in $[0, 1]$. Following a function $0.5 + 0.3 \times \sin(3 \times \pi \times x)$, instances under the curve are classified as positive while the other instances are classified as negative. At every 20,000 instances, a drift will occur and then the classification is reversed.
- **Wave**: This dataset uses the random tree generator to create a dataset with a single sudden concept drift evenly distributed over 100K examples of incoming data and sudden drift is located at 50K. The wave data set is defined by 21 numerical attributes.
- **$RBFG_R$** : $RBFG_{Gradual}$ uses the radial basis function generator to create a dataset with four gradual recurring drifts over 1,000,000 examples. Each concept of the generated drifts contains four decision classes.
- **$Tree_R$** : $Tree_{Recurrent}$ uses the random tree generator to create a dataset containing four sudden recurring drifts and these four drifts are evenly distributed over 1,000,000 examples. This dataset is defined by five nominal and five numerical attributes.

Real-World Data Streams

We select synthetic datasets to evaluate all the works in different scenarios. On the other hand, we share the common claim that when using real datasets, it is not clear if there is any drift or when drifts occur. Therefore, the real dataset serves to evaluate and compare the methods in a real-life scenario rather than in a concrete drift situation.

In addition, there is no ground truth available regarding concept drift in real-world data streams. That is, we do not know whether concept drift occurs or where it occurs in

these data streams [56, 68]. Consequently, we cannot measure the detection delay, true positives, false positives, and false negatives of the drift detectors. We can only evaluate the average of detection runtime, detection memory usage, and classification accuracy.

- **Electricity (Elec):** In data stream classification, the electricity dataset is the most widely used in this field. This dataset comprises energy prices gained from the electricity market where the data are influenced by season, supply, weather, demand and time of usage. The electricity dataset comprises 45,312 examples defined by seven features.
- **Airlines:** This is a real-world dataset which is widely used in data stream classification. This dataset involves predicting whether a given flight will be delayed using the data of the scheduled departure. The airlines dataset comprises 539,383 examples defined by seven attributes.

Table 4.4: Characteristics of Each Dataset.

| Dataset | No.Inst | No.Attrs | No.Cls | Noise | No.Drifts | Drift Type | Drift Points |
|--------------------|---------|----------|--------|-------|-----------|------------|--------------|
| SEA_{Binary} 10K | 100K | 3 | 2 | 10% | 1 | sudden | 10K |
| SEA_{Binary} 20K | 100K | 3 | 2 | 10% | 1 | sudden | 20K |
| SEA_{Binary} 50K | 100K | 3 | 2 | 10% | 1 | sudden | 50K |
| SEA_{Multi} 10K | 100K | 3 | 4 | 10% | 1 | sudden | 10K |
| AGR | 100K | 9 | 2 | 10% | 3 | sudden | 25K |
| Mixed | 100K | 4 | 2 | 10% | 4 | sudden | 20K |
| Sine1 | 100K | 2 | 2 | 10% | 4 | sudden | 20K |
| Sine2 | 100K | 2 | 2 | 10 | 4 | sudden | 20K |
| Wave | 100K | 21 | - | 10 | 1 | sudden | 50K |
| RBF_{GR} | 1M | 20 | 4 | 0 | 4 | gradual | not required |
| $Tree_R$ | 1M | 10 | 4 | 0 | 4 | recurring | not required |
| Airlines | 539,383 | 7 | 2 | - | - | unknown | - |
| Elec | 45,312 | 8 | 2 | - | - | unknown | - |

4.6 Experimental Setup and Evaluation

The proposed algorithm and the ones used for comparison were implemented in Java as part of the MOA framework [129]. The experiments were conducted on a machine equipped with Intel Core i7 @ 3.4 GHz with 16GB of RAM running Windows 10. To make the comparison more meaningful, we set the same parameter values for all the algorithms. For DMDDM, we use the Hoeffding tree (HT), also known as VFDT and perceptron (PER) as our incremental classifiers. In addition, we use the PH test parameters ($\lambda = 100$, $\delta = 0.1$) as proposed in [130] and the forgetting factor ($\alpha = 0.9996$). To make a fair comparison, all the compared drift detectors are run using the Hoeffding tree (HT) and perceptron with the default parameters as set in MOA (or as in the original papers).

The main objective of this work is to propose a concept drift detector that meets the predictive model's four requirements of *delay detection*, *true detection*, *detection runtime*, *memory usage* in addition to *false alarm*, *false negative* and *accuracy* which are used to evaluate the performance of the proposed concept drift detector and the other detectors. Therefore, as we use synthetic datasets, we are able to determine the location of drifts.

Fig.4.9 illustrates the DMDDM evaluation. The straight arrow represents the main stream of incoming data and the crosses show the real position of drifts. We calculate the delay of drift detection by defining a value Δ that represents the length of the acceptable drift delay. This value works as a threshold to locate the distance of the detected drift from the real position of the drift. Therefore, by considering value Δ , we describe the following measures.

- Detection Delay : The number of examples between the actual position of the drift and the detected drift.
- True Detection : A detector detects a drift occurring at time t and within $[t + \Delta]$.
- False Alarm : A detector falsely signals a drift outside $[t + \Delta]$.
- False Negative : The number of drifts which are missed by a detector from the total drifts in the incoming data.

The length of an acceptable drift delay Δ has been set to 250 on the four synthetic

datasets used in [111]. In addition to the above measures, detection runtime (in milliseconds) and memory usage (in bytes) are also considered as detailed below. Finally, the accuracy of the classifiers used by the compared drift detection methods is also considered.

- Detection Runtime : The time that is required to detect drift.
- Memory : Memory requirement.
- Accuracy : Accuracy of classifiers after drift is detected (calculated by counting the number of correct predictions). This value is displayed by MOA.

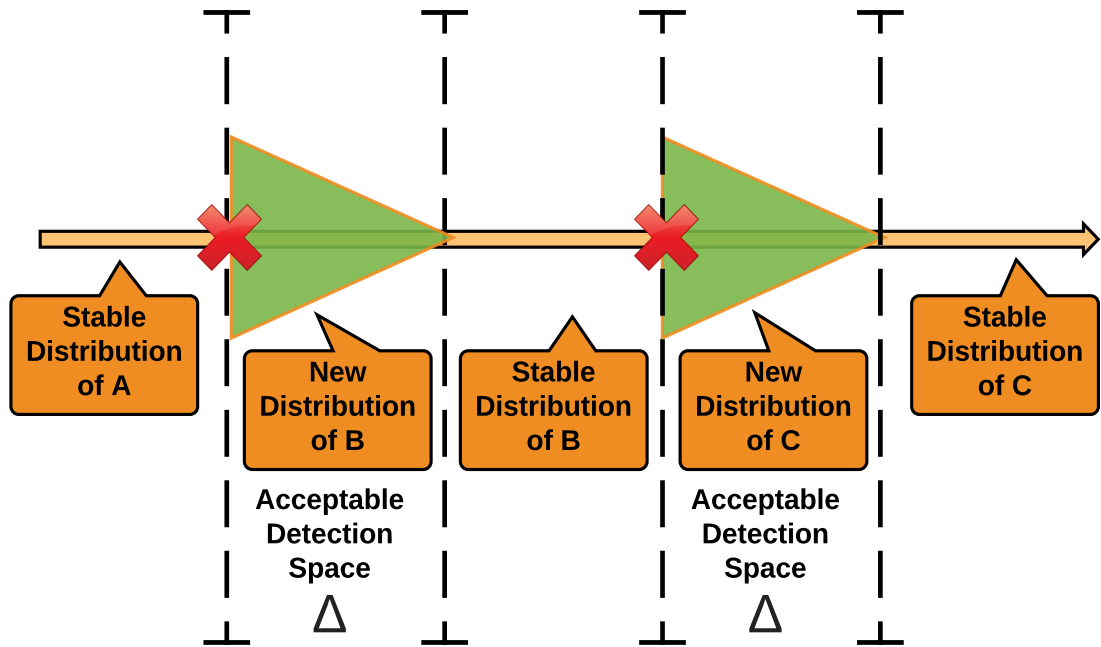


Figure 4.9: Illustration of the Evaluation

Finally, in all the experiments, incoming data are processed prequentially, which means testing the instances first and then using them to train the model. To do this, we run each detector 100 times and then average the delay detection, true drift detection, false alarm detection, and false negatives.

Intuitively, the drift detector with the lowest delay detection, runtime detection and memory usage and which is robust to noise is preferred.

4.7 Applications

The work in [86] undertook an extensive survey on real-world applications/uses where concept drift is a crucial challenge in each supervised and unsupervised learning task. In [86], the applications/uses are divided into the following four groups:

- **Observing and Management** - Observing and management frequently make use of both supervised and unsupervised learning approaches to identify irregular and adversarial behaviours on the net, in telecommunications, computer networks, and monetary transactions.
- **Private assistance and information management** - Private assistance and managing the information of applications / software include recommendation systems, organization and the categorization of textual information, and client profiling for marketing and advertising.
- **Decision Making** - Decision making contains financial as well as bio-medical software applications. The ground truth is frequently postponed, i.e., the correct answer as to whether the decision was right becomes available only after a particular time.
- **Artificial Intelligence (AI)** - AI covers a wide spectrum of systems capable of working together in dynamic environments to accomplish a job. The AI algorithms are widely used in robots, smart houses and mobile vehicles.

Its good to mentioned here that the aim of this thesis is not to solve all different practical applications but instead we are proposing techniques, so these techniques can be applicable to these applications, as long the data meet the requirement of data structure used by the proposed techniques.

4.7.1 Observing and Management

Monitoring and control applications fall into two groups, namely monitoring against adversarial actions and monitoring for management purposes.

Monitoring against Adversarial Actions

Computer Security – Intrusion detection is one of the common monitoring problems, in which undesirable entry to computer systems has to be halted. Adversarial actions are the key source of concept drift identified by intrusion detection systems. The problems and directions for intrusion detection are discussed in [131]. [132] offers ensemble methods to diagnose adversarial activities. Artificial immune systems are commonly viewed as intrusion detection [133].

Telecommunications – Adversarial behaviours also occur in the telecommunications marketplace in the form of fraud and intrusion. The mobile masquerade detection issue, from the study perspective, is very closely related to intrusion detection. The work in [134] aims to prevent adversaries from gaining unauthorized access to personal details. The sources of concept drift are adversarial behaviour attempting to get over the control and change the behaviour of respectable end users.

Finance - Financial businesses employ data mining methods to keep track of streams of transactions (e.g., credit cards, in addition to online banking) to alert for potential frauds. Both supervised and unsupervised learning methods are employed for the detection of fraudulent transactions [135]. The primary challenges might arise from incorrect information labelling, misinterpretation of reputable transactions, and extremely tall imbalanced classes (i.e., few frauds vs. legitimate actions), along with users' behavioural changes.

Monitoring for Management

Monitoring for management usually makes use of streaming data from detectors. It is characterized by very high volumes of data in addition to real-time decision making.

Transportation – Traffic management systems use data mining to identify website traffic states, e.g., automobile density in a specific place or accidents [136]. Transportation systems are dynamic due to distinct movement patterns. Website traffic patterns change seasonally and also permanently, therefore the methods need to be capable of managing concept drift.

Positioning – Concept drift could possibly be encountered in the remote sensing of fixed geographic places. Interactive road monitoring techniques help cartographers to annotate road sections in aerial photographs [137]. For this particular method, modification detection is necessary to sequentially identify and extract road segments which is non-trivial as road features vary over time. When it comes to position recognition [138] or exercise recognition [139], the powerful dynamics of environments may be the cause of concept drift.

Industrial monitoring – In industrial monitoring programs, e.g., production or service monitoring, various practices of users or changes in their behaviours may be regarded as concept drift.

4.7.2 Personal Assistance and Information Management

These apps manage and personalize the flow of information and are classified into personal assistance, customer profiling, and information management.

Personal Assistance: Personal assistance uses deal with the end-user modelling process aiming to personalize the flow of a user 's information, which is described as information filtering. Operator modelling uses are document categorization [140], net customization [141], and spam filtering [142] with regard on the present interests of a user. Changes in the user 's interests might also bring about concept drift.

Customer Profiling: The objective of client profiling is usually to sector clients based on their interests, even though the reason for concept drift might be due to changes in their interests and preferences. Advertising and marketing, social community analysis and professional recommendation methods are uses of client profiling. Adaptive learning strategies are utilized around customer segmentation , social networking sites evaluation [143], shopping bin studies, and film suggestions.

Information Management: Information management applications are used in *document organization* and *economics*.

Document Organization: The process of document organization is to extract substantial components from document streams, e.g., e-mails or news to link them with other things. Because the subjects of documents and associated vocabulary might change

over time, temporal ordering is essential. The Latent Dirichlet Allocation (LDA) model [144] for probabilistic subject modelling was built recently with a period dimension [145] assessed the dynamics of subjects for posts in the Science magazine from 1881 to 1999. The emergence, peak, and decline of subjects were detailed and the subject vocabulary representation was created. Intuitively, this is akin to including the time within the initial observation.

Economics: Concept drift appears in macroeconomic forecasts [146] and in forecasting the phases of a business cycle [147]. The data are drifting primarily as a result of a huge number of influencing factors, which makes it infeasible for the models to take these factors into account.

4.7.3 Decision Making

Financial and biomedical systems apply adaptive learning to decision-making problems.

Finance: Bankruptcy prediction or credit scoring are generally assumed to be a stationary problem [148]. The occurrence of concept drift is triggered by latent aspects, e.g., community demands and movements, which are not considered while a decision model is trained. As an initial effort, the work in [149] applied decision models for bankruptcy prediction in light of various financial factors.

Biomedicine: Biomedical systems are subject to concept drift for the adaptive dynamics of microorganisms [150]. The impact of antibiotics on a patient typically reduces as time passes since microorganisms mutate and produce resistance to antibiotics. If a patient is given an antibiotic when it is not required, resistance may develop and antibiotics may no longer be beneficial when they are essential. Changes in disease progression could be caused by changes in the drug being utilized [151]. Adaptive learning can be utilized to explore antimicrobial or antibiotic resistance in nosocomial infection, i.e., an infection which results from therapy in clinics. Resistance changes as time passes.

4.7.4 Artificial Intelligence (AI)

In artificial intelligence uses, automata communicate with the surroundings to follow the most effective actions for the following techniques. The automata can adjust their knowledge over time because they usually act in environments which are dynamic. The issue of concept drift also is known as a dynamic environment in the artificial intelligence group.

Mobile systems and robotics: Ubiquitous Knowledge Discovery (UKD) handles distributed and movable methods, functioning inside a complicated, dynamic, and unstable environment. [152] created the NextLocation framework to predict other places associated with a movable user. Adaptivity to an evolving environment was addressed in the field of robotics [153], e.g., developing a robot player for soccer.

Intelligent systems: Smart solutions learn by using streams of data in real time and adapt their models depending on the present scenario. Smart houses should be adaptive to meet users' requirements [154].

Virtual reality: Virtual reality should be empowered with mechanisms to deal with concept drift. In game design, the adversarial actions of players who are cheating might be one of the concept drift sources [155]. In games simulating fights, strategies and skills differ across different users.

4.8 Summary

The fundamental concepts of data stream mining were discussed in this chapter. We detailed the main differences between batch settings and data stream settings in Section 2.1. A batch is a finite and static dataset whereas a data stream is an infinite sequence of rapidly arriving instances. Furthermore, the concept of learning may change in the data stream setting. Section 2.1.3 explained the batch/offline and incremental/online learning modes. The offline learning mode is appropriate for a batch problem where the distribution of data is static. The online learning mode, on the other hand, processes instances one-by-one from a data stream continuously and updates the predictive model accordingly.

We formalized the data stream classification problem in Section 2.2.1 by describing the assumptions, the requirements and the life cycle of data stream mining. The assumptions are concerned with the nature of data streams and the sufficiency of the learning algorithms. Online classification algorithms have to process an example at a time and only once, using a limited amount of memory and a limited amount of time, and need to be ready to predict at any point. As presented in 2.2.4, the life cycle of data stream mining consists of three stages of processing instances, learning from instances, and utilizing the model. We described the (base) incremental learners, e.g., naive Bayes, Hoeffding trees and perceptron in Section 2.2.5.

We differentiated evolving data streams from stationary data streams by describing the concept drift phenomenon in Section 2.3.1. We applied the Bayesian decision theory to define real concept drift and virtual concept drift notions in Section 2.3.1.1. The underlying data distribution and the target concept change when real concept drift occurs. Further, we illustrated the patterns of concept drift, namely, abrupt, gradual, incremental, and re-occurring in Section 2.3.1.2.

In Section 2.3.2, we explained adaptive learning as an incremental learning solution which is able to detect concept drift and to adapt its model to the new distribution. We then compared the blind and informed adaptive algorithms. A blind solution adapts its classification model without any explicit concept drift detection, whereas an informed algorithm initially detects concept drift prior to updating its model. The adaptation may appear as either complete or partial replacement. As to the complete replacement, the outdated model is discarded and a new model is trained from scratch. We further discussed in Section 2.3.3 that adaptive learners must fulfill three requirements, namely minimum false positive and false negative rates, short detection delay, and robustness to noise. We presented three groups of drift detection methods, namely, sequential analysis-based approaches, statistical-based methods and window-based methods in Section 2.3.4.

We provided definitions of synthetic, real-world and semi-real-world data streams in Section 2.4. Then, we discussed the evaluation settings for adaptive learning. The incremental holdout and prequential evaluation procedures were described in Section 2.5.1. The holdout procedure assesses the model periodically, whereas the prequential procedure evaluates the model once a new instance arrives. We then studied classification, drift detection and resource consumption measures. Finally, we covered applications of adaptive

learning and concept drift detection in Section 2.6.

In the next chapter, we introduce the diversity measure as a new drift detection method in data streaming (DMDDM), fast reaction to sudden concept drift in the absence of class labels (DMDDM-S), and a combination of information entropy and ensemble classification to detect concept drift in data streams (EDE) for the accurate detection of concept drift with a shorter delay compared to the state-of-the-art.

Chapter 5

Diversity Measure as a New Drift Detection Method in Data Streaming for the Binary Classification Problem

This chapter comprises two of our research papers:

- “*Diversity measure as a new drift detection method in data streaming*”, published in the Knowledge-Based Systems Journal, Elsevier (Mahdi, Osama A et al., 2020). This paper introduced a novel method, the disagreement measure as a fully supervised drift detection method (DMDDM) which reacts rapidly to sudden concept drift in less time and with less memory consumption compared to state-of-the-art drift detectors. We present DMDDM in Section 5.2.
- “*Fast Reaction to Sudden Concept Drift in the Absence of Class Labels*”, published in Applied Sciences Journal, MDPI (Mahdi, Osama A et al., 2020). This paper introduced DMDDM-S as an extension of DMDDM, which reacts rapidly to sudden concept drift in the absence of class labels (semi-supervised). We present DMDDM-S in Section 5.3.

5.1 Problem Statement

As discussed in Section 4.3.1, the classification accuracy of data streams may decline due to the existence of concept drift. To keep accuracy constant, an adaptive learning mechanism which employs drift detection methods to discover these concept drifts and update their

models accordingly is needed [8]. Recall that most of the existing drift detectors, including all the aforementioned methods, evaluate the prediction results by analysing the error rate (accuracy) and its corresponding standard deviation and find the difference between the means of the sub-windows or compare the accuracy of a model with different time windows, etc., [42]. However, such methods either suffer from a high cost in terms of memory or run time or they are not fast enough in terms of detection speed. Furthermore, according to the authors of [7], the existing works make an optimistic assumption that all incoming data are labelled and the class labels are available immediately. However, such an assumption is not always valid. Therefore, a lack of class labels aggravates the problem of concept drift detection.

Consequently, having a concept drift detection method which reacts rapidly to concept drift in less time and with less memory consumption is needed to enhance the condition of adaptive learning. In addition, reacting to concept drift in the absence of class labels, where the true label of an example is not necessary, is required. Recall that we formalized these problems and their corresponding research objectives in Section 2.1.1 as follows:

- **Research Problem 1.1:** *Will the classification accuracy of data stream models reduce due to the concept drift phenomenon in evolving data ?*. Therefore, the task is to signal for concept drift in less time and with less memory consumption, keeping the accuracy of the data stream models constant.
- **Research Objective 1.1:** To propose a novel concept drift detection method which detects concept drift more accurately and with less time and memory consumption compared to the other drift detectors. To do this, we empirically compare our proposed drift detection method with the existing ones using synthetic and real-world data streams considering different performance measures, e.g., detection delay, true detection, memory and accuracy.
- **Research Problem 1.2:** *Can a lack of class labels aggravate the problem of concept drift detection in data streams?*. Therefore, in the absence of class labels, the task is to signal concept drifts in less time and with less memory consumption, keeping the accuracy of the data stream models constant.
- **Research Objective 1.2:** To propose a semi-supervised drift detection method which detects concept drift in the absence of class labels more accurately, compared to other

fully supervised drift detectors.

To address these research problems, we introduce a method called diversity measure as a new drift detection method (DMDDM). The proposed method combines one of the diversity measures, disagreement measure, known from static learning in streaming scenarios with the Page-Hinkley test and uses these calculations to detect drifts. We present DMDDM in Section 5.2. Then, we describe the fast reaction to sudden concept drift in the absence of class labels (DMDDM-S), as an extension of DMDDM in Section 5.3. We separately evaluate the proposed methods against the state-of-the-art in each section to discuss the main idea underpinning each. We experimentally show our methods react rapidly to concept drift in less time, with less memory consumption even in the absence of class labels compared to the existing ones.

5.2 Diversity Measure as a New Drift Detection Method (DMDDM)

Diversity is an important characteristic of ensembles in the standard, static data context. Measuring diversity can be useful to analyse the effectiveness of a diversity-inducing method. Therefore, many researchers consider diversity to prune a number of component classifiers [43–45]. In addition, there have been scant attempts at promoting diversity, for example, the work in [46] investigates the effect of using diversity in online ensemble learning and amends the Poisson distribution that has been used in online bagging for reactions to drift. However, the researchers used the adjusted ensemble to measure accuracy not diversity. Therefore, to the best of our knowledge, this work is the first supervised drift detection method to measure the diversity of component classifiers directly and use it as a base for drift detection. In contrast, the previous approaches used classification accuracy to detect drifts.

The diversity of component classifiers can be calculated in pairs, for example, let $X = x_1, \dots, x_n$ be a labelled data set and $y'_v = [y'_v(x_1), \dots, y'_v(x_n)]$ an n -dimensional binary vector that represents the output of a classifier h_v , such that $y'_v(x_j) = 1$, if h_v correctly predicts the class label and 0 otherwise. Table 5.1 presents (as it calls oracle outputs) all the possible outcomes for a pair of classifiers h_u and h_v , such that $h_u \neq h_v$, where N^{ab} is the

number of instances $x_j \in X$ for which $y'_u(x_j) = a$ and $y'_v(x_j) = b$. Therefore, all the probabilities of N^{ab} are as follows:

- N^{10} indicates number of examples where C_i predicts class 1 and C_j predicts class 0.
- N^{01} indicates number of examples where C_j predicts class 1 and C_i predicts class 0.
- N^{11} indicates number of examples where C_i predicts class 1 and C_j predicts class 1.
- N^{00} indicates number of examples where C_i predicts class 0 and C_j predicts class 0.

Table 5.1: The Correlation of a Pair of Classifiers (2×2)

| $h_u = h_v$ | $h_u \text{correct}(1)$ | $h_u \text{incorrect}(0)$ |
|---------------------------|-------------------------|---------------------------|
| $h_v \text{correct}(1)$ | N^{11} | N^{10} |
| $h_v \text{incorrect}(0)$ | N^{01} | N^{00} |

We now focus on the disagreement measure which was used in [156] to characterize the diversity between a base classifier and a complementary classifier, and then in [157] to measure diversity in decision forests. Formally speaking, the disagreement measure is defined as the ratio of the number of inconsistent decisions over the total number of observations. In other words, this measure is defined based on the intuition that two diverse classifiers perform differently on the same training data. Also, we can say it is the ratio between the number of observations for which one classifier is correct and the other is incorrect. Consequently, the diversity measure reflects the variety of reactions of classifiers to stream changes. The disagreement measure is probably the most intuitive measure of diversity between a pair of classifiers [51]. Consequently, the diversity between two base classifiers (h_u and h_v) using the disagreement measure is calculated by Equation.5.1:

$$D_{u:v} = N^{10} + N^{01} \quad (5.1)$$

According to [130] and as mentioned earlier, the PH test considers a variable m_T which measures the accumulated differences between observed values e (error estimates). Two basic approaches to calculate these values prequentially (incrementally with forgetting) are *fading factors* and *sliding windows*.

The fading factors approach is used in this work. Over time, the fading factors remove outdated information by multiplying the former summary by a factor and then

adding a new value calculated using the incoming data. Other approaches use the sliding windows to hold at each time point a set of d most current examples in order to limit the number of analyzed examples. Therefore, we calculate the *fading sum* $S_{x,\alpha}$ and *fading increment* N_α at time t from a stream of objects x , as follows:

$$S_{x:\alpha}(t) = x^t + \alpha \times S_{x:\alpha}(t - 1) \quad (5.2)$$

$$N_\alpha(t) = 1 + \alpha \times N_\alpha(t - 1) \quad (5.3)$$

Then, the *fading average* is computed at observation i :

$$M_\alpha(t) = \frac{S_{x:\alpha}(t)}{N_\alpha(t)} \quad (5.4)$$

According to [130], the fading factors approach uses less time and memory compared with the sliding windows approach. Therefore, in this work, the observed value with the fading factor is the studied diversity measure instead of error rate. By applying the value of diversity from Equation.5.5 in Equation.5.6, we have the following equations.

$$S_{u:v:\alpha}(t) = D_{u:v} + \alpha \times S_{u:v:\alpha}(t - 1) \quad (5.5)$$

$$M_\alpha(t) = \frac{S_{u:v,\alpha}(t)}{N_\alpha(t)} \quad (5.6)$$

Equation.5.6 can be used with the PH test to monitor the diversity of a pair of classifiers. the PH test signals a drift whenever the predictions of components (h_u and h_v) start to disagree in an unusual way. In other words, the PH test detects a significant increase in diversity.

Equation.5.7 is used to calculate the cumulative difference m_T between the observed values from Equation.5.7 and their mean till the current moment t :

$$m_T = \sum_{t=1}^T (x_t - x'_T - \delta) \quad (5.7)$$

where $x'_T = \frac{1}{T} \sum_{t=1}^T x_t$ and δ correspond to the magnitude of changes that are allowed. The minimum value of this variable is also computed via Equation.5.8:

$$M_T = \min(m_t, t = 1 \dots T). \quad (5.8)$$

As a final step, the test monitors the difference between M_T and m_T as follows:

$$PH_T = m_T - M_T \quad (5.9)$$

When this difference is greater than a given threshold (λ), a drift is signaled.

Algorithm 3: Pseudocode of the Diversity Measure As a New Drift Detection Method (DMDDM)

Require: S : data stream of examples (labelled),
Forgetting factor $\alpha : 0 < \alpha < 1$
Admissible change: $\delta = 0.1$,
Drift threshold: $\lambda = 100$
 $M_T : 1.0D$
Result: $\text{Drift} \in \{\text{TRUE}, \text{FALSE}\}$

```

1 for each example  $x^t \in S$  do
2    $C_v\text{prediction} = \text{get prediction using } x^t$ ;
3    $C_u\text{prediction} = \text{get prediction using } x^t$ ;
4   if  $C_v\text{prediction} = 0.0$  and  $C_u\text{prediction} = 1.0$  then
5      $b++$ ;
6   end
7   if  $C_u\text{prediction} = 0.0$  and  $C_v\text{prediction} = 1.0$  then
8      $c++$ ;
9   end
10   $\text{Disagreement}, D_{u,v} = b + c/L$ ;
11   $S_{u:v,\alpha}(t) = D_{u:v} + \alpha \times S_{u:v,\alpha}(t - 1)$ ;
12   $N_\alpha(t) = 1 + \alpha \times N_\alpha(t - 1)$ ;
13   $M_\alpha(t) = \frac{S_{u:v,\alpha}(t)}{N_\alpha(t)}$ ;
14   $\text{SumDiversity} = \text{SumDiversity} + M_\alpha(t)$ ;
15   $m_T = (m_T + M_\alpha(t) - (\text{SumDiversity}/\text{instancesSeen}) - \delta)$ ;
16   $M_T = \text{Min}(M_T, m_T)$ ;
17   $\text{PH}_{test} = m_T - M_T$ ;
18  if  $\text{PH}_{test} > \lambda$  then
19    Return TRUE
20  else
21    Return FALSE
22  end
23  incrementally train  $C_v$  and  $C_u$  with  $x^t$ ;
24 end

```

The DMDDM approach is presented in Algorithm 3 and its framework is shown in Fig.5.1. First, the algorithm processes each example from the data stream and obtains the predictions for a pair of classifiers on each example line (lines 1-3). Then the algorithm builds the oracle output table, as mentioned in Table 5.1. From Table 5.1, we need to find the two cases (N^{10} and N^{01}) where the pair of classifiers performs differently on the same training data x^t . Once we observe this disagreement, we count the number of observations on which the classifier is correct and incorrect, shown in lines 4 to 9, respectively. These steps represent the first phase of the DMDDM framework (prediction phase). Phase two (concept drift detection phase) uses these observed predictions using a disagreement measure with the PH test to detect drifts.

In line 10, we apply the disagreement measure (Equation.5.1) by aggregating these observations and dividing it by the number of component classifiers. Then, the fading factor approach is applied from lines 11 to 13 (Equation.5.5, Equation.5.3 and Equation.5.6). In lines 11 and 12, the fading sum and fading increment are calculated, respectively.

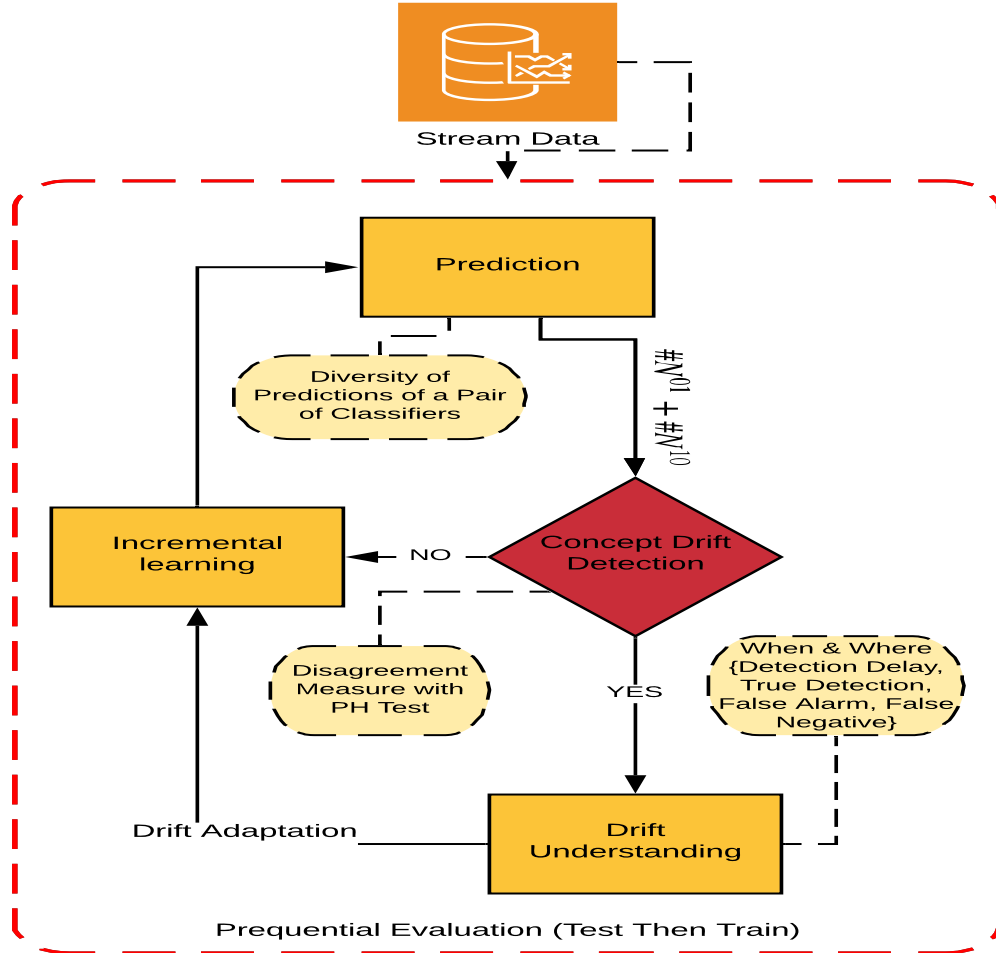


Figure 5.1: Framework of DMDDM

With the fading sum and fading increment, we use the value of diversity from line 10 as the observed value instead of the error estimates that were used in the original PH test. In line 13, the fading average is calculated. To monitor the diversity of a pair of classifiers, from lines 14 to 22, the PH test considers a variable m_T , which measures the accumulated difference between the observed value of diversity and their mean up to the current moment. After each monitoring, there is a step for checking and signaling for a drift, if the value of variation between the current m_T and the smallest value up to this moment M_T is larger than a predefined threshold. If there is a drift, phase three (drift understanding phase) starts, which evaluates the detected drift in terms of delay detection, true detection, false alarm and false negative. The evaluation method is explained in Section 4.6. Finally, whether there is a drift or not, phase four (incremental learning phase) / line 23 is used to incrementally train

the model and keep it up-to-date.

5.2.1 Experimental Evaluation

This section details the results of the experiments and the analyses of each drift detector. The experiments show the detection delay, true detection, false alarm, false negative, detection runtime, memory usage and accuracy in the presence of 10% noise over the five tested datasets. We highlight our and the best results in bold. In addition, Table 5.2 shows the abbreviations of the measures that are used in Tables 5.3-5.6.

Table 5.2: Abbreviations of the Names of the Measures.

| Name of Measure | Abbreviation |
|--------------------------------|--------------|
| Average Delay Detection | ADD |
| Average True Detection | ATD |
| Average False Alarm | AFA |
| Average False Negative | AFN |
| Average Detection Runtime (MS) | DRMS |
| Average Memory Usage (Byte) | MUB |
| Average Accuracy | ACC |

Tables 5.3 (a) and (b) show the results of the experiments using the Mixed dataset. Note that DMDDM is able to detect drift in the shortest time compared with the other detectors. In addition, the results show that the method which detected drift the fastest was DMDDM, followed by Wtest, MDDMs, FHDDM, ADWIN, Atest and SeqDrift2 in ascending order of delay detection. Other methods such as PHtest and DDM are only able to detect some drifts. In relation to the computational time and memory usage, it is clearly seen that SeqDrift2 and ADWIN have the highest memory usage and the longest runtime was recorded by ADWIN, as shown in Table 5.3 (b). In addition to having the lowest delay detection, DMDDM has the lowest computational time and memory usage.

Table 5.4, Table 5.5 and Table 5.6 (a) and (b) show the results of the experiments using the Sine1, Agrawal and SEA datasets, respectively. The results of the Agrawal and the SEA datasets are similar. For example, for both datasets, DMDDM has the lowest average delay detection followed by MDDMs, Wtest, FHDDM and Atest, respectively. In addition, these successfully detect all three drifts in the Agrawal dataset, while DMDDM was the only one able to detect the single drift in the SEA dataset. On the other hand, for the Sine1 dataset, the four top performing methods for delay detection and average

real drift are Wtest, DMDDM, MDDMs and FHDDM. Regarding computational time and memory usage, SeqDrift2 and ADWIN require more memory than the other detectors, while the longest runtime belongs to ADWIN. The reason for this is that SeqDrift2 and ADWIN consume more memory for storing the prediction results in the sliding windows or repositories. Furthermore, SeqDrift2 and ADWIN also use more computational time due to sub-window compression or reservoir sampling procedures.

Regarding false alarms and accuracy, DMDDM has better accuracy on the Mixed, Sine1 and SEA datasets, and was among the best on the Agrawal dataset. In terms of false alarms, DMDDM has the highest readings due to the sensitivity of the parameter values of DMDDM. For example, as shown in Fig.5.2 and Fig.5.3, increasing λ (100-200-300) entails fewer false alarms, but might delay change or miss detection, resulting in a trade-off between false alarms and delay change detection. Even with this increase, DMDDM is still among the best in terms of delay detection and lowest in terms of time and memory consumption. Furthermore, despite this trade-off, the accuracy of DMDDM is still constant over the sensitivity of parameters, as shown in Fig.5.4. As mentioned in [130], the trade-off exists due to the use of the PH test; one possible solution to overcome this trade-off is to control the diversity of the disagreement method using two fading factors instead of one, as used in this work. Such a solution is left for future consideration.

In the third experiment, we use a real-world dataset. As mentioned earlier, in real-world datasets, the ground truth for drifts is not available, thus, we are not able to measure the average delay detection and the real drift in this section. Consequently, we only evaluate memory usage and classification accuracy. Table 5.7 summarizes the results of the experiments on the aforementioned dataset. For the Electricity dataset, the classification accuracy of DMDDM is among the highest. On the other hand, in terms of memory usage, DMDDM is the lowest compared to the other methods.

Table 5.3: Results of Mixed dataset with 10% noise

| (a) | | | | | |
|------------|-----------|--------------|----------|------|----------|
| Classifier | Detectors | ADD | ATD | AFA | AFN |
| HT&PER | DMDDM | 35.81 | 4 | 7.62 | 0 |
| HT | FHDDM | 48.52 | 4 | 0 | 0.01 |
| | DDM | 214.65 | 1.85 | 0.18 | 2.16 |
| | ADWIN | 64.56 | 4 | 0.84 | 0 |
| | Wtest | 36.2 | 3.99 | 0 | 0.01 |
| | PH Test | 240.97 | 1.04 | 0 | 2.96 |
| | SeqDrift | 200 | 4 | 0 | 0 |
| | Atest | 71.54 | 4 | 0 | 0 |
| | MDDM-A | 43.81 | 3.99 | 0 | 0.01 |
| | MDDM-E | 41.45 | 3.99 | 0 | 0.01 |
| | MDDM-G | 41.86 | 3.99 | 0 | 0.01 |
| PER | FHDDM | 49.01 | 4 | 0 | 0 |
| | DDM | 232.51 | 1.57 | 0.04 | 2.43 |
| | ADWIN | 66.48 | 4 | 1.61 | 0 |
| | Wtest | 36.16 | 4 | 0.01 | 0 |
| | PH Test | 247.36 | 0.22 | 0 | 3.78 |
| | SeqDrift | 200 | 4 | 0 | 0 |
| | Atest | 92.13 | 3.84 | 0 | 0.16 |
| | MDDM-A | 43.9 | 4 | 0 | 0 |
| | MDDM-E | 42.13 | 3.5 | 0 | 0.009 |
| | MDDM-G | 42.14 | 3.5 | 0 | 0.009 |

| (b) | | | | |
|------------|-----------|-------------|------------|--------------|
| Classifier | Detectors | DRMS | MUB | ACC |
| HT&PER | DMDDM | 0.68 | 168 | 84.16 |
| HT | FHDDM | 7.4 | 1048 | 84.06 |
| | DDM | 1.95 | 472 | 81.58 |
| | ADWIN | 50.91 | 2280.48 | 82.31 |
| | Wtest | 6.77 | 1624 | 84.00 |
| | PH Test | 1.17 | 1240 | 80.44 |
| | SeqDrift | 4.2 | 80824.24 | 83.46 |
| | Atest | 12 | 1176 | 84.04 |
| | MDDM-A | 40.55 | 1336 | 84.06 |
| | MDDM-E | 28.49 | 1288 | 84.05 |
| | MDDM-G | 21.25 | 1344 | 84.05 |
| PER | FHDDM | 6.43 | 1048 | 82.25 |
| | DDM | 1.45 | 472 | 82.60 |
| | ADWIN | 53.38 | 2321.44 | 82.59 |
| | Wtest | 6.99 | 1624 | 82.54 |
| | PH Test | 0.99 | 1240 | 82.61 |
| | SeqDrift | 3.9 | 81549.36 | 82.63 |
| | Atest | 11.71 | 1176 | 82.58 |
| | MDDM-A | 41.54 | 1336 | 82.61 |
| | MDDM-E | 27.88 | 1288 | 82.38 |
| | MDDM-G | 20.72 | 1344 | 82.61 |

Table 5.4: Results of Sine1 dataset with 10% noise

| (a) | | | | | |
|------------|-----------|--------------|----------|------|------|
| Classifier | Detectors | ADD | ATD | AFA | AFN |
| HT&PER | DMDDM | 39.05 | 4 | 9.01 | 0 |
| HT | FHDDM | 48.14 | 4 | 0 | 0 |
| | DDM | 175.28 | 2.88 | 0.35 | 1.12 |
| | ADWIN | 63.84 | 4 | 0.26 | 0 |
| | Wtest | 35.69 | 4 | 0 | 0 |
| | PH Test | 238.37 | 1.31 | 0 | 2.69 |
| | SeqDrift | 200 | 4 | 0 | 0 |
| | Atest | 57.04 | 4 | 0.01 | 0 |
| | MDDM-A | 43.03 | 4 | 0 | 0 |
| | MDDM-E | 41.02 | 4 | 0 | 0 |
| | MDDM-G | 40.99 | 4 | 0 | 0 |
| PER | FHDDM | 47.54 | 4 | 0 | 0 |
| | DDM | 156.43 | 3.98 | 0.13 | 0.02 |
| | ADWIN | 63.84 | 4 | 1.17 | 0 |
| | Wtest | 35.66 | 4 | 0 | 0 |
| | PH Test | 249.76 | 0.06 | 0 | 3.94 |
| | SeqDrift | 200 | 4 | 0 | 0 |
| | Atest | 61.59 | 4 | 0 | 0 |
| | MDDM-A | 42.80 | 4 | 0 | 0 |
| | MDDM-E | 40.82 | 4 | 0 | 0 |
| | MDDM-G | 40.84 | 4 | 0 | 0 |

| (b) | | | | |
|------------|-----------|-------------|------------|--------------|
| Classifier | Detectors | DRMS | MUB | ACC |
| HT&PER | DMDDM | 0.58 | 168 | 87.99 |
| HT | FHDDM | 8.26 | 1048 | 87.92 |
| | DDM | 2.5 | 472 | 86.348 |
| | ADWIN | 50.24 | 2316.80 | 87.89 |
| | Wtest | 6.93 | 1624 | 87.92 |
| | PH Test | 1.19 | 1240 | 83.64 |
| | SeqDrift | 3.71 | 82746.85 | 87.89 |
| | Atest | 11.59 | 1176 | 87.92 |
| | MDDM-A | 38.49 | 1336 | 87.90 |
| | MDDM-E | 28.65 | 1288 | 87.87 |
| | MDDM-G | 20.31 | 1344 | 87.86 |
| PER | FHDDM | 7.07 | 1048 | 88.15 |
| | DDM | 1.47 | 472 | 88.15 |
| | ADWIN | 52.87 | 2313.51 | 88.15 |
| | Wtest | 6.70 | 1624 | 88.16 |
| | PH Test | 1.13 | 1240 | 85.8 |
| | SeqDrift | 3.46 | 83244.00 | 88.14 |
| | Atest | 11.29 | 1176 | 88.15 |
| | MDDM-A | 41.07 | 1336 | 88.15 |
| | MDDM-E | 28.33 | 1288 | 88.15 |
| | MDDM-G | 20.77 | 1344 | 87.7 |

Table 5.5: Results of AGR dataset with 10% noise

| (a) | | | | | |
|------------|-----------|--------------|----------|------|------|
| Classifier | Detectors | ADD | ATD | AFA | AFN |
| HT&PER | DMDDM | 27.12 | 3 | 8.54 | 0 |
| HT | FHDDM | 67.79 | 3 | 0 | 0 |
| | DDM | 233.99 | 1 | 0 | 2 |
| | ADWIN | 91.09 | 3 | 5 | 0 |
| | Wtest | 63.34 | 2.79 | 0 | 0.21 |
| | PH Test | 250 | 0 | 0 | 3 |
| | SeqDrift | 200 | 3 | 0 | 0 |
| | Atest | 70.80 | 3 | 0 | 0 |
| | MDDM-A | 57.82 | 3 | 0 | 0 |
| | MDDM-E | 56.81 | 3 | 0 | 0 |
| | MDDM-G | 56.81 | 3 | 0 | 0 |
| PER | FHDDM | 129.66 | 2 | 0 | 1 |
| | DDM | 250 | 0 | 0 | 3 |
| | ADWIN | 112.10 | 3 | 9.45 | 0 |
| | Wtest | 105.24 | 1.97 | 1.92 | 1.03 |
| | PH Test | 250 | 0 | 0 | 3 |
| | SeqDrift | 200 | 3 | 0 | 0 |
| | Atest | 250 | 0 | 0 | 3 |
| | MDDM-A | 121.32 | 2 | 0 | 1 |
| | MDDM-E | 112.32 | 2 | 0 | 1 |
| | MDDM-G | 112.32 | 2 | 0 | 1 |

| (b) | | | | |
|------------|-----------|------------|------------|-------------|
| Classifier | Detectors | DRMS | MUB | ACC |
| HT&PER | DMDDM | 1.8 | 168 | 87 |
| HT | FHDDM | 8.06 | 1048 | 87.09 |
| | DDM | 2.39 | 472 | 76.35 |
| | ADWIN | 52.12 | 2288.16 | 88.99 |
| | Wtest | 6.67 | 1624 | 83.75 |
| | PH Test | 1.61 | 1240 | 71.4 |
| | SeqDrift | 4.13 | 97567.76 | 89.3 |
| | Atest | 12.38 | 1176 | 89.2 |
| | MDDM-A | 40.63 | 1336 | 87.1 |
| | MDDM-E | 27.50 | 1288 | 87.1 |
| | MDDM-G | 21.80 | 1344 | 87.1 |
| PER | FHDDM | 8.46 | 1048 | 38.8 |
| | DDM | 2.83 | 472 | 38.80 |
| | ADWIN | 51.63 | 2392.40 | 38.80 |
| | Wtest | 7.7 | 1624 | 38.80 |
| | PH Test | 1.99 | 1240 | 38.80 |
| | SeqDrift | 4.44 | 106543.28 | 38.80 |
| | Atest | 12.80 | 1176 | 38.80 |
| | MDDM-A | 43.97 | 1336 | 38.80 |
| | MDDM-E | 26.27 | 1288 | 38.80 |
| | MDDM-G | 21.20 | 1344 | 38.80 |

Table 5.6: Results of SEA dataset with 10% noise

| (a) | | | | | |
|------------|-----------|--------------|----------|------|------|
| Classifier | Detectors | ADD | ATD | AFA | AFN |
| HT&PER | DMDDM | 81.38 | 1 | 1.92 | 0 |
| HT | FHDDM | 219.28 | 0.26 | 0 | 0.73 |
| | DDM | 248.05 | 0.01 | 0 | 0.98 |
| | ADWIN | 244.82 | 0.16 | 0.07 | 0.83 |
| | Wtest | 235.03 | 0.11 | 0 | 0.88 |
| | PH Test | 250 | 0 | 0 | 1 |
| | SeqDrift | 229.29 | 0.41 | 0 | 0.58 |
| | Atest | 241.56 | 0.12 | 0 | 0.87 |
| | MDDM-A | 209.29 | 0.33 | 0 | 0.66 |
| | MDDM-E | 209.76 | 0.32 | 0 | 0.67 |
| | MDDM-G | 209.76 | 0.32 | 0 | 0.67 |
| PER | FHDDM | 207.32 | 0.37 | 0 | 0.62 |
| | DDM | 250 | 0 | 0 | 1 |
| | ADWIN | 248.18 | 0.02 | 0.05 | 0.97 |
| | Wtest | 198.43 | 0.40 | 0 | 0.59 |
| | PH Test | 250 | 0 | 0 | 1 |
| | SeqDrift | 245.45 | 0.09 | 0 | 0.90 |
| | Atest | 242.17 | 0.07 | 0 | 0.93 |
| | MDDM-A | 200.26 | 0.40 | 0 | 0.59 |
| | MDDM-E | 193.95 | 0.43 | 0 | 0.56 |
| | MDDM-G | 194.89 | 0.42 | 0 | 0.57 |

| (b) | | | | |
|------------|-----------|-------------|------------|--------------|
| Classifier | Detectors | DRMS | MUB | ACC |
| HT&PER | DMDDM | 0.48 | 168 | 89.20 |
| HT | FHDDM | 8.93 | 1048 | 89.13 |
| | DDM | 2.46 | 472 | 89.11 |
| | ADWIN | 58.55 | 2576.44 | 89.08 |
| | Wtest | 7.18 | 1624 | 89.12 |
| | PH Test | 1.40 | 1240 | 89.08 |
| | SeqDrift | 4.12 | 341247.5 | 89.09 |
| | Atest | 12.53 | 1176 | 89.14 |
| | MDDM-A | 34.49 | 1336 | 88.34 |
| | MDDM-E | 28.05 | 1288 | 89.13 |
| | MDDM-G | 21.26 | 1344 | 89.13 |
| PER | FHDDM | 7.32 | 1048 | 82.96 |
| | DDM | 1.23 | 472 | 82.12 |
| | ADWIN | 55.54 | 2466.48 | 82.11 |
| | Wtest | 6.16 | 1624 | 81.85 |
| | PH Test | 0.88 | 1240 | 82.07 |
| | SeqDrift | 3.10 | 183948.84 | 82.45 |
| | Atest | 11.12 | 1176 | 82.55 |
| | MDDM-A | 37.82 | 1336 | 83.48 |
| | MDDM-E | 28.34 | 1288 | 83.6 |
| | MDDM-G | 21.16 | 1344 | 83.41 |

Table 5.7: The results of the Electricity dataset

| Classifier | Detects | Accuracy | Memory |
|------------|----------|--------------|------------|
| HT&PRE | DMDDM | 84.12 | 168 |
| HT | FHDDM | 84.03 | 1048 |
| | DDM | 85.39 | 472 |
| | ADWIN | 83.15 | 1821.93 |
| | Wtest | 85.08 | 1624 |
| | PH Test | 82.01 | 1240 |
| | SeqDrift | 82.76 | 10484.15 |
| | Atest | 85.76 | 1176 |
| | MDDM-A | 84.16 | 1336 |
| | MDDM-E | 84.74 | 1288 |
| | MDDM-G | 84.65 | 1344 |
| PRE | FHDDM | 77.4 | 1048 |
| | DDM | 78.96 | 472 |
| | ADWIN | 78.12 | 2336.68 |
| | Wtest | 76.47 | 1624 |
| | PH Test | 79.2 | 1240 |
| | SeqDrift | 78.26 | 41475.58 |
| | Atest | 77.05 | 1176 |
| | MDDM-A | 77.05 | 1336 |
| | MDDM-E | 77.25 | 1288 |
| | MDDM-G | 77.25 | 1344 |

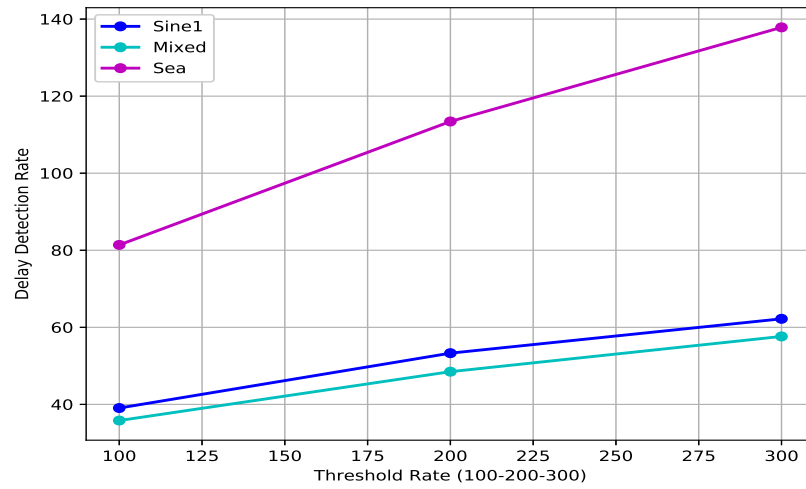


Figure 5.2: Delay Detection Rate

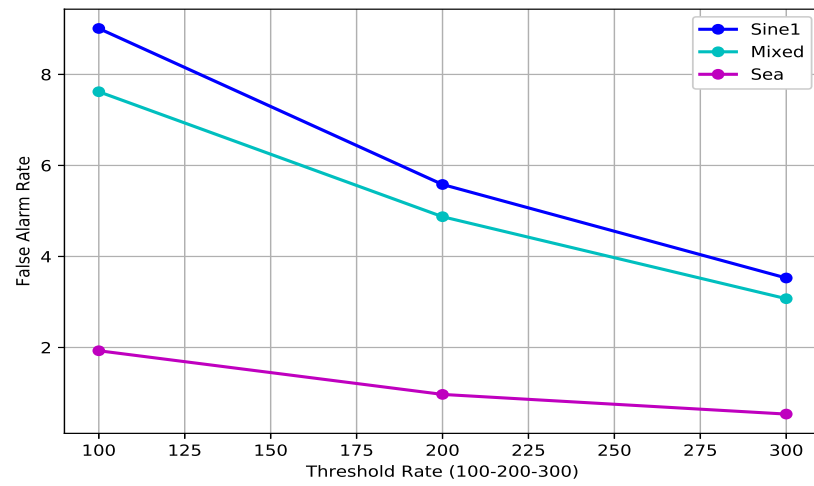


Figure 5.3: False Alarm Rate

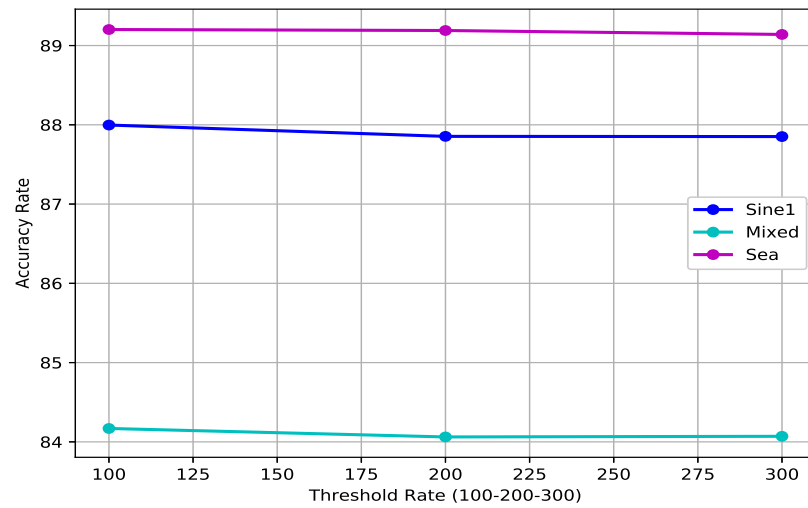


Figure 5.4: Accuracy Rate

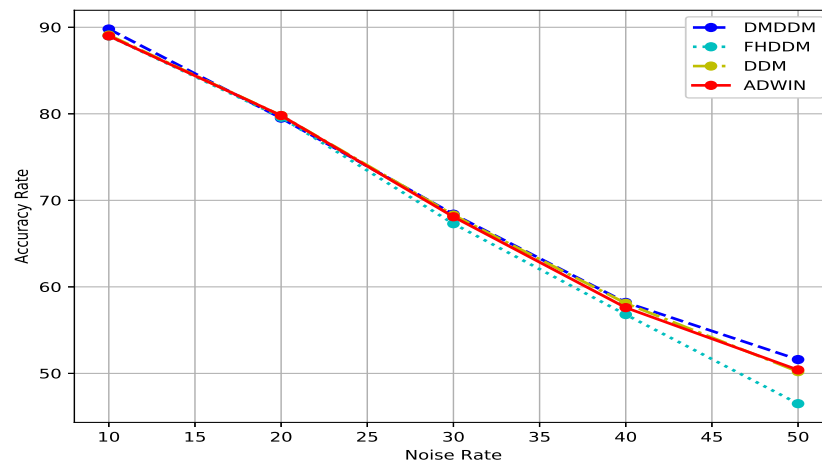


Figure 5.5: Effect of Noise Rate

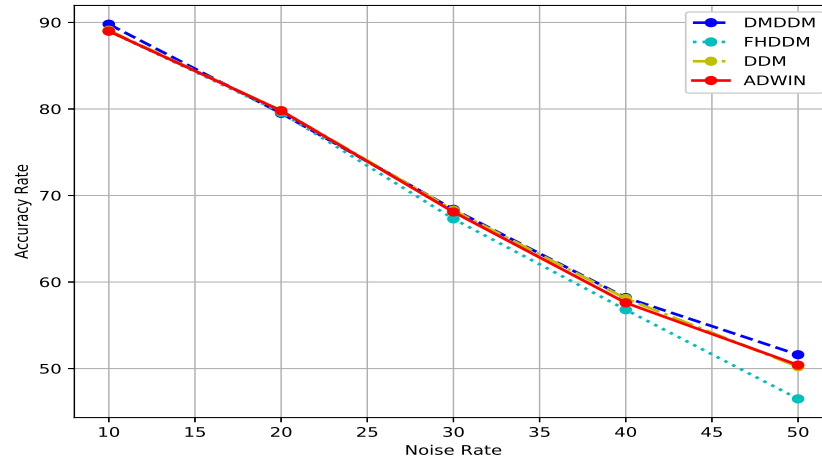


Figure 5.6: Effect of Noise Rate

In addition to having the lowest delay detection, it is clearly seen from Tables 5.2-5.5 that DMDDM has the lowest computational time and memory usage. The main reason for this is because DMDDM maintains a lower number of variables than the others which results in less memory usage and less execution runtime to update these variables compared to the others which use sliding windows which means more memory and time is used to detect drifts. In relation to time complexity, most of the drift detectors including ours have constant time complexity. The exceptions are ADWIN and SeqDrift which have logarithmic complexity.

On the other hand, the noise ratio in the literature and also in this work is always 10%. We believe this ratio represents the normal or acceptable ratio for any dataset. Without doubt, increasing the noise ratio affects the performance of any model, as demonstrated in the experiments. Fig.5.5 shows the performance of DMDDM and several others. It is clearly seen that the performance of each detector dropped significantly when the noise ratio increased from 10% to 50%.

Furthermore, the certainty of data is one of the assumptions that is used in this field, however detecting drifts is the main concern because of its impact on performance. Due to privacy protection, data loss, network errors and so on, it is very common to have uncertainty during data streaming. However, all the works in the literature including this work applied the assumption of the certainty of data.

Consequently, considering these assumptions in addition to the size of the data, being the era of big data, opens the door for further research questions.

5.3 Fast Reaction to Sudden Concept Drift in the Absence of Class Labels (DMDDM-S)

We introduced an extension of DMDDM in (Mahdi , Osama et al., 2020), called the Fast Reaction to Sudden Concept Drift in the Absence of Class Labels (DMDDM-S). DMDDM-S extends the idea of DMDDM when the class labels of the incoming data stream are not available with the concept drift problem.

Informally, for a pair of classifiers in a binary classification problem, each component gives one of two possible predictions for each example, either 0 (negative class) or 1 (positive class). If we take the predictions of each component, we can calculate the disagreement between these predictions. Therefore, it is about the disagreement between the predictions of pairs of component classifiers, regardless of the true labels.

First, we propose DMDDM to detect concept drift in a semi-supervised environment (DMDDM-S). The main advantage of calculating diversity is that for binary classification, the true label of an example is not necessary to determine whether components disagree. Second, we apply the proposed drift detector to detect sudden drifts when the class labels of incoming data are not available. To the best of our knowledge, this is the first work that uses such a method to detect concept drift. Third, we adopt k-prototype clustering as a solution to label the unlabelled data and use the newly labelled data along with the labelled ones to retrain the model to be consistent with the current concept. We show that when only 50% of data is labelled, the proposed drift detector can detect drifts faster and with minimal consumption in terms of memory and run time than the existing methods which use 100% of labelled data.

The DMDDM-S approach is presented in Algorithm 4 and its framework is shown in Fig5.7. First, the algorithm processes each example from the data stream and obtains the predictions for a pair of classifiers on each example line (lines 1–3). Then, the algorithm builds the outputs, as shown Table 5.1. As shown Table 5.1, we need to find the two cases (N^{10} and N^{01}) where the pair of classifiers performs differently. Once we observe this disagreement, we count the number of observations on which one classifier is correct and the other is incorrect, shown in lines 4 to 9, respectively. These steps represent the first phase of the DMDDM-S framework (prediction phase).

Phase two (concept drift detection phase) uses these observed predictions using a disagreement measure with the PH test to detect drifts. In line 10, we apply the disagreement measure (Equation. 5.1) by aggregating these observations and dividing it by the number of component classifiers. Then, the fading factor approach is applied from lines 11 to 13 (Equation. 5.5, Equation. 5.3 and Equation. 5.6). In lines 11 and 12, the fading sum and fading increment are calculated, respectively. With the fading sum and fading increment, we use the value of diversity from line 10 as the observed value instead of the error estimates that were used in the original PH test. In line 13, the fading average is calculated. To monitor the diversity of a pair of classifiers from lines 14 to 19, the modified PH test considers a variable m_T , which measures the accumulated difference between the observed value of diversity and their mean up to the current moment (Equation. 5.7). After each observation, the PH test checks whether the difference between the current m_T and the smallest value up to this moment M_T is greater than a given threshold (Equation. 5.9). If the difference exceeds the predefined threshold, a drift is signalled. When there is a drift, we need to label the current unlabelled data to use them to retrain the model and keep it consistent.

Therefore, line 20 combines the two windows of the labelled (W_{ld}) and unlabelled data (W_{uld}) and sends the result to K-prototype clustering in order to label the unlabelled data in line 21, the third phase of the DMDDM-S framework. When there is a drift, phase four (drift understanding) starts, which evaluates the detected drift in terms of delay detection, true detection, false alarm and false negative. The evaluation method is explained in Figure 4.10.

Then, lines (22–24) incrementally train the current model and keep it up-to-date with the newly labelled data (N_{ld}). Lines (26–39) check and handle the availability of each class, phase five (class is missing?). With each observation of x^t and whether there is a drift or not, we need to check if x^t is labelled or not. If the class of x^t is missing, first line 27 checks if (W_{uld}) reached the pre-defined probability, removes the oldest instance in (W_{uld}) and adds the newest one to (W_{uld}); otherwise, x^t is added to (W_{uld}) directly in line 30. On the other hand, if the class is not missing, line 33 will incrementally train the current model with the current labelled instance. Finally, lines (35–39) check the dynamic window of labelled data (W_{ld}), and If the size of this window reaches the pre-defined probability, we start removing the oldest one and add the newest labelled data; otherwise, x^t is added to (W_{ld}).

Algorithm 4: Pseudocode of Diversity Measure as a Drift Detection Method in a Semi- Supervised Environment (DMDDM-S)

Require: S : data stream of examples (50% labelled),
 Forgetting factor α : $0 < \alpha < 1$
 Admissible change: $\delta = 0.1$,
 Drift threshold: $\lambda = 100$
 Base Classifiers (Hoeffding Tree, Perceptron): $L = 2$
 M_T : 1.0D
 $|W_{ld}|=100$
 $|W_{uld}|=100$
 $b, c = 0$

Result: Drift $\in \{\text{TRUE}, \text{FALSE}\}$

```

1  for each example  $x^t \in S$  do
2       $C_v$  prediction = get prediction using  $x^t$ ;
3       $C_u$  prediction = get prediction using  $x^t$ ;
4      if  $C_v$  prediction = 0.0 and  $C_u$  prediction = 1.0 then
5          b++;
6      end
7      if  $C_u$  prediction = 0.0 and  $C_v$  prediction = 1.0 then
8          c++;
9      end
10     Disagreement,  $D_{u,v} = b + c/L$ ;
11      $S_{u,v,\alpha}(t) = D_{u,v} + \alpha \times S_{u,v,\alpha}(t - 1)$ ;
12      $N_\alpha(t) = 1 + \alpha \times N_\alpha(t - 1)$ ;
13      $M_\alpha(t) = \frac{S_{u,v,\alpha}(t)}{N_\alpha(t)}$ ;
14     SumDiversity = SumDiversity +  $M_\alpha(t)$ ;
15      $m_T = (m_T + M_\alpha(t) - (\text{SumDiversity}/\text{instancesSeen}) - \delta)$ ;
16      $M_T = \min(M_T, m_T)$ ;
17      $PH_{test} = m_T - M_T$ ;
18     if  $PH_{test} > \lambda$  then
19         Return TRUE ;
20          $W_{ld} \leftarrow W_{ld} \cup W_{uld}$  ;
21          $N_{ld} = K - \text{PrototypeClustering}(W_{ld})$  ;
22         for each example  $x^t \in N_{ld}$  do
23             train classifier  $C_v$  and  $C_u$  using  $x^t$ 
24         end
25     end
26     if class is missing then
27         if  $|W_{uld}| = |uld|$  then
28             remove oldest instance in  $W_{uld}$  and add  $x^t$  to  $W_{uld}$ 
29         else
30              $W_{uld} \leftarrow W_{uld} \cup x^t$ 
31         end
32     else
33         Incrementally train classifier  $C_v$  and  $C_u$  using  $x^t$ 
34     end
35     if  $|W_{ld}| = |ld|$  then
36         remove oldest instance in  $W_{ld}$  and add  $x^t$  to  $W_{ld}$ 
37     else
38          $W_{ld} \leftarrow W_{ld} \cup x^t$ 
39     end
40 end
    
```

5.3.1 Experimental Evaluation

The performance of DMDDM-S is evaluated against various drift detectors, namely, DDM, EDDM, FHDDM, PH Test, SEED, STEPD and RDDM. The main reasons for choosing these supervised methods over semi-supervised methods are: (i) the limited access to the source code of some works, and (ii) we want to show the strengths of DMDDM-S, which can detect drifts faster when only 50% of the data is labelled with minimal consumption in terms of memory and run time than the existing methods that use 100% of labelled data.

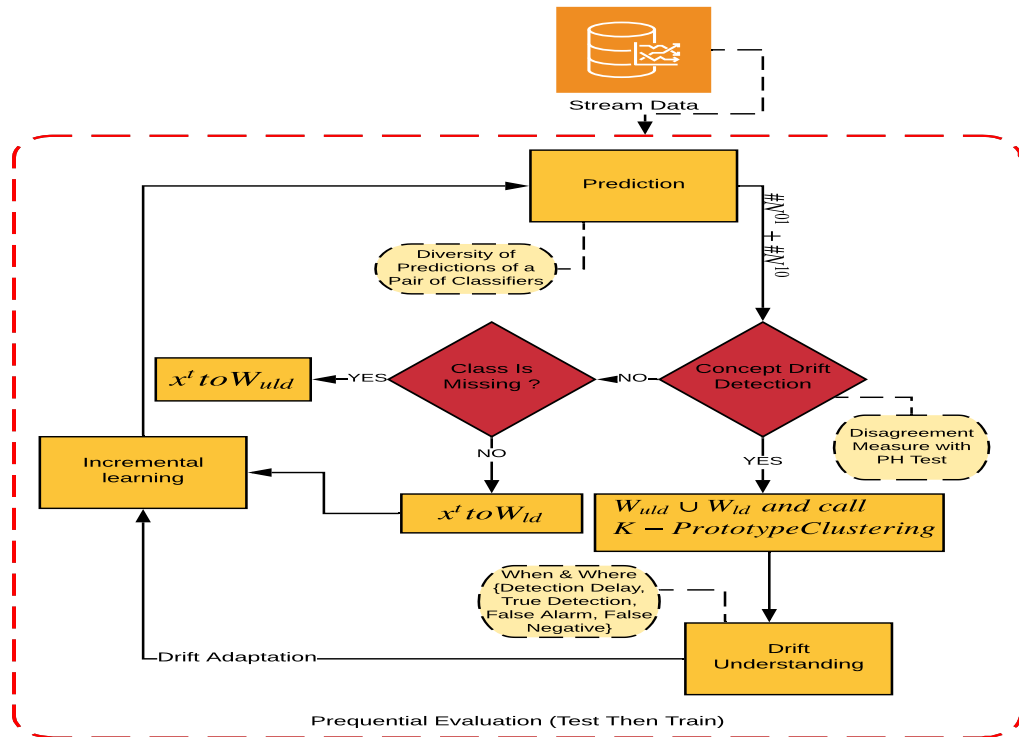


Figure 5.7: Framework of DMDDM-S.

This section presents the results of the experiments and the analyses of each drift detector, with Tables 5.8 – 5.10 showing the results of the experiments using Sine1, Sine2 and the Mixed datasets. The results of the Sine1 and Mixed datasets are similar. For example, the method that detects drift the fastest is STEPD, followed by our proposed method, DMDDM-S, and then FHDDM, SEED and RDDM in ascending order of delay detection. In addition, all the methods detect all the drifts correctly. In relation to Sine2, DMDDM-S has the lowest average delay detection followed by STEPD, FHDDM, and SEED, respectively. In relation to computational time and memory usage, it is clearly seen that SEED, RDDM and FHDDM have the highest memory usage compared to the others. This is because these methods require more memory for storing the prediction results in the

sliding windows or repositories and they use more computational time due to sub-window compression or reservoir sampling procedures. In addition to being the fastest and second fastest in delay detection, DMDDM-S has the lowest computational time and memory usage. The main reason for this is because DMDDM-S maintains a small number of variables compared with the others, which results in less memory usage and less execution runtime to update these variables.

Table 5.8: Results of Sine1 dataset with 10% noise.

| Sine1 (A) | | Sine1 (B) | | | | |
|------------|----------|-----------|-------|-------|----------|--------------|
| Classifier | Detector | Delay | TP | Time | Memory | MeanAccuracy |
| HT | DMDDM-S | 36.375 | 4 | 1.6 | 168 | 86.631 |
| | FHDDM | 47.375 | 4 | 7.7 | 1048 | 85.242 |
| | DDM | 196.225 | 2 | 3.3 | 472 | 66.633 |
| | PHTest | 238.275 | 1.2 | 2.1 | 1240 | 66.211 |
| | STEPD | 27.05 | 4 | 6.4 | 936 | 87.047 |
| | SEED | 58.4 | 4 | 12 | 3572.588 | 86.969 |
| | RDDM | 93 | 4 | 2.6 | 8656 | 86.894 |
| | EDDM | 244.525 | 0.1 | 1.6 | 144 | 83.34 |
| Pre | FHDDM | 46.562 | 4 | 7.75 | 1048 | 87.177 |
| | DDM | 154.636 | 4 | 2.545 | 472 | 86.870 |
| | PHTest | 249.568 | 0.090 | 1.363 | 1240 | 72.199 |
| | STEPD | 27.35 | 4 | 5.2 | 936 | 87.199 |
| | SEED | 56.8 | 4 | 11.5 | 3593.608 | 87.098 |
| | RDDM | 99.875 | 4 | 2.6 | 8656 | 87.075 |
| | EDDM | 250 | 0 | 1.6 | 144 | 72.181 |

Regarding false alarm, DMDDM-S has relatively high readings, due to the parameter sensitivity of DMDDM-S. For example, as shown in Fig 5.8 a,c, increasing λ (100–200–300) will entail fewer false positives (as shown in Fig 5.8 a) but might delay change or miss detection (as shown in Fig 5.8 c), resulting in a trade-off between false positives and delay change detection. Even with this increase, DMDDM-S is one of the highest in terms of delay detection and incurs the lowest time and memory consumption.

Furthermore, despite this trade-off, the accuracy of DMDDM-S is constant over the sensitivity of parameters, as shown in Fig 5.8 e. As mentioned in [130], the trade-off exists with the use of the PH test; consequently, one possible solution to overcome this trade-off is to control the diversity of the disagreement method using two fading factors

Table 5.9: Results of Sine2 dataset with 10% noise

| Sine2 (A) | | Sine2 (B) | | | | |
|------------|----------|-----------|-----|------|----------|--------------|
| Classifier | Detector | Delay | TP | Time | Memory | MeanAccuracy |
| HT & Pre | DMDDM-S | 23.75 | 4 | 1.7 | 168 | 78.294 |
| | FHDDM | 52.125 | 4 | 9.2 | 1048 | 79.834 |
| | DDM | 209.2 | 3.6 | 1.5 | 472 | 77.399 |
| | PHTest | 230 | 0 | 1.9 | 1240 | 57.738 |
| | STEPD | 33.1 | 4 | 6.4 | 936 | 79.855 |
| | SEED | 62.4 | 4 | 11.4 | 3638.354 | 79.729 |
| | RDDM | 134.575 | 4 | 3.8 | 8656 | 79.55 |
| | EDDM | 250 | 0 | 2 | 144 | 57.738 |
| Pre | FHDDM | 56.675 | 4 | 10.5 | 1048 | 74.843 |
| | DDM | 246.25 | 0.3 | 2.1 | 472 | 74.269 |
| | PHTest | 250 | 0 | 1 | 1240 | 49.882 |
| | STEPD | 41.725 | 4 | 6.5 | 936 | 74.851 |
| | SEED | 70.4 | 4 | 10.6 | 3688.576 | 74.813 |
| | RDDM | 189.625 | 3.4 | 2 | 8656 | 74.623 |
| | EDDM | 250 | 0 | 1.4 | 144 | 74.256 |

instead of one as used in this work. Such a solution is left for future work. Moreover, Fig 5.8 b,d,f shows the stability of each detector in terms of accuracy and mean accuracy over the data stream, where DMDDM-S is among the best and most stable compared to the others.

Table 5.10: Results of Mixed dataset with 10% noise.

| Mixed (A) | | Mixed (B) | | | | |
|-------------------|-----------------|------------------|-----------|-------------|---------------|---------------------|
| Classifier | Detector | Delay | TP | Time | Memory | MeanAccuracy |
| HT & Pre | DMDDM-S | 36.45 | 4 | 1.2 | 168 | 83.171 |
| | FHDDM | 48.1 | 4 | 9 | 1048 | 72.767 |
| | DDM | 214.575 | 1.8 | 1.9 | 427 | 69.828 |
| | PHTest | 236.025 | 1.3 | 1.6 | 1240 | 67.876 |
| | STEPD | 28.7 | 4 | 7.6 | 936 | 83.373 |
| | SEED | 60 | 4 | 11.1 | 3609.606 | 83.294 |
| | RDDM | 100.125 | 4 | 3.4 | 8656 | 83.265 |
| | EDDM | 250 | 0 | 2.1 | 144 | 57.858 |
| Pre | FHDDM | 47.45 | 4 | 7.6 | 1048 | 82.125 |
| | DDM | 220.925 | 2.1 | 1.9 | 427 | 79.316 |
| | PHTest | 246.55 | 0.2 | 1.4 | 1240 | 76.743 |
| | STEPD | 30.725 | 4 | 7 | 936 | 82.143 |
| | SEED | 60 | 4 | 11 | 3641.589 | 82.065 |
| | RDDM | 117.575 | 4 | 2.5 | 8656 | 82.006 |
| | EDDM | 244.65 | 0.1 | 1.5 | 144 | 76.483 |

5.4 Summary

In this chapter, we introduced the diversity measure as a new drift detection method in data streaming for binary classification problem (DMDDM) and fast reaction to sudden concept drift in the absence of class labels (DMDDM-S).

In Section 5.2, we presented DMDDM which was designed to react to sudden concept drift in the binary classification problem. The key novelty of the proposed algorithm is that it is the first supervised drift detection method to measure the diversity of component classifiers directly and use it as a base for drift detection. In contrast, the previous approaches used classification accuracy to detect drifts. Such a novel approach allows DMDDM to react effectively to sudden concept drift in the binary classification problem in a rapid manner, using minimal computational resources. We carried out an experimental analysis to evaluate and compare DMDDM with ten state-of-the-art data stream drift detector methods.

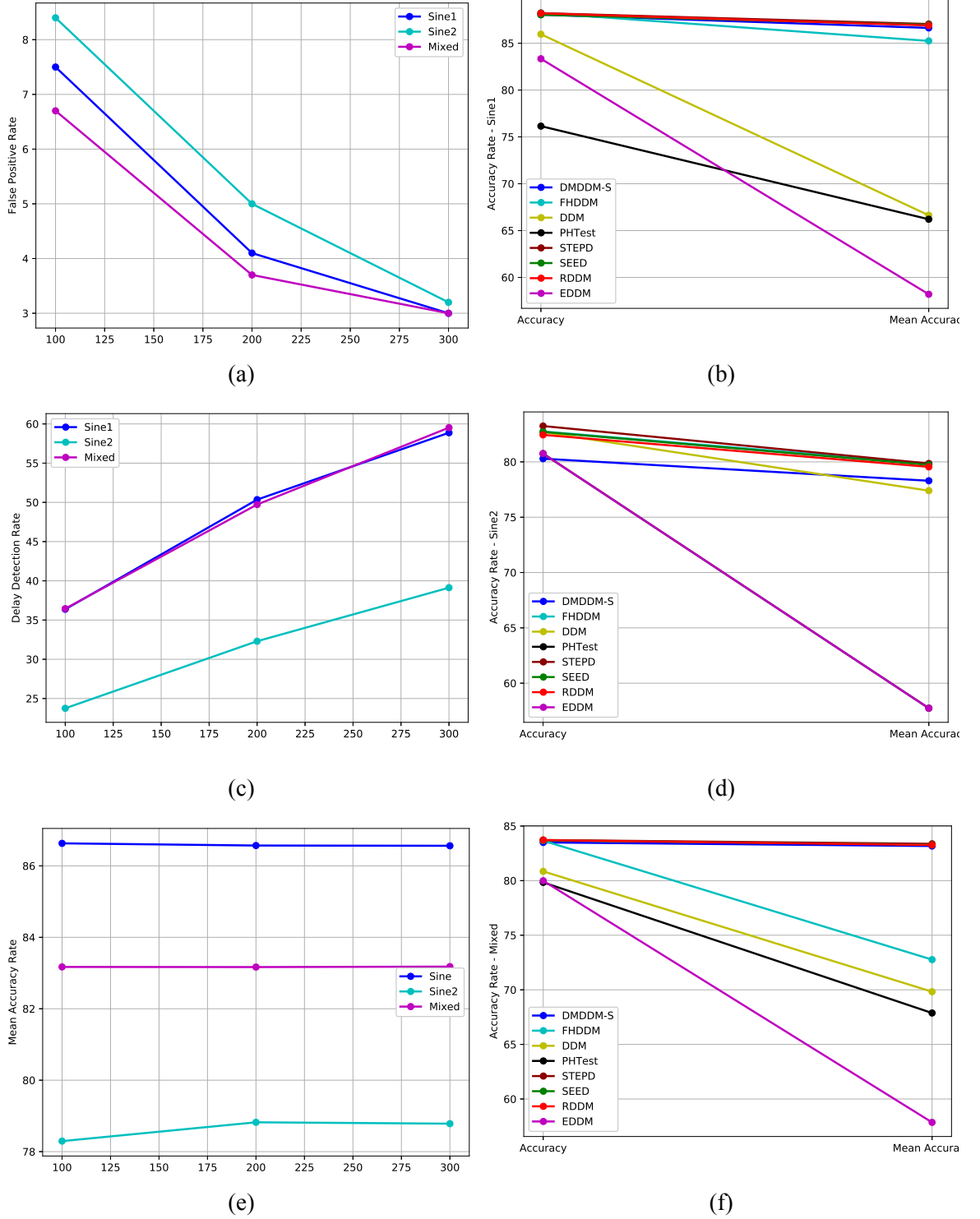


Figure 5.8: Changing DMDDM-S threshold (a,c,e) and the stability of each detector's accuracy (b,d,f).

The final results confirm that DMDDM efficiently handled concept drift in a rapid manner and used minimal computational resources compared to the state-of-the-art.

In Section 5.3, we presented DMDDM-S, which was designed to react to sudden concept drift in the absence of class labels. Most existing works make an optimistic assumption that all incoming data are labelled and the class labels are available immediately.

However, such an assumption is not always valid. Therefore, a lack of class labels aggravates the problem of concept drift detection. With this motivation, we propose a drift detector that reacts naturally to sudden drifts in the absence of class labels. In a novel way, the proposed detector reacts to concept drift in the absence of class labels, where the true label of an example is not necessary. Instead of monitoring the error estimates, the proposed detector, DMDDM-S, monitors the diversity of a pair of classifiers, where the true label of an example is not necessary to determine whether components disagree.

We also conducted an experimental analysis evaluating and comparing DMDDM-S with seven state-of-the-art data stream drift detector methods. The proposed algorithm and the ones used for comparison are implemented in Java as part of the

Massive Online Analysis. (We had to remove the label from some training instances (50%) to simulate a semi-supervised environment and where a semi-supervised setting assumes some input (training), the instances will not be labelled). The performance of DMDDM-S is evaluated against seven supervised drift detectors. The main reasons for choosing the supervised methods over semi-supervised methods are: (i) limited access to the source

code of some works, and (ii) we want to show the strengths of DMDDM-S, which can detect drifts faster when only 50% of the data is labelled with minimal consumption in terms of memory and run time than the existing methods that use 100% of labelled data.

The results of the experiments on the synthetic data sets indicate that with a lack of class labels, DMDDM-S detects drifts with shorter delay and with minimal detection run-time and memory usage compared to the existing methods that use 100% of labelled data.

Chapter 6

A Hybrid Block-Based Ensemble Framework

This chapter presents one of our research papers, which is:

- “*A Hybrid Block-Based Ensemble Framework for the Multi-Class Problem to React to Different Types of Drifts*”, paper submitted to Journal Cluster Computing on 30 July 2020. This paper introduced a novel method, the disagreement measure as a fully supervised drift detection method (DMDDM), which reacts rapidly to sudden concept drift in less time and with less memory consumption compared to the start-of-the-art drift detectors. We present DMDDM in Section 4.2.

6.1 Problem Statement

As previously mentioned, the problem of changing the definitions of classes over time decreases the performance (accuracy) of a predictive model which has been trained using old instances [7, 126]. Also, processing the multi-class problem is computationally more expensive, particularly in the presence of concept drift in data streams where the data is changing over time, and this aggravates the problem of a loss of performance during the process of drift detection in data streams [49]. In addition, ensemble approaches which process instances in blocks may not react to sudden changes sufficiently quickly. Consequently, developing a concept drift detection method that processes the multi-class problem which reacts rapidly to concept drift in less time and with less memory consumption is needed to enhance the condition of adaptive learning. Recall that we formalized these

problems and its corresponding research objective in Section 2.1 as follows:

- **Research Problem 2.1:** *How can DMDDM detect concept drift in multi-class classification?*. The task is to signal concept drifts in less time and with less memory consumption, keeping the accuracy of the data stream models constant.
- **Research Objective 2.1:** To devise a new formalism that facilitates a way to detect concept drifts in the multi-class problem with more accuracy and less time memory consumption compared to other drift detectors. To do this, we empirically compare our proposed drift detection method with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., detection delay, true detection, memory, and accuracy.
- **Research Problem 2.2:** *Can we use the block-based ensemble approach to enhance the reaction to sudden drifts and respond to different types of concept drift?*. Therefore, the task is to keep the accuracy of the data stream models constant.
- **Research Objective 2.2:** To devise a hybrid block-based ensemble which is a framework for multi-class classification in evolving data streams. To do this, we empirically compare our proposed framework with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., memory, time and accuracy.

To address these research problems, we introduce a hybrid block-based ensemble (HBBE) which is a fully supervised multi-class classification framework for classification in evolving data streams. HBBE maintains an ensemble of classifiers and builds a new model using a new block of samples in data streams when it detects concept drift via an online drift director. To do this, we put forward an online drift detector for a multi-class problem (ODDK) which reacts rapidly to concept drift in less time and with less memory consumption, and works in parallel with HBBE. We present the details of HBBE in Section 4.2.

6.2 Framework for the Multi-Class Classification Problem to react to different types of drifts

The proposed HBBE approach combines an online drift detector which processes instances online (instance by instance) and block-based weighting to deal with different types of drifts. Figure 6.1 shows the proposed block-based ensemble framework combined with a drift detector. The framework comprises the following:

- Online drift detector (instance by instance) enhances the reaction of the ensemble to sudden drift.
- Block-based ensemble: After d examples – evaluate and updates component classifiers of the ensemble incrementally and add a new member if necessary, which should improve the ensemble’s reactions to gradual drift.
- If a drift is detected, a nominee classifier is built using the most recent examples, it is weighted and then the nominee classifier is added to the ensemble according to $\theta()$. The existing ensemble components are also re-weighted after each drift.
- When the concept is stable and no drifts occur, the framework works similarly to the normal block-based ensemble.
- The outputs of the hypotheses from the online drift detector and the batch learners are integrated by a weighted majority vote using a suitability measure.

The next sub-section details the proposed HBBE. First, we discuss the proposed online drift detector as a drift detector for the K-class problem, followed by a discussion of the details of the HBBE which combines the best of the proposed online drift detectors and block-based weighting to react to different types of drifts.

6.2.1 Online Drift Detector for the K-Class Problem (ODDK)

The online drift detector for the K-class problem (ODDK) uses a pair of base learners/classifiers to detect drifts in evolving data streams. We have already considered

a preliminary version of a pair of base learners/classifiers as a drift detector (DMDDM, DMDDM-S), where a disagreement measure has been used with a PH test to detect sudden drifts in the binary classification problem. In this current work, we incorporate a number of modifications, resulting in new contributions. First, it comes with a new formalism that facilitates the way to detect concept drifts in the multi-class problem. Second, it is combined with a block-based ensemble in a mechanism that enhances its ability to react to sudden drifts and other types of drifts. As a result, the analysis of ODDK has led to interesting findings compared to the state-of-the-art.

Let $X = x_1, \dots, x_n$ be data set labels and $y'_v = [y'_v(x_1), \dots, y'_v(x_n)]$ which represents the n -dimensional binary vector of the classifier h_v output, such that $y'_v(x_j) = 1$, if h_v predicts the class label successfully, and 0 otherwise. Therefore, for a pair of classifiers h_u and h_v , Table 6.1 shows all the possible outcomes for the binary classification problem, where N^{ab} is the number of instances $x_j \in X$ for which $y'_u(x_j) = a$ and $y'_v(x_j) = b$.

Table 6.1: Output of a Pair of Classifiers (2×2) for the Binary Classification Problem.

| $h_u = h_v$ | $h_u \text{correct}(1)$ | $h_u \text{incorrect}(0)$ |
|---------------------------|-------------------------|---------------------------|
| $h_v \text{correct}(1)$ | N^{11} | N^{10} |
| $h_v \text{incorrect}(0)$ | N^{01} | N^{00} |

In preliminary versions of this work (DMDDM, DMDDM-S) and for the binary classification problem, the disagreement measure $D_{v,u}$ Equation 6.1 has been used. However, for multi-class classification problems, using the output of Table 6.1 would be unsuccessful to measure the differences between a pair of classifiers that incorrectly predict the same instance using different labels. Consequently, as the first contribution of this work, we propose a new way of tracking the classifiers' exact predictions instead of only the dichotomy correct/incorrect, as has been done in preliminary versions of this work. Therefore, to capture the variation of a pair of classifiers precisely, we construct a table $C_{i,j}$, where each value at the intersection of row i and column j holds the number of instances $x \in X$, where $h_v(x) = i$ and $h_u(x) = j$. Table 6.2 shows an example of contingency table $C_{i,j}$ for the k -class problem.

$$D_{u:v} = N^{10} + N^{01} \quad (6.1)$$

From Table 6.2, the concomitant decisions of the pair of classifiers are stored in the

Table 6.2: Output of a Pair of Classifiers for the Multi-class Classification Problem

| | $h_u(x) = 0$ | $h_u(x) = 1$ | ... | $h_u(x) = (k - 1)$ |
|--------------------|--------------|--------------|-----|--------------------|
| $h_v(x) = 0$ | C_{00} | C_{01} | ... | $C_{0(k-1)}$ |
| $h_v(x) = 1$ | C_{10} | C_{11} | ... | $C_{1(k-1)}$ |
| ... | ... | ... | ... | ... |
| $h_v(x) = (k - 1)$ | $C_{(k-1)0}$ | $C_{(k-1)1}$ | ... | $C_{(k-1)(k-1)}$ |

diagonal in matrix $C_{i,j}$. Thus, in order to weight their similarity, Equation 6.2 is used to find the summation of its values and divide it by the total number of instances n .

$$\Theta = 1/n \sum_{i=0}^K (C_{i,j}) \quad (6.2)$$

In addition, in order to signal if there is a drift, we use the PH test from our preliminary work, as shown in Equation 6.3 and Equation 6.4

$$m_T = \sum_{t=1}^T (x_t - x'_T - \delta) \quad (6.3)$$

$$PH_T = m_T - M_T \quad (6.4)$$

The ODDK approach is presented in Algorithm 5. As a first step, the algorithm starts processing each example and finds the predictions for the pair of classifiers (lines 1–3). Then, the algorithm starts constructing a contingency table $C_{i,j}$, as shown in Table 6.2, such that the value at the intersection of row i and a column j stores the number of instances $x \in X$, where $h_v(x) = i$ and $h_u(x) = j$. Then, the concomitant decisions of the pair from the diagonal in matrix $C_{i,j}$, are used to weight their similarity sum of its values and divide it by the number of instances n (Line 4-5).

Lines (6-7) calculate the fading sum and fading increment to find the fading average (line 8). Then, the diversity of C_v and C_u is monitored (9-14), where the PH test looks at the variable m_T which is used to measure the accumulated variation of the observed value of the diversity of C_v and C_u and their mean, up to the current moment. After each iteration, there is a step that checks if there is a drift, if the value between m_T and M_T is larger than a predefined threshold. If a drift is detected, phase three (drift understanding phase) takes action by evaluating the drift that has been detected by calculating the following metrics,

Algorithm 5: Pseudocode of the Online Drift Detector for K-Class Problem (ODDKP)

Require: S : data stream of examples (labelled),
 Forgetting factor $\alpha : 0 < \alpha < 1$
 Admissible change: $\delta = 0.1$,
 Drift threshold: $\lambda = 100$
 $M_T : 1.0D$
Result: Drift $\in \{\text{TRUE}, \text{FALSE}\}$

```

1 for each example  $x^t \in S$  do
2    $C_v\text{prediction} = \text{get prediction using } x^t$ ;
3    $C_u\text{prediction} = \text{get prediction using } x^t$ ;
4    $P[C_v\text{prediction}][C_u\text{prediction}]++$ 
5    $Sum+ = P[i][i]$ 
6    $S_{u:v,\alpha}(t) = Sum + \alpha \times Sum(t-1)$ ;
7    $N_\alpha(t) = 1 + \alpha \times N_\alpha(t-1)$ ;
8    $M_\alpha(t) = \frac{S_{u:v,\alpha}(t)}{N_\alpha(t)}$ ;
9    $SumDiversity = SumDiversity + M_\alpha(t)$ ;
10   $m_T = (m_T + M_\alpha(t) - (SumDiversity/instancesSeen) - \delta)$ ;
11   $M_T = \text{Min}(M_T, m_T)$ ;
12   $PH_{test} = m_T - M_T$ ;
13  if  $PH_{test} > \lambda$  then
14    | Return TRUE
15  else
16    | Return FALSE
17  end
18  incrementally train  $C_v$  and  $C_u$  with  $x^t$ ;
19 end

```

namely: delay detection, true detection, false alarm and false negative. The process of evaluating the detected drift is explained in Equation. 4. Phase four (incremental learning phase) keeps the model up-to-date by incrementally training the model (line18).

6.2.2 Hybrid Block-Based Ensemble (HBBE)

Block-based ensembles are a collection of component classifiers that work together to achieve greater predictive performance, in which data is processed in blocks of a pre-defined size. In such a scenario, when a new block of data arrives, all the existing component classifiers of the ensemble are evaluated and their combination weights are updated. At the same time, a new learner/ classifier is trained on the new block of data and added to the ensemble and replaces the weakest one based on the evaluation results of the ensemble. In addition, the accuracy weighted ensemble (AWE) [115] can be considered as the first work using such ensembles. However, as they rely on the existence of concept drifts within a

pre-defined size of blocks, block-based ensembles may not act quickly enough to changes in the concepts of incoming data. In the case of sudden drifts, block-based ensemble approaches may respond slowly due to using outdated blocks to train classifiers. On the other hand, relying on data blocks of a small size can help to some extent in responding to sudden concept drift, however doing so may harm the ensemble performance and increase computational costs in the interval of stability.

As a result, we investigate the notion that combining an online drift detector and a block-based ensemble in one framework enhances the ensemble's response to sudden concept drift with a reasonable balance with computational costs. Consequently, the proposed framework can be considered a HBBE which reacts to sudden drift via the online drift detector and reacts to gradual drift by updating the components of the ensemble after every block of data.

Let S denote a data stream in chunks S_1, S_2, \dots, S_n where each chunk is of equal size and C_i represents some classifier for S_i . The weight of classifier C_i is the estimated prediction error using the most recent data S_n . Since S_n is a data stream and produces examples in the form (x, c) where x is the feature vector and c is the class label, the classification error of C_i is $1 - f_c^I(x)$ where $f_c^I(x)$ is the probability that x is an example of class c . As such, the mean square error of C_i is given by

$$MSE_i = 1 / (s_n \sum_{(x,c) \in s_n}^T \int_c^i (x))^2 \quad (6.5)$$

Should a classifier predict randomly, then the mean square error can be given as:

$$MSE_r = \sum_c p(c)(1 - p(c))^2 \quad (6.6)$$

A random classifier contains no meaningful knowledge of the data, rather, it makes predictions simply at random. Therefore, MSE_r is used as a threshold for weighting and classifiers whose error rate is at least equal to MSE_r are discarded. The weight of a classifier C_i is given as:

$$w_i = MSE_r - MSE_i \quad (6.7)$$

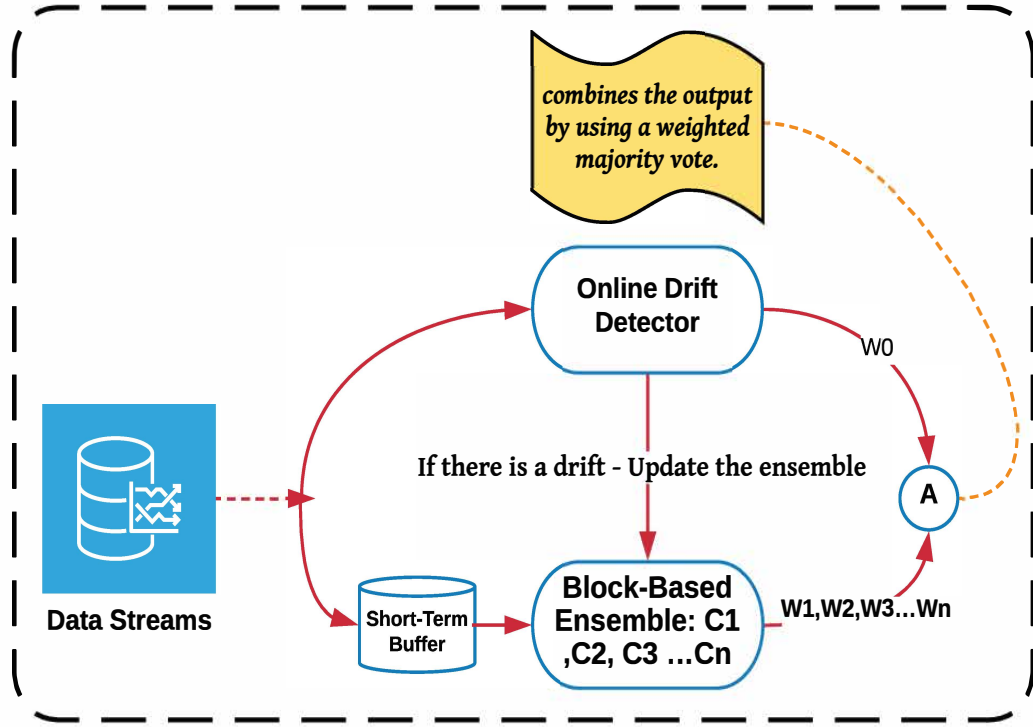


Figure 6.1: The Framework of Hybrid Block-Based Ensemble

Algorithm 6 uses the block-based ensemble and the online drift detector as a hybrid block-based ensemble. The algorithm starts by processing examples of the data stream online (instance by instance) (line 1) and then holds each incoming example in a buffer of predefined size d (line 2). Line 3 uses the online drift detector (Algorithm 1). In addition, if Algorithm 1 detects a drift or the block size reaches its predefined size, then a nominated classifier is built and weighted using the buffer (line 4) which should hold the most recent examples of the data stream. After this, all the ensemble components are weighted using the buffer line (5). Lines 7 and 8 add the nominated classifier to the ensemble if the size of the ensemble is less than the predefined k , otherwise, the weakest component is substituted in the ensemble lines 9 and 10. In such a scenario, a faster online reaction to sudden drift and gradual changes is sought.

Algorithm 6: Pseudocode of the Block-Based Ensemble and Drift detector (HBBE)

Require: S : data stream of examples (labelled),
 D: drift detector
 k: number of ensemble members
 B: example buffer of size d
 (Q): classifier quality measure
 t: example number
 $M_T : 1.0D$
Result: E: ensemble of k weighted classifiers and 1 classifier with a drift detector

```

1 for each example  $x^t \in S$  do
2    $B \leftarrow B \cup \{x^t\}$ 
3   if drift detected (Algorithm 1) OR  $|B| = d$  then
4     build and weight candidate classifier  $C'$  using B and Q(6);
5     weight all classifiers  $C^i$  in ensemble E using B and Q(8);
6   end
7   if  $|E| < k$  then
8      $E \leftarrow E \cup \{C'\}$ 
9   else
10    if  $Q()' > Q()$  then
11      replace weakest ensemble member with  $C'$ ;
12    end
13  end
14 end
  
```

6.2.3 Experimental Evaluation

This section details the results and the analysis of both the online drift detector and the HBBE. Tables 6.3 (a) and 6.4 (a) present the results of using the online drift detector in terms of detection delay, true positives, false positives, and false negatives. Tables 6.3 (b) and 6.4 (b) show the detection runtime, memory usage and accuracy.

Tables 6.3 (a, b) show the results and the analysis using the wave dataset. It can be clearly seen that the proposed online drift detector (ODDKP) detected the single drift faster than the other detectors with 0 false positives/false alarms and false negatives. In addition, the results show that the methods using HT which detected drift the fastest were $MDDM_A$, STEPDP (H) and STEPDP (H). However, $MDDM_A$ and STEPDP (H) recorded a high rate of false positives/ false alarms and false negatives compared to ODDKP. On the other hand, the methods using PER which detected drift the fastest were FHDDM (P) followed by ODDKP. Again, FHDDM recorded more false positives/ false alarms compared to ODDKP. Furthermore, Tables 6.4 (a, b) show the results and the analysis using the SEA

dataset, where our online drift detector was much faster to detect drift compared to the others. In addition, our drift detector is able to detect all the drifts compared to the other drift detectors which fail to detect all the drifts. Figs 5 and 6 show the average detection delay for each detector using both classifiers.

In relation to the time and memory use of each drift detector using both the Wave and SEA dataset, SEED, ADWIN and RDDM have the highest memory use and the longest runtime was recorded by ADWIN, as shown in Fig 6.2. This is because these detectors require extra memory for storing the prediction results of the classifiers in sliding windows or repositories and this results in using extra computational time due to sub-window compression or reservoir sampling procedures. In addition, compared to the other detectors, our online drift detector has the lowest cost in terms of computational time and memory use. The main reason for these good results is because our online drift detector retains a small number of variables which results in less memory use and less runtime to update these variables.

Finally, with respect to the classification accuracy of each drift detector, Figs 6.3 and 6.4 show the accuracy of the proposed online drift detector compared to the others. ODDKP achieved higher accuracy compared to the average of each drift detector.

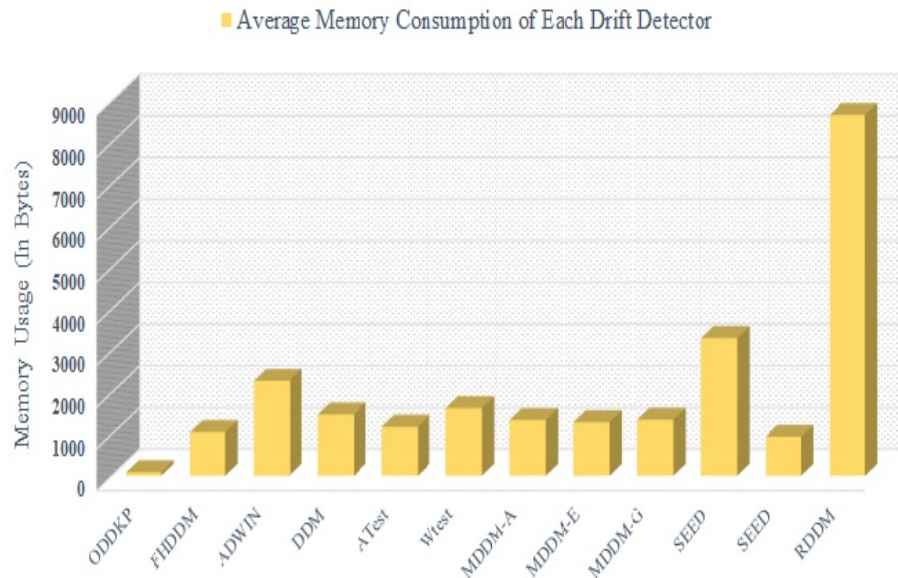


Figure 6.2: Memory Consumption

Table 6.3: Results of Wave Dataset

| Table 3 (a) | | | | | |
|--------------------|-----------|-----------|----------|-----|----------|
| Classifier | Detectors | ADD | ATD | AFA | AFN |
| HT&PER | ODDKP | 34 | 1 | 0 | 0 |
| HT | FHDDM | 77 | 1 | 7 | 0 |
| | ADWIN | 144 | 1 | 6 | 0 |
| | DDM | 250 | 0 | 0 | 1 |
| | ATes | 250 | 0 | 2 | 1 |
| | Wtest | 36 | 1 | 10 | 0 |
| | MDDM-A | 16 | 1 | 9 | 0 |
| | MDDM-E | 73 | 1 | 9 | 0 |
| | MDDM-G | 66 | 1 | 9 | 0 |
| | SEED | 80 | 1 | 5 | 0 |
| | STEPD | 26 | 1 | 33 | 0 |
| | RDDM | 250 | 0 | 11 | 1 |
| PER | FHDDM | 20 | 1 | 2 | 0 |
| | ADWIN | 250 | 0 | 93 | 1 |
| | DDM | 250 | 0 | 0 | 1 |
| | ATest | 85 | 1 | 1 | 0 |
| | Wtest | 92 | 1 | 59 | 0 |
| | MDDM-A | 104 | 1 | 16 | 0 |
| | MDDM-E | 47 | 1 | 17 | 0 |
| | MDDM-G | 47 | 1 | 13 | 0 |
| | SEED | 250 | 0 | 107 | 1 |
| | STEPD | 250 | 0 | 73 | 1 |
| | RDDM | 169 | 1 | 3 | 0 |

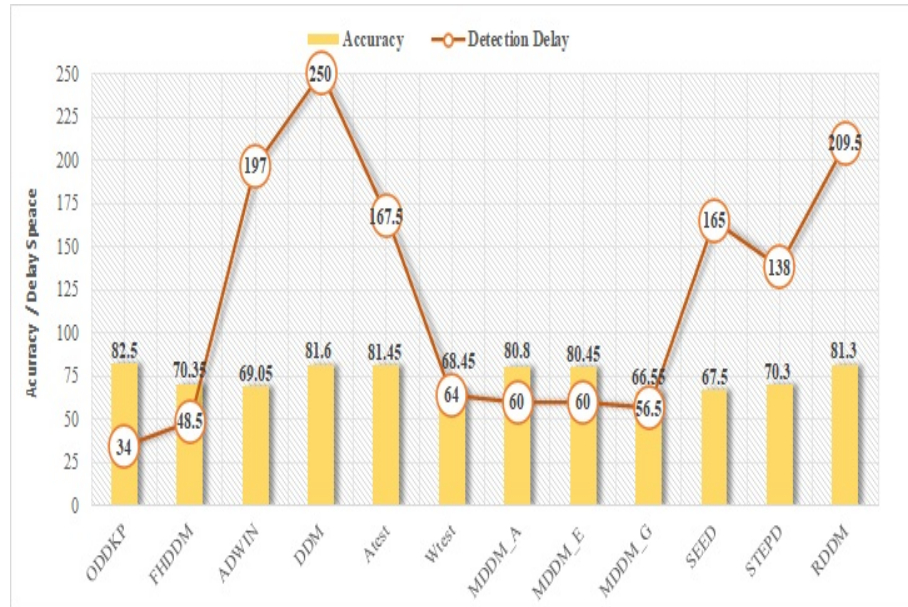


Figure 6.3: Average Accuracy and Delay Detection (Wave dataset)

On the other hand, Tables 6.5 - 6.7 show the results of the proposed HBBE framework in terms of classification accuracy, time and memory usage, respectively. As Table 6.5 shows, for the datasets with sudden drift (Wave and SEA) our method outperforms all the other algorithms followed by AUE2. In addition, we note that the accuracy of the

| Table 3 (b) | | | | |
|-------------|-----------|----------|------------|--------------|
| Classifier | Detectors | DRMS | MUB | ACC |
| HT&PER | ODDKP | 3 | 168 | 82.50 |
| HT | FHDDM | 12 | 1048 | 79.20 |
| | ADWIN | 75 | 2530.84 | 78.80 |
| | DDM | 2 | 472 | 79.80 |
| | ATest | 20 | 1176 | 79.40 |
| | Wtest | 8 | 1624 | 79.20 |
| | MDDM-A | 47 | 1336 | 79.00 |
| | MDDM-E | 24 | 1288 | 79.20 |
| | MDDM-G | 22 | 1344 | 79.10 |
| | SEED | 12 | 3704.032 | 77.90 |
| | STEPD | 13 | 936 | 79.60 |
| | RDDM | 2 | 8656 | 79.20 |
| PER | FHDDM | 10 | 1048 | 61.50 |
| | ADWIN | 55 | 2023.144 | 59.30 |
| | DDM | 2 | 2472 | 83.40 |
| | ATest | 8 | 1176 | 83.50 |
| | Wtest | 12 | 1624 | 57.70 |
| | MDDM-A | 25 | 1336 | 82.60 |
| | MDDM-E | 23 | 1288 | 81.70 |
| | MDDM-G | 23 | 1344 | 54.00 |
| | SEED | 6 | 2900.752 | 57.10 |
| | STEPD | 14 | 936 | 61.00 |
| | RDDM | 4 | 8656 | 83.40 |

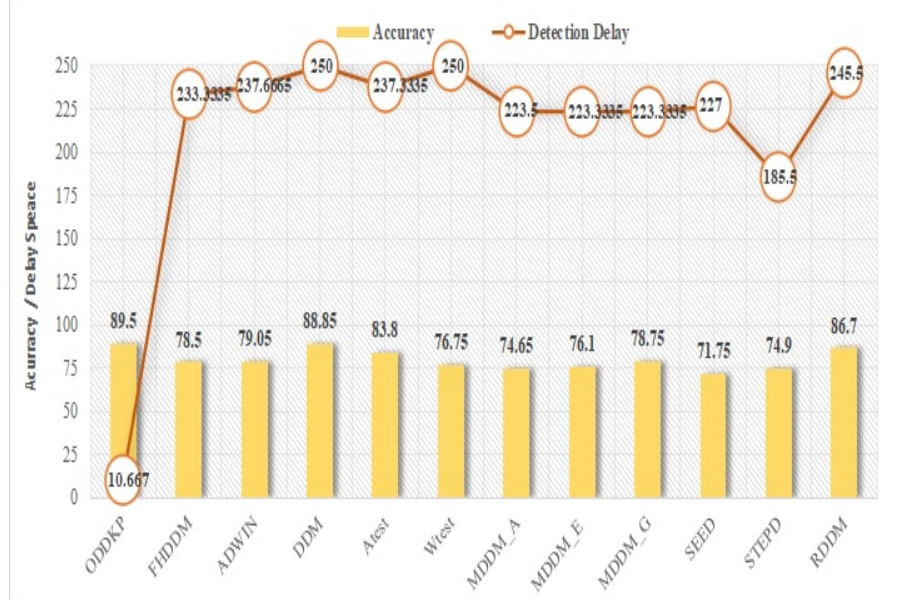


Figure 6.4: Average Accuracy and Delay Detection (SEA Dataset)

ensemble has improved compared to using the drift detector alone. This is due to adopting an online drift detector with the block-based ensemble. For the RBF_{GR} and $Tree_R$, AUE2 had the best performance followed by the proposed hybrid framework. However, our hybrid framework appears to be more precise in the case of sudden drift (Wave and SEA datasets). This is due to adopting the online drift detector which has a faster response to

Table 6.4: Results of SEA Dataset

| Table 4 (a) | | | | | |
|--------------------|-----------|----------------|----------|-----|-----|
| Classifier | Detectors | ADD | ATD | AFA | AFN |
| HT&PER | ODDKP | 10.667 | 3 | 4 | 0 |
| HT | FHDDM | 222 | 2 | 3 | 1 |
| | ADWIN | 225.333 | 1 | 112 | 2 |
| | DDM | 250 | 0 | 4 | 3 |
| | ATest | 224.667 | 1 | 6 | 2 |
| | Wtest | 250 | 0 | 11 | 3 |
| | MDDM-A | 222 | 2 | 7 | 1 |
| | MDDM-E | 196.667 | 2 | 9 | 1 |
| | MDDM-G | 196.667 | 2 | 9 | 1 |
| | SEED | 204 | 1 | 150 | 2 |
| | STEPD | 121 | 3 | 353 | 0 |
| | RDDM | 241 | 1 | 166 | 2 |
| PER | FHDDM | 244.667 | 1 | 133 | 2 |
| | ADWIN | 250 | 0 | 308 | 3 |
| | DDM | 250 | 0 | 0 | 3 |
| | ATest | 250 | 0 | 13 | 3 |
| | Wtest | 250 | 0 | 408 | 3 |
| | MDDM-A | 225 | 1 | 146 | 2 |
| | MDDM-E | 250 | 0 | 182 | 3 |
| | MDDM-G | 250 | 0 | 176 | 3 |
| | SEED | 250 | 0 | 374 | 3 |
| | STEPD | 250 | 0 | 445 | 3 |
| | RDDM | 250 | 0 | 35 | 3 |

sudden concept drift compared to most block-based ensemble approaches. In addition, our framework clearly outperforms all the other algorithms, especially on the Electricity dataset where ours is the most accurate followed by WMA, DWM, AUE2 and AWE, however all the algorithms perform almost identically on the Airline dataset. Second, with respect to time, DWM consumes the least amount of time followed by AWE, whereas .NSE consumes the most time. Furthermore, it is clear that processing an ensemble online is the best in respect to classification accuracy, however it results in a weak performance in respect to runtime. Third, as shown in Table 6.7, in most cases, DWM achieved minimal memory use, while NSE consumed the most memory. The memory usage of DWM is lower than the others because DWM adds and removes component classifiers in response to the global performance of the entire ensemble and the local performance of individual components. In addition, it is clear that our proposed algorithms are not the best of all the other methods, because our methods combine an online drift detector with the block-based ensemble.

| Table 4 (b) | | | | |
|--------------------|-----------|----------|------------|--------------|
| Classifier | Detectors | DRMS | MUB | ACC |
| HT&PER | ODDKP | 4 | 168 | 89.50 |
| HT | FHDDM | 93 | 1048 | 89.00 |
| | ADWIN | 563 | 2575.158 | 88.40 |
| | DDM | 29 | 472 | 89.80 |
| | ATest | 120 | 1176 | 89.70 |
| | Wtest | 89 | 1624 | 88.90 |
| | MDDM-A | 435 | 1336 | 89.90 |
| | MDDM-E | 306 | 1288 | 89.60 |
| | MDDM-G | 247 | 1344 | 89.60 |
| | SEED | 102 | 3598.547 | 89.00 |
| | STEPD | 55 | 936 | 88.80 |
| PER | RDDM | 41 | 8656 | 89.20 |
| | FHDDM | 96 | 1048 | 68.00 |
| | ADWIN | 513 | 2360.202 | 69.70 |
| | DDM | 19 | 472 | 87.90 |
| | ATest | 99 | 1176 | 77.90 |
| | Wtest | 72 | 1624 | 64.60 |
| | MDDM-A | 374 | 1336 | 59.40 |
| | MDDM-E | 275 | 1288 | 62.60 |
| | MDDM-G | 221 | 1344 | 67.90 |
| | SEED | 98 | 3507.966 | 54.50 |
| | STEPD | 68 | 936 | 61.00 |
| | RDDM | 21 | 8656 | 84.20 |

Table 6.5: Classification accuracy of the different algorithms

| | HBBE | AUE2 | AWE | DWM | .NSE |
|-------------------------|-------|-------|-------|-------|-------|
| <i>Wave</i> | 84.30 | 83.30 | 79.80 | 79.40 | 77.90 |
| <i>Sea</i> | 90.20 | 90.10 | 88.40 | 88.90 | 85.60 |
| <i>RBF_{GR}</i> | 90.70 | 91.50 | 88.60 | 90.10 | 87.30 |
| <i>Tree_R</i> | 82.40 | 82.80 | 55.10 | 50.90 | 42.00 |
| <i>Electricity</i> | 85.02 | 77.54 | 71.01 | 79.60 | 71.50 |
| <i>Airline</i> | 64.70 | 64.10 | 57.40 | 64.00 | 61.10 |

Table 6.6: Runtime of the different algorithms.

| | HBBE | AUE2 | AWE | DWM | .NSE |
|-------------------------|--------|--------|--------|--------|---------|
| <i>Wave</i> | 10.45 | 10.20 | 16.75 | 14.41 | 33.25 |
| <i>Sea</i> | 47.11 | 41.16 | 26.80 | 28.69 | 2210.42 |
| <i>RBF_{GR}</i> | 32.21 | 28.94 | 27.87 | 43.18 | 2139.36 |
| <i>Tree_R</i> | 159.71 | 157.06 | 89.24 | 62.44 | 2977.83 |
| <i>Electricity</i> | 3.23 | 2.19 | 1.80 | 0.63 | 2.01 |
| <i>Airline</i> | 454.57 | 453.71 | 161.48 | 145.97 | 609.35 |

6.3 Summary

In this chapter, we introduced a hybrid block-based ensemble framework. We furthermore compared it with the state-of-the-art methods. In section 6.2, we have presented the

Table 6.7: Memory usage of the different algorithms.

| | HBBE | AUE2 | AWE | DWM | .NSE |
|-------------------------|-------|-------|------|------|--------|
| <i>Wave</i> | 2.31 | 0.85 | 0.51 | 0.30 | 0.32 |
| <i>Sea</i> | 2.14 | 1.94 | 0.24 | 0.08 | 1.02 |
| <i>RBF_{GR}</i> | 1.20 | 0.85 | 0.51 | 0.30 | 147.71 |
| <i>Tree_R</i> | 8.15 | 7.66 | 0.63 | 0.15 | 6.29 |
| <i>Electricity</i> | 0.89 | 0.85 | 0.39 | 0.04 | 0.28 |
| <i>Airline</i> | 65.65 | 65.47 | 7.50 | 7.19 | 33.88 |

framework for the multi-class classification problem to react to different types of drifts. This chapter presented a HBBE framework which brings together the best of the online drift detectors and block-based weighting with a view to enhancing reaction to sudden drifts and to respond to other types of concept drift. Additionally, this chapter proposed an online drift detector to detect concept drift in a timely manner with less memory consumption. We proposed a new way of calculating diversity which has been designed for a K-class problem, while the preliminary version of our work (DMDDM and DMDDM-S) addressed the binary classification problem. The final results of the experimental evaluations on well-known synthetic and real-world datasets through a comprehensive comparison of eleven drift detectors and five ensemble approaches indicated that our proposed algorithms perform significantly better than other drift detectors and ensemble approaches.

Chapter 7

Adapting Related Knowledge for Detecting Concept Drift

This chapter presents two of our research papers, which are:

- “*Combination of Information Entropy and Ensemble Classification for Detecting Concept Drift in Data Stream*”, published in Proceedings of the Australasian Computer Science, (Mahdi, Osama A et al., 2018). This paper presents the entropy-based ensemble (EBE) for dealing with two main kinds of concept drifts: sudden and gradual drifts in labelled data. The learning procedures of the model are processed in blocks of the same size. We present EBE in Section 5.2.
- “*KAPPA as Drift Detector in Data Stream*”, Paper under preparation to be submitted to Conference Proceeding. This paper introduces KAPPA as drift detector, which reacts rapidly to sudden concept drift. We present KAPPA in Section 5.3.

7.1 Problem Statement

The classifier’s error rate and the ensemble are used in most of the previous works to manage classification accuracy as a criterion for judging whether concept drift is occurring or not. Information entropy and KAPPA are effective ways of measuring uncertainty and level of agreement, respectively, and they are suitable to detect concept drift in a reliable, fast, and computationally efficient way.

Consequently, in this chapter, first we introduce the proposed model called entropy-based ensemble (EBE) which is based on incorporating entropy as a drift detector into an ensemble. The combination is based on the following assumption:

- **Research Assumption:** *In data streaming, if the distribution of block point i is similar to the distribution of block point $i + 1$, this means that the stream is stable and there is no drift and vice versa.*

In addition, the proposed EBE does not build a new classifier for every new block of data, instead it builds a new classifier only when there is a drift. The new classifier will be trained on more recent instances and added to the ensemble. Therefore, this mechanism results in a low computational cost.

- **Research Problem 3.1:** *Is measuring uncertainty in data streams instead of the classifier's error rate suitable for detecting concept drift in a reliable, fast, and computationally efficient way?* Therefore, the task is to keep the accuracy of the data stream models constant.
- **Research Objective 3.1:** To propose a model called entropy-based ensemble (EBE) which is based on incorporating entropy as a drift detector into the evolving ensemble in a reliable, fast, and computationally efficient way. To do this, we empirically compare our proposed framework with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., memory, time and accuracy.
- **Research Problem 3.2:** *Contrary to the disagreement measure that was used in Chapter 4, is measuring the level of agreement using KAPPA suitable to detect concept drift when different classifiers access data items?* The task is to signal concept drift in less time and with less memory consumption, keeping the accuracy of the data stream models constant.
- **Research Objective 3.2:** To propose a drift detector based on KAPPA calculations. To do this, we empirically compare our proposed drift detector with our proposed drift detector from **Problem I** using synthetic data streams, considering different performance measures, e.g., delay detection, true positives and the mean accuracy.

7.2 Entropy in Data Streams

A data stream can be defined as a sequence consisting of sequentially ordered tuples d_i in time t_i where $i \in (1, 2, 3, \dots)$. Each tuple d_i consists of S feature streams s and one label stream l , formally $d_i := (s_i, l_i)$, where s_i is the vector of all feature stream instances at time t_i . In this direction, we make use of entropy in the context of data streams. Entropy is well-known from information theory [158] as a measure for information content and its application, thus, it is self-evident that we make use of it, mainly because of its symmetry and additive properties. It is defined in Equation.7.1. To compare the distribution dissimilarity using entropy (by counting and comparing all instances with respect to their features and class membership), we compare the old and new blocks of instances of data streams. If it results in a value of 1, this indicates the two distributions are equal, otherwise they are different. Entropy in the context of data streams is calculated using Equation.7.2:

$$H(x) = \sum_{t=i}^i P_i \log_2 P_i \quad (7.1)$$

$$H(x) = \sum_{t=i}^i P(x) \log_2 P(x) \quad (7.2)$$

Where x is a discrete random variable, p_i is the probability of occurrence of x_i and $P(x)$ is the probability mass function of x .

To illustrate the related notions of data streams and entropy, we provide the following definitions.

Definition 1. A certain number of instances n are organized as a data set according to the time sequence, so we call the data set a data block, which is denoted by $b_1, b_2, b_3, \dots, b_n$, where n is the size of the data block.

Definition 2. For any two-time points i and j , if the distribution of block point i is not similar to the distribution of block point j $D_i \neq D_j$, this means the stream is not stable and there is a drift, otherwise, the distribution is stable.

In order to use entropy in the context of detecting concept changes in data streams, two blocks over the data stream are tracked; one is the earlier block of data, and the other is current instances of stream. An algorithm for detecting concept changes compares the entropies of the earlier and current blocks. If the entropies differ by more than a defined amount, then a change is deemed to have occurred. The proposed entropy-based ensemble (EBE) is a drift detection method operating on blocks of data. The combination of entropy and an ensemble is presented in Algorithm 7. First, the algorithm builds a classifier which is added to the ensemble (line 2 and 3). Then the entropy for $block_i$ and $block_{i+1}$ is calculated (line 4 to 7). After calculating the entropy of $block_i$, we combine the instances of $block_i$ and $block_{i+1}$ in one block called JointEntropy. Line 8 checks if the result is less than the predefined threshold (predefined threshold = 95), meaning drift has occurred. The algorithm checks if the ensemble size is below the maximum number of classifiers (which is predefined as 10) and if so, it creates a new classifier based on the current block of data and is added to the ensemble (line 9 to 12). Otherwise, it removes the poorest classifier from the ensemble and creates a new one that is added to the ensemble (line 13 to 16). It trains the ensemble component incrementally on the current block in line 18. Lastly, if the result matches the predefined threshold, there is no need to create a new classifier, rather it trains the ensemble component incrementally to keep the model updated with the current blocks of data (line 22).

7.2.1 Experimental Evaluation

The main aim of this experiment is to evaluate the efficiency of proposed algorithm in terms of processing time, memory used and accuracy of classification and compares it with the prequential evaluation after adding concept drift.

From Table 7.1, we can see that using prequential evaluation, accuracy has dropped from 88.232% to 83.831% and 83.754% for gradual and sudden drifts, respectively. This drop in accuracy is clearly due to the presence of concept drift. In the other hand, EBE achieved an accuracy of 81.4%, however, in the presence of concept drift, the percentage of accuracy increased to 85.5% for gradual drift and 83.0% for sudden drift. The difference in accuracy between gradual and sudden drift reflects the behavior of the two types of drifts. For example, a smaller block size is more suitable for sudden drift because it allows a faster response to the changes, whereas a larger block size is more suitable for gradual drift. In

Algorithm 7: Pseudocode of the Entropy-Based Ensemble (EBE)

Require: S : Data stream of examples partitioned into blocks B ,
 k : number of ensemble members,
Result: E : Ensemble of k incremental classifiers

```

1  $E \leftarrow \emptyset$ 
2  $Cb_i \leftarrow \text{Newcomponentclassifierbuiltin } B_i$ 
3  $E \leftarrow E \cup Cb_i$ 
4 for all data Blocks  $B_i \in S$  do
5   Calculate Entropy 1 on  $B_i$ 
6   Calculate Entropy 2 on  $B_{i+1}$  // JointEntropy
7   Result of Entropy Entropy1 // Entropy2
8   if Result < Threshold then
9     if Ensemble size <  $K$  then
10      Create new classifier  $C_i$  on the current block  $B_i$ ;
11       $E \leftarrow E \cup Cb_i$ 
12    else
13      Remove the poorest classifier  $C_i$  from the Ensemble
14      Create new classifier on the current block
15       $E \leftarrow E \cup Cb_i$ 
16    end
17    for each example in block  $B_i$  do
18      Incrementally train classifiers  $C_i$  in the ensemble with  $B_i$ 
19    end
20  else
21    for each example in block  $B_i$  do
22      Incrementally train classifiers  $C_i$  in the ensemble with  $B_i$ 
23    end
24  end
25 end

```

addition, we believe that the time and memory usage of EBE are relatively high compared to ppquential evaluation due to the different ways the EBE algorithm handles concept drift.

Table 7.1: Comparison of EBE and prequential evaluation with and without concept drift.

| Method | Accuracy | Time | Memory Usage |
|-----------------------------|----------|------------|--------------|
| Prequential Without CD | 88.232% | 0.109375 S | |
| Prequential With Gradual CD | 83.831% | 0.140625 S | 33.555 MB |
| With Sudden CD | 83.754% | 0.125 S | |
| EBE without CD | 81.4% | 2.328125 S | |
| EBE with Gradual CD | 85.5% | 2.328125 S | 74.784 MB |
| With Sudden CD | 83.0 % | 2.171875 S | |

7.3 Inter-rater Agreement, k , in Data Streams

Recall that learning from data streams in the presence of concept drift is one of the biggest challenges in contemporary machine learning. Since data class distributions may change through the progress of the stream, KAPPA provides better insight than the other metrics to detect concept drift in data class distribution. First, as KAPPA is a strict measure that quickly drops in case of incorrect predictions, this makes it much more useful than using accuracy/error-rate which only introduces small changes. Second, the main reason concept drift occurs is due to changes/drifts in data class distribution as the stream progresses. KAPPA is capable of capturing the competence of the components reflecting the possibly varying data class distribution with time [51, 52]

In addition, KAPPA is a statistic that is commonly used for handling the problem of imbalanced classification [159–161]. It evaluates the competence of a classifier by measuring the inter-rater agreement between successful predictions and the statistical distribution of data classes, correcting agreements that occur by mere statistical chance [162].

Formally, a statistic developed as a measure of inter-rater reliability, called k , can be used when different raters (here classifiers) assess data items to measure the level of agreement while correcting for chance [51].

Recall that in Table 5.1 for c class labels, k is defined on the $C \times C$ coincidence matrix M of the two classifiers. The entry of M is the proportion of the data set, which D_i labels as w_k and D_j labels as w_s . The agreement between D_i and D_j is given by

$$k_{i,j} = \frac{\sum_k m_{kk} - ABC}{1 - ABC} \quad (7.3)$$

where $\sum_{t=1} m_{kk}$ is the observed agreement between the classifiers and ABC is agreement-by-chance.

$$ABC = \sum_k \left(\sum_s m_{k,s} \right) \left(\sum_s m_{s,k} \right) \quad (7.4)$$

Low values of k signify higher disagreement and hence higher diversity. If calculated on the 2×2 joined oracle output space using probabilities,

$$k_{i,j} = \frac{2(ac - bd)}{(a + b)(c + d)(a + c)(b + d)} \quad (7.5)$$

For the purpose of this work, we divide 5.5 by 2 as we are using a pair of classifier.

$$k_{i,j} = \left(\frac{2(ac - bd)}{(a + b)(c + d)(a + c)(b + d)} \right) / 2 \quad (7.6)$$

Table 7.2: The Correlation of a Pair of Classifiers (2×2)

| $h_u = h_v$ | $h_u correct(1)$ | $h_u incorrect(0)$ |
|--------------------|------------------|--------------------|
| $h_v correct(1)$ | N^{11} | N^{10} |
| $h_v incorrect(0)$ | N^{01} | N^{00} |

Algorithm 8: Pseudocode of the KAPPA as Drift Detector in Data Stream

Require: S : data stream of examples (labelled),
 Forgetting factor $\alpha : 0 < \alpha < 1$
 Admissible change: $\delta = 0.1$,
 Drift threshold: $\lambda = 100$
 $M_T : 1.0D$
Result: $\text{Drift} \in \{\text{TRUE}, \text{FALSE}\}$

```

1 for each example  $x^t \in S$  do
2    $C_v\text{prediction} = \text{get prediction using } x^t$ ;
3    $C_u\text{prediction} = \text{get prediction using } x^t$ ;
4   if  $C_v\text{prediction} = 0.0$  and  $C_u\text{prediction} = 1.0$  then
5      $b++$ ;
6   end
7   if  $C_v\text{prediction} = 1.0$  and  $C_u\text{prediction} = 0.0$  then
8      $c++$ ;
9   end
10  if  $C_v\text{prediction} = 0.0$  and  $C_u\text{prediction} = 0.0$  then
11     $a++$ ;
12  end
13  if  $C_v\text{prediction} = 1.0$  and  $C_u\text{prediction} = 1.0$  then
14     $d++$ ;
15  end
16   $k_{i,j} = (2(ac - bd)/(a + b)(c + d)(a + c)(b + d))/2$ ;
17   $S_{u:v,\alpha}(t) = k_{i,j} + \alpha \times S_{u:v,\alpha}(t - 1)$ ;
18   $N_\alpha(t) = 1 + \alpha \times N_\alpha(t - 1)$ ;
19   $M_\alpha(t) = \frac{S_{u:v,\alpha}(t)}{N_\alpha(t)}$ ;
20   $\text{SumDiversity} = \text{SumDiversity} + M_\alpha(t)$ ;
21   $m_T = (m_T + M_\alpha(t) - (\text{SumDiversity}/\text{instancesSeen}) - \delta)$ ;
22   $M_T = \text{Min}(M_T, m_T)$ ;
23   $PH_{test} = m_T - M_T$ ;
24  if  $PH_{test} > \lambda$  then
25    Return TRUE
26  else
27    Return FALSE
28  end
29  incrementally train  $C_v$  and  $C_u$  with  $x^t$ ;
30 end

```

The KAPPA approach is presented in Algorithm 8. First, the algorithm processes each example from the data stream and obtains the predictions for a pair of classifiers on each example line (lines 1-15). As the main idea behind these 15 steps, KAPPA tries to find all the possibilities which could exist using a pair of classifiers, as shown in Table 3.1.

Then, the algorithm builds the oracle output table, as shown in Table 3.1. From Table 3.1, it can be seen that we need to find all cases (N^{10} , N^{01} , N^{00} and N^{11}). Once we observe all the cases, we count the number of observations on which the classifier is correct and

incorrect for each case. Line (15) in our updated equation of KAPPA finds the agreement between the pair of classifiers. Then, the fading factor approach is applied from lines 17 to 19 (Equation.3.5, Equation.3.3 and Equation.3.6). In lines 17 and 16, the fading sum and fading increment are calculated, respectively.

With the fading sum and fading increment, we use the value of KAPPA from line 16 as the observed value instead of the error estimates that were used in the original PH test. In line 19, the fading average is calculated. To monitor the diversity of a pair of classifiers, from lines 20 to 28, the PH test considers a variable m_T , which measures the accumulated difference between the observed value of diversity and their mean up to the current moment. After each monitoring, there is a step for checking and signaling for a drift, if the value of variation between the current m_T and the smallest value up to this moment M_T is larger than a predefined threshold. If there is a drift, phase three (drift understanding phase) starts, which evaluates the detected drift in terms of delay detection, true detection, false alarm and false negative. The evaluation method is explained in **Section 4.6**. Finally, whether there is a drift or not, phase four (incremental learning phase) / line 29 is used to incrementally train the model and keep it up-to-date.

7.3.1 Experimental Evaluation

This subsection presents the outcomes of the analyses and the experiments of the proposed drift detector using the KAPPA measure. - Fig 7.1 (a - e) compares the outcomes of the experiments using delay detection, true positives and mean accuracy based on five well-known datasets, namely: Mixed, Sine1, SEA 10K, SEA 20K and SEA 50K.

First, in relation to the experiments using the Mixed and Sine1 datasets, for the Mixed dataset, DMDDM was able to detect four drifts successfully and slightly faster than KAPPA, 4 and 3.9 in terms of true positive (TP) and 35.11 and 4 in terms of delay detection, respectively. In addition, similar to the Mixed dataset, the results for Sine1 show that DMDDM is still slightly better than KAPPA in terms of delay detection, true positive and mean accuracy.

On the other hand, the last three datasets, SEA 10K, SEA 20K and SEA 50K, showed very promising results, with KAPPA achieving significant results compared to DMDDM. From Fig 7.1 (c - e) and in terms of delay detection, KAPPA was much faster than DMDDM, whereas in terms of detecting drift, both KAPPA and DMDDM were successfully able to detect a single drift in all datasets. Finally, in terms of mean accuracy, both drift detectors achieve almost identical results.



Figure 7.1: Changing DMDDM-S threshold (a,c,e) and the stability of each detector's accuracy (b,d,f).

7.4 Summary

In this chapter, we introduced the combination of information entropy and ensemble classification in addition to KAPPA as drift detector.

First, in Section 7.2, we presented the combination of information entropy and ensemble classification for detecting concept drift in data streams which deals with two kinds of concept drift: sudden and gradual in labelled data. The proposed algorithm is called entropy-based ensemble (EBE) for classifying data streaming. EBE is based on incorporating information entropy as a drift detector into the evolving ensemble. The main objective of the proposed algorithm is to detect two types of concepts drifts, namely: sudden and gradual drifts and handling them in an incremental way. Two experiments scenarios were conducted using synthetic data sets which were compared to prequential evaluation. The first is to generate data sets without concept drift and the second with concept drift. The results show that EBE improves accuracy in the presence of concept drift (sudden and gradual drift).

Additionally, in Section 7.3, we presented KAPPA as a drift detector in data stream mining, which was designed to react to sudden concept drift. Since data class distributions may change through the progress of the stream, KAPPA provides better insight than other metrics to detect concept drift. First, this is because KAPPA is a strict measure which quickly drop in the event of incorrect predictions, making it much more useful than using accuracy/ error-rate that only introduces small changes. Second, the main reason for concept drift is due to changes/drifts in data class distribution as the stream progresses. Therefore, Kappa is capable of capturing the competence of the components reflecting the possibly varying data class distribution with time. The final results confirm that KAPPA efficiently handles and detects concept drifts, where the results showed it has a comparable performance with DMDDM.

Chapter 8

Conclusion and Future Work

This chapter concludes the thesis and provides further research directions for this topic.

8.1 Conclusions

This thesis concerns drift detectors and learning ensemble classifiers from concept-drifting data streams. As mentioned in Section 2.1, our main aim was to propose efficient novel drift detectors and a hybrid block-based ensemble which are capable of reacting to sudden and various types of concept drift. This involved analyzing the properties of both drift detection strategies and block-based ensembles in the context of real concept drift. We believe the main goal as well as the aforementioned sub-tasks have been accomplished. To support this claim, this thesis contributes to the state-of-the-art in the area of data stream mining in relation to concept drift. Specifically, five new concept drift detection methods and a new ensemble approach were proposed in Chapters 5, 6, and 7, respectively namely DMDDM and DMDDM-S, ODDK and HBBE, and EBE and KAPPA.

We summarise the findings of this study as follows:

Research Objective 1.1: To propose a novel concept drift detection method which detects concept drift more accurately and with less time and memory consumption compared to other drift detectors. To do this, we empirically compare our proposed drift detection with the existing ones using synthetic and real-world data streams considering different performance measures, e.g., detection delay, true detection, memory, and accuracy.

To achieve Research Objective 1.1: we introduced a new drift detector, called

DMDDM, which adapts the disagreement measure and uses its calculations with the Page-Hinkley test to detect concept drift. DMDDM is proposed to calculate the diversity of a classifier's response according to the evolving incoming data. Hence, according to the statistical PH test, instead of monitoring the error estimates, we monitor the diversity of a pair of classifiers using the fading factor. In this way, the PH test is triggered whenever the predictions of components (h_u and h_v) start to disagree in an unusual way. We evaluated the proposed method on synthetic and real-world datasets using three sets of experiments. The results of the experiments on the synthetic datasets indicate that DMDDM achieved the main requirements of a model operating in a changing environment. The proposed method detects drifts with shorter delay and with minimal detection runtime and memory usage compared to the existing methods.

Research Objective 1.2: To propose a semi-supervised drift detection method which detects concept drift in the absence of class labels more accurately compared to other fully supervised drift detectors.

To achieve Research Objective 1.2: we introduced a new drift detector, called DMDDM-S, which adapts the disagreement measure and uses its calculations with the PH test to detect concept drift. DMDDM-S is proposed to calculate the diversity of classifier responses according to the evolving incoming data. Therefore, instead of monitoring the error estimates, DMDDM-S monitors the diversity of a pair of classifiers using the fading factor. In this way, the PH test is triggered whenever the predictions of components (h_u and h_v) start to disagree in an unusual way. The results of the experiments on the synthetic datasets indicate that with a lack of class labels, DMDDM-S detects drifts with shorter delay and with minimal detection runtime and memory usage compared to the existing methods.

Research Objective 2.1: To devise a new formalism that facilitates a way to detect concept drifts in the multi-class problem with more accuracy, in less time and with less memory consumption compared to other drift detectors. To do this, we empirically compare our proposed drift detection method with the existing ones using synthetic and real-world data streams, considering different performance measures, e.g., detection delay, true detection, memory, and accuracy.

Research Objective 2.2: To devise a hybrid block-based ensemble which is a framework for multi-class classification in evolving data streams. To do this, we empirically compare our proposed framework with the existing ones using synthetic and real-world data

streams, considering different performance measures, e.g., memory, time and accuracy.

To achieve Research Objectives 2.1 and 2.2: By studying and analyzing the problem of concept drift in data streams, we propose a hybrid block-based ensemble paradigm (HBBE) which aims to combine the best of online drift detector (ODDK) and block-based weighting in order to enhance the reaction to sudden drifts and to respond to other types of concept drift. This paper proposed an online drift detector to capture concept drifts in a timely manner with less memory consumption. We proposed a new way of calculating diversity which was designed for a K-Class Problem, while the preliminary version of our works (DMDDM and DMDDM-S) was to address the binary classification problem. Furthermore, two experiments were conducted, the first to evaluate the online drift detector against eleven well-known concept drift detectors and the second to evaluate the hybrid block-based ensemble against existing ensemble approaches. The analysis of the experimental results show that our proposed algorithms perform better than other drift detectors and ensemble approaches.

Research Objective 3.1: To propose a model called entropy-based ensemble (EBE) which incorporates entropy as a drift detector into the evolving ensemble in a reliable, fast, and computationally efficient way. To do this, we empirically compare our proposed framework with the existing ones using synthetic and real-world data streams considering different performance measures, e.g., memory, time and accuracy.

To achieve Research Objective 3.1: We presented an algorithm, called entropy-based ensemble (EBE) to classify data streams. EBE is based on incorporating information entropy as a drift detector into the evolving ensemble. The objective of the proposed algorithm is to detect two types of concepts drifts, namely: sudden and gradual drifts and to handle them in an incremental way. Two experiments were conducted using synthetic data sets and a comparison was made with prequential evaluation. The first generated data sets without concept drift and the second generated data sets with concept drift. The results show that EBE improves accuracy in the presence of both sudden and gradual concept drift.

8.2 Future Work

Regarding future research, an abundance of fascinating avenues are available we explain as follow.

- *Semi-supervised or unsupervised drift detection and adaptation.* The current drift detection adaptation techniques assume that accurate labels of data instances are obtainable immediately after prediction, which means both detection and adaptation procedures are supervised. Nevertheless, in real world scenarios true labels, might not be readily available. Hence, learning ways to enhance drift detection and adaptation algorithms for semi-unsupervised conditions is crucial.
- *Concept-oriented data filtering.* Concept drift issues not only occur in data stream learning, they can also be ubiquitous in training and testing batch-based learning, if the training and testing data is gathered within a particular time interval instead of at a time point. For instance, in the training and testing data collected in 2017 for client churn prediction, the readily available data included a number of concepts. The knowledge patterns might differ in different months or perhaps over several days, and also the most useful information for client churn prediction in 2018 might simply be found in December of 2017. Even though the cross-validation strategy can reduce the overfitting issues of the model built based on the whole dataset of 2017, it might not be the right answer. Concept drift concerns the current challenges of overfitting and underfitting problems. One possible solution is concept-oriented data filtering.
- *Video stream concept drift analysis.* Since videos are a type of streaming data, it could be easy to utilize several of the newly suggested techniques for video examination associated uses. Additionally, the research studies are associated with the connections and the differences involving concept drift adaptation and visible domain adaptation strategies. Finally, handling concept drift is an urgent and important issue. It is a key technique in achieving adaptive systems. Future research on the adaptivity of machine learning techniques and systems to address concept drift has great prospects
- *Class Imbalance issue.* Class imbalance is an established issue in machine learning in which the minority class contains a small prior probability compared to the majority class. To address this concern, detecting any drift with regard to the minority class could be difficult.

- *Data streams in big data.* Detecting different types of concept drift in a big data stream is very challenging, where improving the accuracy, and minimizing the processing time and memory usage are desirable.

Finally, handling concept drift is an urgent and important issue. It is a key technique in achieving adaptive systems. The future research on the adaptivity of machine learning techniques and systems to concept drift has great prospects.

Chapter A

Appendix: Pseudocodes of Online Learning Algorithms

A.1 Naive Bayes

Algorithm A.1 Naive Bayes (NB) Pseudocode

```

1: function INITIALIZE(attributes, labels)
2:   (A, Y)  $\leftarrow$  (attributes, labels)
3:   foreach class c in C do
4:      $N(c) = 0$ 
5:    $n = 0$ 
6:   return

7: function TRAIN(x, y)
8:   Increase  $n$  by 1
9:   Increase number  $N(y)$  by 1
10:  Calculate probability  $P(y) = N(y)/n$ 
11:  foreach attribute  $a_i \in A$  that holds the value of  $x_i$  in x do
12:    Increase number  $N(x_i|y)$  by 1
13:    foreach class c in C do
14:      Calculate probability  $P(x_i|c) = N(x_i|c)/N(c)$ 
15:  return

16: function TEST(x)
17:  foreach class c in C do
18:    Calculate likelihood  $P(\mathbf{x}|c) = \prod_{k=1}^{|\mathbf{x}|} P(x_k|c)$ 
19:    Calculate relative-likelihood  $Rel(c|\mathbf{x}) = P(\mathbf{x}|c) \cdot P(c)$ 
20:  return the class with the maximum relative-likelihood

```

Figure A.1: Pseudocode of Naive Bayes

A.2 Decision Stump

Algorithm A.2 Decision Stump (DS) Pseudocode

```

1: function INITIALIZE(attributes, labels)
2:   (A, Y)  $\leftarrow$  (attributes, labels)
3:   foreach class c in C do
4:      $N(c) = 0$ 
5:     foreach value v of a  $\in A$  do
6:        $N(v|c) = 0$ 
7:   return

8: function TRAIN(x, y)
9:   Increase n by 1
10:  Increase  $N(y)$  by 1
11:  for attribute  $a_i \in A$  that holds the value of  $x_i \in \mathbf{x}$  do
12:    Increase number  $N(x_i|y)$  by 1
13:  foreach class c in Y do
14:     $Entropy_{Total} += -(N(c)/n) \cdot \log_2(N(c)/n)$ 
15:  foreach attribute a  $\in A$  do
16:    for each value of v of attribute a do
17:      foreach c in C do
18:         $P(c|v) = N(c|v)/N(v)$ 
19:         $I(v) += -P(c|v) \cdot \log_2 P(c|v)$ 
20:       $Entropy_A += (N(v)/n) \times I(v)$ 
21:     $Gain_A = Entropy_{Total} - Entropy_A$ 
22:  Assign the attribute with the highest ‘gain’ as the stump.
23:  return

24: function TEST(x)
25:  Sort x into a leaf based on the attribute assigned as the stump.
26:  return the label at the leaf

```

Figure A.2: Pseudocode of Decision Stump

A.3 Hoeffding Tree

Algorithm A.3 Hoeffding Tree (HT) Pseudocode

output classifier:
 HT a decision tree

classifier variables:
 S a sequence of examples
 X a set of discrete attributes
 δ allowed of probability of choosing incorrect attribute at any given node
 τ a user specified tie threshold
 G(.) a split evaluation function, e.g. information gain

```

1: function INITIALIZE(attributes, delta, tau)
2:   (X,  $\delta$ ,  $\tau$ )  $\leftarrow$  (attributes, delta, tau)
3:   Create a HT with a single leaf  $l_1$  (the root)
4:   for each class  $y_k$  do
5:     for each value of  $x_{ij}$  of each attribute  $X_i \in \mathbf{X}$  do
6:       Let  $n_{ijk}(l_1) = 0$ 
7:   return

8: function TRAIN(S)
9:   for each example (x, y) in S do
10:    Sort(x, y) into a leaf  $l$  using VFDT
11:    for each  $x_{ij}$  in x such that  $X_i \in \mathbf{X}$  do
12:      Increment  $n_{ijk}(l)$ 
13:    Label  $l$  with the majority class among the examples seen so far at  $l$ .
14:    if the examples seen so far at  $l$  are not all of the same class then
15:      Compute  $\overline{G}_l(X_i)$  for each attribute  $X_i \in \mathbf{X}$ 
16:      Let  $X_a$  be the attribute with the highest  $\overline{G}_l$ 
17:      Let  $X_b$  be the attribute with the second-highest  $\overline{G}_l$ 
18:      Compute bound  $\varepsilon = \sqrt{\frac{R^2}{2n_l} \ln \frac{1}{\delta}}$ 
19:      if  $X_a \neq X_b$  and ( $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \varepsilon$  or  $\varepsilon < \tau$ ) then
20:        Replace  $l$  by an internal node that splits on  $X_a$ 
21:        for each branch of the split do
22:          Add a new leaf  $l_m$ , and let  $X_m = \mathbf{X} - \{X_a\}$ 
23:          for each class  $y_k$  and each value  $x_{ij}$  of each attribute  $X_i \in X_m$  do
24:            Let  $n_{ijk}(l_m) = 0$ 
25:   return

26: function TEST(x)
27:   Traverse HT considering x
28:   Let  $c$  be the label of stopping node
29:   return c

```

Figure A.3: Pseudocode of Hoeffding Tree

A.4 Perceptron

Algorithm A.4 Perceptron (PR) Pseudocode

```

1: function INITIALIZE([learning rate], labels)
2:    $Y \leftarrow \text{labels}$ 
3:    $\eta \leftarrow [\text{learning rate}]$ 
4:   Let  $w$  be randomly initialized

5: function TRAIN( $\mathbf{X}, y$ )
6:   foreach class  $c$  in  $Y$  do
7:     if  $c = y$  then
8:        $\delta = (1 - \tilde{f}_{w_i}(\mathbf{x})) \cdot \tilde{f}_{w_i}(\mathbf{x}) \cdot (1 - \tilde{f}_{w_i}(\mathbf{x}))$ 
9:     else
10:       $\delta = (0 - \tilde{f}_{w_i}(\mathbf{x})) \cdot \tilde{f}_{w_i}(\mathbf{x}) \cdot (1 - \tilde{f}_{w_i}(\mathbf{x}))$ 
11:       $w_c = w_c + \eta \cdot \delta \cdot \mathbf{x}$ 

12: function TEST( $\mathbf{x}$ )
13:   return  $\text{argmax}_c \tilde{f}_{w_c}(\mathbf{x})$ 

```

Figure A.4: Pseudocode of Perceptron

A.5 K-Nearest Neighbours

Algorithm A.5 K-Nearest Neighbours (K-NN) Pseudocode

```

1: function INITIALIZE([num. of neighbours], [window size])
2:    $K \leftarrow [\text{num. of neighbours}]$ 
3:    $n \leftarrow [\text{window size}]$ 
4:    $W \leftarrow \text{Initialize a window with size of } n$ 

5: function LOAD( $\mathbf{x}, y$ )
6:   if window is full then
7:     drop an element from the tail of the window
8:   add instance  $(\mathbf{x}, y)$  into the window

9: function TEST( $\mathbf{x}$ )
10:  find the  $K$ -nearest neighbours of instance  $\mathbf{x}$ 
11:  [predicted class]  $\leftarrow$  majority class among  $K$ -nearest neighbours
12:  return [predicted class]

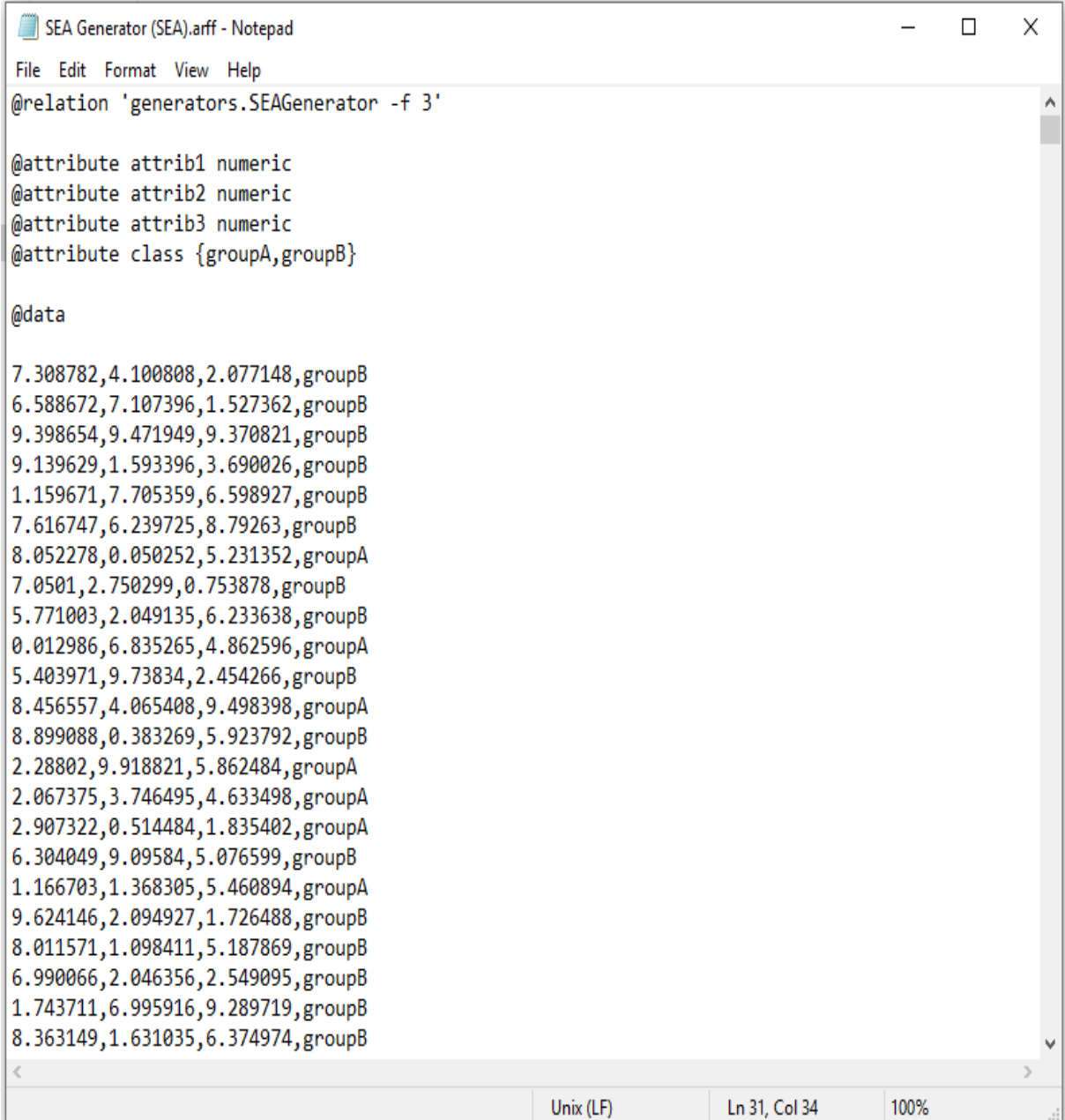
```

Figure A.5: Pseudocode of K-Nearest Neighbours

Chapter B

Appendix: Samples of Generated Datasets

B.1 SEA Generator (SEA)



```
SEA Generator (SEA).arff - Notepad
File Edit Format View Help
@relation 'generators.SEAGenerator -f 3'

@attribute attrib1 numeric
@attribute attrib2 numeric
@attribute attrib3 numeric
@attribute class {groupA,groupB}

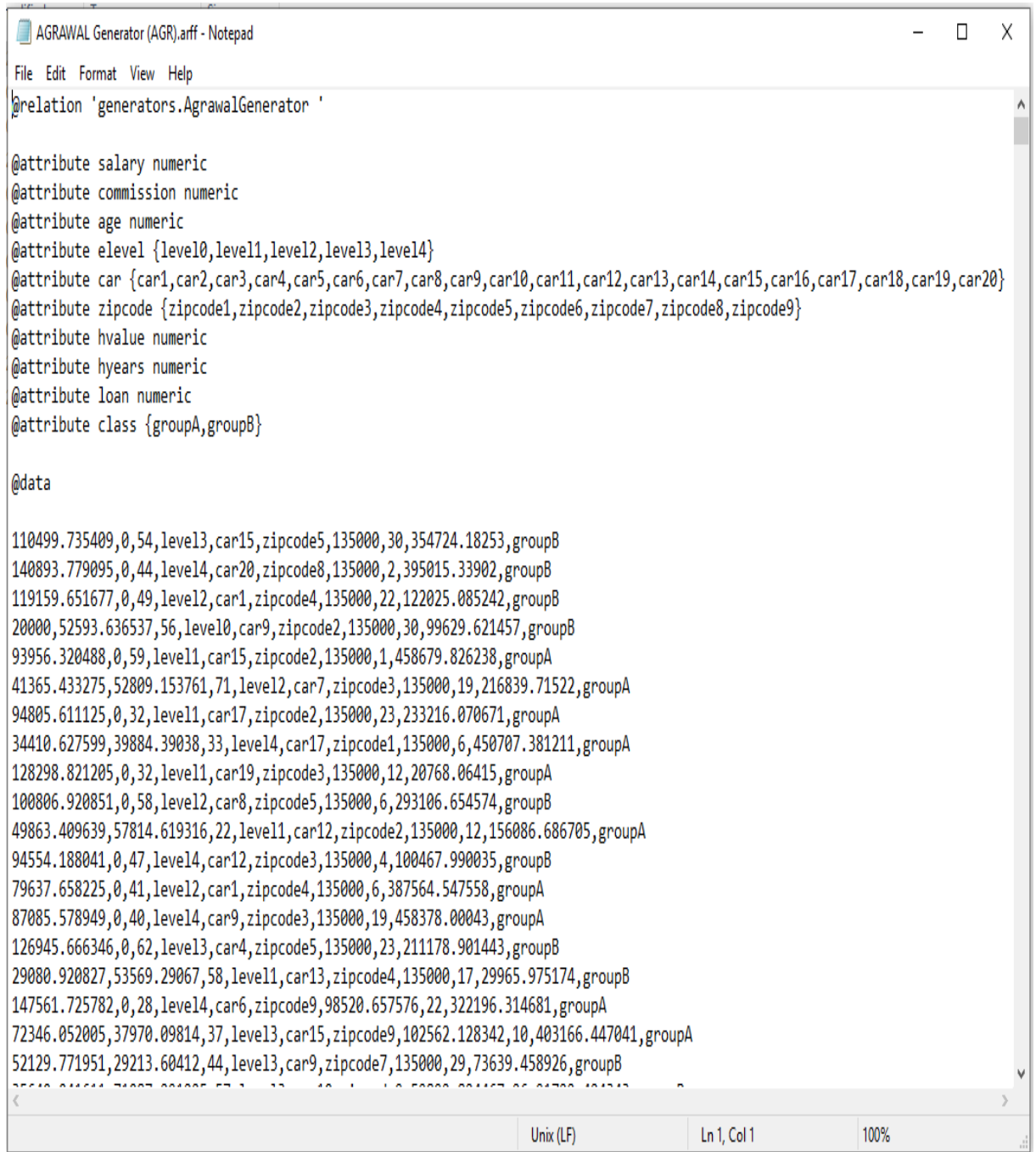
@data

7.308782,4.100808,2.077148,groupB
6.588672,7.107396,1.527362,groupB
9.398654,9.471949,9.370821,groupB
9.139629,1.593396,3.690026,groupB
1.159671,7.705359,6.598927,groupB
7.616747,6.239725,8.79263,groupB
8.052278,0.050252,5.231352,groupA
7.0501,2.750299,0.753878,groupB
5.771003,2.049135,6.233638,groupB
0.012986,6.835265,4.862596,groupA
5.403971,9.73834,2.454266,groupB
8.456557,4.065408,9.498398,groupA
8.899088,0.383269,5.923792,groupB
2.28802,9.918821,5.862484,groupA
2.067375,3.746495,4.633498,groupA
2.907322,0.514484,1.835402,groupA
6.304049,9.09584,5.076599,groupB
1.166703,1.368305,5.460894,groupA
9.624146,2.094927,1.726488,groupB
8.011571,1.098411,5.187869,groupB
6.990066,2.046356,2.549095,groupB
1.743711,6.995916,9.289719,groupB
8.363149,1.631035,6.374974,groupB

Unix (LF) Ln 31, Col 34 100%
```

Figure B.1: Sample of SEA Generator

B.2 AGRAWAL Generator (AGR)



```

AGRAWAL Generator (AGR).arff - Notepad
File Edit Format View Help
relation 'generators.AgrawalGenerator'

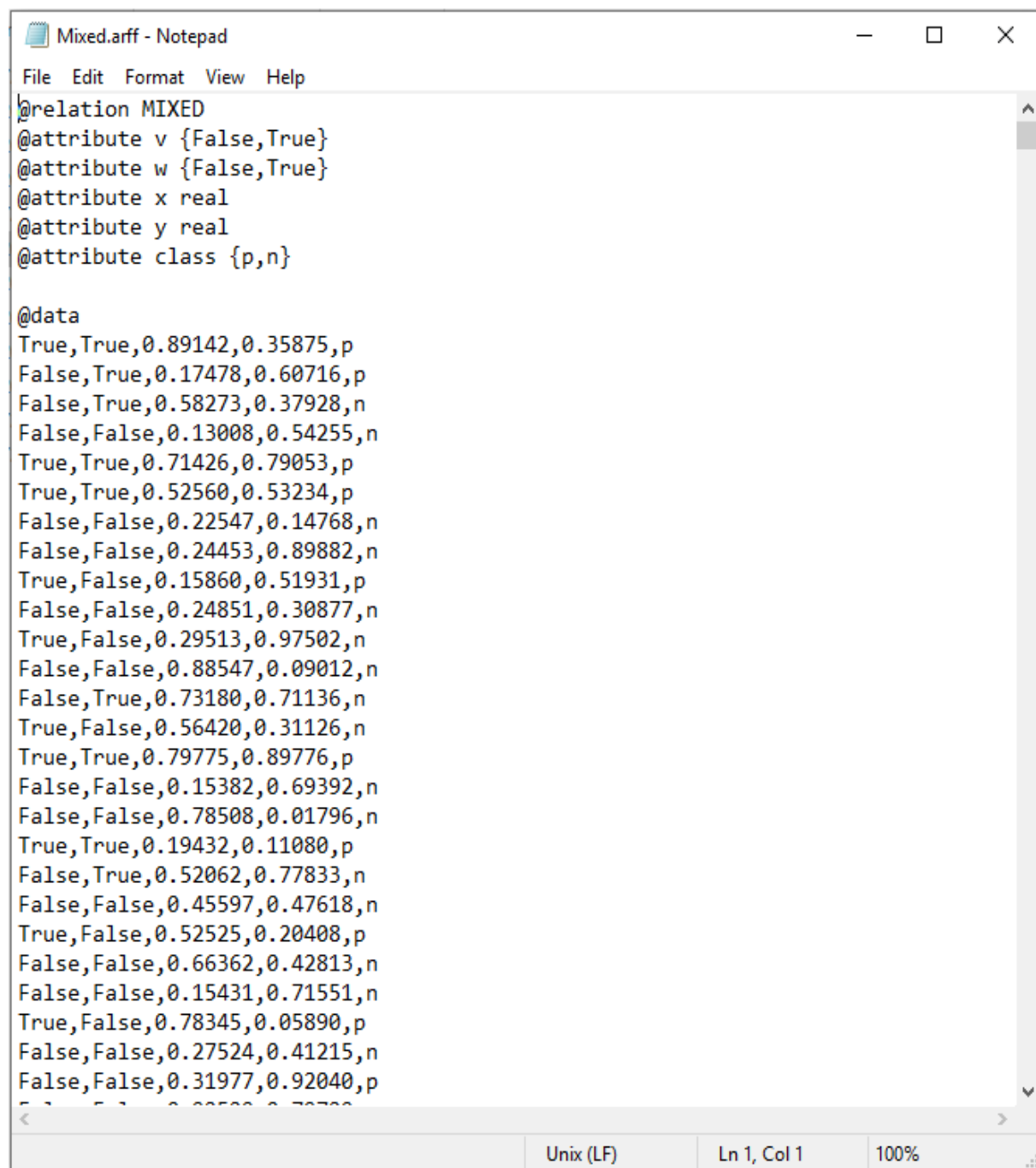
@attribute salary numeric
@attribute commission numeric
@attribute age numeric
@attribute elevel {level0,level1,level2,level3,level4}
@attribute car {car1,car2,car3,car4,car5,car6,car7,car8,car9,car10,car11,car12,car13,car14,car15,car16,car17,car18,car19,car20}
@attribute zipcode {zipcode1,zipcode2,zipcode3,zipcode4,zipcode5,zipcode6,zipcode7,zipcode8,zipcode9}
@attribute hvalue numeric
@attribute hyears numeric
@attribute loan numeric
@attribute class {groupA,groupB}

@data
110499.735409,0,54,level3,car15,zipcode5,135000,30,354724.18253,groupB
140893.779095,0,44,level4,car20,zipcode8,135000,2,395015.33902,groupB
119159.651677,0,49,level2,car1,zipcode4,135000,22,122025.085242,groupB
20000.52593.636537,56,level0,car9,zipcode2,135000,30,99629.621457,groupB
93956.320488,0,59,level1,car15,zipcode2,135000,1,458679.826238,groupA
41365.433275,52809.153761,71,level2,car7,zipcode3,135000,19,216839.71522,groupA
94805.611125,0,32,level1,car17,zipcode2,135000,23,233216.070671,groupA
34410.627599,39884.39038,33,level4,car17,zipcode1,135000,6,450707.381211,groupA
128298.821205,0,32,level1,car19,zipcode3,135000,12,20768.06415,groupA
100806.920851,0,58,level2,car8,zipcode5,135000,6,293106.654574,groupB
49863.409639,57814.619316,22,level1,car12,zipcode2,135000,12,156086.686705,groupA
94554.188041,0,47,level4,car12,zipcode3,135000,4,100467.990035,groupB
79637.658225,0,41,level2,car1,zipcode4,135000,6,387564.547558,groupA
87085.578949,0,40,level4,car9,zipcode3,135000,19,458378.00043,groupA
126945.666346,0,62,level3,car4,zipcode5,135000,23,211178.901443,groupB
29080.920827,53569.29067,58,level1,car13,zipcode4,135000,17,29965.975174,groupB
147561.725782,0,28,level4,car6,zipcode9,98520.657576,22,322196.314681,groupA
72346.052005,37970.09814,37,level3,car15,zipcode9,102562.128342,10,403166.447041,groupA
52129.771951,29213.60412,44,level3,car9,zipcode7,135000,29,73639.458926,groupB
35540.844544,71007.881005,57,1,13,40,1,1,3,50000,80,1457,86,61700,101113,groupA

```

Figure B.2: Sample of AGRAWAL Generator

B.3 Mixed



```

Mixed.arff - Notepad
File Edit Format View Help
@relation MIXED
@attribute v {False,True}
@attribute w {False,True}
@attribute x real
@attribute y real
@attribute class {p,n}

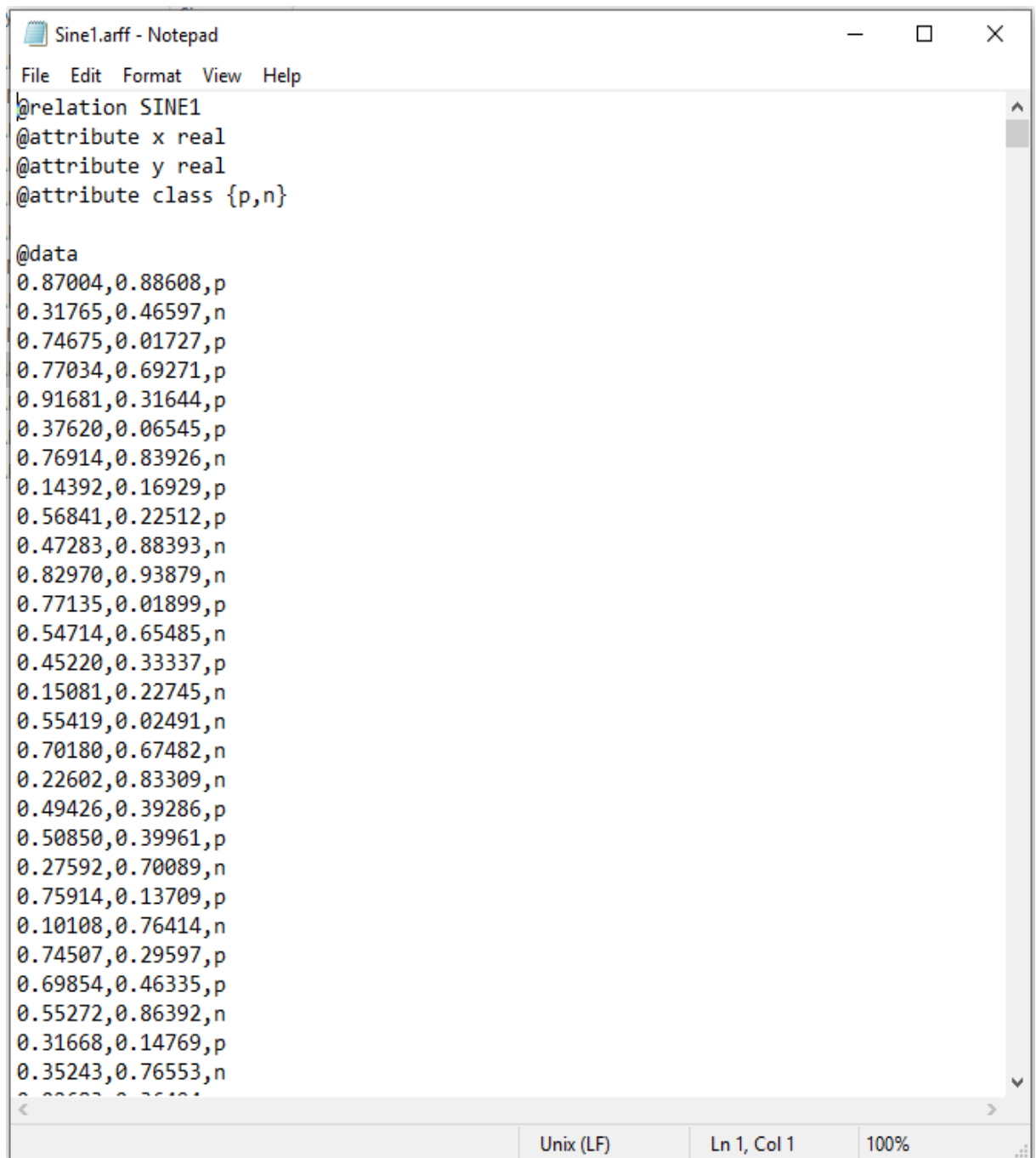
@data
True,True,0.89142,0.35875,p
False,True,0.17478,0.60716,p
False,True,0.58273,0.37928,n
False,False,0.13008,0.54255,n
True,True,0.71426,0.79053,p
True,True,0.52560,0.53234,p
False,False,0.22547,0.14768,n
False,False,0.24453,0.89882,n
True,False,0.15860,0.51931,p
False,False,0.24851,0.30877,n
True,False,0.29513,0.97502,n
False,False,0.88547,0.09012,n
False,True,0.73180,0.71136,n
True,False,0.56420,0.31126,n
True,True,0.79775,0.89776,p
False,False,0.15382,0.69392,n
False,False,0.78508,0.01796,n
True,True,0.19432,0.11080,p
False,True,0.52062,0.77833,n
False,False,0.45597,0.47618,n
True,False,0.52525,0.20408,p
False,False,0.66362,0.42813,n
False,False,0.15431,0.71551,n
True,False,0.78345,0.05890,p
False,False,0.27524,0.41215,n
False,False,0.31977,0.92040,p
False,False,0.00500,0.70700,p

```

Unix (LF) Ln 1, Col 1 100%

Figure B.3: Sample of Mixed

B.4 Sine1

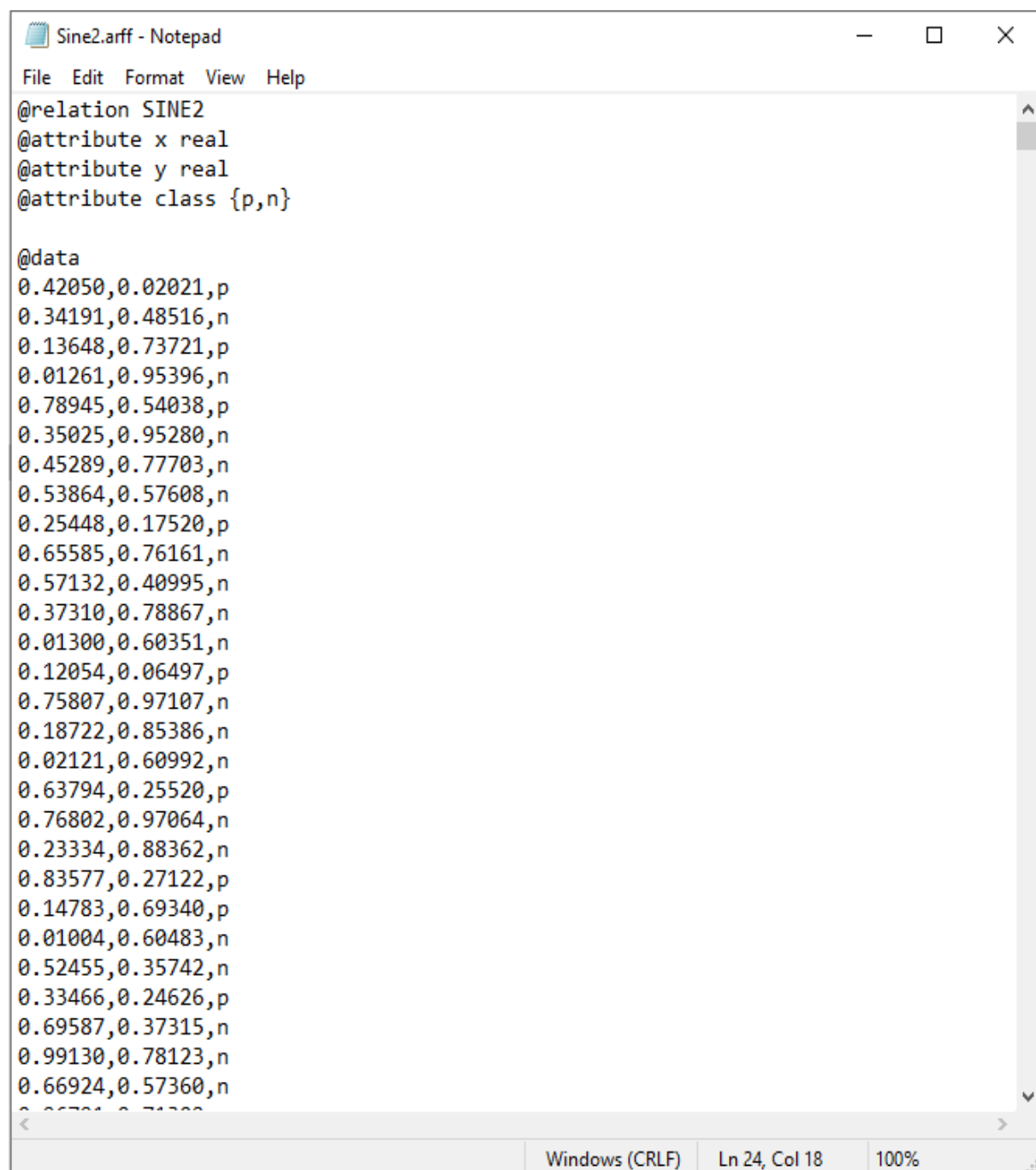


```
Sine1.arff - Notepad
File Edit Format View Help
@relation SINE1
@attribute x real
@attribute y real
@attribute class {p,n}

@data
0.87004,0.88608,p
0.31765,0.46597,n
0.74675,0.01727,p
0.77034,0.69271,p
0.91681,0.31644,p
0.37620,0.06545,p
0.76914,0.83926,n
0.14392,0.16929,p
0.56841,0.22512,p
0.47283,0.88393,n
0.82970,0.93879,n
0.77135,0.01899,p
0.54714,0.65485,n
0.45220,0.33337,p
0.15081,0.22745,n
0.55419,0.02491,n
0.70180,0.67482,n
0.22602,0.83309,n
0.49426,0.39286,p
0.50850,0.39961,p
0.27592,0.70089,n
0.75914,0.13709,p
0.10108,0.76414,n
0.74507,0.29597,p
0.69854,0.46335,p
0.55272,0.86392,n
0.31668,0.14769,p
0.35243,0.76553,n
0.66602,0.36401,n
```

Figure B.4: Sample of Sine1

B.5 Sine2



```
Sine2.arff - Notepad
File Edit Format View Help
@relation SINE2
@attribute x real
@attribute y real
@attribute class {p,n}

@data
0.42050,0.02021,p
0.34191,0.48516,n
0.13648,0.73721,p
0.01261,0.95396,n
0.78945,0.54038,p
0.35025,0.95280,n
0.45289,0.77703,n
0.53864,0.57608,n
0.25448,0.17520,p
0.65585,0.76161,n
0.57132,0.40995,n
0.37310,0.78867,n
0.01300,0.60351,n
0.12054,0.06497,p
0.75807,0.97107,n
0.18722,0.85386,n
0.02121,0.60992,n
0.63794,0.25520,p
0.76802,0.97064,n
0.23334,0.88362,n
0.83577,0.27122,p
0.14783,0.69340,p
0.01004,0.60483,n
0.52455,0.35742,n
0.33466,0.24626,p
0.69587,0.37315,n
0.99130,0.78123,n
0.66924,0.57360,n
0.00000,0.00000,n
```

Windows (CRLF) Ln 24, Col 18 100%

Figure B.5: Sample of Sine2

B.6 Wave

```
Wave.arff - Notepad
File Edit Format View Help
@relation 'generators.WaveformGenerator'

@attribute att1 numeric
@attribute att2 numeric
@attribute att3 numeric
@attribute att4 numeric
@attribute att5 numeric
@attribute att6 numeric
@attribute att7 numeric
@attribute att8 numeric
@attribute att9 numeric
@attribute att10 numeric
@attribute att11 numeric
@attribute att12 numeric
@attribute att13 numeric
@attribute att14 numeric
@attribute att15 numeric
@attribute att16 numeric
@attribute att17 numeric
@attribute att18 numeric
@attribute att19 numeric
@attribute att20 numeric
@attribute att21 numeric
@attribute class {class1,class2,class3}

@data

-0.092386,-0.362416,1.163335,1.578042,0.163761,0.268926,0.715832,1.361112,0.3748
-0.352228,-0.656318,1.669843,1.355256,1.95123,3.34644,4.754577,2.728011,3.409077
-1.119112,0.535469,-1.161217,-0.751262,-0.13546,-0.301284,-0.799115,0.397832,0.2
0.426168,-1.04388,0.426038,-0.249019,0.324346,1.120927,1.3601,0.497332,0.976824,
-1.994486,1.105256,1.598198,0.622171,0.695333,1.870191,3.038291,3.644945,5.17685
0.677905,0.927419,-1.594822,-0.421577,-0.043433,-1.538777,0.716338,1.013481,3.18
-0.648082,0.928569,0.825683,-2.515433,-0.045903,-0.290725,0.957661,0.516516,2.12
0.102332,1.106881,0.70015,0.005505,0.677035,1.700332,1.700102,1.600050,1.102332
<
Unix (LF) Ln 1, Col 1 100%
```

Figure B.6: Sample of Wave

B.7 RBF GR

```
RBF_GR.arff - Notepad
File Edit Format View Help
@relation 'generators.RandomRBFGenerator -c 4 -a 20'

@attribute att1 numeric
@attribute att2 numeric
@attribute att3 numeric
@attribute att4 numeric
@attribute att5 numeric
@attribute att6 numeric
@attribute att7 numeric
@attribute att8 numeric
@attribute att9 numeric
@attribute att10 numeric
@attribute att11 numeric
@attribute att12 numeric
@attribute att13 numeric
@attribute att14 numeric
@attribute att15 numeric
@attribute att16 numeric
@attribute att17 numeric
@attribute att18 numeric
@attribute att19 numeric
@attribute att20 numeric
@attribute class {class1,class2,class3,class4}

@data

0.31746413517119204,0.18885698499550185,0.9484388993060071,0.8429947861613085,0.
0.7735136037431646,0.5856795724996723,0.5191157959948152,0.47210052355734994,1.1
0.5282473701595095,0.9692400720777171,-0.12878364441196088,0.5453169394831847,0.
0.6584705539476755,0.7427245345868463,0.3578176993700374,1.0035171162679608,0.42
0.9031124701564174,0.2579691459020286,0.7599693421935043,0.9657884348160793,0.59
-0.1528430649926002,0.9685111920783267,0.6231607472980613,0.4757460878278988,-0.
0.7325601413228381,0.44863548914825985,-0.08712092991985931,0.824073407860623,1.
0.7949211069366092,0.8363790927101669,0.43358769285277443,0.3412492497617531,0.2
0.6888888888888889,0.6888888888888889,0.6888888888888889,0.6888888888888889,0.6888888888888889
```

Figure B.7: Sample of RBF GR

B.8 Tree R

```

Tree_R.arff - Notepad
File Edit Format View Help
@relation 'generators.RandomTreeGenerator -c 4'

@attribute nominal1 {value1,value2,value3,value4,value5}
@attribute nominal2 {value1,value2,value3,value4,value5}
@attribute nominal3 {value1,value2,value3,value4,value5}
@attribute nominal4 {value1,value2,value3,value4,value5}
@attribute nominal5 {value1,value2,value3,value4,value5}
@attribute numeric1 numeric
@attribute numeric2 numeric
@attribute numeric3 numeric
@attribute numeric4 numeric
@attribute numeric5 numeric
@attribute class {class1,class2,class3,class4}

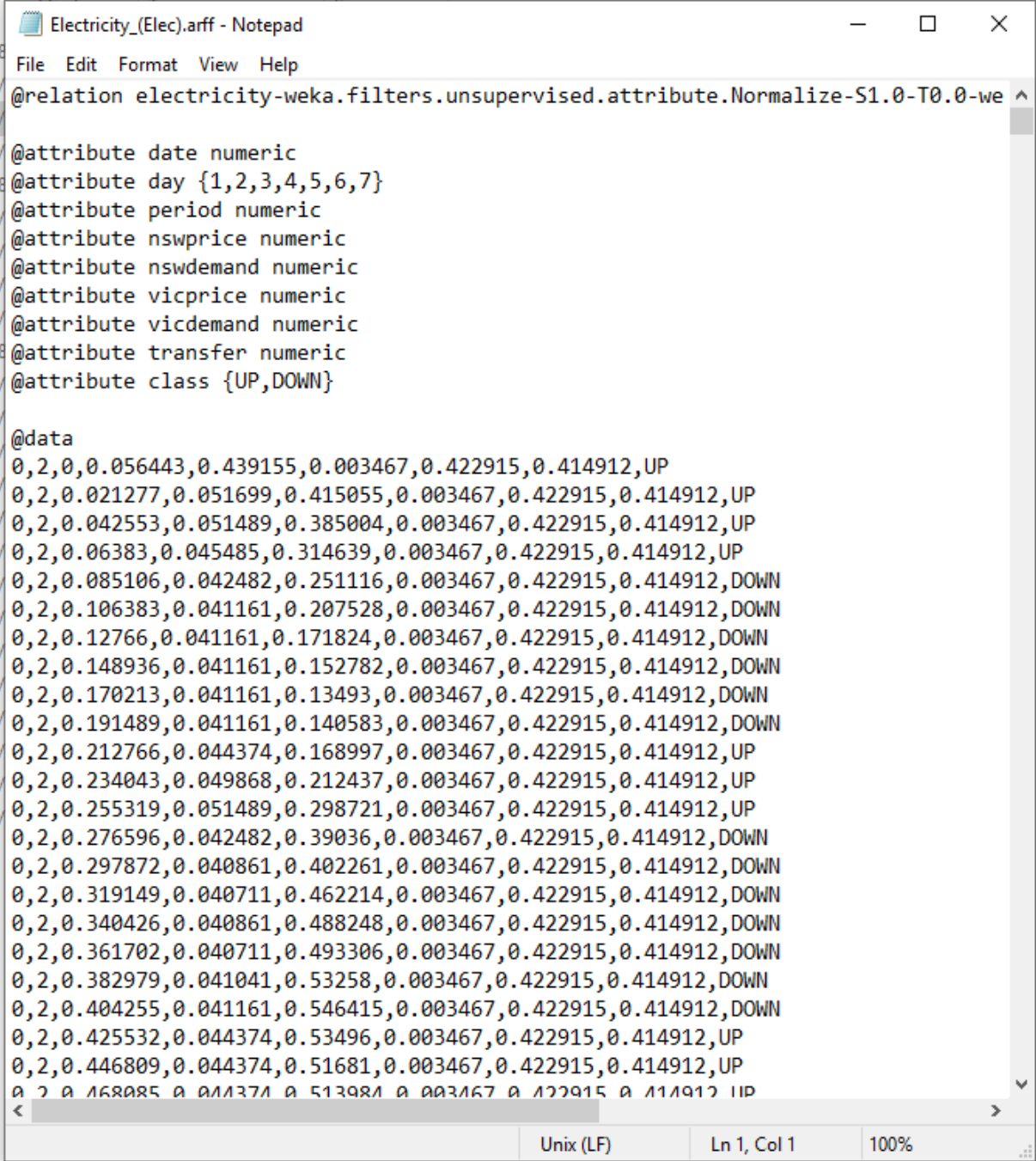
@data

value1,value4,value3,value4,value5,0.0362353821503677,0.6588672394716129,0.71073
value5,value3,value3,value2,value5,0.3971743421847056,0.34751802920311026,0.2940
value1,value5,value1,value2,value4,0.7616746929019993,0.623972537215988,0.879262
value3,value1,value4,value1,value3,0.14202270321592614,0.481728301575598,0.54455
value1,value2,value2,value4,value4,0.6835265034592647,0.48625958231173216,0.4875
value3,value3,value2,value2,value3,0.43200687989353914,0.2331557947513334,0.8899
value3,value2,value5,value1,value4,0.5862483975935397,0.9622148157170044,0.26515
value5,value4,value4,value4,value3,0.9989808147679288,0.6304048639228539,0.90958
value4,value4,value1,value2,value2,0.4946194136502534,0.01538437917072677,0.3589
value2,value5,value2,value2,value2,0.6990066224344805,0.20463555220568552,0.2549
value5,value3,value4,value3,value2,0.09014655892705203,0.437422687374017,0.32078
value2,value2,value2,value4,value1,0.7233716462158025,0.5094929196703692,0.98710
value1,value1,value1,value2,value4,0.7077681792059891,0.26590316146240056,0.0634
value3,value2,value1,value5,value2,0.679935420191025,0.280001615782666,0.6039836
value5,value2,value2,value5,value4,0.8257518968619348,0.5397618625049699,0.64183
value4,value1,value2,value2,value4,0.4577001830735099,0.8624497547615505,0.90192
value2,value5,value4,value5,value3,0.6304712233889196,0.552807920683817,0.543419
value5,value2,value3,value5,value1,0.3377209272639704,0.6073350789607563,0.31542
value4,value3,value1,value2,value4,0.0420465272087327,0.8242939221199226,0.57742
value4,value1,value5,value2,value4,0.7716011091589238,0.3061827881053399,0.88059

```

Figure B.8: Sample of Tree R

B.9 Electricity (Elec)

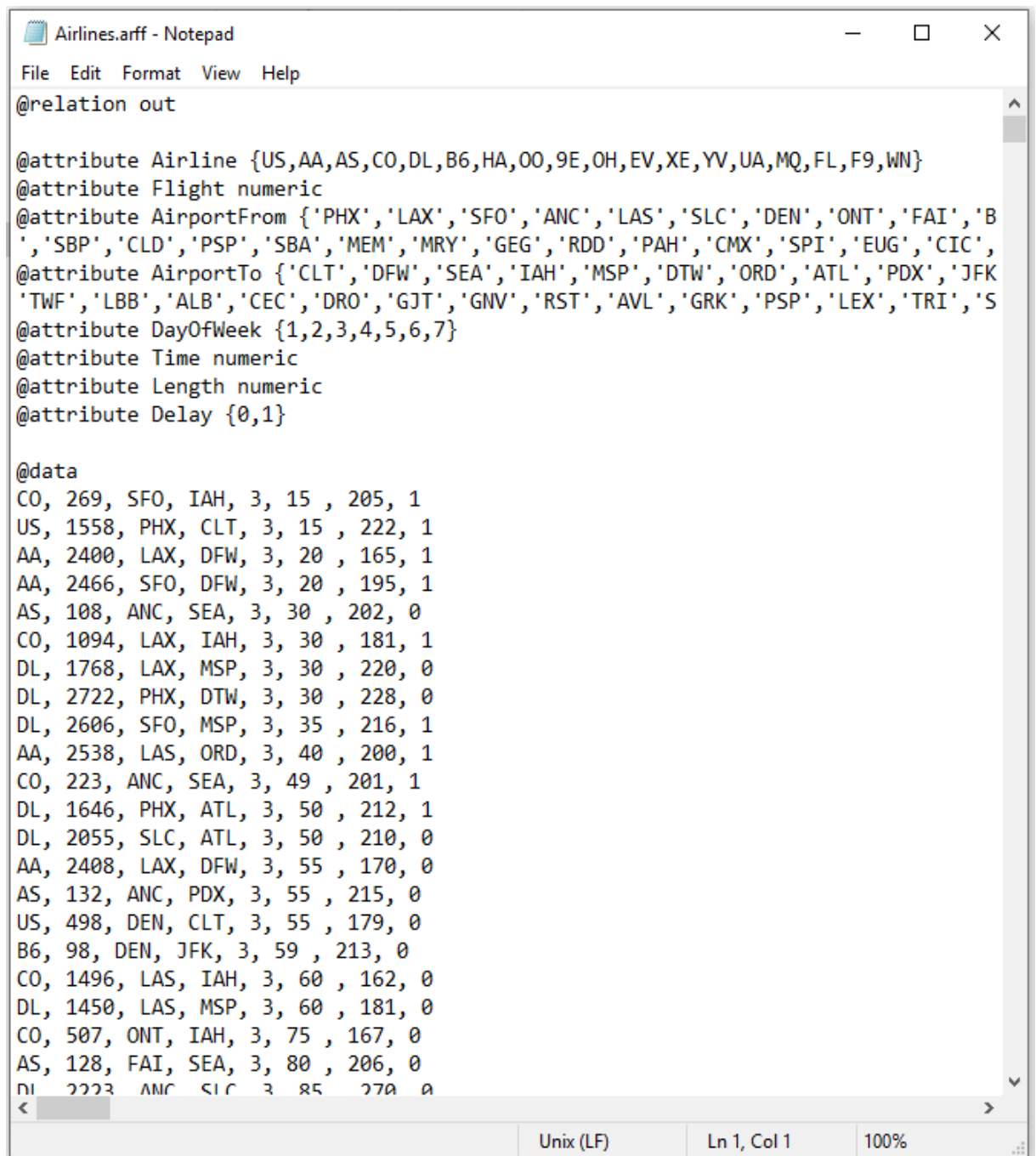


```
Electricity_(Elec).arff - Notepad
File Edit Format View Help
@relation electricity-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0-we
@attribute date numeric
@attribute day {1,2,3,4,5,6,7}
@attribute period numeric
@attribute nswprice numeric
@attribute nswdemand numeric
@attribute vicprice numeric
@attribute vicedemand numeric
@attribute transfer numeric
@attribute class {UP,DOWN}

@data
0,2,0,0.056443,0.439155,0.003467,0.422915,0.414912,UP
0,2,0.021277,0.051699,0.415055,0.003467,0.422915,0.414912,UP
0,2,0.042553,0.051489,0.385004,0.003467,0.422915,0.414912,UP
0,2,0.06383,0.045485,0.314639,0.003467,0.422915,0.414912,UP
0,2,0.085106,0.042482,0.251116,0.003467,0.422915,0.414912,DOWN
0,2,0.106383,0.041161,0.207528,0.003467,0.422915,0.414912,DOWN
0,2,0.12766,0.041161,0.171824,0.003467,0.422915,0.414912,DOWN
0,2,0.148936,0.041161,0.152782,0.003467,0.422915,0.414912,DOWN
0,2,0.170213,0.041161,0.13493,0.003467,0.422915,0.414912,DOWN
0,2,0.191489,0.041161,0.140583,0.003467,0.422915,0.414912,DOWN
0,2,0.212766,0.044374,0.168997,0.003467,0.422915,0.414912,UP
0,2,0.234043,0.049868,0.212437,0.003467,0.422915,0.414912,UP
0,2,0.255319,0.051489,0.298721,0.003467,0.422915,0.414912,UP
0,2,0.276596,0.042482,0.39036,0.003467,0.422915,0.414912,DOWN
0,2,0.297872,0.040861,0.402261,0.003467,0.422915,0.414912,DOWN
0,2,0.319149,0.040711,0.462214,0.003467,0.422915,0.414912,DOWN
0,2,0.340426,0.040861,0.488248,0.003467,0.422915,0.414912,DOWN
0,2,0.361702,0.040711,0.493306,0.003467,0.422915,0.414912,DOWN
0,2,0.382979,0.041041,0.53258,0.003467,0.422915,0.414912,DOWN
0,2,0.404255,0.041161,0.546415,0.003467,0.422915,0.414912,DOWN
0,2,0.425532,0.044374,0.53496,0.003467,0.422915,0.414912,UP
0,2,0.446809,0.044374,0.51681,0.003467,0.422915,0.414912,UP
0,2,0.468085,0.044374,0.513984,0.003467,0.422915,0.414912,UP
```

Figure B.9: Sample of Electricity (Elec)

B.10 Airline



```
Airlines.arff - Notepad
File Edit Format View Help
@relation out

@attribute Airline {US,AA,AS,CO,DL,B6,HA,OO,9E,OH,EV,XE,YV,UA,MQ,FL,F9,WN}
@attribute Flight numeric
@attribute AirportFrom {'PHX','LAX','SFO','ANC','LAS','SLC','DEN','ONT','FAI','B
','SBP','CLD','PSP','SBA','MEM','MRY','GEG','RDD','PAH','CMX','SPI','EUG','CIC',
@attribute AirportTo {'CLT','DFW','SEA','IAH','MSP','DTW','ORD','ATL','PDX','JFK
'TWF','LBB','ALB','CEC','DRO','GJT','GNV','RST','AVL','GRK','PSP','LEX','TRI','S
@attribute DayOfWeek {1,2,3,4,5,6,7}
@attribute Time numeric
@attribute Length numeric
@attribute Delay {0,1}

@data
CO, 269, SFO, IAH, 3, 15 , 205, 1
US, 1558, PHX, CLT, 3, 15 , 222, 1
AA, 2400, LAX, DFW, 3, 20 , 165, 1
AA, 2466, SFO, DFW, 3, 20 , 195, 1
AS, 108, ANC, SEA, 3, 30 , 202, 0
CO, 1094, LAX, IAH, 3, 30 , 181, 1
DL, 1768, LAX, MSP, 3, 30 , 220, 0
DL, 2722, PHX, DTW, 3, 30 , 228, 0
DL, 2606, SFO, MSP, 3, 35 , 216, 1
AA, 2538, LAS, ORD, 3, 40 , 200, 1
CO, 223, ANC, SEA, 3, 49 , 201, 1
DL, 1646, PHX, ATL, 3, 50 , 212, 1
DL, 2055, SLC, ATL, 3, 50 , 210, 0
AA, 2408, LAX, DFW, 3, 55 , 170, 0
AS, 132, ANC, PDX, 3, 55 , 215, 0
US, 498, DEN, CLT, 3, 55 , 179, 0
B6, 98, DEN, JFK, 3, 59 , 213, 0
CO, 1496, LAS, IAH, 3, 60 , 162, 0
DL, 1450, LAS, MSP, 3, 60 , 181, 0
CO, 507, ONT, IAH, 3, 75 , 167, 0
AS, 128, FAI, SEA, 3, 80 , 206, 0
DL, 2223, ANC, SLC, 3, 85 , 270, 0
<
Unix (LF) Ln 1, Col 1 100%
```

Figure B.10: Sample of Airlines

References

- [1] J. Gama, *Knowledge discovery from data streams* (Chapman and Hall/ CRC Press, 1st ed, 2010).
- [2] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, “Advances in knowledge discovery and data mining,” (American Association for Artificial Intelligence, 1996).
- [3] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining* (Pearson Education India, 2016).
- [4] M. Bramer, *Principles of data mining*, vol. 180 (Springer, 2007).
- [5] C. C. Aggarwal, *Data streams: models and algorithms*, vol. 31 (Springer Science & Business Media, 2007).
- [6] R. S. M. de Barros, “Advances in data stream mining with concept drift,” (2017).
- [7] S. Wares, J. Isaacs, and E. Elyan, “Data stream mining: methods and challenges for handling concept drift,” *SN Applied Sciences* **1**(11), 1412 (2019).
- [8] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM computing surveys (CSUR)* **46**(4), 1–37 (2014).
- [9] A. Chakrabarti, G. Cormode, and A. McGregor, “A near-optimal algorithm for computing the entropy of a stream,” in *SODA*, vol. 7, pp. 328–335 (Citeseer, 2007).
- [10] H. Wang, W. Fan, P. S. Yu, and J. Han, “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 (2003).
- [11] J. Gama and M. M. Gaber, *Learning from data streams: processing techniques in sensor networks* (Springer, 2007).
- [12] N. Trigoni, A. Guitton, and A. Skordylis, “Learning from Data Streams: Processing Techniques in Sensor Networks. Chapter 6: Querying of Sensor Data,” (2007).

- [13] K. Lorincz, D. J. Malan, T. R. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, “Sensor networks for emergency response: challenges and opportunities,” *IEEE pervasive Computing* **3**(4), 16–23 (2004).
- [14] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, “Habitat monitoring with sensor networks,” *Communications of the ACM* **47**(6), 34–40 (2004).
- [15] J. Smailović, M. Grčar, N. Lavrač, and M. Žnidaršič, “Stream-based active learning for sentiment analysis in the financial domain,” *Information sciences* **285**, 181–203 (2014).
- [16] M. J. Procopio, J. Mulligan, and G. Grudic, “Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments,” *Journal of Field Robotics* **26**(2), 145–175 (2009). <https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20279>, URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20279>.
- [17] G. Kreml, I. Žliobaite, D. Brzeziundefinedski, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, and J. Stefanowski, “Open Challenges for Data Stream Mining Research,” *SIGKDD Explor. Newsl.* **16**(1), 1–10 (2014). URL <https://doi.org/10.1145/2674026.2674028>.
- [18] A. S. Iwashita and J. P. Papa, “An Overview on Concept Drift Learning,” *IEEE Access* **7**, 1532–1547 (2019).
- [19] P. Domingos and G. Hulten, “Mining High-Speed Data Streams,” in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’00, p. 71–80 (Association for Computing Machinery, New York, NY, USA, 2000). URL <https://doi.org/10.1145/347090.347107>.
- [20] W. N. Street and Y. Kim, “A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification,” in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’01, p. 377–382 (Association for Computing Machinery, New York, NY, USA, 2001). URL <https://doi.org/10.1145/502512.502568>.
- [21] L. I. Kuncheva, “Classifier Ensembles for Changing Environments,” in *Multiple Classifier Systems*, F. Roli, J. Kittler, and T. Windeatt, eds., pp. 1–15 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004).
- [22] Y. Koren, “Collaborative Filtering with Temporal Dynamics,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’09, p. 447–456 (Association for Computing Machinery, New York, NY, USA, 2009). URL <https://doi.org/10.1145/1557019.1557072>.

-
- [23] Y. Ding and X. Li, “Time Weight Collaborative Filtering,” in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM ’05, p. 485–492 (Association for Computing Machinery, New York, NY, USA, 2005). URL <https://doi.org/10.1145/1099554.1099689>.
 - [24] A. Tsymbal, “The problem of concept drift: definitions and related work,” (2004).
 - [25] Janardan and S. Mehta, “Concept drift in Streaming Data Classification: Algorithms, Platforms and Issues,” *Procedia Computer Science* **122**, 804 – 811 (2017). 5th International Conference on Information Technology and Quantitative Management, ITQM 2017, URL <http://www.sciencedirect.com/science/article/pii/S1877050917326881>.
 - [26] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. (Prentice Hall Press, USA, 2009).
 - [27] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)* (Springer-Verlag, Berlin, Heidelberg, 2006).
 - [28] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning* (The MIT Press, 2012).
 - [29] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011).
 - [30] P. Flach, *Machine Learning: The art and science of algorithms that make sense of data* (Cambridge University Press, United Kingdom, 2012).
 - [31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (A Bradford Book, Cambridge, MA, USA, 2018).
 - [32] M. Bramer, *Principles of Data Mining*, 2nd ed. (Springer Publishing Company, Incorporated, 2013).
 - [33] J. Han, *Data Mining: Concepts and Techniques* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005).
 - [34] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective* (Cambridge University Press, USA, 2011).
 - [35] T. M. Mitchell, *Machine Learning*, 1st ed. (McGraw-Hill, Inc., USA, 1997).
 - [36] M. Sayed-Mouchaweh, *Learning from Data Streams in Dynamic Environments*, 1st ed. (Springer Publishing Company, Incorporated, 2015).
 - [37] A. Liu, J. Lu, F. Liu, and G. Zhang, “Accumulating Regional Density Dissimilarity for Concept Drift Detection in Data Streams,” *Pattern Recogn.* **76(C)**, 256–272 (2018). URL <https://doi.org/10.1016/j.patcog.2017.11.009>.

-
- [38] N. Lu, J. Lu, G. Zhang, and R. Lopez de Mantaras, “A Concept Drift-Tolerant Case-Base Editing Technique,” *Artif. Intell.* **230**(C), 108–133 (2016). URL <https://doi.org/10.1016/j.artint.2015.09.009>.
- [39] N. Lu, G. Zhang, and J. Lu, “Concept Drift Detection via Competence Models,” *Artif. Intell.* **209**, 11–28 (2014). URL <https://doi.org/10.1016/j.artint.2014.01.001>.
- [40] A. Liu, Y. Song, G. Zhang, and J. Lu, “Regional Concept Drift Detection and Density Synchronized Drift Adaptation,” in *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI’17*, p. 2280–2286 (AAAI Press, 2017).
- [41] A. Liu, G. Zhang, and J. Lu, “Fuzzy time windowing for gradual concept drift adaptation,” in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6 (2017).
- [42] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, “A Survey on Ensemble Learning for Data Stream Classification,” *ACM Comput. Surv.* **50**(2) (2017). URL <https://doi.org/10.1145/3054925>.
- [43] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, “Efficient Online Evaluation of Big Data Stream Classifiers,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’15*, p. 59–68 (Association for Computing Machinery, New York, NY, USA, 2015). URL <https://doi.org/10.1145/2783258.2783372>.
- [44] J. a. Gama, R. Sebastião, and P. P. Rodrigues, “On Evaluating Stream Learning Algorithms,” *Mach. Learn.* **90**(3), 317–346 (2013). URL <https://doi.org/10.1007/s10994-012-5320-9>.
- [45] I. Žliobaitė, A. Bifet, J. Read, B. Pfahringer, and G. Holmes, “Evaluation methods and decision theory for classification of streaming data with temporal dependence,” *Machine Learning* **98**, 455–482 (2014).
- [46] I. Stoica, D. Song, R. A. Popa, D. A. Patterson, M. W. Mahoney, R. H. Katz, A. D. Joseph, M. I. Jordan, J. M. Hellerstein, J. E. Gonzalez, K. Goldberg, A. Ghodsi, D. E. Culler, and P. Abbeel, “A Berkeley View of Systems Challenges for AI,” *CoRR* **abs/1712.05855** (2017). 1712.05855, URL <http://arxiv.org/abs/1712.05855>.
- [47] R. S. M. Barros and S. G. T. C. Santos, “A large-scale comparison of concept drift detectors,” *Information Sciences* **451–452**, 348 – 370 (2018). URL <http://www.sciencedirect.com/science/article/pii/S0020025518302743>.

-
- [48] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with Drift Detection,” in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, eds., pp. 286–295 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004).
- [49] H. Abdulsalam, D. B. Skillicorn, and P. Martin, “Classification Using Streaming Random Forests,” *IEEE Transactions on Knowledge and Data Engineering* **23**(1), 22–36 (2011).
- [50] J. Z. Kolter and M. A. Maloof, “Dynamic weighted majority: a new ensemble method for tracking concept drift,” in *Third IEEE International Conference on Data Mining*, pp. 123–130 (2003).
- [51] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms* (Wiley-Interscience, New York, NY, USA, 2004).
- [52] A. Cano and B. Krawczyk, “Kappa Updated Ensemble for drifting data stream mining,” *Machine Learning* **109**(1), 175–218 (2020).
- [53] M. Kubat and G. Widmer, “Adapting to Drift in Continuous Domains (Extended Abstract),” in *Proceedings of the 8th European Conference on Machine Learning, ECML’95*, p. 307–310 (Springer-Verlag, Berlin, Heidelberg, 1995). URL https://doi.org/10.1007/3-540-59286-5_74.
- [54] M. Baena-Garc, J. del Campo Ávila, A. Bifet, R. Gavald, and R. Morales-Bueno, “Early Drift Detection Method,” (2005).
- [55] K. Nishida and K. Yamauchi, “Detecting Concept Drift Using Statistical Testing,” in *Discovery Science*, V. Corruble, M. Takeda, and E. Suzuki, eds., pp. 264–269 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007).
- [56] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, “New Ensemble Methods for Evolving Data Streams,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’09*, p. 139–148 (Association for Computing Machinery, New York, NY, USA, 2009). URL <https://doi.org/10.1145/1557019.1557041>.
- [57] I. Frías-Blanco, J. d. Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, “Online and Non-Parametric Drift Detection Methods Based on Hoeffding’s Bounds,” *IEEE Transactions on Knowledge and Data Engineering* **27**(3), 810–823 (2015).
- [58] M. M. Black and R. J. Hickey, “Classification of Customer Call Data in the Presence of Concept Drift and Noise,” in *Proceedings of the First International Conference on Computing in an Imperfect World*, Soft-Ware 2002, p. 74–87 (Springer-Verlag, Berlin, Heidelberg, 2002).

-
- [59] T. M. Mitchell, J. G. Carbonell, and R. S. Michalski, eds., *Machine Learning: A Guide to Current Research* (Kluwer Academic Publishers, USA, 1986).
- [60] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, “A Survey on Data Stream Clustering and Classification,” *Knowl. Inf. Syst.* **45**(3), 535–569 (2015). URL <https://doi.org/10.1007/s10115-014-0808-1>.
- [61] A. Bifet and R. Kirkby, “Data stream mining a practical approach,” (2009).
- [62] P. M. Domingos and G. Hulten, “Catching up with the Data: Research Issues in Mining Data Streams,” in *DMKD* (2001).
- [63] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning*, 1st ed. (Springer Publishing Company, Incorporated, 2011).
- [64] A. Bifet, “Adaptive Learning and Mining for Data Streams and Frequent Patterns,” *SIGKDD Explor. Newsl.* **11**(1), 55–56 (2009). URL <https://doi.org/10.1145/1656274.1656287>.
- [65] J. a. Gama, R. Sebastião, and P. P. Rodrigues, “Issues in Evaluation of Stream Learning Algorithms,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’09, p. 329–338 (Association for Computing Machinery, New York, NY, USA, 2009). URL <https://doi.org/10.1145/1557019.1557060>.
- [66] A. Bifet and R. Gavaldà, “Learning from time-changing data with adaptive windowing,” in *Proceedings of the 2007 SIAM international conference on data mining*, pp. 443–448 (SIAM, 2007).
- [67] A. Bifet, G. Holmes, B. Pfahringer, and E. Frank, “Fast Perceptron Decision Tree Learning from Evolving Data Streams,” in *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD’10, p. 299–310 (Springer-Verlag, Berlin, Heidelberg, 2010). URL https://doi.org/10.1007/978-3-642-13672-6_30.
- [68] D. T. J. Huang, Y. S. Koh, G. Dobbie, and A. Bifet, “Drift Detection Using Stream Volatility,” in *Machine Learning and Knowledge Discovery in Databases*, A. Appice, P. P. Rodrigues, V. Santos Costa, C. Soares, J. Gama, and A. Jorge, eds., pp. 417–432 (Springer International Publishing, Cham, 2015).
- [69] W. Iba and P. Langley, “Induction of One-Level Decision Trees,” in *Proceedings of the Ninth International Workshop on Machine Learning*, ML ’92, p. 233–240 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992).
- [70] R. C. Holte, “Very Simple Classification Rules Perform Well on Most Commonly Used Datasets,” *Mach. Learn.* **11**(1), 63–90 (1993). URL <https://doi.org/10.1023/A:1022631118932>.

-
- [71] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. (O'Reilly Media, Inc., 2009).
- [72] J. Cattlet, "Megainduction: machine learning on very large databases," (1991).
- [73] W. Hoeffding, *Probability Inequalities for sums of Bounded Random Variables*, pp. 409–426 (Springer New York, New York, NY, 1994). URL https://doi.org/10.1007/978-1-4612-0865-5_26.
- [74] Y. Freund and R. E. Schapire, "Large Margin Classification Using the Perceptron Algorithm," in *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT' 98*, p. 209–217 (Association for Computing Machinery, New York, NY, USA, 1998). URL <https://doi.org/10.1145/279943.279985>.
- [75] D. Shen, G. Wu, and H.-I. Suk, "Deep Learning in Medical Image Analysis," *Annual Review of Biomedical Engineering* **19**(1), 221–248 (2017). PMID: 28301734, <https://doi.org/10.1146/annurev-bioeng-071516-044442>, URL <https://doi.org/10.1146/annurev-bioeng-071516-044442>.
- [76] B. W. Silverman and M. C. Jones, "E. Fix and J.L. Hodges (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951)," (1989).
- [77] K. Mouratidis and D. Papadias, "Continuous Nearest Neighbor Queries over Sliding Windows," *IEEE Transactions on Knowledge and Data Engineering* **19**(6), 789–803 (2007).
- [78] D. Brzezinski and J. Stefanowski, "Ensemble Diversity in Evolving Data Streams," in *Discovery Science*, T. Calders, M. Ceci, and D. Malerba, eds., pp. 229–244 (Springer International Publishing, Cham, 2016).
- [79] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, and K. Ghédira, "Discussion and review on evolving data streams and concept drift adapting," *Evolving Systems* **9**(1), 1–23 (2018).
- [80] S. J. Morshed, J. Rana, and M. Milrad, "Real-Time Data Analytics: An Algorithmic Perspective," in *International Conference on Data Mining and Big Data*, pp. 311–320 (Springer, 2016).
- [81] J. C. Schlimmer and R. H. Granger, "Incremental Learning from Noisy Data," *Mach. Learn.* **1**(3), 317–354 (1986). URL <https://doi.org/10.1023/A:1022810614389>.
- [82] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Mach. Learn.* **23**(1), 69–101 (1996). URL <https://doi.org/10.1023/A:1018046501280>.

-
- [83] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)* (Wiley-Interscience, USA, 2000).
 - [84] J. Gao, W. Fan, J. Han, and P. S. Yu, “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 (SIAM, 2007).
 - [85] M. G. Kelly, D. J. Hand, and N. M. Adams, “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 (ACM, 1999).
 - [86] I. Žliobaitė, “Learning under concept drift: an overview,” ArXiv Preprint ArXiv:1010.4784 (2010).
 - [87] W. Fan, Y. an Huang, H. Wang, and P. S. Yu, “Active Mining of Data Streams,” in *SDM* (2004).
 - [88] M. Salganicoff, “Tolerating Concept and Sampling Shift in Lazy Learning Using Prediction Error Context Switching,” *Artif. Intell. Rev.* **11**(1–5), 133–155 (1997). URL <https://doi.org/10.1023/A:1006515405170>.
 - [89] J. Gao, W. Fan, J. Han, and P. S. Yu, “A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions,” in *SDM* (2007).
 - [90] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle, “A Case-Based Technique for Tracking Concept Drift in Spam Filtering,” *Know.-Based Syst.* **18**(4–5), 187–195 (2005). URL <https://doi.org/10.1016/j.knosys.2004.10.002>.
 - [91] G. Widmer and M. Kubat, “Effective Learning in Dynamic Environments by Explicit Context Tracking,” in *Proceedings of the European Conference on Machine Learning*, ECML ’93, p. 227–243 (Springer-Verlag, Berlin, Heidelberg, 1993).
 - [92] M. M. Lazarescu, S. Venkatesh, and H. H. Bui, “Using Multiple Windows to Track Concept Drift,” *Intell. Data Anal.* **8**(1), 29–59 (2004).
 - [93] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection: A Survey,” *ACM Comput. Surv.* **41**(3) (2009). URL <https://doi.org/10.1145/1541880.1541882>.
 - [94] L. L. Minku, A. P. White, and X. Yao, “The Impact of Diversity on Online Ensemble Learning in the Presence of Concept Drift,” *IEEE Trans. on Knowl. and Data Eng.* **22**(5), 730–742 (2010). URL <http://dx.doi.org/10.1109/TKDE.2009.156>.
 - [95] P. Kosina, J. a. Gama, and R. Sebastião, “Drift Severity Metric,” in *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, p. 1119–1120 (IOS Press, NLD, 2010).

-
- [96] A. Pesaranghader, H. Viktor, and E. Paquet, “McDiarmid Drift Detection Methods for Evolving Data Streams,” 2018 International Joint Conference on Neural Networks (IJCNN) pp. 1–9 (2018).
- [97] A. Liu, G. Zhang, and J. Lu, “Fuzzy time windowing for gradual concept drift adaptation,” in *Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on*, pp. 1–6 (IEEE, 2017).
- [98] C. Alippi, W. Qi, and M. Roveri, “Learning in Nonstationary Environments: A Hybrid Approach,” in *International Conference on Artificial Intelligence and Soft Computing*, pp. 703–714 (Springer, 2017).
- [99] Z. Ahmadi and S. Kramer, “Modeling recurring concepts in data streams: a graph-based framework,” *Knowledge and Information Systems* **55**(1), 15–44 (2018).
- [100] P. Dhaliwal and M. Bhatia, “Effective Handling of Recurring Concept Drifts in Data Streams,” *Indian Journal of Science and Technology* **10**(30) (2017).
- [101] S. Sakthithasan and R. Pears, “Capturing recurring concepts using discrete Fourier transform,” *Concurrency and Computation: Practice and Experience* **28**(15), 4013–4035 (2016).
- [102] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, “Learning in nonstationary environments: A survey,” *IEEE Computational Intelligence Magazine* **10**(4), 12–25 (2015).
- [103] P. Sidhu and M. Bhatia, “A novel online ensemble approach to handle concept drifting data streams: diversified dynamic weighted majority,” *International Journal of Machine Learning and Cybernetics* **9**(1), 37–61 (2018).
- [104] Y. Geng and J. Zhang, “An Ensemble Classifier Algorithm for Mining Data Streams Based on Concept Drift,” in *Computational Intelligence and Design (ISCID), 2017 10th International Symposium on*, vol. 2, pp. 227–230 (IEEE, 2017).
- [105] P.-X. Loeffel, A. Bifet, C. Marsala, and M. Detyniecki, “Droplet Ensemble Learning on Drifting Data Streams,” in *International Symposium on Intelligent Data Analysis*, pp. 210–222 (Springer, 2017).
- [106] C. McDiarmid, *On the method of bounded differences*, p. 148–188, London Mathematical Society Lecture Note Series (Cambridge University Press, 1989).
- [107] R. Pears, S. Sakthithasan, and Y. S. Koh, “Detecting Concept Change in Dynamic Data Streams,” *Mach. Learn.* **97**(3), 259–293 (2014). URL <https://doi.org/10.1007/s10994-013-5433-9>.
- [108] J. S. Vitter, “Random Sampling with a Reservoir,” *ACM Trans. Math. Softw.* **11**(1), 37–57 (1985). URL <http://doi.acm.org/10.1145/3147.3165>.

- [109] S. Bernstein, “The theory of probabilities,” (1946).
- [110] R. S. M. de Barros, D. R. de Lima Cabral, P. Gonçalves, and S. G. T. de Carvalho Santos, “RDDM: Reactive drift detection method,” *Expert Syst. Appl.* **90**, 344–355 (2017).
- [111] A. Pesaranghader and H. L. Viktor, “Fast hoeffding drift detection method for evolving data streams,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 96–111 (Springer, 2016).
- [112] D. T. J. Huang, Y. S. Koh, G. Dobbie, and R. Pears, “Detecting Volatility Shift in Data Streams,” in *2014 IEEE International Conference on Data Mining*, pp. 863–868 (2014).
- [113] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, “Online and non-parametric drift detection methods based on Hoeffding’s bounds,” *IEEE Transactions on Knowledge and Data Engineering* **27**(3), 810–823 (2015).
- [114] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, “Exponentially weighted moving average charts for detecting concept drift,” *Pattern Recognition Letters* **33**(2), 191 – 198 (2012). URL <http://www.sciencedirect.com/science/article/pii/S0167865511002704>.
- [115] H. Wang, W. Fan, P. S. Yu, and J. Han, “Mining Concept-drifting Data Streams Using Ensemble Classifiers,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’03, pp. 226–235 (ACM, New York, NY, USA, 2003). URL <http://doi.acm.org/10.1145/956750.956778>.
- [116] B. Krawczyk and M. Woźniak, “Reacting to different types of concept drift with adaptive and incremental one-class classifiers,” in *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*, pp. 30–35 (IEEE, 2015).
- [117] N. Littlestone and M. K. Warmuth, “The Weighted Majority Algorithm,” *Inf. Comput.* **108**, 212–261 (1994).
- [118] K. Nishida, K. Yamauchi, and T. Omori, “ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments,” in *Proceedings of the 6th International Conference on Multiple Classifier Systems*, MCS’05, p. 176–185 (Springer-Verlag, Berlin, Heidelberg, 2005). URL https://doi.org/10.1007/11494683_18.
- [119] M. Deckert, “Batch Weighted Ensemble for Mining Data Streams with Concept Drift,” in *Proceedings of the 19th International Conference on Foundations of Intelligent Systems*, ISMIS’11, p. 290–299 (Springer-Verlag, Berlin, Heidelberg, 2011).

-
- [120] R. Elwell and R. Polikar, “Incremental Learning of Concept Drift in Nonstationary Environments,” *IEEE Transactions on Neural Networks* **22**(10), 1517–1531 (2011).
- [121] A. Blum and T. Mitchell, “Combining Labeled and Unlabeled Data with Co-training,” in *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT’ 98*, pp. 92–100 (ACM, New York, NY, USA, 1998). URL <http://doi.acm.org/10.1145/279943.279962>.
- [122] and, “Tri-training: exploiting unlabeled data using three classifiers,” *IEEE Transactions on Knowledge and Data Engineering* **17**(11), 1529–1541 (2005).
- [123] X. Zhu, “Semi-Supervised Learning Literature Survey,” (2006).
- [124] Z. Huang, “Clustering large data sets with mixed numeric and categorical values,” in *In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 21–34 (1997).
- [125] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, “New Ensemble Methods for Evolving Data Streams,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’09*, pp. 139–148 (ACM, New York, NY, USA, 2009). URL <http://doi.acm.org/10.1145/1557019.1557041>.
- [126] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with drift detection,” in *Brazilian Symposium on Artificial Intelligence*, pp. 286–295 (Springer, 2004).
- [127] M. K. Olorunnimbe, H. L. Viktor, and E. Paquet, “Intelligent Adaptive Ensembles for Data Stream Mining: A High Return on Investment Approach,” in *Proceedings of the 4th International Conference on New Frontiers in Mining Complex Patterns, NFMCP’15*, p. 61–75 (Springer, Gewerbestrasse 11 CH-6330, Cham (ZG), CHE, 2015).
- [128] M. K. Olorunnimbe, H. L. Viktor, and E. Paquet, “Dynamic adaptation of online ensembles for drifting data streams,” *Journal of Intelligent Information Systems* **50**(2), 291–313 (2018).
- [129] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive Online Analysis,” *Journal of Machine Learning Research* **11**, 1601–1604 (2010). URL <http://portal.acm.org/citation.cfm?id=1859903>.
- [130] J. Gama, R. Sebastião, and P. P. Rodrigues, “On evaluating stream learning algorithms,” *Machine Learning* **90**(3), 317–346 (2013). URL <https://doi.org/10.1007/s10994-012-5320-9>.
- [131] A. Patcha and J.-M. Park, “An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends,” *Comput. Netw.* **51**(12), 3448–3470 (2007). URL <https://doi.org/10.1016/j.comnet.2007.02.001>.

- [132] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, “A Multi-partition Multi-chunk Ensemble Technique to Classify Concept-Drifting Data Streams,” in *Advances in Knowledge Discovery and Data Mining*, T. Theeramunkong, B. Kijssirikul, N. Cercone, and T.-B. Ho, eds., pp. 363–375 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2009).
- [133] J. Kim, P. J. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross, “Immune System Approaches to Intrusion Detection — a Review,” *Natural Computing: An International Journal* **6**(4), 413–466 (2007). URL <https://doi.org/10.1007/s11047-006-9026-4>.
- [134] O. Mazhelis and S. Puuronen, “Comparing Classifier Combining Techniques for Mobile-Masquerader Detection,” in *The Second International Conference on Availability, Reliability and Security (ARES’07)*, pp. 465–472 (2007).
- [135] R. J. Bolton and D. J. H., “Statistical fraud detection: A review,” *Statistical Science* **17**, 2002 (2002).
- [136] F. A. Crespo and R. Weber, “A methodology for dynamic data mining based on fuzzy clustering,” *Fuzzy Sets Syst.* **150**, 267–284 (2005).
- [137] J. Zhou, L. Cheng, and W. Bischof, “Prediction and Change Detection In Sequential Data for Interactive Applications,” in *National Conference on Artificial Intelligence (AAAI)*, pp. 805–810 (AAAI, 2008).
- [138] J. Luo, A. Pronobis, B. Caputo, and P. Jensfelt, “Incremental learning for place recognition in dynamic environments,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 721–728 (2007).
- [139] L. Liao, D. J. Patterson, D. Fox, and H. Kautz, “Learning and Inferring Transportation Routines,” *Artif. Intell.* **171**(5–6), 311–331 (2007). URL <https://doi.org/10.1016/j.artint.2007.01.006>.
- [140] F. Mourão, L. Rocha, R. Araújo, T. Couto, M. Gonçalves, and W. Meira, “Understanding Temporal Aspects in Document Classification,” in *Proceedings of the 2008 International Conference on Web Search and Data Mining, WSDM ’08*, p. 159–170 (Association for Computing Machinery, New York, NY, USA, 2008). URL <https://doi.org/10.1145/1341531.1341554>.
- [141] P. De Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash, “AHA! The Adaptive Hypermedia Architecture,” in *Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia, HYPERTEXT ’03*, p. 81–84 (Association for Computing Machinery, New York, NY, USA, 2003). URL <https://doi.org/10.1145/900051.900068>.

-
- [142] F. Fdez-Riverola, E. L. Iglesias, F. Díaz, J. R. Méndez, and J. M. Corchado, “Applying Lazy Learning Algorithms to Tackle Concept Drift in Spam Filtering,” *Expert Syst. Appl.* **33**(1), 36–48 (2007). URL <https://doi.org/10.1016/j.eswa.2006.04.011>.
 - [143] N. Lathia, S. Hailes, and L. Capra, “KNN CF: A Temporal Social Network,” in *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys '08*, p. 227–234 (Association for Computing Machinery, New York, NY, USA, 2008). URL <https://doi.org/10.1145/1454008.1454044>.
 - [144] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research* **3**(Jan), 993–1022 (2003).
 - [145] C. Wang, D. Blei, and D. Heckerman, “Continuous time dynamic topic models,” *arXiv preprint arXiv:1206.3298* (2012).
 - [146] R. Giacomini and B. Rossi, “Detecting and predicting forecast breakdowns,” *The Review of Economic Studies* **76**(2), 669–705 (2009).
 - [147] R. Klinkenberg, “Meta-Learning, Model Selection, and Example Selection in Machine Learning Domains with Concept Drift.” in *LWA*, vol. 2005, pp. 164–171 (2005).
 - [148] P. R. Kumar and V. Ravi, “Bankruptcy prediction in banks and firms via statistical and intelligent techniques—A review,” *European journal of operational research* **180**(1), 1–28 (2007).
 - [149] T. K. Sung, N. Chang, and G. Lee, “Dynamics of modeling in data mining: interpretive approach to bankruptcy prediction,” *Journal of management information systems* **16**(1), 63–85 (1999).
 - [150] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, “Dynamic integration of classifiers for handling concept drift,” *Information fusion* **9**(1), 56–68 (2008).
 - [151] M. Black and R. Hickey, “Detecting and adapting to concept drift in bioinformatics,” in *International Symposium on Knowledge Exploration in Life Science Informatics*, pp. 161–168 (Springer, 2004).
 - [152] J. B. Gomes, C. Phua, and S. Krishnaswamy, “Where will you go? mobile data mining for next place prediction,” in *International Conference on Data Warehousing and Knowledge Discovery*, pp. 146–158 (Springer, 2013).
 - [153] M. J. Procopio, J. Mulligan, and G. Grudic, “Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments,” *Journal of Field Robotics* **26**(2), 145–175 (2009).

-
- [154] P. Rashidi and D. J. Cook, “Keeping the resident in the loop: Adapting the smart home to the user,” *IEEE Transactions on systems, man, and cybernetics-part A: systems and humans* **39**(5), 949–959 (2009).
 - [155] D. Charles, M. Mcneill, M. McAlister, M. Black, A. Moore, K. Stringer, J. Kücklich, and A. Kerr, “Player-centred game design: Player modelling and adaptive digital games,” (2005).
 - [156] D. B. Skalak, “The Sources of Increased Accuracy for Two Proposed Boosting Algorithms,” in *In Proc. American Association for Arti Intelligence, AAAI-96, Integrating Multiple Learned Models Workshop*, pp. 120–125 (1996).
 - [157] T. K. Ho, “The Random Subspace Method for Constructing Decision Forests,” *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(8), 832–844 (1998). URL <http://dx.doi.org/10.1109/34.709601>.
 - [158] C. Shannon, “A mathematical theory of communication,” *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **5**, 3–55 (2001).
 - [159] C. Ferri, J. Hernández-Orallo, and R. Modroiu, “An experimental comparison of performance measures for classification,” *Pattern Recognition Letters* **30**(1), 27–38 (2009).
 - [160] L. A. Jeni, J. F. Cohn, and F. De La Torre, “Facing imbalanced data—recommendations for the use of performance metrics,” in *2013 Humaine association conference on affective computing and intelligent interaction*, pp. 245–251 (IEEE, 2013).
 - [161] D. Brzezinski, J. Stefanowski, R. Susmaga, and I. Szczęch, “Visual-based analysis of classification measures and their properties for class imbalanced problems,” *Information Sciences* **462**, 242–261 (2018).
 - [162] A. Cano, A. Zafra, and S. Ventura, “Weighted data gravitation classification for standard and imbalanced data,” *IEEE transactions on cybernetics* **43**(6), 1672–1687 (2013).