A Reservoir of Adaptive Algorithms for Online Learning from Evolving Data Streams

by

Ali Pesaranghader

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
Doctorate in Philosophy degree in Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

To my beloved Mother,

Abstract

Continuous change and development are essential aspects of *evolving* environments and applications, including, but not limited to, smart cities, military, medicine, nuclear reactors, self-driving cars, aviation, and aerospace. That is, the fundamental characteristics of such environments may evolve, and so cause dangerous consequences, e.g., putting people lives at stake, if no reaction is adopted. Therefore, learning systems need to apply intelligent algorithms to *monitor* evolvement in their environments and *update* themselves effectively. Further, we may experience fluctuations regarding the performance of learning algorithms due to the nature of incoming data as it continuously evolves. That is, the current efficient learning approach may become deprecated after a change in data or environment. Hence, the question '*how to have an efficient learning algorithm over time against evolving data?*' has to be addressed. In this thesis, we have made two contributions to settle the challenges described above.

In the machine learning literature, the phenomenon of (distributional) change in data is known as *concept drift*. Concept drift may shift decision boundaries, and cause a decline in accuracy. Learning algorithms, indeed, have to detect concept drift in evolving data streams and replace their predictive models accordingly. To address this challenge, *adaptive* learners have been devised which may utilize drift detection methods to locate the drift points in dynamic and changing data streams. A drift detection method able to discover the drift points quickly, with the lowest false positive and false negative rates, is preferred. False positive refers to incorrectly alarming for concept drift, and false negative refers to not alarming for concept drift. In this thesis, we introduce three algorithms, called as the Fast Hoeffding Drift Detection Method (FHDDM), the Stacking Fast Hoeffding Drift Detection Method (FHDDMS), and the McDiarmid Drift Detection Methods (MDDMs), for detecting drift points with the minimum delay, false positive, and false negative rates. FHDDM is a sliding window-based algorithm and applies Hoeffding's inequality (Hoeffding, 1963) to detect concept drift. FHDDM slides its window over the prediction results, which are either 1 (for a correct prediction) or 0 (for a wrong prediction). Meanwhile, it compares the mean of elements inside the window with the maximum mean observed so far; subsequently, a significant difference between the two means, upper-bounded by the Hoeffding inequality, indicates the occurrence of concept drift. The FHDDMS extends the FHDDM algorithm by sliding multiple windows over its entries for a better drift detection regarding the detection delay and false negative rate. In contrast to FHDDM/S, the MDDM variants assign weights to their entries, i.e., higher weights are associated with the most recent entries in the sliding window, for faster detection of concept drift. The rationale is that recent examples reflect the ongoing situation adequately. Then, by putting higher weights on the latest entries, we may detect concept drift quickly. An MDDM algorithm bounds the difference between the weighted mean of elements in the sliding window and the maximum weighted mean seen so far, using McDiarmid's inequality (McDiarmid, 1989). Eventually, it alarms for concept

drift once a significant difference is experienced. We experimentally show that FHDDM/S and MDDMs outperform the state-of-the-art by representing promising results in terms of the adaptation and classification measures.

Due to the evolving nature of data streams, the performance of an adaptive learner, which is defined by the *classification*, *adaptation*, and *resource consumption* measures, may fluctuate over time. In fact, a learning algorithm, in the form of a (classifier, detector) pair, may present a significant performance before a concept drift point, but not after. We define this problem by the question '*how can we ensure that an efficient classifier-detector pair is present at any time in an evolving environment?*' To answer this, we have developed the TORNADO framework which runs various kinds of learning algorithms simultaneously against evolving data streams. Each algorithm *incrementally* and *independently* trains a predictive model and updates the statistics of its drift detector. Meanwhile, our framework monitors the (classifier, detector) pairs, and recommends the efficient one, concerning the classification, adaptation, and resource consumption performance, to the user. We further define the holistic CAR measure that integrates the classification, adaptation, and resource consumption measures for evaluating the performance of adaptive learning algorithms. Our experiments confirm that the most efficient algorithm may differ over time because of the developing and evolving nature of data streams.

Acknowledgements

I praise to the Lord Almighty for his blessings and mercy. I am deeply grateful to my beloved parents, Dr. Majid Pesaranghader and Nasrin Hakimian, for all sacrifices they made for me. Mom, you will be with us forever.

I appreciate the supervision, the patience, and the kindness of Dr. Herna L. Viktor, during the past four years, which made this work possible. I extend my sincere thanks to Dr. Eric Paquet for his more than generous guidance and support. It has been an honor to work with both of you.

My dear siblings, Narges and Ahmad, thank you for your presence, encouragement, inspiration, love, and care throughout this and other chapters of my life.

I cannot express enough how thankful I am to my friends, Mohammad Ali, Nicolino, Carson, Shiyi, Megan, Linda, Gabriel, Bukky, Mohammad, amongst others, who were with me after the loss of my mother. Yue, Kenniy, Sarah, John, and Richard, thank you for sharing your invaluable thoughts and time with me during this journey.

My gratitude also goes to Dr. Ali Ghorbani, Dr. Hong Yu Guo, Dr. Nancy Samaan, Dr. Ana-Maria Cretu, Dr. James R. Green, and Dr. Iluju Kiringa for their invaluable and constructive comments which helped me to improve my thesis for the final submission.

Finally, I wish to acknowledge the financial support by the Canadian Natural Sciences and Engineering Research Council (NSERC) as well as the Ontario Trillium Scholarship (OTS).

Table of Contents

List of Tables **xii**

List of Figures **xv**

List of Algorithms **xvii**

List of Abbreviations **xviii**

I Prologue 1

1 Introduction 2

1.1 Background 2

1.2 Research Problems and Motivations 5

1.2.1 Problem I: Concept Drift Detection for Adaptive Learning 5

1.2.2 Problem II: Online Multi-strategy Learning 6

1.2.3 Motivations 8

1.3 Research Scope 9

1.4 Research Methodology 10

1.5 Research Accomplishments and Deliverables 10

1.6 Thesis Organization 11

II Fundamentals 12

2 Adaptive Machine Learning 13

2.1 Machine Learning 13

2.1.1 Batch Setting . . . 13
2.1.2 Data Stream Setting . . . 13
2.1.3 Learning Modes . . . 14
2.2 Online Classification . . . 15
2.2.1 Data Stream Classification . . . 16
2.2.2 Assumptions . . . 16
2.2.3 Requirements . . . 17
2.2.4 Online Classification Cycle . . . 18
2.2.5 Online Classification Algorithms . . . 18
2.2.5.1 Naive Bayes . . . 19
2.2.5.2 Decision Stump . . . 21
2.2.5.3 Hoeffding Trees . . . 22
2.2.5.4 Perceptron . . . 23
2.2.5.5 K-Nearest Neighbours . . . 25
2.2.5.6 Discussion . . . 26
2.3 Adaptive Classification . . . 27
2.3.1 Concept Drift Phenomenon . . . 27
2.3.1.1 Formal Definition . . . 27
2.3.1.2 Concept Drift Patterns . . . 29
2.3.1.3 Concept Drift Terminology . . . 31
2.3.2 Adaptation Approaches . . . 31
2.3.3 Adaptation Requirements . . . 33
2.3.4 Concept Drift Detection Methods . . . 34
2.3.4.1 Cumulative Sum Variants . . . 34
2.3.4.2 Drift Detection Method . . . 35
2.3.4.3 Early Drift Detection Method . . . 35
2.3.4.4 Reactive Drift Detection Method . . . 36
2.3.4.5 Adaptive Windowing . . . 36
2.3.4.6 SeqDrift2 Detector . . . 36
2.3.4.7 Drift Detection Methods based on Hoeffding's Bound . . . 36

2.3.4.8 Discussion . . . 37
2.4 Data Streams . . . 38
2.4.1 Synthetic Data Streams . . . 38
2.4.1.1 Concept Drift Simulation . . . 40
2.4.2 Real-World Data Streams . . . 41
2.4.3 Benchmarking Data Streams . . . 42
2.5 Evaluation Settings . . . 43
2.5.1 Evaluation Procedures . . . 43
2.5.1.1 Incremental Holdout . . . 44
2.5.1.2 Predictive Sequential . . . 45
2.5.1.3 Comparison . . . 45
2.5.2 Classification Measures . . . 46
2.5.3 Drift Detection Measures . . . 47
2.5.3.1 Correctness Measures . . . 47
2.5.3.2 Drift Detection Delay . . . 49
2.5.4 Resource Consumption Measures . . . 49
2.6 Applications . . . 49
2.6.1 Monitoring and Control . . . 50
2.6.1.1 Monitoring against Adversary Actions . . . 50
2.6.1.2 Monitoring for Management . . . 51
2.6.2 Personal Assistance and Information Management . . . 51
2.6.2.1 Personal Assistance . . . 51
2.6.2.2 Customer Profiling . . . 51
2.6.2.3 Information Management . . . 52
2.6.3 Decision Making . . . 52
2.6.3.1 Finance . . . 52
2.6.3.2 Biomedicine . . . 52
2.6.4 AI and Robotics . . . 53
2.6.4.1 Mobile systems and robotics . . . 53
2.6.4.2 Intelligent systems . . . 53
2.6.4.3 Virtual reality . . . 53
2.6.5 Discussion . . . 53
2.7 Summary . . . 54

III Contributions **56**

3 Fast Hoeffding and McDiarmid Drift Detection Methods **57**

3.1 Problem Statement . . . 58
3.2 Fast Hoeffding Drift Detection Method . . . 59
3.2.1 Sensitivity of Parameters . . . 61
3.2.2 Experimental Evaluation . . . 62
3.2.2.1 Experiments against Abrupt Concept Drift . . . 63
3.2.2.2 Experiments against Gradual Concept Drift . . . 65
3.2.2.3 Discussion . . . 68
3.3 Stacking Fast Hoeffding Drift Detection Methods . . . 69
3.3.1 Stacking Fast Hoeffding Drift Detection Method . . . 69
3.3.2 Additive FHDDMS . . . 71
3.3.3 Sensitivity of Parameters . . . 72
3.3.4 Experimental Evaluation . . . 72
3.3.4.1 Experiments against Abrupt Concept Drift . . . 73
3.3.4.2 Experiments against Gradual Concept Drift . . . 75
3.3.4.3 Discussion . . . 76
3.4 McDiarmid Drift Detection Methods . . . 77
3.4.1 McDiarmid Drift Detection Methods (MDDMs) . . . 77
3.4.2 Discussion on MDDM Variants . . . 81
3.4.3 Sensitivity of Parameters . . . 81
3.4.4 Experimental Evaluation . . . 81
3.4.4.1 Experiments against Abrupt Concept Drift . . . 82
3.4.4.2 Experiments against Gradual Concept Drift . . . 84
3.4.4.3 Discussion . . . 84
3.5 Complementary Evaluations . . . 86
3.5.1 Evaluation against Synthetic Streams of Bits . . . 86
3.5.2 Evaluation against Real-world Data Streams . . . 88
3.6 Summary . . . 91

4 Adaptive Multi-Strategy Learning **93**

4.1 Problem Statement . . . 93

4.2 Multi-strategy Learning . . . 96

4.3 Performance Measures . . . 97

4.3.1 Single-purpose Measures . . . 97

4.3.2 The EMR Measure . . . 98

4.3.3 The CAR Measure . . . 99

4.4 TORNADO: Reservoir of Multi-Strategy Learning . . . 101

4.5 Experimental Study of TORNADO Framework . . . 104

4.5.1 Synthetic Data Streams . . . 105

4.5.1.1 Top 20 (Classifier, Detector) Pairs . . . 105

4.5.1.2 Illustration of Pair Recommendation . . . 107

4.5.2 Semi-real-world Data Streams . . . 112

4.5.2.1 Top 20 (Classifier, Detector) Pairs . . . 112

4.5.2.2 Illustration of Pair Recommendation . . . 114

4.5.3 Discussion . . . 118

4.6 Summary . . . 118

IV Epilogue **120**

5 Conclusion and Future Work **121**

5.1 Conclusion . . . 121

5.2 Future Work . . . 123

V APPENDICES **126**

A Pseudocodes of Online Learning Algorithms **127**

A.1 Naive Bayes . . . 128

A.2 Decision Stump . . . 129

A.3 Hoeffding Tree . . . 130

A.4 Perceptron . . . 131

A.5 K-Nearest Neighbours . . . 131

B Complementary Tables **132**

C Theoretical Proofs **144**

C.1 FHDDM Bounds 145

C.1.1 False Positive Bound 145

C.1.2 False Negative Bound 146

References **148**

List of Tables

2.1 Batch/Traditional Setting vs. Data Stream Setting 14
2.2 Advantages and Disadvantages of Learning Algorithms 26
2.3 Concept Drift Terminology 31
2.4 The CIRCLES Data Stream 39
2.5 Summary of Synthetic Data Streams 41
2.6 Shifting Classes to Simulate Concept Drift 43
2.7 Characteristics of Applications 54

3.1 FHDDM and Sensitivity of Parameters 62
3.2 Hoeffding Tree and FHDDM against Data Streams with Abrupt Drift . . . 64
3.3 Naive Bayes and FHDDM against Data Streams with Abrupt Drift 65
3.4 Hoeffding Tree and FHDDM against Data Streams with Gradual Drift . . 66
3.5 Naive Bayes and FHDDM against Data Streams with Gradual Drifts . . . 67
3.6 Hoeffding Tree with FHDDMS against Data Streams with Abrupt Drift . . 74
3.7 Naive Bayes with FHDDMS against Data Streams with Abrupt Drift . . . 74
3.8 Hoeffding Tree and FHDDMS against Data Streams with Gradual Drift . . 75
3.9 Naive Bayes and FHDDMS against Data Streams with Gradual Drift . . . 76
3.10 Hoeffding Tree and MDDMs against Data Streams with Abrupt Drift . . . 83
3.11 Naive Bayes and MDDMs against Data Streams with Abrupt Drift 83
3.12 Hoeffding Tree and MDDMs against Data Streams with Gradual Drift . . . 84
3.13 Naive Bayes and MDDMs against Data Streams with Gradual Drift 85
3.14 Experiments against Synthetic Streams of Bits with Abrupt Drift 87
3.15 Experiments against Synthetic Streams of Bits with Gradual Drift 88
3.16 Hoeffding Tree and Naive Bayes against Real-world Data Streams 90

4.1 Top 20 Pairs with Highest Average Scores against Data Streams with Abrupt Concept Drift 105
4.2 Top 20 Pairs with Highest Average Scores against CIRCLES and LED Data Streams with Gradual Concept Drift 106
4.3 Top 20 Pairs with Highest Average Scores against CIRCLES and LED Data Streams with Gradual Concept Drift 113

B.1 Hoeffding Tree and DDMs against SINE1 Data Stream with Abrupt Drift . 133
B.2 Hoeffding Tree and DDMs against MIXED Data Stream with Abrupt Drift 133
B.3 Hoeffding Tree and DDMs against CIRCLES Data Stream with Gradual Drift 134
B.4 Hoeffding Tree and DDMs against LED Data Stream with Gradual Drift . 134
B.5 Naive Bayes and DDMs against SINE1 Data Stream with Abrupt Drift . . 135
B.6 Naive Bayes and DDMs against MIXED Data Stream with Abrupt Drift . . 135
B.7 Naive Bayes and DDMs against CIRCLES Data Stream with Gradual Drift 136
B.8 Naive Bayes and DDMs against LED Data Stream with Gradual Drift . . 136
B.9 Top 20 Pairs with Highest Average Scores against SINE1 Data Stream with Abrupt Concept Drift 137
B.10 Top 20 Pairs with Highest Average Scores against SINE2 Data Stream with Abrupt Concept Drift 137
B.11 Top 20 Pairs with Highest Average Scores against MIXED Data Stream with Abrupt Concept Drift 138
B.12 Top 20 Pairs with Highest Average Scores against STAGGER Data Stream with Abrupt Concept Drift 138
B.13 Top 20 Pairs with Highest Average Scores against CIRCLES Data Stream with Gradual Concept Drift (1) 139
B.14 Top 20 Pairs with Highest Average Scores against CIRCLES Data Stream with Gradual Concept Drift (2) 139
B.15 Top 20 Pairs with Highest Average Scores against LED Data Stream with Gradual Concept Drift (1) 140
B.16 Top 20 Pairs with Highest Average Scores against LED Data Stream with Gradual Concept Drift (2) 140
B.17 Top 20 Pairs with Highest Average Scores against ADULT Data Stream with Gradual Concept Drift (1) 141
B.18 Top 20 Pairs with Highest Average Scores against ADULT Data Stream with Gradual Concept Drift (2) 141

B.19 Top 20 Pairs with Highest Average Scores against NURSERY Data Stream with Gradual Concept Drift (1) 142
B.20 Top 20 Pairs with Highest Average Scores against NURSERY Data Stream with Gradual Concept Drift (2) 142
B.21 Top 20 Pairs with Highest Average Scores against SHUTTLE Data Stream with Gradual Concept Drift (1) 143
B.22 Top 20 Pairs with Highest Average Scores against SHUTTLE Data Stream with Gradual Concept Drift (2) 143

List of Figures

1.1 Research Methodology Lattice 10

2.1 The Taxonomy of Learning Modes 15
2.2 The Cycle of Online Classification 18
2.3 Decision Stump Example 21
2.4 Perceptron Structure 23
2.5 Real Concept Drift vs. Virtual Concept Drift 29
2.6 Patterns of Concept Drifts 30
2.7 Adaptation Methods 31
2.8 The SINE1 Data Stream 38
2.9 The CIRCLES Data Stream 39
2.10 Concept Drift Simulation using Sigmoid Function 40
2.11 4-Fold Cross Validation Example 44
2.12 Holdout vs. Prequential 46
2.13 Illustration of Counting TP, FP and FN 48

3.1 FHDDM: Illustrative Example 60
3.2 Stacking Fast Hoeffding Drift Detection Method (FHDDMS) 69
3.3 Additive FHDDMS Approach ($\text{FHDDMS}_{\text{add}}$) 71
3.4 FHDDMS and $\text{FHDDMS}_{\text{add}}$: Illustration Example 71
3.5 McDiarmid Drift Detection Method (MDDM) 78

4.1 Effect of Distributional Change on Performance of Different Classifiers . . . 94
4.2 The TORNADO Framework 102
4.3 Recommendation of (Classifier, Detector) Pairs over Time 103

4.4 Pair-Color Map 104
4.5 Recommended Classifier+Detector Pairs out of 75 Pairs against SINE1 and SINE2 Data Streams with Abrupt Concept Drifts 108
4.6 Recommended Classifier+Detector Pairs out of 75 Pairs against MIXED and STAGGER Data Streams with Abrupt Concept Drifts 109
4.7 Recommended Classifier+Detector Pairs out of 75 Pairs against CIRCLES Data Stream with Gradual Concept Drifts 110
4.8 Recommended Classifier+Detector Pairs out of 75 Pairs against LED Data Stream with Gradual Concept Drifts 111
4.9 Recommended Classifier+Detector Pairs out of 75 Pairs against ADULT Data Stream with Gradual Concept Drifts 115
4.10 Recommended Classifier+Detector Pairs out of 75 Pairs against NURSERY Data Stream with Gradual Concept Drifts 116
4.11 Recommended Classifier+Detector Pairs out of 75 Pairs against SHUTTLE Data Stream with Gradual Concept Drifts 117

List of Algorithms

3.1 Fast Hoeffding Drift Detection Method (FHDDM) 61
3.2 Stacking Fast Hoeffding Drift Detection Method (FHDDMS) 70
3.3 McDiarmid Drift Detection Method (MDDM) 80
A.1 Naive Bayes (NB) Pseudocode 128
A.2 Decision Stump (DS) Pseudocode 129
A.3 Hoeffding Tree (HT) Pseudocode 130
A.4 Perceptron (PR) Pseudocode 131
A.5 K-Nearest Neighbours (K-NN) Pseudocode 131

List of Abbreviations

ADWIN: Adaptive Windowing
AI: Artificial Intelligence
ANN: Artificial Neural Networks
BI: Business Intelligence
CAR: Classification-Adaptation-Resource Consumption
CUSUM: Cumulative Sum
CVFDT: Concept-adapting Very Fast Decision Tree
DDM: Drift Detection Method
DS: Decision Stump
EDDM: Early Drift Detection Method
EMR: Error-Memory-Runtime
EWMA: Exponentially Weighted Moving Average
FHDDM: Fast Hoeffding Drift Detection Method
FHDDMS: Stacking Fast Hoeffding Drift Detection Method
FHDDMS$_{\mathbf{add}}$: Additive Stacking Fast Hoeffding Drift Detection Method
FN: False Negative
FP: False Positive
HT: Hoeffding Tree
HDDM: Drift Detection Method based on Hoeffding's bound
IoT: Internet of Things
K-NN: K Nearest Neighbours
MAP: Maximum a posteriori
MDDM: McDiarmid Drift Detection Method
ML: Machine Learning
MOA: Massive Online Analysis
MSE: Mean Squared Error
NB: Naive Bayes
PAC: Probably Approximately Correct
RDDM: Reactive Drift Detection Method
PH: PageHinkley
PR: Perceptron
ROI: Return on Investment
SVM: Support Vector Machine
TN: True Negative
TP: True Positive
UCI: University of California, Irvine
USFS: US Forest Service
VFDT: Very Fast Decision Trees
WEKA: Waikato Environment for Knowledge Analysis

Part I

Prologue

Chapter 1

Introduction

1.1 Background

Artificial intelligent automata have, undoubtedly, emerged in every aspect of human life, from daily web searches and video games to medical diagnosis and cognitive sciences. An intelligent automaton perceives its environment and considers actions to maximize the chance of success to some predefined goals (Russell et al., 2003). *Machine Learning* (ML), as the core of Artificial Intelligence (AI), has been manifested since the early 1950s when the Turing Test[1] was introduced (Turing, 1950). The Checker Program, the first learning program able to improve its performance after each play, was created by Arthur Samuel in the 50s (Samuel, 1959). Rosenblatt (1957) designed the Perceptron machine for the image recognition task in 1957. It was a starting point for developing advanced Artificial Neural Networks (ANN) later in the 1980s (Rumelhart et al., 1988). Ross Quinlan proposed ID3, as one of the first decision tree learning algorithms, readily interpreted by a human, in 1986 (Quinlan, 1986). Advanced learning algorithms, e.g., Support Vector Machine (SVM) (Cortes and Vapnik, 1995), Hoeffding Trees[2] (Domingos and Hulten, 2000), and ensemble meta-learning techniques, e.g., Bagging (Breiman, 1996) and Adaptive Boosting (Freund and Schapire, 1995), were introduced later. Wojtusiak (2012) defines Machine Learning as follows:

> "Machine Learning is a scientific discipline that concerns developing learning capabilities in computer systems. A computer system learns if it improves its performance or knowledge due to experience, or if it adapts to a changing environment." – Janusz Wojtusiak, 2012.

As stressed above, both the *improvement* in performance after some experiences and the *adaption* to changes are fundamental parts of machine learning. Although most definitions

1 In the Turing test, an intelligent machine has to be indistinguishable from a human.
2 They are also called as Very Fast Decision Trees (VFDT) in the literature.

of machine learning revolve around the former case, e.g., in (Samuel, 1959; Han et al., 2011), the latter one is essential when learning happens in dynamic and evolving environments.

Supervised learning, unsupervised learning, and reinforcement learning are the three general machine learning problems (Russell et al., 2003; Bishop, 2006). Supervised Learning algorithms induce a predictive model using labelled data which is known as training data (Mohri et al., 2012). Then, the predictive model is utilized for labelling new instances. Classification and regression are examples of supervised learning where the output labels are discrete and continuous, respectively (Han et al., 2011; Flach, 2012). Unsupervised learning discovers hidden patterns in unlabelled data. Clustering is a kind of unsupervised learning; that reveals hidden patterns in data by grouping similar data points into separate clusters (Bishop, 2006). The discovered clusters may later be used to define labels for a classification task (Han et al., 2011). Reinforcement learning develops intelligent agents able to communicate with their environments and consequently take optimal actions. An agent learns to choose the optimal action through receiving feedback, i.e., reward or penalty, from its environment (Sutton and Barto, 1998). In this thesis, our concentration is on the classification task, particularly against data streams where instances continuously arrive over time. The reader should note that more information about the scope of this work is available in Section 1.3.

A classic supervised learner initially trains a predictive model using a preprocessed dataset, known as the training set; and then it runs the model against a test set to evaluate its performance (Han et al., 2011; Flach, 2012). Validation sets may also be used to ensure there is no overfitting to the training data (Bishop, 2006; Han et al., 2011). Finally, the predictive model is applied in the real world to predict the output (or label) of new unseen data. Although memory and time are assumed to be unlimitedly available for a traditional supervised learning task, this assumption may not be practical in the era of Big Data and the Internet of Things (IoT) where data arrive in a huge *volume* with high *velocity* possibly from a *variety* of resources (Domingos and Hulten, 2000; Bifet et al., 2009; Krempl et al., 2014; De Mauro et al., 2016). Currently, over $4,100$ million people, i.e., more than 50% of the world's population, use the Internet and publicly generate data every day (Stats, 2018). The volume of data generated in 2015 was estimated to be 10 exabytes, and it will be four times greater by 2020 (Gantz and Reinsel, 2012). Furthermore, data are not only in forms of batches anymore but also (infinite) streams, and the decision makers should perform in a timely manner. Classification algorithms should, therefore, train their models in real-time without accommodating all streaming data in the main memory (Bifet et al., 2009; Gama et al., 2014). Hence, to address this challenge, there has been a move from traditional learning to *incremental/online learning* where learning algorithms are able to build classification models incrementally, either by continuous update or retraining from scratch as new instances or batches arrive (Žliobaitė et al., 2012; Gama et al., 2014). The formal definition of incremental learning, by Gama et al. (2014), is provided as follows:

> "Incremental algorithms process input examples one-by-one and update the sufficient statistics stored in the model after receiving each example. Incremental algorithms may have random access to previous examples or selected examples. In such a case these algorithms are called incremental algorithms with partial memory." – João Gama et al., 2014.

Please note that online learning is a particular case of incremental learning where each example is usually discarded after being processed (Gama et al., 2014). We provide more information regarding incremental/online learning in Section 2.1.3.

Learning from evolving data streams is a challenging task because of not only managing memory and time efficiently but also possible changes in the distribution of data (Baena-Garcıa et al., 2006; Bifet and Gavalda, 2007). The phenomenon of distributional changes in data is known as *concept drift* in the literature (Schlimmer and Granger Jr, 1986; Widmer and Kubat, 1996). The quality of learning may diminish, i.e., the classification error-rate may increase when concept drift takes place (Gama et al., 2004; Nishida and Yamauchi, 2007). Therefore, incremental learning algorithms need to monitor data and adapt themselves according to the new distribution when concept drift happens (Kuncheva, 2008; Žliobaitė, 2010). *Adaptive learning* algorithms, recognized as advanced incremental learning techniques, employ drift detection methods to find the drift points (Gama et al., 2014). Classification models are updated or retrained from scratch once concept drift is detected. Hollis (2012) describes the goal of adaptation as follows:

> "Adaptation is a ubiquitous term, used in the humanities and sciences alike, to refer to the act or process of changing - or, indeed, to the change itself - so as to become better suited to a new situation, or in a new application." – Karen L. Hollis, 2012.

Furthermore, Gama et al. (2014) define adaptive learning, more specifically for the machine learning tasks, as below:

> "Adaptive learning refers to updating predictive models online during their operation to react to concept drift. Adaptive learning algorithms can be seen as advanced incremental learning algorithms that are able to adapt to the evolution of the data-generating process over time." – João Gama et al., 2014.

In an (incremental) adaptive learning task, the current optimal classifier may not be optimal *'all'* the time against evolving or completing data streams (Wolpert and Macready, 1997; Žliobaitė, 2010). This implies that if, for example, model M is optimal for interval T, we are not able to claim it will also stay optimal for the next intervals. As a consequence, we need a solution to provide the optimal model, preferably in near real-time, to the user

over time. The reader should note that the definition of *optimality* may be subjective and may depend on the domain of application. In this thesis, we use the '*optimal*' term to refer to the most '*efficient*' learning algorithm concerning the *classification*, *adaptation*, and *resource consumption* measures. In most studies, the classification accuracy is used, as a decisive measure, to define the best or optimal model, e.g., in (Gama et al., 2004; Baena-Garcıa et al., 2006; Bifet and Gavalda, 2007; Nishida and Yamauchi, 2007). *Multi-strategy learning* applies multiple homogeneous or heterogeneous learners, simultaneously in parallel, against data streams[3]. As instances are processed, the optimal model, which is defined based on the performance measures, are provided to the user. The formal definition of multi-strategy learning by Brazdil (2012) is:

> "Multi-strategy learning is concerned with developing learning methods and systems that combine different inferential and/or representational strategies in solving a given learning task." – Pavel B. Brazdil, 2012.

The focus of this section was to briefly explain the key concepts, from machine learning to multi-strategy learning, which may appear in this thesis. We declare the research problems for this thesis in the next section.

1.2 Research Problems and Motivations

This thesis addresses two research problems related to adaptive learning and multi-strategy learning against evolving data streams. We, firstly, introduce new drift detection methods to improve the quality of adaptation regarding various performance measures. Secondly, we create a reservoir of adaptive learning algorithms for multi-strategy learning, able to provide the user with the optimal classifier-detector pair, i.e., the most efficient pair regarding the predetermined measures, every moment against data streams. We state and discuss each research problem in the following subsections.

1.2.1 Problem I: Concept Drift Detection for Adaptive Learning

Recall that learning from evolving data streams is challenging due to distributional changes in data, i.e., the *concept drift* phenomenon. Learning algorithms have to adapt themselves to the new distributions for keeping the classification error-rate low. Drift detection methods, as the main component of adaptive learning algorithms, are responsible for detecting concept drifts with the least delay as soon as they appear in data streams. Such methods should also avoid high *false positive* and *false negative* rates as they process input data

3 Please note that this is our solution provided in this work.

(Sakthithasan et al., 2013; Pears et al., 2014). False positive refers to false alarms for concept drift, whereas false negative means ignoring a real concept drift. False positive entails keeping resource busier, whereas false negative causes loss in the classification accuracy (Gama et al., 2004; Bifet and Gavalda, 2007; Žliobaitė et al., 2015; Huang et al., 2015). Additionally, based on the probably approximately correct (PAC) learning model (Valiant, 1984; Mitchell, 1997), a high false positive rate may not let accuracy increase since an insignificant amount of data would be used for training. The drift detection methods should not assume the incoming input data, e.g., prediction results, follow a specific distribution function as the nature of streaming data is dynamic (Frías-Blanco et al., 2015). Recall that (adaptive) online learning algorithms need to process data in real-time without exhausting the main memory. Therefore, execution runtime and memory occupation of drift detection methods are supposed to be affordable. Finally, an adaptive learning algorithm should obtain a higher accuracy, compared to a non-adaptive algorithm, to be considered beneficial for a learning task from an evolving data stream. We express the research problem and its corresponding research objective for addressing the just mentioned demands as follows:

> **Research Problem 1** is that *the quality of adaptive learning may diminish due to the concept drift phenomenon in evolving data.* Therefore, the challenge is to alarm for concept drifts as soon as they happen, with low false positive and false negative rates, regardless of the incoming data distribution function, to keep the classification error-rate minimal.
>
> **Research Objective 1** is *to introduce novel concept drift detection methods which detect drift points more accurately compared to the state-of-the-art.* For that, we *empirically* compare our methods with the existing algorithms against synthetic and real-world data streams, by considering different performance measures, e.g., detection delay, false positive, and false negative.

We introduce our Fast Hoeffding Drift Detection Method (FHDDMs), Stacking Fast Hoeffding Drift Detection Method (FHDDMS), and McDiarmid Drift Detection Methods (MDDMs), for addressing the aforesaid research problem, in Chapters 3.

1.2.2 Problem II: Online Multi-strategy Learning

In an evolving environment, the learning algorithm with the highest accuracy at the current time may show a low accuracy in the future. That may happen because the nature of data is dynamic, or data are completing. For example, consider three learning algorithms of L_1, L_2, and L_3. Assume L_2 has shown the highest accuracy since the first instance until time t_c when concept drift happens. After reacting to concept drift, L_1 becomes the learner with the highest accuracy. That implies the most suitable learning algorithm may differ over time, in an evolving environment. To this end, we design a reservoir of various adaptive

learning algorithms that offers the most *efficient* learner, by multi-strategy learning, at any time against an evolving data stream. The reservoir runs various kinds of learning algorithms *simultaneously* and keeps track of the optimal one at the current time or interval. Please also note that we consider a learner with the current highest performance as the optimal learner while the input instances are processed. Recall that, the optimality is subjective and depends on the domain of application. For example, one may consider classification accuracy, or any other performance measure, to define the optimal model. We state the research problem and the research objective regarding this issue as follows:

> **Research Problem 2.1** is that *a current optimal learning algorithm may not be optimal in the very near future due to the dynamic nature of evolving environments.* We express such a challenge as follows: "*how can we guarantee that an optimal model is available at every time step against evolving data?*"
>
> **Research Objective 2.1** is *to develop a reservoir of various adaptive learning algorithms, for multi-strategy learning against evolving data streams*, able to run different kinds of learning algorithms in *parallel* continuously and to recommend the optimal model, at the current moment or interval, to the user.

Moreover, one may study two or more performance measures together and then set a compromise among them. For example, consider model M_1 with the accuracy of 90.5% and the runtime of $2,000$ milliseconds per instance, and model M_2 with the accuracy of 91.8% and the runtime of $10,000$ milliseconds. The user may compromise accuracy for runtime for a (near) real-time decision-making application, e.g., emergency response and security (Bolton and Hand, 2002; Thuraisingham, 2006), and considers model M_1. We, thus, need comprehensive measures able to summarize all individual performance measures together and set (necessary) trade-offs among them. In this work, we are interested in the classification, adaptation, and resource consumption (i.e., memory usage and execution runtime) performance measures. Memory usage and the execution runtime are critical in various domains, e.g., wireless sensors and mobile devices, where main memory and CPU are limited (Phung et al., 2007; Mahmood et al., 2013; Gaber et al., 2014a). Finally, we define this problem as below:

> **Research Problem 2.2** is that *the classification error-rate does not cover all scenarios when it comes to evaluating the optimality of learning algorithms against evolving data streams.* For that, we need to go beyond the classification error-rate to be able to recommend an optimal solution to the user, considering the current situation.
>
> **Research Objective 2.2** is *to define a comprehensive measure which not only combines classification error-rate with adaptation and resource consumption measures* but also *lets us compromise specific individual measures for others*, for ranking and recommending the classification algorithms.

In Chapter 4, we address the aforementioned problems by introducing the TORNADO framework and the CAR performance measure.

1.2.3 Motivations

After explaining the research problems and the research objectives, we now discuss our motivations as follows:

- *The Need for Accurate Drift Detection Methods* – Automatic and intelligent adaption is vital in evolving applications, e.g., including, but not limited to, medicine, nuclear reactors, self-driving cars, and smart cities; otherwise, paying a substantial cost, possibly, in terms of people lives, horrific accidents, and environmental hazards would be inevitable. Hence, *blindly* retraining at specific time points is not an appropriate solution since it does not guarantee an immediate and accurate detection of concept drift, and subsequently an indispensable reaction. Therefore, detection methods able to react to concept drift on time with the minimum false positive and false negative rates are indeed required. The existing methods, however, showed poor performance in our preliminary assessments and early studies, and such an observation persuaded us to devise new drift detection methods for detecting concept drift effectively.

- *Evaluation of Adaptation Methods* – Although many adaptive learning works have been evaluated regarding the classification accuracy (Gama et al., 2004; Baena-Garcıa et al., 2006; Nishida and Yamauchi, 2007; Ross et al., 2012; Maciel et al., 2015), we should not just rely on accuracy since it does not reflect how well a drift detection method works. It was, therefore, a motivation for us to go beyond accuracy and evaluate drift detection methods in terms of the other performance measures, e.g., detection delay, detection false positive, and detection false negative as well.

- *The Need for Online Multi-Strategy Learning* – The question of '*Which learning algorithm accurately represents the given data?*' is ubiquitous for every application. We face two challenges to answer this question for adaptive online learning. Firstly, a classification model is continuously updated, without storing streaming data. That means the data which are already processed, are not be available in the future (Bifet et al., 2011). Further, we cannot claim the current suitable model may appropriately reflect the data in the future (Wolpert and Macready, 1997). Thus, the comparison of classification models, as in the traditional way, is rather impractical against real-world streaming environments. We found adaptive multi-strategy learning beneficial in order to track the classification model with the highest performance, for decision making, over time.

- *The Great Pool of Applications* – This work has the potential to attract the attention of International and Canadian manufacturers which are interested to invest in data stream mining applications in telecommunication (Mazhelis and Puuronen, 2007; Hilas, 2009), security and intrusion detection (Sadoddin and Ghorbani, 2008, 2009), transportation (Crespo and Weber, 2005; Moreira, 2008), maritime systems (Idiri and Napoli, 2012; St-Hilaire and Hadzagic, 2013), forestry (Cortez and Morais, 2007; Sanquetta et al., 2013), smart homes and cities (Anguita, 2001; Rashidi and Cook, 2009), health care and biomedicine (Tsymbal et al., 2008; Poh et al., 2009), and military (Ceruti, 2003; Seifert, 2007). Traditional machine learning algorithms may not be able to react to the dynamic nature of such applications in a real-time fashion while keeping the performance of learning as high as possible. Eventually, we show that our reservoir of adaptive learning algorithms for multi-strategy learning, as a solution, may help to improve those applications.

1.3 Research Scope

The reader may find the scope of our concentration, data, experiments, and evaluations as follows:

- *Thesis Concentration* – The primary concentration of this thesis is on *classification* and *adaptation* against evolving streaming data.
- *Data Characteristics* – The data streams, used in our experiments, are essentially cross-sectional data. That is, we performed our experiments against not time-series data streams[4]. Furthermore, the data streams are assumed to be already labelled. We consider addressing the challenges imposed by the delayed labels, unlabelled instances, and time-series data as future work.
- *Experiments* – Our experiments are over synthetic and real-world data streams which are frequently used in the literature (Kubat and Widmer, 1995; Gama et al., 2004; Baena-Garcıa et al., 2006; Nishida and Yamauchi, 2007; Bifet et al., 2009; Frías-Blanco et al., 2015). The data streams contain numerical or categorical attributes. The concept drift may appear abruptly or gradually in the synthetic data streams. We added noise to the synthetic data streams to confirm our methods are robust against noisy data. Please note that drift detectors are beneficial if they distinguish concept drift from noise (Black and Hickey, 2002; Krempl et al., 2014; Gama et al., 2014). Finally, we process only one single stream per task.

4 Time-series data consist of long sequences of numeric data traced through time, e.g., recorded at equal time intervals. Examples of such data are stock markets and medical observations (Han et al., 2011).

- *Evaluations* – We have considered the drift detection delay, false positive and false negative rates, runtime, and memory usage as well as the classification accuracy to assess the drift detection methods for adaptive learning. As for adaptive multi-strategy learning, we study the classification error-rate, the adaptation measures, and memory usage and runtime.

1.4 Research Methodology

We followed an incremental methodology for this thesis. It comprised seven general tasks, as depicted in Fig. 1.1. We began with the literature review, and the data collection and/or generation task. We then designed our drift detection methods for adaptive learning as well as a reservoir for adaptive multi-strategy learning. The adjustment of parameters, experiments, and discussion were the oncoming tasks. The central hexagon, i.e., the analysis task, had an influential role, because we discussed pros and cons of existing methods and established our developments based on that. Finally, the process was not like a queue where one task waits for the other ones, but rather like a lattice where all tasks were, interactively, fulfilled.

Fig. 1.1. Research Methodology Lattice

1.5 Research Accomplishments and Deliverables

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

- **Journal Paper**: *"Reservoir of Diverse Adaptive Learners and Stacking Fast Hoeffding Drift Detection Methods for Evolving Data Streams"*, published in the Machine Learning Journal, Springer (Pesaranghader et al., 2018a).

- **Conference Proceeding**: *"McDiarmid Drift Detection Methods for Evolving Data Streams"*, presented at International Joint Conference on Neural Network (IJCNN 2018) and published in the proceedings (Pesaranghader et al., 2018b).

- **Conference Proceeding**: *"Fast Hoeffding Drift Detection Method for Evolving Data Streams"*, presented at European Conference on Machine Learning and Principles of Knowledge Discovery and Databases (ECML-PKDD 2016) and published in the proceedings (Pesaranghader and Viktor, 2016).

- **Conference Proceeding**: *"A Framework for Classification in Data Streams using Multi-strategy Learning"*, presented at International Conference on Discovery Science (DS 2016) and published in the proceedings (Pesaranghader et al., 2016).

- **The Tornado Framework**: Recall that we developed the Tornado framework for adaptive multi-strategy learning from dynamic and evolving data streams (i.e., to address Research Problem II). We explain the structure of the framework and how it operates in Chapter 4. The source of this framework is available to public[5].

- **The Source Codes of Our Methods for the MOA Framework[6]**: We also implemented the Fast Hoeffding Drift Detection Method (FHDDM), the Stacking Fast Hoeffding Drift Detection Method (FHDDMS), and the McDiarmid Drift Detection Method (MDDM) in Java for the MOA framework, and made their source codes available to public[7].

1.6 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 explains the fundamental concepts of adaptive learning and includes the literature review. We introduce our new drift detection methods in Chapter 3. Extensive experiments are performed against the synthetic and real-world data streams to compare the introduced methods with the state-of-the-art. We present our Tornado framework, as a reservoir of adaptive learning algorithms, for adaptive multi-strategy learning in Chapter 4. We also define the EMR and CAR measures for ranking the adaptive learners against evolving data streams. The chapter ends with experiments on different data streams to represent how the Tornado framework operates and recommends the optimal models to the user over time. Finally, we conclude the thesis and discuss future work in Chapter 5.

5 The Tornado framework is available at `https://github.com/alipsgh/tornado`.

6 MOA is a framework for data stream mining. It is developed in Java at the University of Waikato, New Zealand (Bifet et al., 2010a).

7 The source codes of our drift detection methods for the MOA framework is available at `https://github.com/alipsgh/codes_for_moa`.

Part II

Fundamentals

Chapter 2

Adaptive Machine Learning

2.1 Machine Learning

Machine learning may take place in either batch setting or data stream setting. Traditional machine learning algorithms and techniques are founded based on the batch setting until the late 1990s when movements towards online learning, including data stream mining, began as the result of fast growing data. Different learning approaches are available for each setting. We explain both settings in the following subsections.

2.1.1 Batch Setting

The batch setting[1] assumes data are already collected and preprocessed. That means the data are resident, static, and stored in the memory. The size of data is in order of hundreds or even less. Hence, to make the best use of a dataset, it may be processed multiple times for the learning task. There are typically no memory or runtime constraints in batch learning (Nguyen et al., 2015). The concept of learning is static, i.e., the distribution of data does not change. Eventually, learning is done in an offline mode, meaning that the learning task is carried out after data collection (Nguyen et al., 2015).

2.1.2 Data Stream Setting

A data stream is a sequence of data samples rapidly arriving in a massive volume. Each sample is processed only once and then usually discarded. Learning algorithms may address this constraint by buffering samples for a short-term for future training or testing (Gama et al., 2014). Accommodating all the data into the main memory is not practical, and the

1 The batch setting is also referred to as the traditional, or offline, setting in the literature (Gama et al., 2014; Nguyen et al., 2015).

learning task should be done rapidly in a near *real-time* fashion (Bifet et al., 2011; Domingos and Hulten, 2001). Moreover, the concept of learning may change over time due to the evolving nature of environments. In this setting, learning happens in an incremental or online way.

We summarize and compare the settings mentioned above in Table 2.1. Recall that the size of data is in the order of hundreds in the batch setting, while it is in the magnitude of millions in the data stream setting. As to the number of scans, data may be processed multiple times in the batch setting. On the other hand, data are often processed in a single round in the data stream setting due to the limited memory bottleneck. The *memory*, *time*, and *concept of learning* are three critical dimensions of stream mining, which a learning algorithm has to address; otherwise, it may not be considered as suitable for the learning task. Furthermore, because the whole data are not available at a time in the data stream setting, a learning algorithm models a function which is an approximation of the real function for decision making. Finally, the mode of learning is incremental for data stream mining. Table 2.1 is similar to Table 1 in (Nguyen et al., 2015).

Table 2.1. Batch/Traditional Setting vs. Data Stream Setting

Criterion	**Batch/Traditional**	**Data Stream**
Size	Hundreds	Millions
Num. Scans	Multiple	Single
Memory	Unlimited	**Limited**
Time	Unlimited	**Real-time**
Concept	Static	**Evolving**
Model	Accurate	Approximate
Learning Mode	Offline	Incremental/Online

In the next subsection, we compare the learning modes of offline and incremental/online in detail.

2.1.3 Learning Modes

We present the taxonomy of machine learning modes in Fig. 2.1. Machine learning holds two general modes of *Batch Learning* and *Incremental Learning*, described as follows:

- **Batch learning (or offline learning)**: Batch learning algorithms are used for the batch setting, where the data are finite, static, and already preprocessed for the data mining task (Han et al., 2011). A batch learning task includes two *separated* phases of (1) *learning*, where a classification model is built, and (2) *classification*, where the classification model is tested and then used for labelling new instances. Please note that the learning phase is often called as the *training* phase in the literature. There is no further learning, once a model is trained (Sammut and Webb, 2011).

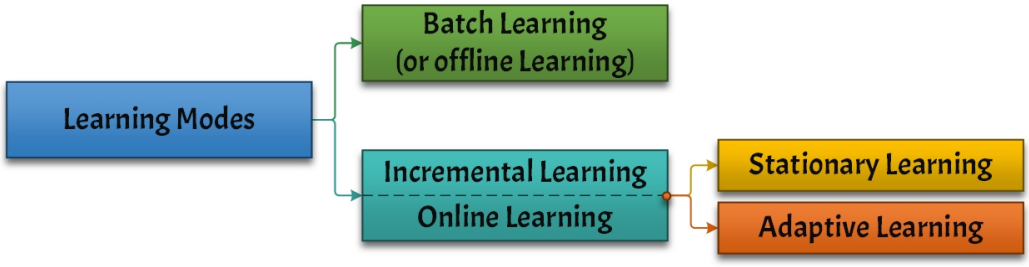


Fig. 2.1. The Taxonomy of Learning Modes

- **Incremental Learning:** Incremental learning algorithms are utilized for the data stream setting, where data arrive rapidly in a massive volume instance-by-instance. An incremental algorithm updates its classification model by processing instances one by one. Instances may be buffered for future use. Since the incremental learners must behave in real-time, unlike in batch learning, the training and testing phases happen together continuously (Bifet, 2009; Gama et al., 2004; Domingos and Hulten, 2000). Although incremental learning and online learning are often used interchangeably for each other in the literature, online learning is a kind of incremental learning where each instance is processed only once and then discarded (Gama et al., 2014). Learning could happen in either *stationary* or *evolving* environments. In the former case, the classification models are updated through *completion*, while in the latter case, they are updated through not only completion but also *adaptation*. Completion reflects the new instances, which are following the same distribution of previous ones, into the model, whereas adaptation deals with concept drift, typically, by retraining the model from scratch. The reader should note that the online learning terminology is interchangeably used for stationary learning in the literature, when concept drift and adaptation are absent.

Recall that the main focus of this thesis is on *classification* and *adaptation* against evolving data streams, as explained in Section 1.3. We review the fundamental concepts regarding online classification and adaption in Sections 2.2 and 2.3, respectively.

2.2 Online Classification

We categorized the learning tasks into two general groups of *batch/offline learning* and *incremental/online learning* and described each group in detail. We also discussed and compared two modes of online learning, i.e., *stationary learning* and *adaptive learning*. Since adaptive learning intrinsically shares fundamental concepts and requirements with (stationary) online learning, we study the necessary notions for online classification in this section before describing adaptive learning in Section 2.3.

2.2.1 Data Stream Classification

Recall that data stream classification is the task of building a model, using the available data (i.e., the training data), for predicting the label of unseen examples. We provide the formal definition of data stream classification as follows:

> Let stream S be a sequence of instances, as $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_t, y_t)$, arriving one-by-one over time. The pair $(\mathbf{x}_t, y_t)$ represents an instance arriving at time t, where $\mathbf{x}_t$ is a vector that holds the values of k attributes as $\mathbf{x} = (x_1, x_2, ..., x_k)$, and y_t is a *class label* from a *finite* set of m class labels as $y_t \in \{c_1, c_2, ..., c_m\}$. Assume a target function $y_t = f(\mathbf{x}_t)$ which maps an input vector to a class label. The learning task is to incrementally build a model $\tilde{f}$ that approximates the function f while instances are processed. An approximation that *maximizes* the classification accuracy is preferred.

For data stream classification, some assumptions are held regarding the nature of data streams. We review the assumptions in Section 2.2.2. Incremental learners should also comply with four basic requirements for the data stream classification task; we study the requirements in Section 2.2.3. Finally, we illustrate the life cycle of data stream classification in Section 2.2.4.

2.2.2 Assumptions

Generally, six fundamental assumptions are currently held for the task of data stream classification in the literature (Bifet et al., 2011; Domingos and Hulten, 2001). We describe the assumptions as follows:

1. The data have a fixed number of attributes. A high number of attributes may slow the learning task and increase memory consumption.

2. The total number of instances or records is substantial compared to the number of attributes. In fact, learning algorithms are supposed to be able to handle an *infinite* amount of data without exhausting memory resources.

3. The number of class labels must be small. More class labels introduce more statistics for inducing a classification model (Bifet and Kirkby, 2009). As a consequence, since the values of the statistics are continuously updated over time, mining a data stream with a large number of classes is computationally expensive. That is, computational complexity grows linearly with the number of classes.

4. The size of data is typically larger than the available memory. Therefore, loading all the data in memory is not feasible.

5. The learning algorithms need to accomplish training and testing phases in near real-time since instances of data streams arrive very fast.

6. The concept of learning is assumed to be stationary or evolving. Recall that concept drift happens when the underlying distribution of data changes.

The first three assumptions discuss the nature of data streams, whereas, the last three assumptions discuss what learning algorithms should consider for classification against data streams.

2.2.3 Requirements

The main challenges for data stream classification are imposed by the limited amount of computational resources and the concept drift phenomenon. The classification algorithms need to fulfill four basic requirements to make the learning task from data streams possible (Domingos and Hulten, 2001; Gama et al., 2009; Bifet et al., 2011), which are:

- **Requirement 1 (R1): Process an example at a time, and inspect it only once** – Recall that instances of a data stream arrive one after another, and they are processed only once in the order of arrival. That is, random access to the instances is not possible. An example is discarded once processed. Although this is an essential requirement for data stream mining, a learning algorithm can *internally* store instances, for a short time, for further use without violating Requirement 2.

- **Requirement 2 (R2): Use a limited amount of memory** – The reason behind training classification models incrementally is that the size of data is substantially larger than the size of available memory. That is, we cannot store a massive amount of data in limited memory. Therefore, an upper bound should be defined on the memory usage to avoid any possible memory exhaustion. A learning algorithm may only use the main memory for keeping the *current model*.

- **Requirement 3 (R3): Work in a limited amount of time** – Learning algorithms should process instances as fast as they arrive. That is, the learners need to work faster than the rate of arriving data. Any failure would inevitably cause loss of information. We, therefore, need to set an upper bound on the amount of time allocated for processing an instance.

- **Requirement 4 (R4): Be ready to predict at any point** – An ideal learning algorithm builds a classification model able to predict the label of unseen instances within a reasonable amount of time. So, *anytime prediction*, meaning a class label is available at any point of time, is essential for data stream classification.

2.2.4 Online Classification Cycle

We illustrate the cycle of the data stream classification task in Fig. 2.2. It comprises three phases of *Processing*, *Learning*, and *Utilizing* (Bifet and Kirkby, 2009), that are described as follows:

1. ***Processing*:** Data stream instances are processed, and then, passed to the learning phase. This phase complies Requirement 1.
2. ***Learning*:** The learning algorithm updates its predictive model by training every new instance. It is also ensured that Requirements 2 and 3 are not violated, by not exceeding the memory and runtime bounds.
3. ***Utilizing*:** The model is used to predict the labels of unseen instances. Requirement 4 should be guaranteed by the readiness of the model for anytime prediction.

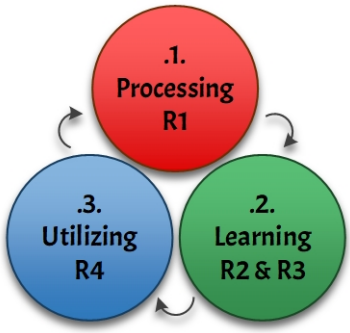


Fig. 2.2. The Cycle of Online Classification
(Bifet et al., 2011)

2.2.5 Online Classification Algorithms

This section represents a set of incremental/online learning algorithms frequently used as benchmarks in the data stream mining literature (Bifet and Gavalda, 2007; Bifet et al., 2010b; Frías-Blanco et al., 2015; Gama et al., 2004; Huang et al., 2015). We cover the *Naive Bayes*, *Decision Stump*, *Hoeffding Trees*, *Perceptron*, and *K-Nearest Neighbours* algorithms. Except for Hoeffding Trees, all other learning algorithms are the adapted versions of their classic algorithms for data stream mining. That is, the implementations of classic algorithms are modified for incremental/online learning. We explain the online algorithms in the following subsections.

2.2.5.1 Naive Bayes

Bayesian classifiers use statistical theorems to predict the probabilities of class memberships, i.e., they calculate the probability of belonging to a particular class for a given example. Bayesian classification is based on *Bayes' theorem*, Equation (2.1), which is named after Thomas Bayes, an English decision theorist. Bayesian classifiers have shown *high accuracy* as well as *high speed* when used against massive datasets (Han et al., 2011). Furthermore, experimental studies have indicated that *Naive Bayes*, i.e., the most basic Bayesian algorithm, is comparable in performance with other learners, such as decision trees and neural networks (Han et al., 2011).

The Naive Bayes algorithm assumes that values of attributes are independent of each other while it calculates the probabilities of class memberships. This assumption is known as *class conditional independence*.

Let $\mathbf{x}$ be an instance, also referred to as an *evidence*, that holds a set of values according to attributes, and let C denote the *target variables* which we are interested to predict, e.g., the class of instance $\mathbf{x}$. For classification, we have to calculate $P(\mathrm{C}|\mathbf{x})$, which indicates the probability that instance $\mathbf{x}$ belongs to a particular target variable (or class). $P(\mathrm{C}|\mathbf{x})$ is the *posterior probability* which is calculated by Bayes' theorem, as shown in Equation (2.1). As the equation shows, we need to calculate three components of $P(\mathbf{x}|\mathrm{C})$, $P(\mathrm{C})$, and $P(\mathbf{x})$. $P(\mathbf{x}|\mathrm{C})$ is called the *likelihood function*, and it is the probability of observing instance $\mathbf{x}$ among all instances with a particular target variable (or class). $P(\mathrm{C})$ is the *prior probability* showing the probability of observing each class, and $P(\mathbf{x})$ is the probability of experiencing instance $\mathbf{x}$ in data.

$$P(\mathrm{C}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathrm{C}) \cdot P(\mathrm{C})}{P(\mathbf{x})} \tag{2.1}$$

Suppose that there exist m classes, i.e., $\mathrm{C} = \{c_1, c_2, ..., c_m\}$. Let $\mathbf{x}$ be an example, represented as a vector of k attributes like $\mathbf{x} = (x_1, x_2, ..., x_k)$, and its class label is supposed to be predicted. To predict the label, the Naive Bayes algorithm calculates the posterior probability, using Bayes' theorem, for every class c_i by Equation (2.2).

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i) \times P(c_i)}{P(\mathbf{x})} \tag{2.2}$$

The class holding the *maximum* value of $P(c_i|\mathbf{x})$ defines the class label of instance $\mathbf{x}$. In the literature, it is called the *maximum a posteriori* (MAP) decision rule (Flach, 2012). Therefore, we must calculate the posterior probability for every class c_i, and then pick the one with the *maximum* posteriori probability as the label of the instance $\mathbf{x}$.

We may ignore $P(\mathbf{x})$ in our calculations since it is constant for all classes (Han et al., 2011; Flach, 2012). The prior probability of class c_i is estimated by $P(c_i) = |c_i|/n$, where

$|c_i|$ is the number of instances with class c_i after processing n instances. Since Naive Bayes assumes the values of the attributes are conditionally independent of one another, the likelihood $P(\mathbf{x}|c_i)$ is calculated by Equation (2.3).

$$P(\mathbf{x}|c_i) = \prod_{k=1}^{|\mathbf{x}|} P(x_k|c_i) \tag{2.3}$$

where, $P(x_k|c_i)$ is the probability of having instances with the value of x_k for attribute a_k, conditioned on class c_i, after processing n instances.

Finally, any class that maximizes the product value of $P(\mathbf{x}|c_i) \times P(c_i)$, also referred to as *relative-likelihood*, has the maximum posterior probability (Flach, 2012; Han et al., 2011). Therefore, that class will be considered as the label of instance $\mathbf{x}$. That means class y_l will be predicted as the label for instance $\mathbf{x}$ if and only if its $P(\mathbf{x}|y_l) \times P(y_l)$ is the greatest relative-likelihood.

For online learning, we summarize the training and testing tasks of the incremental Naive Bayes algorithm as follows:

- **Training task:** Assume $(\mathbf{x}, y)$ is the current instance, from a data stream, where $\mathbf{x} = (x_1, x_2, ..., x_k)$, e.g., x_k is the value of k^{th} attribute, and y is the class label and $y \in \mathrm{C}$. Naive Bayes increases the total number of instances processed so far by one, and it updates the prior probability for the class y, i.e., $P(y)$. Next, the posterior probability of $P(\mathbf{x}|c_i)$ for every class c_i is updated. At this point, instance $\mathbf{x}$ is trained, and all statistics are up-to-date.

- **Testing task:** Assume $\mathbf{x}$ is an instance that Naive Bayes predicts its label. First, Naive Bayes calculates the *likelihood* of $\mathbf{x}$ by

  $$P(\mathbf{x}|c_i) = \prod_{k=1}^{|\mathbf{x}|} P(x_k|c_i)$$

  for every class c_i. Subsequently, it calculates the relative-likelihoods by

  $$Rel(c_i|\mathbf{x}) = P(\mathbf{x}|c_i) \cdot P(c_i)$$

  for every class c_i. Eventually, the instance is labelled by the class that results in the maximum relative-likelihood.

The pseudocode of the incremental Naive Bayes algorithm is presented in Algorithm A.1. The INITIALIZE function assigns the attributes and class labels. The TRAIN function processes every new instance $(\mathbf{x}, y)$, and updates the statistics accordingly. The TEST function calculates the likelihoods as well as the relative-likelihoods for predicting the class label. The class with the highest relative-likelihood is returned as the label.

2.2.5.2 Decision Stump

A decision stump models a one-level decision tree[2] (Iba and Langley, 1992), i.e., a decision stump is a decision tree only with the root node. A decision stump makes a prediction using the values of an attribute that is assigned as the root. For example, Fig. 2.3 represents a decision stump for predicting whether a person plays a game outside based on the humidity level. Decision stumps are also called *1-rules* algorithms (Holte, 1993).

Considering the type of the attribute assigned as the root, several cases are possible: (1) For *nominal* attributes, one may build a stump which contains a leaf for each possible value (Bird et al., 2009) or a stump with two leaves, one corresponds to a chosen category, and the other leaf corresponds to all the other categories. The missing value may be treated as another category as well. (2) For a *continuous* attribute, usually, a threshold is defined, and the stump holds two leaves, for values below and above the threshold. It is also possible to have multiple thresholds, and as a result, the stump may contain three or more leaves. Fig. 2.3 represents a decision stump where attribute 'Humidity' is selected as the stump.

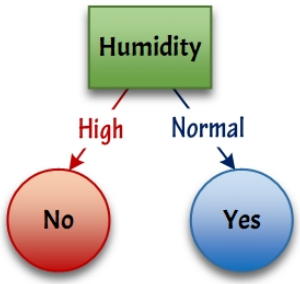


Fig. 2.3. Decision Stump Example

Attribute selection measures are used to identify an attribute as the stump. Attribute selection measures are used to rank the attributes based on their informativeness to build a decision tree in a top-down way. That is, the most informative attribute is selected as the root. Other attributes create the next levels of the tree (Han et al., 2011). Similarly, in a decision stump, the attribute with the highest rank is chosen as the stump.

As to an incremental learning scenario, while instances are processed, we need to update the statistics required for the attribute selection measures. Assuming *information gain* as the attribute selection measure, total entropy and entropies of all attributes are updated while instances are processed one-by-one. The reader may refer to (Han et al., 2011) for the calculation of information gain. Finally, the attribute with the highest gain is considered as the stump. The stump is then used to predict the label of a new instance. We present

2 A decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label (Han et al., 2011).

this process in Algorithm A.2. The total entropy is updated by lines 13 and 14. Then, the entropies and gains of attributes are updated, lines 15 and 21. Eventually, the attribute with the highest gain is assigned as the stump. The TEST function is used to sort the new instance $\mathbf{x}$ into a leaf and consider it as the predicted label.

2.2.5.3 Hoeffding Trees

Domingos and Hulten (2000) introduced the Hoeffding Tree algorithm[3] for learning from extremely large, and possibly infinite, data streams. The Hoeffding Tree algorithm mines data stream incrementally, without keeping instances in the main memory, and builds potentially very complex decision tree models with affordable computational cost.

The Hoeffding Tree algorithm is founded on the observation made by Catlett (1991), saying 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, Hoeffding Tree considers the early instances for choosing the root test; once the root attribute is chosen, the succeeding instances are passed down and used for choosing the appropriate attributes as test nodes in the next levels. Hoeffding Tree guarantees that, with a high probability, the attribute chosen as a node after processing n examples would be the same to the one which would be chosen after processing infinite examples. For this case, Domingos and Hulten (2000) applied the Hoeffding inequality (Hoeffding, 1963) 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 highest observed G after seeing n examples, and X_b be the second-best attribute. Let $\Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b)$ 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 $\Delta\overline{G} > \varepsilon$, where ε is calculated by:

$$\varepsilon = \sqrt{\frac{R^2}{2n} ln \frac{1}{\delta}} \tag{2.4}$$

The range R is equal to 1 for probability, and equal to $log(|C|)$ for information gain where $|C|$ is the number of classes.

Domingos and Hulten (2000) proved that Hoeffding Tree algorithm generates trees that are asymptotically close to the ones produced by a batch learner. Please note that when two or more attributes have very close G's, many examples would potentially be needed to decide between the attributes to find the best split. That is wasteful because it makes little difference in the quality of the tree. Hence, Hoeffding Tree determines that there is a tie among attributes, and split on the current best attribute if $\Delta\overline{G} < \varepsilon < \tau$, where τ is a user-specified threshold.

3 It is also known as Very Fast Decision Tree (VFDT) algorithm in the literature.

The pseudocode of Hoeffding Tree is available in Algorithm A.3. The counts n_{ijk} are the sufficient statistics needed for most attribute selection measures. Pre-pruning is carried out by considering a '*null*' attribute $X_\emptyset$ at each node. A split is made if it is found to be better than the not splitting case, with confidence $1-\delta$, according to G. Meanwhile, at *any time point*, the TEST function may be called to use the current tree to for the prediction task (Domingos and Hulten, 2000).

2.2.5.4 Perceptron

The classic Perceptron algorithm is used for *binary* classification (Freund and Schapire, 1998). 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. 2.4 represents a general structure of a Perceptron network, where an instance $\mathbf{x}$ with k attributes, i.e., $\mathbf{x} = (x_1, x_2, ..., x_k)$, is fed to the Perceptron. Each attribute is associated with a weight. The weighted sum of inputs, i.e., $\sum_{i=1}^{k} w_i x_i$, 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 $\mathbf{W}^T\mathbf{X}$ where $\mathbf{W}$ is a weight vector as $\mathbf{W} = (w_1, w_2, ..., w_k)$. Following Bifet et al. (2010b), we use the *Sigmoid function*[45] as the activation function in our Perceptron algorithm. It is worth mentioning that Perceptron can be used to model *linearly separable* functions (Zhou et al., 2017).

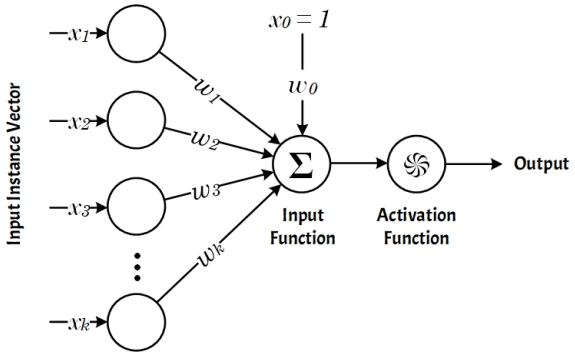


Fig. 2.4. Perceptron Structure

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-square error (MSE) is considered as a cost function, and calculated by Equation (2.5), where n is the number of instances in the training set, y_j is the real class and $\tilde{f}_{\mathbf{w}}(\mathbf{x}_j)$ is the activation function that outputs the prediction.

4 The Sigmoid function is $\sigma(x) = 1/(1+e^{-x})$. The return value monotonically increases from 0 to 1.
5 The Sigmoid function is also known as the Logistic function in the literature.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{j}^{n}(y_j - \tilde{f}_{\mathbf{w}}(\mathbf{x}_j))^2 \tag{2.5}$$

For minimizing the cost function $J(\mathbf{w})$, the *gradient descent* optimizer may be used to update the weights as instances are processed until it converges to a minimum point. The update rule is as defined follows:

$$\mathbf{w} = \mathbf{w} - \eta\nabla J(\mathbf{w}) \tag{2.6}$$

where $\mathbf{w}$ is the weight vector and $\nabla J(\mathbf{w})$ 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., $\nabla J(\mathbf{w})$, is measured by Equation (2.7).

$$\nabla J = -\sum_{j}^{n}(y_j - \tilde{f}_{\mathbf{w}}(\mathbf{x}_j))\nabla\tilde{f}_{\mathbf{w}}(\mathbf{x}_j) \tag{2.7}$$

Assuming the Sigmoid function as the activation function, the gradient of the *hypothesis* function[6], i.e., $\nabla\tilde{f}_{\mathbf{w}}(\mathbf{x}_j)$, is calculated by Equation (2.8).

$$\nabla\tilde{f}_{\mathbf{w}}(\mathbf{x}_j) = \tilde{f}_{\mathbf{w}}(\mathbf{x}_j)(1 - \tilde{f}_{\mathbf{w}}(\mathbf{x}_j))\mathbf{x}_j \tag{2.8}$$

Using Equations (2.6) to (2.8), the final weight update rule is:

$$\mathbf{w} = \mathbf{w} + \eta\sum_{j}^{n}(y_j - \tilde{f}_{\mathbf{w}}(\mathbf{x}_j))\tilde{f}_{\mathbf{w}}(\mathbf{x}_j)(1 - \tilde{f}_{\mathbf{w}}(\mathbf{x}_j))\mathbf{x}_j \tag{2.9}$$

Equation (2.9) is used to update weights in a batch learning mode, where there is a training set with n instances. In a data stream scenario, this update rule would not be practical. As an alternative, the *stochastic gradient descent* optimizer is used for updating the weight vector for every instance of the data stream (Bifet et al., 2010b). Using the *stochastic gradient descent* optimizer, the weights are updated as follows:

$$\mathbf{w} = \mathbf{w} + \eta \cdot (y - \tilde{f}_{\mathbf{w}}(\mathbf{x}))\tilde{f}_{\mathbf{w}}(\mathbf{x})(1 - \tilde{f}_{\mathbf{w}}(\mathbf{x}))\mathbf{x} \tag{2.10}$$

6 Please note the derivative of Sigmoid function, $\sigma(x)$, is $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

Recall that the Perceptron algorithm is only usable for a binary classification task. For a multi-class problem, however, we can train one Perceptron for each class (Bifet et al., 2010b). That is, for a problem with m class labels, we have m Perceptrons and m weight vectors as $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_m$. If we want to classify an unseen instance $\mathbf{x}$, we need to make m predictions as $\tilde{f}_{\mathbf{w}_1}(\mathbf{x})$, $\tilde{f}_{\mathbf{w}_2}(\mathbf{x})$, ..., $\tilde{f}_{\mathbf{w}_m}(\mathbf{x})$ using m Perceptrons. Finally, the predicted class is the one with with greatest value of $\tilde{f}$, i.e., $argmax_c \tilde{f}_{\mathbf{w}_c}(\mathbf{x})$. The pseudocode of Perceptron is presented in Algorithm A.4. The reader should note that the Sigmoid function was considered as the activation function for the pseudocode.

2.2.5.5 K-Nearest Neighbours

The classification algorithms, which were discussed so far, are examples of *eager learners*. Firstly, an eager learner trains a model. The model is, then, used to predict the label of a new instance. That is, the learned model is ready and eager to be used for classification. On the contrary, a *lazy learner* stores training instances in memory and waits until it is given an unseen test instance (Han et al., 2011). Finally, it predicts the label of a test instance based on the similarity between the given instance and the training instances. Lazy learners are also known as *instance-based learners*, implying that they rely on instances other than a model to predict the labels.

K-Nearest Neighbours (K-NN) is a *lazy learner*, introduced by (Sillverman et al., 1951). To predict the label of instance $\mathbf{x}$, a K-nearest neighbour classifier searches the existing instances to find the K *closest* instances to the instance $\mathbf{x}$. These K training instances are called as 'K-nearest neighbours' of the instance $\mathbf{x}$. The most common label among 'K-nearest neighbours' is considered as the label of instance $\mathbf{x}$. *Closeness* is defined in terms of a distance measure, such as the Euclidean distance. The Euclidean distance between two instances $\mathbf{x}_p = (x_1^p, x_2^p, ..., x_k^p)$ and $\mathbf{x}_q = (x_1^q, x_2^q, ..., x_k^q)$ is:

$$dist(\mathbf{x}_\mathrm{p}, \mathbf{x}_\mathrm{q}) = \sqrt{\sum_{i=1}^{k} (x_i^p - x_i^q)^2} \tag{2.11}$$

It is necessary to normalize the values of every attribute before using Equation (2.11). This aids to prevent attributes with large ranges to outweigh attributes with smaller ranges. The *min-max normalization* is typically used to map all values of all attributes in the range [0, 1]. This normalization technique is formalized as follows:

$$v' = \frac{v - min_A}{max_A - min_A} \tag{2.12}$$

where min_A and max_A are the minimum and maximum values of attribute A. Also, v and v' are the original and normalized values, respectively. For nominal attributes, the *Hamming*

distance (Hamming, 1950) may be used. The Hamming distance between two instances is the number of attributes at which the corresponding values are different. For example, consider two examples of $\mathbf{x}_p = (a, b, b, c)$ and $\mathbf{x}_q = (a, c, a, b)$. The distance between these two examples is $(0 + 1 + 1 + 1) = 3$. The Euclidean distance and the Hamming distance may be used together to measure closeness when numeric and nominal attributes are both present in data.

K-Nearest Neighbours are labour intensive and extremely slow when the training data are large. That is, they are expensive in terms of memory usage and execution runtime (Han et al., 2011). It becomes more challenging in a data stream setting because learning algorithms must use a limited amount of memory (Requirement 2) and must work in a limited amount of time (Requirement 3). To address this, it is possible to use a sliding window with a size of n for holding the n most recent instances of a data stream (Mouratidis and Papadias, 2007). For predicting the label of a new instance, the K-nearest neighbours are found among instances in the sliding window. The majority class among K-nearest neighbours is considered as the label of the new instance. The pseudocode of a K-NN with a sliding window is represented in Algorithm A.5.

2.2.5.6 Discussion

We summarize the pros and cons of the learning algorithms that we discussed in Table 2.2. Naive Bayes and Perceptron are efficient in terms of memory usage and runtime compared to other algorithms. Despite the class conditional independence assumption, Naive Bayes have shown comparable accuracy to other algorithms in the literature (Han et al., 2011). Decision Stump may be susceptible to underfitting. Perceptron is not able to adequately model nonlinear problems. Models generated by Decision Stump and Hoeffding Trees can be easily interpreted. Hoeffding Trees typically consume more memory compared to Naive Bayes and Perceptron. Although K-Nearest Neighbours algorithm is easy to implement and interpret, it has an extensive memory usage and execution runtime.

Table 2.2. Advantages and Disadvantages of Learning Algorithms

Algorithm	Advantages	Disadvantages
Naive Bayes	Simple, efficient, effective and robust to noisy data	Independence assumption is not often hold in the real-world
Decision Stump	Simple, and understandable	Susceptible to underfitting
Hoeffding Trees	Easy to understand, robust to noisy data	Consuming more memory compared to other algorithms
Perceptron	Simple and Efficient	Cannot model nonlinear problems
K-Nearest Neighbours	Easy to implement, Understandable	Extensive memory usage and execution runtime

In the literature, Hoeffding Tree and Naive Bayes are often used, due to their high accuracy, robustness to noise, and ease of interpretability, with drift detection algorithms, for learning from evolving data streams (Bifet et al., 2009, 2010b; Frías-Blanco et al., 2015; de Lima Cabral and de Barros, 2018; de Barros et al., 2018; Duong et al., 2018a,b). Hence, we consider Hoeffding Tree and Naive Bayes as the base learners when we assess our drift detection methods against the state-of-the-art in Chapter 3. We also employ all algorithms described above for multi-strategy learning in Chapter 4. The reader should note that we rely on only individual learners than an ensemble of them, e.g., OzaBag and OzaBoost (Oza and Russell, 2001), for our experiments. In an ensemble scenario, a number of classifiers participate in predicting the label of a new instance, for example, by a majority voting solution. We may consider the ensemble algorithms as an avenue for future work.

2.3 Adaptive Classification

Recall that *adaptive learning* addresses the concept drift challenge (Gama et al., 2014), by retraining the predictive models, either partially or globally, once a drift is experienced.

We provide a formal definition for the concept drift phenomenon in Section 2.3.1, and then explain the adaptation solutions for handling concept drift in Section 2.3.2. Section 2.3.3 studies the requirements for adaptive learning. Finally, we cover the state-of-the-art methods for drift detection in Section 2.3.4.

2.3.1 Concept Drift Phenomenon

In evolving/non-stationary environments, the underlying distribution of data may change over time leading to the phenomenon of *concept drift* (Schlimmer and Granger Jr, 1986; Widmer and Kubat, 1996). For example, a shift in one's interest in the genre of movies can be considered as concept drift. As a consequence of concept drift, decision makers may become deprecated and obsolete, as their accuracy may drop.

We provide the formal definition of concept drift in Section 2.3.1.1. The general assumption regarding concept drift is that we may not be able to predict when it happens. However, in some real-world applications, it is possible to anticipate concept drift before it occurs by considering correlated environmental or political events (Gama et al., 2014). We also cover different patterns of concept drift in Section 2.3.1.2.

2.3.1.1 Formal Definition

The Bayesian Decision Theory (Duda et al., 2012) is commonly employed in describing the classification problem based on the prior probability distribution of classes, i.e., $p(y)$, and

the class conditional probability distribution, i.e., $p(\mathbf{x}|y)$, for all $y \in \{c_1, ..., c_m\}$, where c_m represents the m^{th} class label (Žliobaitė, 2010; Gama et al., 2014). The classification decision is defined by the posterior probabilities of the classes. The posterior probability associated with class c_i, given instance $\mathbf{x}$, is calculated by:

$$p(c_i|\mathbf{x}) = \frac{p(c_i) \cdot p(\mathbf{x}|c_i)}{p(\mathbf{x})} \tag{2.13}$$

where $p(\mathbf{x}) = \sum_{i=1}^{m} p(c_i) \cdot p(\mathbf{x}|c_i)$ is the marginal probability distribution. Formally, concept drift occurs in between time point t_0 and time point t_1 if:

$$\exists \mathbf{x} : p_{t_0}(\mathbf{x}, y) \neq p_{t_1}(\mathbf{x}, y) \tag{2.14}$$

where p_{t_0} and p_{t_1} denote the joint probability distributions at time t_0 and t_1, respectively, for $\mathbf{x}$ and y (Gama et al., 2014). The joint probability distribution of $\mathbf{x}$ and y is defined by $p(\mathbf{x}, y) = p(y|\mathbf{x}) \cdot p(\mathbf{x}) = p(\mathbf{x}|y) \cdot p(y)$. Equation (2.14) implies that the data distribution at time t_0 and t_1 are distinct. Equation (2.13) may be considered to declare changes in the data distribution (Kelly et al., 1999; Gao et al., 2007; Gama et al., 2014), as follows:

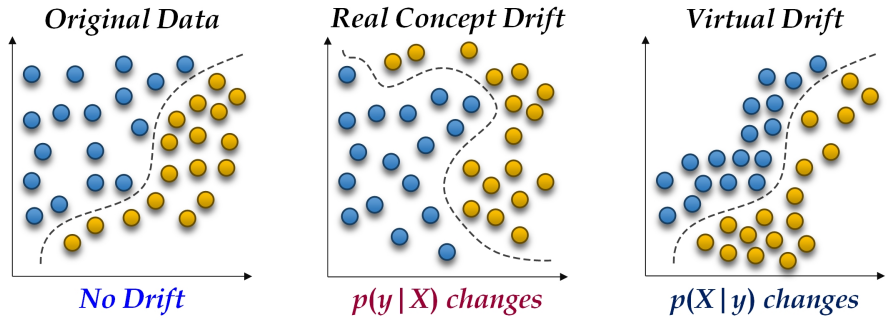
- A change may happen in the prior probability distribution $p(y)$,
- A change may happen in the class conditional probability distribution $p(\mathbf{x}|y)$,
- A change may happen in the posterior probability distribution $p(y|\mathbf{x})$, thus affecting the classification decision boundaries.

Widmer and Kubat (1993) categorized changes in the data distribution into two groups of *real concept drift* and *virtual concept drift*. A real concept drift affects the target concept, whereas, a virtual drift may appear as the completion of the concept of learning. Žliobaitė (2010) and Gama et al. (2014) formalized these two types of drifts as follows:

1. *Real concept drift* refers to changes in $p(y|\mathbf{x})$, which affects the target concept of learning (as shown in Fig. 2.5 (b)). This kind of drift is also referred to as *concept shift* in (Salganicoff, 1997) and *conditional change* in (Gao et al., 2007).

2. *Virtual concept drift* emerges by changes in $p(\mathbf{x})$, and subsequently in $p(\mathbf{x}|y)$, but not necessarily in $p(y|\mathbf{x})$ (as shown in Fig. 2.5 (c)). A virtual drift is a change in the distribution of the incoming data without altering the concept of learning. This type of drift is also called as *sampling shift* (Salganicoff, 1997), *temporary drift* (Lazarescu et al., 2004), and *feature change* (Gao et al., 2007).

In practice, a virtual drift that changes the prior probability distribution of classes, i.e., $p(y)$, may appear in a combination with a real concept drift. In such a scenario, the target concept is also affected.



(a) Original Data **(b)** Real Drift **(c)** Virtual Drift

Fig. 2.5. Real Concept Drift vs. Virtual Concept Drift

Gama et al. (2014) used the following scenario as an example to explain real concept drift and virtual drift in a real-world application. Consider an online stream of news articles on real estate. A user wants to classify the incoming news into relevant and not relevant. Suppose, at the moment, the user is looking for a new apartment. That means the news on dwelling houses is relevant, and the news on holiday homes is irrelevant. If the editor of the news changes, the writing style changes as well, but the dwelling houses remain relevant for the user. This scenario corresponds to virtual drift. Suppose due to a crisis more articles on dwelling houses and fewer articles on holiday homes appear, but the editor, the writing style, and the interests of the user remain the same, this situation corresponds to drift in prior probabilities of the classes. On the contrary, if the user has bought a house and starts looking for a holiday destination, dwelling houses become irrelevant and holiday places become relevant. Although the writing style and the prior probabilities stay the same, the scenario corresponds to the real concept drift.

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 (Gama et al., 2014). 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.

2.3.1.2 Concept Drift Patterns

A concept drift may appear in different patterns. As illustrated in Fig. 2.6, concept drift may happen *abruptly*, *gradually*, *incrementally*, or it may *re-occur* again over time.

Abrupt concept drift happens as a sudden change from one concept to another, as depicted in Fig. 2.6 (a). For example, John's favorite book genre may suddenly change from science fiction (sci-fi) to mystery. Another example is the replacement of a sensor with another that has a different calibration in a chemical plant (Gama et al., 2014).

In contrast to abrupt drift, gradual concept drift happens with a slower transition from one concept to another. That is, during the transition, we may experience examples from two concepts. For example, John does not get interested in mystery book suddenly, but rather gradually over time. That means he is interested in science fiction books. Then, he becomes eager to read mystery books as well. He reads mystery books beside science fiction ones. As time passes, he becomes more interested in mystery books until he eventually becomes a fan of mystery books. Fig. 2.6 (b) demonstrates gradual concept drift.

In incremental concept drift, many intermediate concepts happen in a transition from one concept to another one. In the John example, his interest may change from sci-fi to sci-fi + adventure, from sci-fi + adventure to adventure, from adventure to adventure + mystery, and finally from adventure + mystery to mystery. In the case of sensors, a sensor may slowly wear off and becomes less accurate. Fig. 2.6 (c) illustrates incremental concept drift.

There is yet another type of concept drift referred to as re-occurring concept (Žliobaitė, 2010). As Fig. 2.6 (d) presents a previously active concept reappears after some time. A re-occurring concept is different from common seasonality notion in a way that it is not periodic (Žliobaitė, 2010). Back to our example, John is currently interested in sci-fi books. He finds out that his favorite author published a few books with the genre of mystery. So, he reads those books for a couple of months. Then, he wins some coupons for buying sci-fi books. He again starts readying sci-fi books which he bought by the coupons.

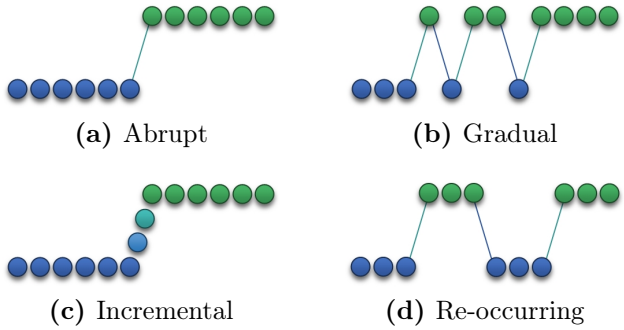


Fig. 2.6. Patterns of Concept Drifts

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 (Gama et al., 2014). This is a critical observation since a decision model would be 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. Chandola et al. (2009) accomplished a survey on outlier and anomaly detection in data.

2.3.1.3 Concept Drift Terminology

The concept drift phenomenon has been studied by various research communities including data mining, machine learning, statistics, and information retrieval (Žliobaitė, 2010). This phenomenon, however, may be known by different terms in each community. Table 2.3 lists the terms that correspond with concept drift phenomenon in different research areas.

Table 2.3. Concept Drift Terminology
(Žliobaitė, 2010)

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

2.3.2 Adaptation Approaches

Recall that (online) learning algorithms must have the ability to adjust themselves to the new situations in non-stationary environments, where concept drift is not avoidable. Otherwise, their classification models become obsolete, and subsequently, their error-rates increase. Hence, the ability to adapt to concept drift has to considered as an extension for incremental learners when they train their predictive models, instance-by-instance, against evolving data streams (Giraud-Carrier, 2000).

Adaptive learning may be seen as advanced incremental learning which is able to detect distributional changes or concept drift in data streams, and to adapt the predictive model accordingly (Gama et al., 2014). Adaptation methods fall into two major groups of *blind* and *informed*, as shown in Fig. 2.7.

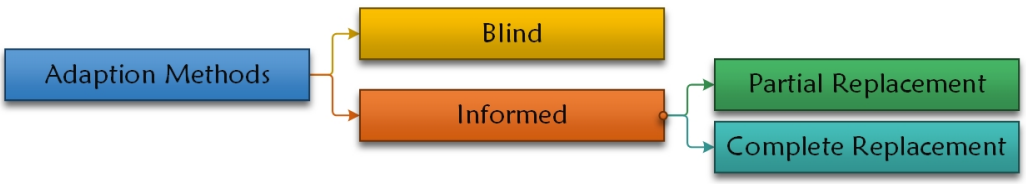


Fig. 2.7. Adaptation Methods

Blind Adaptation – The blind adaptive algorithms update their classification models without any explicit detection of concept drift. These algorithms may use a sliding window, for example, with a size of n, to hold the n most recent instances from a data stream. As the window is slid over the instances, the predictive model is *periodically* retrained. The weighting techniques may also be used to emphasize the importance of recent instances (Widmer and Kubat, 1996; Klinkenberg and Renz, 1998; Lanquillon, 2001).

The blind algorithms are proactive, meaning that they request an update before a concept drift even occurs. Moreover, they forget information periodically, at a constant speed, without considering whether any concept drift has happened or not. As a result, some advantageous patterns may be lost. Frequent blind updates may also keep the computational resources busier.

Informed Adaptation – The informed adaptive algorithms are reactive, which means they update their classification models once an alarm has triggered for concept drift by a drift detector (Bifet and Gavalda, 2006; Hulten et al., 2001). Drift detectors may be separate from the adaptive algorithm or be integrated with the algorithm. The latter case is referred to as *model integrated detectors* (Gama et al., 2006; Ikonomovska et al., 2011). The reaction to an alarm for concept drift may happen in the form of *partial replacement* or *global replacement*.

- *Complete/Global Replacement* – It refers to the full reconstruction of predictive models. That is, the outdated model is discarded, and a new model is trained from scratch once a concept drift is detected. This strategy has been widely used in the literature (Klinkenberg and Joachims, 2000; Street and Kim, 2001; Gama et al., 2004; Zeira et al., 2004). This strategy works for *global classifiers*, e.g., Naive Bayes and Perceptron, as well as *granular classifiers*, e.g., Decision Rules and Decision Trees.

- *Partial/Local Replacement* – Concept drift may affect only some regions of decision space. In such a case, only some parts of the classification model are required to be updated to reflect the partially changed decision space. This strategy works for only *granular classifiers*, e.g., decision trees, and decision rules, where only some parts of models are updated. In a decision tree (or a decision rules model), each node (or rule) covers a hyper-rectangle in the data space. Hence, for such decomposable models, we only need to update those nodes that cover the region of the data space affected by concept drift (Gama et al., 2014). CVFDT by Hulten et al. (2001) and VFDTc by Gama et al. (2006) are examples of incremental decision trees able to partially adapt themselves to concept drift.

The reader should note that explicit/informed adaptation has advantages which blind adaptation does not offer. In many settings, e.g., in Business Intelligence (BI) and medical applications, we may be interested in knowing why and how a concept drift was

experienced, which may let us have the best reaction for similar situations in the future. Therefore, it would be rather impossible to investigate and understand the reasons behind the occurrence of the drift without knowing its time of appearance. Furthermore, detecting concept drift *intelligently* and *accurately* is vital in many applications, including, but not limited to, nuclear reactors, self-driving cars, medical diagnostics, and military. In such applications, adaptation should take place immediately with the least false negative rate of drift detection. Otherwise, detrimental consequences, i.e., loss of life and environmental hazards, would be inevitable.

2.3.3 Adaptation Requirements

Recall that, in evolving settings, learning algorithms need to detect concept drifts and adapt their predictive models for the new situations. As we presented in Section 2.2.3, incremental learners have to follow four requirements, i.e., *processing an example only once* (Requirement 1), *using a limited amount of memory* (Requirement 2), *working in a limited amount of time* (Requirement 3), and *being ready to predict at any point* (Requirement 4) against evolving data streams. While adaptive learners should conform to Requirements 1 to 4, they have to fulfill three additional requirements, to keep the quality of learning high, as follows:

- **Adaptation Requirement 5 (A-R5): Minimum false positive and false negative rates** – An adaptive learning algorithm needs to detect drift points with fewer false positives and fewer false negatives. A high false positive number leads to more model retraining or replacement, leading to keeping resources busier (Žliobaitė et al., 2015). On the other hand, a high false negative number causes decay in the accuracy of classification, since the concept drifts are not detected.

- **Adaptation Requirement 6 (A-R6): Short detection delay** – An adaptive learner needs to detect concept drifts as soon as possible, i.e., with the least delay, and updates its classification model accordingly. Otherwise, as a consequence, the classification accuracy would increase by a delay. If we detect concept drift with a shorter delay, we can use more instances from the new distribution which may lead to having a higher classification accuracy.

- **Adaptation Requirement 7 (A-R7): Robustness to noise** – Adaptive learners must be able to distinguish concept drift from noise. As discussed in Section 2.3.1.2, no adaptation is required if noise is experienced in a data stream. For example, if a false alarm is triggered due to noise, an adaptive approach may cast off its predictive model, and all computations become wasted. False alarms may keep the resources busy as well. Therefore, adaptive algorithms are supposed to be robust against noise.

2.3.4 Concept Drift Detection Methods

The change detection methods refer to the algorithms that are used to *explicitly* detect drift points, and distributional changes, in data streams. They characterize and quantify concept drift by discovering the change points or small time intervals during which changes occur (Basseville et al., 1993). Gama et al. (2014) categorized concept drift detectors into three general groups as follows:

1. *Sequential Analysis based Methods* sequentially evaluate prediction results as they become available. They alarm for concept drift when a pre-defined threshold is met. The Cumulative Sum (CUSUM) and its variant PageHinkley (PH) (Page, 1954), as well as Geometric Moving Average (GMA) (Roberts, 2000) are representatives of this group.

2. *Statistical based Approaches* analyze statistical parameters such as the mean and the standard deviation associated with the predicted results in order to detect concept drifts. The Drift Detection Method (DDM) (Gama et al., 2004), Early Drift Detection Method (EDDM) (Baena-Garcıa et al., 2006), Exponentially Weighted Moving Average (EWMA) (Ross et al., 2012), and Reactive Drift Detection Method (RDDM) (Barros et al., 2017) are members of this group.

3. *Windows based Methods* usually utilize a fixed reference window for summarizing the past information and a sliding window for summarizing the most recent information. A significant difference in between the distributions of these two windows implies the occurrence of concept drift. Statistical tests or mathematical inequalities, with the null-hypothesis indicating that the distributions are equal, are employed. The Adaptive Windowing (ADWIN) (Bifet and Gavalda, 2007), the SeqDrift detectors (Pears et al., 2014), and the Drift Detection Methods based on Hoeffding's Bound ($\text{HDDM}_{\text{A-test}}$ and $\text{HDDM}_{\text{W-test}}$) (Frías-Blanco et al., 2015) are members of this family.

CUSUM and its variant PageHinkley (PH) are some of the pioneer methods in the community. DDM, EDDM, and ADWIN have frequently been considered as benchmarks in the literature (Frías-Blanco et al., 2015; Baena-Garcıa et al., 2006; Bifet and Gavalda, 2007; Huang et al., 2015). The SeqDrift2, HDDMs, and RDDM algorithms present similar performances. Therefore, we consider all these methods for our experimental study in Chapters 3 and 4. We briefly explain the methods in the following sections.

2.3.4.1 Cumulative Sum Variants

Cumulative Sum (CUSUM), by Page (1954), alarms for a change when the mean of the input data significantly deviates from zero. The input of CUSUM may, for instance, be

the prediction error from a Kalman filter (Gama et al., 2014). The CUSUM test has the form $g_t = max(0, g_{t-1} + (x_t - \delta))$, and alarms for concept drift when $g_t > \lambda$. Here, x_t is the current observed value, δ specifies the magnitude of allowed changes, $g_0 = 0$, and λ is a user-defined threshold. The accuracy of CUSUM depends on both δ and λ. Lower values of δ result in a faster detection, but at the cost of more false alarms.

PageHinkley (PH), by Page (1954), is a variant of CUSUM employed for change detection in signal processing (Gama et al., 2014). The test variable m_T is defined as the cumulative difference between the observed values and their mean until the current time T; and is evaluated by $m_T = \sum_{t=1}^{T}(x_t - \bar{x}_T - \delta)$, where $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x_t$ and where δ controls the variation range. The PH method also updates the minimum of m_T, denoted as M_T, using $M_T = min(m_t, t = 1...T)$. A significant difference between m_T and M_T, i.e., $PH_T : m_T - M_T > \lambda$ where λ is a user-defined threshold, indicates a concept drift. A large value of λ typically results in fewer false alarms, but at the expense of increasing the false negative rate.

2.3.4.2 Drift Detection Method

The Drift Detection Method (DDM), by Gama et al. (2004), monitors 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 of $s_t = \sqrt{(p_t(1-p_t)/t)}$ at time t. As instances are processed, DDM updates two variables p_{min} and s_{min} when $p_t + s_t < p_{min} + s_{min}$. DDM warns for a drift when $p_t + s_t \geq p_{min} + 2 * s_{min}$, and alarms for 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.

2.3.4.3 Early Drift Detection Method

The Early Drift Detection Method (EDDM), by Baena-Garcıa et al. (2006), evaluates the distances between wrong predictions to alarm for concept drift. The algorithm assumes that concept drift is more likely to occur when the distances between errors become smaller. EDDM calculates the average distance between two recent errors, i.e., p'_t, with its standard deviation s'_t at time t. It updates two variables p'_{max} and s'_{max} when $p'_t + 2 * s'_t > p'_{max} + 2 * s'_{max}$. The method warns for a drift when $(p'_t + 2 * s'_t)/(p'_{max} + 2 * s'_{max}) < \alpha$, and detects a concept drift when $(p'_t + 2 * s'_t)/(p'_{max} + 2 * s'_{max}) < \beta$. The authors set α and β to 0.95 and 0.90, respectively. The p'_{max} and s'_{max} are reset only when a drift is detected.

2.3.4.4 Reactive Drift Detection Method

The Reactive Drift Detection Method (RDDM), by Barros et al. (2017), addresses the performance loss problem of the Drift Detection Method (DDM) when the sensitivity of the method deteriorates over time, particularly for large concept drifts. Similar to DDM, RDDM evaluates two variables p_t and s_t; and updates p_{min} and s_{min} if $p_t + s_t < p_{min} + s_{min}$. The constants α_w and α_d indicate the warning and drift levels, respectively. Moreover, RDDM holds three variables: max (the maximum size of a concept), min (the reduced size of a stable concept), and $warnLimit$ (the maximum number of instances that limits the warning level). The method warns for a drift when $p_t + s_t > p_{min} + \alpha_w * s_{min}$, and alarms for a drift when either of following three conditions occur (1) $p_t + s_t > p_{min} + \alpha_d * s_{min}$, (2) $num\_instances > max$, or (3) $num\_warnings > warnLimit$.

2.3.4.5 Adaptive Windowing

Adaptive Windowing (ADWIN), by Bifet and Gavalda (2007), slides a window w on the prediction results as they become available, in order to detect drifts. The method examines two sub-windows of sufficient length, i.e., w_0 of size n_0 and w_1 of size n_1 where $w_0 \bullet w_1 = w$. The reader should note that the symbol $\bullet$ represents the concatenation of two windows. A significant difference between the means of two sub-windows indicates a concept drift, i.e., when $|\hat{\mu}_{w_0} - \hat{\mu}_{w_1}| \geq \varepsilon$ where $\varepsilon = \sqrt{\frac{1}{2m} \ln \frac{4}{\delta'}}$, m represents the harmonic mean of n_0 and n_1, and $\delta' = \delta / n$. Here δ is the confidence level while n is the size of window w. Once a drift is detected, elements are removed from the tail of the window until no significant difference is observed.

2.3.4.6 SeqDrift2 Detector

SeqDrift2, by Pears et al. (2014), relies on a reservoir sampling method (Vitter, 1985), as an adaptive sampling strategy, for random sampling from input data. SeqDrift2 stores entries into two repositories called *left* and *right*. As entries are processed over time, the left repository forms a combination of old and new entries by applying the reservoir sampling strategy, while the right repository collects the new entries. SeqDrift2 subsequently finds an upper-bound for the difference in between the means of the two repositories, i.e., $\hat{\mu}_l$ for the left repository and $\hat{\mu}_r$ for the right repository, using the Bernstein inequality (Bernstein, 1946). Finally, a significant difference between the two means suggests a concept drift.

2.3.4.7 Drift Detection Methods based on Hoeffding's Bound

$\text{HDDM}_{\text{A-test}}$, by Frías-Blanco et al. (2015), uses six variables, i.e., $total_n$, $total_c$, n_{min}, c_{min}, n_{max}, and c_{max} to detect concept drift. The variables $total_n$ and $total_c$ hold the total number

of entries and the total number of incorrect predictions, respectively. The variables n_{min} and c_{min} are updated if $c_{min}/n_{min}+cota \geq total_c/total_n+cota_t$, whereas, n_{max} and c_{max} are updated if $c_{max}/n_{max} - cota \leq total_c/total_n - cota_t$. The $cota$ and $cota_t$ are calculated by the Hoeffding inequality (Hoeffding, 1963). A significant difference between c_{min}/n_{min} and $total_c/total_n$, bounded by Hoeffding's inequality, represents the occurrence of concept drift. The reader may note that n_{max} and c_{max} are used for resetting all variables if required. $\text{HDDM}_{\text{W-test}}$, a variant of $\text{HDDM}_{\text{A-test}}$, employs the EMWA forgetting scheme (Ross et al., 2012) to update its statistics. The EMWA scheme is $\hat{X}_t = (1-\lambda) * \hat{X}_{t-1} + \lambda * X_t$ where $t \geq 1$, $\hat{X}_1 = X_1$, and X_t is the entry at time t. The authors suggested that the first and the second methods are ideal for detecting abrupt and gradual drifts, respectively. Finally, the HDDM algorithms are analogous to DDM by (Gama et al., 2004).

2.3.4.8 Discussion

CUSUM and PageHinkley (PH) alarm for concept drift when the deviation of the observed values from their mean exceeds a user-defined threshold. These algorithms are sensitive to the parameter values, resulting in a trade-off between false alarms and detecting true drifts. DDM and EDDM require less memory as only a small number of variables are maintained (Gama et al., 2014). On the other hand, the ADWIN and SeqDrift2 approaches necessitate multiple subsets of the stream which lead to more memory consumption. They are also computationally more expensive, due to the sub-window compression or reservoir sampling procedures. Barros et al. (2017) observed that, in general, RDDM leads to a higher classification accuracy compared to DDM, especially against data streams with gradual concept drift, despite an increase in false positives. EDDM may frequently alarm for concept drift in the early stages of learning if the distances in between wrong predictions are small. HDDM employs the Hoeffding inequality to detect concept drifts. Recall that SeqDrift2 uses the Bernstein inequality in order to detect concept drift. SeqDrift2 considers the sample variance, and it assumes that the sampled data follow a normal distribution. This assumption may be too restrictive, in real-world domains. Further, the Bernstein inequality is conservative and requires a variance parameter, in contrast to, for instance, the Hoeffding inequality. These shortcomings may lead to longer detection delay and a potential loss of accuracy.

Our preliminary experimentation confirmed that the methods mentioned above might cause long drift detection delay, as well as high false positive and false negative rates. A recent large-scale comparison by de Barros and de Carvalho Santos (2018) affirms our early observations. In Chapter 3, we introduce the Fast Hoeffding Drift Detection Method (FHDDM) and the McDiarmid Drift Detection Methods (MDDMs) to address these shortcomings. We also experimentally demonstrate the promising performance of our methods in comparison with the foregoing methods.

2.4 Data Streams

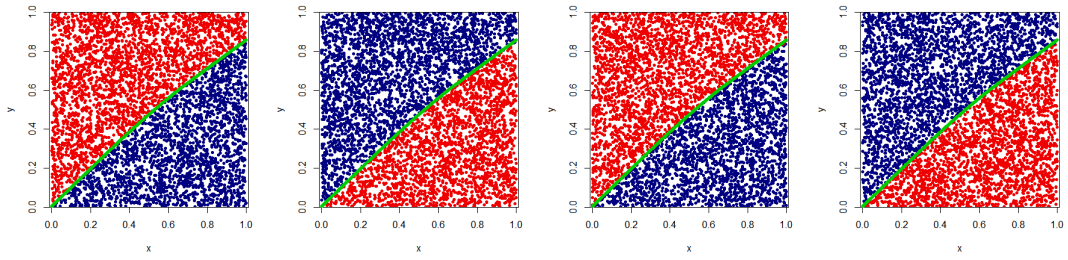
Synthetic and real-world data streams are widely used to evaluate the performance of online and adaptive learners. Synthetic data streams are also known as artificial data streams in the literature. We describe synthetic and real-world data streams in Sections 2.4.1 and 2.4.2, respectively.

As to the real-world data streams, it is not recorded nor identified when concept drift occurs (Bifet et al., 2009; Huang et al., 2015). That is, we do not have the ground truth regarding the drift points for the evaluation of drift detectors. Alternatively, we use *static* real-world datasets to create data streams experiencing concept drift at specific points for our experiments. We describe them as *benchmarking data streams* in Section 2.4.3.

2.4.1 Synthetic Data Streams

SINE1, SINE2, MIXED, STAGGER, CIRCLES and LED are synthetic data streams that are frequently applied in the literature (Kubat and Widmer, 1995; Nishida and Yamauchi, 2007; Frías-Blanco et al., 2015; Gama et al., 2004; Bifet and Gavalda, 2007; Olorunnimbe et al., 2015, 2017). We describe them as follows:

- SINE1 · *with abrupt drift*: The stream has two attributes x and y uniformly distributed in the range $[0, 1]$. The classification function is $y = sin(x)$. Instances are classified as positive if they are under the curve, negative otherwise. Once a drift occurs, the classification is reversed. Fig. 2.8 gives an example of SINE1 data stream with four contexts. A context is the interval between two consecutive drifts.



(a) Context 1 **(b)** Context 2 **(c)** Context 3 **(d)** Context 4

Fig. 2.8. The SINE1 Data Stream

- SINE2 · *with abrupt concept drift*: It holds two attributes x and y that are uniformly distributed in the range $[0, 1]$. The classification function is $0.5 + 0.3 * sin(3\pi x)$. Similar to SINE1, instances are classified as positive if they are under the curve, negative otherwise. At a drift point, the classification scheme is reversed.

- MIXED · *with abrupt drift*: It consists of two numeric attributes x and y, distributed in $[0, 1]$, with two boolean attributes v and w. The instances are classified as positive if at least two of the three following conditions are met: $v, w, y < 0.5 + 0.3 * sin(3\pi x)$. The classification is reversed when concept drift occurs.

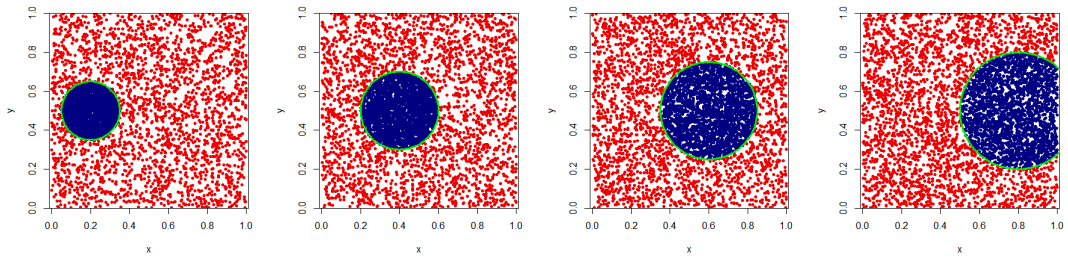
- STAGGER · *with abrupt concept drift*: It contains three nominal attributes, namely $size = \{small, medium, large\}$, $color = \{red, green\}$ and $shape = \{circular, non\text{-}circular\}$. For the first concept (or context), instances are labelled as positive if $(color = red) \wedge (size = small)$. Regarding the second concept, instances are classified as positive if $(color = green) \vee (shape = circular)$. Finally, for the last concept, instances are classified as positive if $(size = medium) \vee (size = large)$.

- CIRCLES · *with gradual drift*: It comes with two attributes x and y which are uniformly distributed in the interval $[0, 1]$. The function of circle $<(x_c, y_c), r_c>$ is $(x - x_c)^2 + (y - y_c)^2 = r_c^2$ where x_c and y_c are its center and radius, respectively. Instances are classified as positive if they are inside the circle. Concept drift happens whenever the classification function, i.e., circle function, changes. Four circles, as listed in Table 2.4, are used to classify instances over time.

Table 2.4. The CIRCLES Data Stream

Center	$(0.2, 0.5)$	$(0.4, 0.5)$	$(0.6, 0.5)$	$(0.8, 0.5)$
Radius	0.15	0.2	0.25	0.3

We illustrate an example of the CIRCLES data stream with four contexts in Fig. 2.9. Concept drift is simulated by moving the circle to right with increasing its radius.



(a) Context 1 **(b)** Context 2 **(c)** Context 3 **(d)** Context 4

Fig. 2.9. The CIRCLES Data Stream

- LED · *with gradual drift*: The objective of this data is to predict the digit on a seven-segment display, where each digit (i.e., 0 to 9) has a 10% chance of being displayed. The data has 7 attributes which define the class and 17 irrelevant attributes. Concept drift is simulated by interchanging relevant attributes (Frías-Blanco et al., 2015).

2.4.1.1 Concept Drift Simulation

We use the synthetic data streams for our experiments in Chapters 3 and 4. Recall that SINE1, SINE2, MIXED, STAGGER, CIRCLES have only 2 class labels, whereas LED has 10 class labels. Each data stream contains $100,000$ instances. Following the convention, we placed the drift points at every $20,000$ instances in SINE1, SINE2, and MIXED, and at every $33,333$ instances in STAGGER with a transition length of $\zeta = 50$ to simulate *abrupt* concept drift. For CIRCLES and LED data streams, we placed the drift points at every $25,000$ instances with a transition length of $\zeta = 500$ to simulate *gradual* concept drift. The reader should note that the transition length defines the speed of change. We have applied the sigmoid function to simulate the transition between two consecutive contexts.

Sigmoid functions are often used to simulate *abrupt* and *gradual* concept drifts (Bifet et al., 2009). We use the sigmoid function $f(t) = 1/(1 + e^{-s(t-t_c)})$, as shown in Fig. 2.10, to calculate the probability of belonging to a new distribution. In this figure, the *length of concept transition* is ζ, and the center of transition is t_c. At time t_c, the slope of the curve is $\tan\alpha = 1/\zeta$, and the derivative of the function is $\tan\alpha = s/4$. That is, we have $s = 4/\zeta$. Finally, the function is written as $f(t) = 1/(1 + e^{-4/\zeta(t-t_c)})$. Using this function, the probabilities of belonging to the new distribution at time t_0 and t_ζ are 0.12 and 0.88, respectively. The probability increases as time passes. We define the speed of concept drift, i.e., abrupt shift or gradual shift, by changing the value of ζ. A shorter ζ results in an *abrupt drift*, while a longer one results in a *gradual drift*.

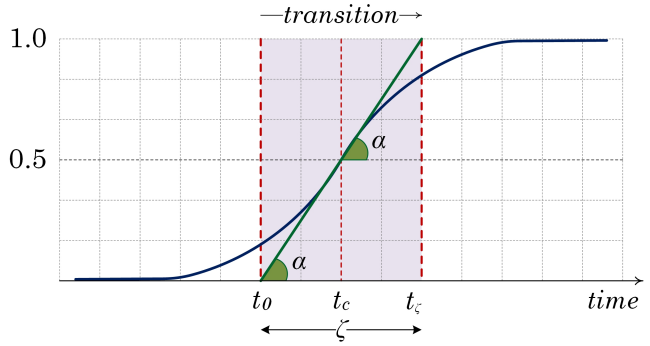


Fig. 2.10. Concept Drift Simulation using Sigmoid Function

We have added 10% class noise[7] to each data stream to observe how robust drift detectors are against noisy data by assessing their ability to distinguish between concept drift and noise. Finally, Table 2.5 summarizes the synthetic data streams.

7 To simulate class noise, we change the true label of an instance with another label.

Table 2.5. Summary of Synthetic Data Streams

Data Stream	Attr.	Attr. Type	Class	Drift Points	ζ	Noise	Drift Type
SINE1	2	Numeric	2	x 20,000	50	10%	abrupt
SINE2	2	Numeric	2	x 20,000	50	10%	abrupt
MIXED	4	Bool. and Num.	2	x 20,000	50	10%	abrupt
STAGGER	3	Nominal	2	x 33,333	50	10%	abrupt
CIRCLES	2	Numeric	2	x 25,000	500	10%	gradual
LED	24	{0,1}	10	x 25,000	500	10%	gradual

2.4.2 Real-World Data Streams

The ELECTRICITY, FOREST COVERTYPE, and POKER HAND data streams, available from the MOA website[8], are often applied in data stream mining literature (Huang et al., 2015; Baena-Garcıa et al., 2006; Bifet and Gavalda, 2007; Frías-Blanco et al., 2015; Olorunnimbe et al., 2015). We describe them as follows:

- ELECTRICITY: It contains 45,312 instances, with 8 attributes, recorded every half an hour for two years from Australian New South Wales Electricity. The classification task is to predict a rise (*Up*) or a fall (*Down*) in the price of electricity. Concept drift may be experienced because of changes in consumption habits, unexpected events, and seasonality (Žliobaitė, 2013).

- FOREST COVERTYPE: It consists of 54 attributes with 581,012 instances describing 7 forest cover types for 30×30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data, for four wilderness areas located in the Roosevelt National Forest of Northern Colorado (Blackard and Dean, 1999). Concept drift may appear as a result of change in the forest cover type.

- POKER HAND: It is composed of 1,000,000 instances, where each instance is an example of five cards drawn from a standard 52 cards deck. Each card is described by two attributes (suit and rank), for a total of ten predictive attributes. The class shows the poker hand (Olorunnimbe et al., 2017).

Tracking Appearance of Concept Drift – There is no ground truth available regarding concept drift in the real-world data streams. That is, we do not know whether concept drift occurs or where it happens in these data streams (Bifet et al., 2009; Huang et al., 2015). Consequently, we cannot measure the detection delay, true positive, false positive, and false negative of drift detectors. We can only evaluate the *average of detection runtime*, *detection memory usage*, and *classification accuracy*.

8 `http://moa.cms.waikato.ac.nz/datasets/`

2.4.3 Benchmarking Data Streams

To the best of our knowledge, there are no real-world datasets publicly available, wherein the locations of concept drifts are identified. As mentioned earlier, there is no ground truth available about concept drift for the ELECTRICITY, FOREST COVERTYPE, and POKER-HAND data streams. There is a consensus among researchers that the location and/or the presence of concept drift in these data streams are unknown (Bifet and Gavalda, 2007; Bifet et al., 2009; Huang et al., 2015; Frías-Blanco et al., 2015). Alternatively, we consider *static* datasets, publicly available from the UCI machine learning repository (Bache and Lichman, 2013), to generate evolving data streams. For that, we may simulate concept drift by switching class labels at the drift points. The reader should note that we call these data stream as *semi-real-world* data streams to distinguish them from real-world data streams. We considered the ADULT (Kohavi, 1996), NURSERY (Zupan et al., 1997), and SHUTTLE (Catlett, 2002) datasets for our experiments. We describe the original datasets as well as the pre-processing steps taken to generate the data streams as follows:

- ADULT: The original dataset has 6 numeric and 8 nominal attributes, 2 class labels, and $48,842$ instances. The dataset was used to predict whether a person earns an annual income greater than $\$50,000$ (Kohavi, 1996).
  ▷ *Pre-processing*: The dataset is imbalanced, and there are $37,155$ instances of class $<= 50K$ as oppose to $11,687$ instances for class $> 50K$. By random sampling, we ensured there are $10,000$ instances for each class, i.e., $20,000$ instances in total, for creating our data stream.

- NURSERY: The dataset holds 8 nominal attributes, 5 classes, and $12,960$ instances. It was designed to predict whether applications for the nursery schools in Ljubljana, Slovenia should be rejected or accepted (Zupan et al., 1997).
  ▷ *Pre-processing*: The class labels of this dataset are 'no_recom', 'recommend', 'very_recom', 'priority', and 'spec_priority'. Since instances of the third and fourth classes are very rare, we ignored them in our sampling, which resulted in a set with $20,000$ instances.

- SHUTTLE: It contains 9 attributes, 7 class labels, and $58,000$ instances. The dataset was designed to predict suspicious situations during a NASA shuttle mission (Catlett, 2002).
  ▷ *Pre-processing*: This dataset is also highly imbalanced. Firstly, the instances of the four minority classes were filtered out. Then, we created a dataset of $20,000$ instances by undersampling and bootstrapping.

Nota Bene – The reader should note that we did not attempt to address any class imbalance problem by sampling, rather we sampled the datasets to prepare sets with $20,000$ instances for creating our data streams.

We simulated concept drift by shifting the class labels after drift points, with a transition length of $w = 500$. Table 2.6 is an example that shows how classes are shifted to simulate concept drift. In Context 1, the instance $\mathbf{x}_1$ is labelled by class c_1. After a concept drift, in Context 2, the instance is labelled by class c_3. Note that the term 'context' refers to the interval between two consecutive concept drifts. Further, the sigmoid function is used to simulate the transition between two contexts, as explained in Section 2.4.1.1.

Table 2.6. Shifting Classes to Simulate Concept Drift

Context 1	Context 2	Context 3
$(\mathbf{x}_1, c_1)$	$(\mathbf{x}_1, c_3)$	$(\mathbf{x}_1, c_2)$
$(\mathbf{x}_2, c_2)$	$(\mathbf{x}_2, c_1)$	$(\mathbf{x}_2, c_3)$
$(\mathbf{x}_3, c_3)$	$(\mathbf{x}_3, c_2)$	$(\mathbf{x}_3, c_1)$

Finally, for our experiments, we made data streams with five contexts, each holding $20,000$ instances, for $100,000$ instances in total.

2.5 Evaluation Settings

We study the evaluation procedures, the classification measures, the drift detection measures, and the resource consumption measures in the following subsections.

2.5.1 Evaluation Procedures

In experimental settings, evaluation procedures determine which instances are used for training and testing a learning algorithm (Bifet et al., 2011).

As to the batch learning scenario, the size of data plays a key role in adopting an evaluation procedure. Learning algorithms are often evaluated by a (ten-fold) *cross-validation* procedure when datasets are small, e.g., with less than a thousand instances, since it makes the maximum use of data by multiple sub-evaluations. Fig. 2.11 illustrates how a four-fold cross-validation procedure works. It partitions the data into four subsets or *folds* of equal size. Subsequently, there are four iterations for both training and testing. The fold that is used for testing is referred to as *validation set*. It is shown by a rectangle in the figure. In each iteration, a model is trained, and its accuracy is also calculated using the validation fold. The final accuracy is the average of accuracies from all iterations. The K-fold cross-validation procedure has the potential of being very slow against huge datasets. Therefore, it is necessary to reduce the numbers of repetitions or folds to complete the task within a reasonable time.

As an alternative, the *holdout* procedure may be considered as a solution for having far fewer computation efforts. This procedure randomly selects, for example, one-third of

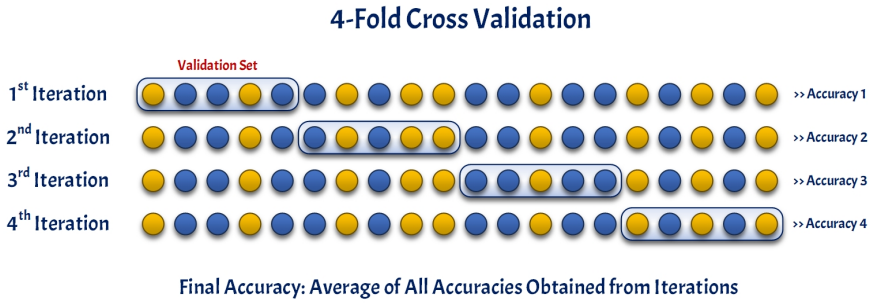


Fig. 2.11. 4-Fold Cross Validation Example

instances as a test set, also called as a holdout set, and use the rest of instances as a training set. The model, built using the training set, is tested on the test set. *Bootstrapping* is another approach that samples the training instances uniformly with replacement from a dataset (Han et al., 2011). In sampling with replacement, once an example is selected for training, it yet remains a candidate for selection, and it may be selected again. Finally, examples that were not selected for training are considered as candidates for testing.

Although cross-validation is typically used for the batch mode, it is impractical for data streams because of its computational cost (in terms of either memory or runtime). The holdout and bootstrapping procedures need the whole data to be completely available for partitioning, which contradicts the nature of a data stream. Moreover, in data stream mining, one of the substantial concern is how to capture accuracy of classification models over time without violating the four requirements discussed in Section 2.2.3.

To address the preceding challenges, the *Incremental Holdout* and *Predictive Sequential* (*Prequential*) procedures are developed for the evaluation of learning algorithms against data streams (Bifet and Kirkby, 2009; Gama et al., 2014). We explain and compare them as follows.

2.5.1.1 Incremental Holdout

Traditional holdout procedure assumes all the data are available to be partitioned into training and test sets, which contradicts with the nature of data streams where instances arrive one-by-one over time; however, the holdout idea is applied in a different way for data stream mining. For example, a classification model is tested *periodically* after training every one thousand or one million instances.

A possible source of holdout examples is newly arrived instances, which have not been used in the training phase yet. That is, the procedure holds a batch of new instances out as a test set. As the stream is processed, the test set is used to evaluate the model in particular periods. Once the testing phase ends, the test set may be used as an extra data source for training. This procedure continues until the learning task comes to a halt.

In stationary environments, a single holdout set may be used for testing. That is, it is not required to create holdout sets periodically from newly arrived instances, since the distribution of data does not change. On the other hand, in evolving environments, we need to update the holdout set once a while.

Holdout set needs to be sufficiently large enough to represents the complexity of the target concept. It is worthwhile to mention that testing the predictive models very often would slow the learning process significantly. It becomes even more severe when the holdout set is very large. Also, if the holdout set is not sufficiently large enough, the outcome of testing may not be reliable. In that case, we may experience fluctuations in the classification accuracy (Bifet et al., 2011). Therefore, the size of the holdout set is a critical parameter.

2.5.1.2 Predictive Sequential

The predictive sequential (i.e., *prequential*) procedure is the often used evaluation strategy in the literature. The prequential procedure first uses a new instance to test the predictive model, and then considers it for training. The classification accuracy is incrementally updated while instances are one-by-one processed. This procedure is also known as interleaved test-and-train (Bifet and Kirkby, 2009).

The prequential procedure assures that a classification model is always being tested on unseen instances. In contrast to the holdout procedure, it makes the maximum use of data, meaning all instances are used for both testing and training. It also ensures a smooth plot of accuracy over time, as the impact of each example becomes increasingly less significant to the overall average (Bifet et al., 2011). In this procedure, the accuracy of a learning algorithm is punished for early mistakes. This effect, however, diminishes after training thousands of instances over time.

2.5.1.3 Comparison

Fig. 2.12 illustratively compares the classification error-rates of the holdout and prequential procedures while instances are processed over time. As the red bullets indicate, the holdout procedure tests the model periodically. For that, the error-rate curve does not look smooth during the early stages. On the contrary, the prequential procedure has a very smooth curve since it tests and trains instances one-by-one sequentially.

One may also notice that the error-rate of the prequential procedure is more pessimistic because the classification model was rather inaccurate during the early stages. Recall that, in the prequential procedure, the average error-rate may be punished by wrong predictions in early stages. On the other hand, because of training a batch of early instances, holdout procedure shows a lower average error-rate.

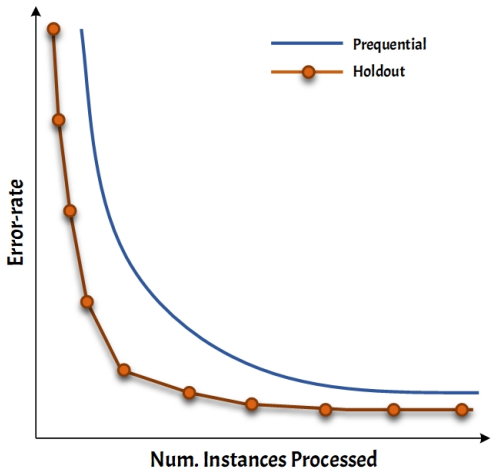


Fig. 2.12. Holdout vs. Prequential

Gama et al. (2009) demonstrated that the holdout error and the prequential error both converge to Optimal Bayes Error as the number of instances increases. That is, the error-rates of the holdout and prequential procedures will be equal after processing infinite instances, i.e., $\lim_{n\to\infty} Er_{holdout}(n) = \lim_{n\to\infty} Er_{prequential}(n)$, where n is the number of instances.

2.5.2 Classification Measures

Accuracy[9], the most often used measure for the classification evaluation, is the fraction of instances that are *correctly* classified, and calculated as follows:

$$accuracy = \frac{TP + TN}{P + N} \tag{2.15}$$

where TP and TN represent the correctly classified positive instances and the correctly classified negative instances, respectively. P and N stand for all positive and all negative instances[10]. Error-rate may be considered as an alternative classification measure which is $1 - accuracy$. It is also calculated by:

$$error\text{-}rate = \frac{FP + FN}{P + N} \tag{2.16}$$

where FP and FN represent the negative instances incorrectly classified as positive and the positive instances incorrectly classified as negative, respectively.

9 We only use accuracy, or error-rate, in this thesis because our data streams are balanced. In the case of streams that are imbalanced, one may employ metrics such as Cohen's Kappa κ, or $\kappa+$, statistics (Bifet and Frank, 2010) as well as the F-measure or G-mean measures (Han et al., 2011).

10 Please note that the instances with the class of interest are called as *positive instances*, and the rest of instances as *negative instances*.

2.5.3 Drift Detection Measures

Drift detection methods, as the core components of adaptive learners, should be evaluated regarding the *correctness measures* and the *delay of detection*. We present a new evaluation approach for concept drift detection concerning the correctness measures in Section 2.5.3.1. The true positive (TP), false positive (FP), and false negative (FN) rates of drift detection indicate how correctly a drift detection operates against an evolving data stream. Finally, we discuss why shorter detection delay is preferred in Section 2.5.3.2.

2.5.3.1 Correctness Measures

The true positive (TP), false positive (FP), and false negative (FN) measures may be used to evaluate the performance of drift detectors. Intuitively, a drift detector with the highest true positive, the lowest false positive and the lowest false negative rates is preferred. Huang et al. (2015) and Bifet and Gavalda (2007) used three types of tests to measure the rates of true positive, false positive, and false negative for a drift detector. For example, to measure the false positive rate, they generated a stream of bits from a stationary (Bernoulli) distribution. If a detector alarms for drifts against the stream, for each alarm, one false positive is counted. Thus, one may use three different tests individually to count the true positive, false positive, and false negative numbers. So, having an approach able to count them in one test, for any stream generated by any probability distribution, is preferred. In (Pesaranghader and Viktor, 2016), we introduced an approach to count the aforementioned measures by defining the *acceptable delay length* notion. The acceptable delay length Δ is a threshold to determine how far a detected drift could be from the true location of drift to be considered as true positive. Considering the acceptable delay length Δ, we explain the true positive, false positive and false negative measures as follows:

- *True Positive* (TP): A drift detector correctly detects a drift occurred at time t if it alarms for that at anytime in $[t - \Delta, t + \Delta]$. We call this range the *acceptable detection interval* of true positive. The true positive rate is defined as the proportion of (actual) drift points that are correctly identified as concept drift.
- *False Positive* (FP): A drift detector falsely detects a drift if the alarm falls outside of an acceptable detection interval. The false positive rate is defined as the proportion of non-drift points that are incorrectly identified as concept drift.
- *False Negative* (FN): A drift detector falsely overlooks a true drift occurred at time t if it does not alarm for the drift in $[t - \Delta, t + \Delta]$. The false negative rate is defined as the proportion of (actual) drift points that are incorrectly ignored as non-concept drift.

The reader should note the acceptable delay length, i.e., Δ, may be tuned empirically or based on prior knowledge regarding the domain. Furthermore, for evaluation of reactive concept drift detectors, the acceptable detection interval is $[t, t + \Delta]$.

Fig. 2.13 is a toy example which illustrates how the true positive, false positive and false negative numbers are counted. The upper stream shows the real locations of drifts, i.e., the squares with a D inside, and the lower stream shows the result of detection at each location. The squares with a T inside represent the drifts detected correctly (true positive), the squares with an F inside represent the points incorrectly considered as concept drift (false positive), and the squares with an N inside indicate undiscovered drifts (false negative). The drift detector signals for concept drift within the first acceptable detection interval and so the true positive number increases. Then, it incorrectly alarms for concept drift, and the false positive number increases. Since the detector does not alarm for concept drift within the second acceptable detection interval, the false negative number increases. The figure shows that the detector incorrectly triggers for concept drift at the very end of the stream.

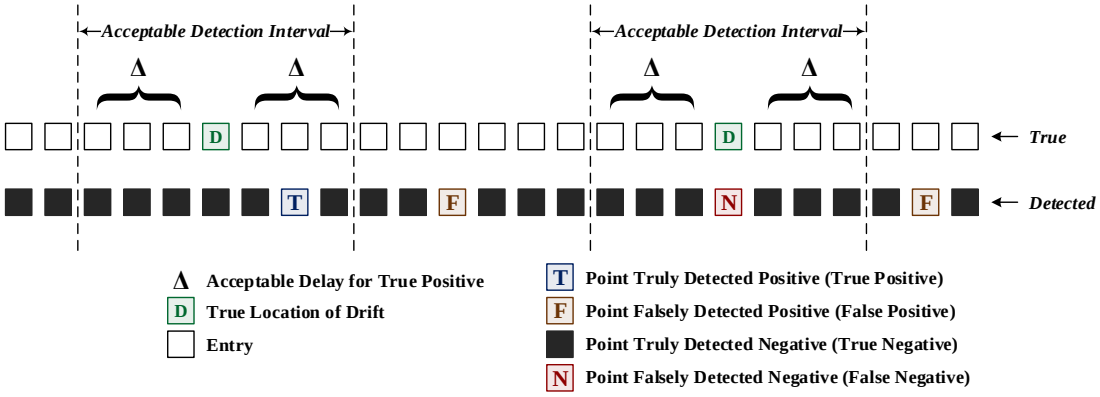


Fig. 2.13. Illustration of Counting TP, FP and FN

One may ask '*why do we need to evaluate false positive and false negative rates?*' For data stream mining, the resources, e.g., CPU, may be kept busy more if a drift detector incorrectly alarms for concept drift repeatedly (Žliobaitė et al., 2015). Further, the error-rate of classification would be high if a drift detector does not accurately detect the location of drifts. In other words, the accuracy of classification typically reduces when the false negative number increases (Huang et al., 2015; Žliobaitė et al., 2015; Bifet and Gavalda, 2007). Therefore, false positive and false negative are essential measures for evaluating concept drift detectors.

2.5.3.2 Drift Detection Delay

The delay of drift detection is also considered as a key performance measure for evaluating drift detection methods, because:

- The correct approximation of the drift location is crucial for further investigation to study what cause was behind that concept drift. Alarming for concept drift with the minimum delay is very critical in many applications, e.g., Business Intelligence (BI) and medical diagnostics. For example, a patient's heartbeat is monitored, and any unusual change must be reported to avoid any lethal risk. Hence, a prompt detection would help a doctor to understand better why such a change has happened, and he could have a careful judgment regarding the condition. The reader may recall that we discussed this case with respect to informed adaptation in Section 2.3.2.
- A shorter detection delay allows the learning algorithms to use more instances from the new distribution for training the classification models. Considering the probably approximately correct (PAC) learning model (Mitchell, 1997), a higher classification accuracy may be achieved by using more instances from a distribution.

2.5.4 Resource Consumption Measures

Recall that (adaptive) online algorithms must perform the task of learning in (near) real-time (Requirement 3) by using a *limited* amount of memory (Requirement 2). Otherwise, if the algorithms violate these requirements, they would not be considered *efficient*. Hence, we have to evaluate (adaptive) algorithms in terms of memory usage and execution runtime (of the learner and the drift detector, together).

Finally, it is worthwhile to mention that qualitative measures, including *interpretability*, *usability*, *transferability*, *robustness*, and *scalability* may also be considered as additional measures to assess learning algorithms (Han et al., 2011).

2.6 Applications

Žliobaitė (2010) conducted an extensive survey on real-world applications where concept drift is a critical challenge in both *supervised* and *unsupervised* learning tasks. Žliobaitė (2010) categorized the applications into four groups of:

- **Monitoring and Control** – Monitoring and control often employ supervised and unsupervised learning techniques to detect abnormal and adversary behaviours on the web, computer networks, telecommunications, and financial transactions.

- **Personal Assistance and Information Management** – Personal assistance and information management applications include recommendation systems, categorization and organization of textual information, and customer profiling for marketing.
- **Decision Making** – Decision making includes finance and bio-medicine applications. The *ground truth* is usually delayed, i.e., the true answer whether the decision was correct becomes available only after a particular time.
- **Artificial Intelligence** – Artificial Intelligence (AI) covers a broad spectrum of systems able to interact with dynamic environments to accomplish a task. The AI algorithms are used in robots, mobile vehicles, and smart houses.

2.6.1 Monitoring and Control

Monitoring and control applications fall into two groups of *monitoring against adversary actions*, and *monitoring for management purposes*.

2.6.1.1 Monitoring against Adversary Actions

Computer Security – Intrusion detection is one of the typical monitoring problems, where unwanted access to computer systems must be halted. Adversary actions are the primary source of concept drift in the intrusion detection systems. The problems and directions for intrusion detection are discussed by Patcha and Park (2007). Masud et al. (2009) proposed ensemble techniques to detect adversary actions. Artificial immune systems are widely considered for intrusion detection (Kim et al., 2007).

Telecommunications – Adversary behaviours also happen in the telecommunications industry, in the form of intrusion or fraud. The mobile masquerade detection problem, from the research perspective, is closely related to the intrusion detection (Mazhelis and Puuronen, 2007) where the goal is to prevent adversaries from unauthorized access to private data. The sources of concept drift are adversary behaviour trying to overcome the control as well as changing behaviour of the legitimate users.

Finance – Financial organizations may apply data mining techniques to monitor streams of transactions (e.g., credit cards, and online banking) to alert for possible frauds. Supervised and unsupervised learning techniques are used for the detection of fraudulent transactions (Bolton and Hand, 2002). The main challenges may be introduced by incorrect data labelling, misinterpretation of legitimate transactions, and very high imbalanced classes (i.e., few frauds vs. legal actions), and users' behavioural changes.

2.6.1.2 Monitoring for Management

Monitoring for management usually uses streaming data from sensors. It is characterized by high volumes of data as well as real-time decision making.

Transportation – Traffic management systems use data mining to determine traffic states, e.g., car density in a particular area or accidents (Crespo and Weber, 2005). Transportation systems are dynamic because of different moving patterns. The traffic patterns are changing seasonally as well as permanently, thus the systems have to be able to handle concept drifts.

Positioning – Concept drift may be experienced in remote sensing of fixed geographic locations. Interactive road tracking system assists cartographers to annotate road segments in aerial photographs (Zhou et al., 2008). In that system, change detection comes into play by recognizing different roads over time. In place recognition (Luo et al., 2007) or activity recognition (Liao et al., 2007), the dynamic nature of environments could be the source of concept drift.

Industrial monitoring – In industrial monitoring applications, e.g., production or service monitoring, and different habits of users, or changes in their behaviours, could be considered as concept drifts.

2.6.2 Personal Assistance and Information Management

These applications organize and/or personalize the flow of information, and they are categorized into *personal assistance*, *customer profiling*, and *information management*.

2.6.2.1 Personal Assistance

Personal assistance applications deal with the user modelling task aiming to personalize the flow of a user's information, which is referred to as *information filtering*. User modelling applications are document categorization (Mourao et al., 2008), web personalization (De Bra et al., 2003), and spam filtering (Fdez-Riverola et al., 2007) with respect to the current interests of a user. Changes in the user's interests may also cause concept drift.

2.6.2.2 Customer Profiling

The goal of customer profiling is to segment the customers according to their interests; while the source of concept drift may be shifts in their interests and preferences. Marketing, social network analysis, and recommendation systems are applications of customer profiling. Adaptive learning solutions are used in customer segmentation (Crespo and Weber, 2005), social networks analysis (Lathia et al., 2008), shopping basket analysis (Rozsypal and Kubat, 2005), and movie recommendation (Koren, 2010).

2.6.2.3 Information Management

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

Document Organization – The task of document organization is to extract meaningful structures from document streams, e.g., news or e-mails, and associate them with different topics. Since the topics of documents and even related vocabulary may change in time, temporal ordering is necessary. The Latent Dirichlet Allocation (LDA) model (Blei et al., 2003) for probabilistic topic modelling was recently equipped with a time dimension (Blei and Lafferty, 2006; Wang et al., 2012). Blei and Lafferty (2006) analyzed the dynamics of topics for articles in the Science magazine from 1881 to 1999. The emergence, peak, and decline of topics were shown, and the topic vocabulary representation was built. Intuitively, this is similar to including the time feature into the original observation.

Economics – Concept drift is common in making macroeconomic forecasts (Giacomini and Rossi, 2009), and predicting the phases of a business cycle (Klinkenberg, 2005). The data are drifting mainly due to a large number of influencing factors, which are not feasible to be taken into prediction models.

2.6.3 Decision Making

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

2.6.3.1 Finance

Bankruptcy prediction or credit scoring are typically assumed to be a stationary problem (Kumar and Ravi, 2007). However, in those problems, the concept drift phenomenon is caused by latent factors, e.g., social demands and movements, which are not considered while a decision model is trained (Harries et al., 1998). As an early work, Sung et al. (1999) applied decision models for bankruptcy prediction under different economic conditions.

2.6.3.2 Biomedicine

Biomedical systems are subject to concept drift for the adaptive nature of microorganisms (Tsymbal et al., 2008). The effect of antibiotics on a patient often diminishes over time since microorganisms mutate and develop resistance to antibiotics. If a patient is treated with an antibiotic when it is not necessary, resistance might develop, and antibiotics might no longer help when they are needed. The changes in disease progression can be triggered by changes in a drug being used (Black and Hickey, 2004). Adaptive learning may be used to discover antimicrobial or antibiotic resistance in nosocomial infections, i.e., the infections which result from the treatment, in hospitals (Jermaine, 2008). The resistance changes over time.

2.6.4 AI and Robotics

In AI applications, automata interact with the environment to adopt the best actions for the next moves. The automata should be able to adapt their knowledge over time as they typically act in dynamic environments. Recall that the problem of concept drift is also known as the *dynamic environment* in the AI community.

2.6.4.1 Mobile systems and robotics

Ubiquitous Knowledge Discovery (UKD) deals with the distributed and mobile systems, operating in a complex, dynamic, and unstable environment. Gomes et al. (2013) have developed the NextLocation framework for predicting the next location of a mobile user. Adaptivity to changing environment has been addressed in robotics (Procopio et al., 2009), e.g., designing a robot-player for soccer (Lattner et al., 2005).

2.6.4.2 Intelligent systems

Smart technologies should learn from streams of data in real-time and should adapt their models based on the current situation. The smart homes must be adaptive to surrounding environments and users' needs (Rashidi and Cook, 2009; Anguita, 2001).

2.6.4.3 Virtual reality

Virtual reality should be empowered with mechanisms to deal with concept drift. In game design, adversary actions of players, who are cheating, might be one of the concept drift sources (Charles et al., 2005). In games simulating fights, the strategies and skills differ across different users (Harries et al., 1998).

2.6.5 Discussion

Žliobaitė (2010) defined five dimensions to compare the applications of adaptive learning. The five dimensions are *decision speed*, *accuracy*, *the cost of mistakes*, *the nature of true labels*, and *adversary activities*. Table 2.7 represents how Žliobaitė (2010) characterized applications discussed above with those five dimensions. The decision speed implies how fast the decision or the output should be made, and it is critical in monitoring and AI applications. For example, in fraud detection, a decision needs to be taken quickly to stop a possible crime. The accuracy of prediction should be high for decision making applications, e.g., medical diagnosis and cancer treatments. Otherwise, the cost of misclassification may be high, e.g., a person's life would be at risk. Personal assistance applications are examples

of *soft-label*[11] problems. Adversary activities, e.g., fraudulent and intrusive actions, are expected in monitoring and control applications more than other applications. Please note that Žliobaitė (2010) made a global comparison among characteristics of applications. Nonetheless, for example, one might expect high decision speed for a personal assistance application.

Table 2.7. Characteristics of Applications
(Žliobaitė, 2010)

Application	Decision Speed	Accuracy	Cost of Mistakes	Labels	Adversary
Monitoring and Control	High	Approximate	Medium	Hard	Active
Personal Assistance	Medium	Approximate	Low	Soft	Low
Decision Making	Low	Precise	High	Delayed	Possible
AI and Robotics	High	Precise	High	Hard	Low

2.7 Summary

The fundamental concepts of data stream mining were discussed in this chapter. We showed the main differences between the batch setting and the data stream setting in Section 2.1. A batch is a finite and static dataset, whereas, a data stream is an infinite sequence of fast 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 learning algorithms. Online classification algorithms have to *process an example at a time, and only once*, *use a limited amount of memory*, *use a limited amount of time*, and *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 (Duda

11 The labels could be either *hard* or *soft*. In a hard-label classification task, an instance belongs only to *one* class. On the other hand, in a soft-label problem, an instance may belong to more than one class with different probabilities.

et al., 2012) to define the *real concept drift* and *virtual concept drift* notions in Section 2.3.1.1. The underlying data distribution, as well as the target concept, are changed when real concept drift happens. 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 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 the adaptive learners must fulfill three requirements of the *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 the definitions of the synthetic, the real-world data, and the semi-real-world 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 the *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 Fast Hoeffding Drift Detection Method (FHDDM), the Stacking Fast Hoeffding Drift Detection Method (FHDDMS), and the McDiarmid Drift Detection Methods (MDDMs) for the accurate detection of concept drift, with a shorter delay, compared to the state-of-the-art.

Part III

Contributions

Chapter 3

Fast Hoeffding and McDiarmid Drift Detection Methods

This chapter represents three of our research papers, which are:

- *"Fast Hoeffding Drift Detection Method for Evolving Data Streams"*, presented at European Conference on Machine Learning and Principles of Knowledge Discovery and Databases (ECML-PKDD 2016) and published in the proceedings (Pesaranghader and Viktor, 2016). This paper introduces the (first generation of) Fast Hoeffding Drift Detection Method (FHDDM) for detecting concept drift with a shorter delay, and lower false positive as well as false negative rates compared to the start-of-the-art methods. We represent FHDDM in Section 3.2.

- *"Reservoir of Diverse Adaptive Learners and Stacking Fast Hoeffding Drift Detection Methods for Evolving Data Streams"*, published in the Machine Learning Journal, Springer (Pesaranghader et al., 2018a). This paper is a twofold work that presents a framework for adaptive multi-strategy learning and introduces the stacking windows-based drift detection methods. We explain the second fold, i.e., the Stacking Fast Hoeffding Drift Detection Method (FHDDMS) family in this chapter, in Section 3.3[1].

- *"McDiarmid Drift Detection Methods for Evolving Data Streams"*, demonstrated at International Joint Conference on Neural Networks (IJCNN 2018) and published in the proceedings (Pesaranghader et al., 2018b). In this research work, we considered the McDiarmid inequality (McDiarmid, 1989) to bound the difference in between the maximum weighted mean and the current weighted mean of elements in the sliding window. We considered the weighted mean, in contrast to the (Pythagorean) means in FHDDM and FHDDMS, for *faster* drift detection. We describe and evaluate the MDDM variants in Section 3.4.

1 Please note that we describe the first fold in Chapter 4.

3.1 Problem Statement

As discussed in Section 2.3.1, the accuracy of classification may decrease due to the *concept drift* phenomenon. To this end, adaptive learning algorithms utilize drift detection methods to find concept drift and retrain their decision models accordingly (Gama et al., 2014).

Recall that drift detection methods should have the least false positive and false negative rates (Requirement 5). A drift detector with a high false positive number (or rate) keeps resources busier because of sending requests frequently for retraining the classification model (Huang et al., 2015; Žliobaitė et al., 2015). On the other hand, a drift detector with a high false negative rate causes decay in the accuracy of classification since it has not detected the drift points. These types of oversights are costly and must be avoided in many applications, e.g., in the fraud detection and emergency response settings.

Furthermore, the drift detectors should alarm for concept drift with the shortest delay (Requirement 6). The correct approximation of a drift point, i.e., identifying the change as soon as possible, is necessary because it helps not only to benefit from more examples, of course from the new distribution, for learning but also to investigate better and realize why/how the drift has happened. Such insights are crucial in Business Intelligence (BI) and medical applications.

Therefore, having drift detection methods able to detect the drift points with fewer false positives, fewer false negatives, and shorter detection delays is essential to improve the quality of adaptive learning. Recall that we formalized this problem and its corresponding research objective in Section 1.2.1 as follows:

> **Research Problem 1** is that *the quality of adaptive learning may diminish due to the concept drift phenomenon in evolving data.* Therefore, the challenge is to alarm for concept drifts as soon as they happen, with low false positive and false negative rates, regardless of the incoming data distribution function, to keep the classification error-rate minimal.
>
> **Research Objective 1** is *to introduce novel concept drift detection methods which detect drift points more accurately compared to the state-of-the-art.* For that, we *empirically* compare our methods with the existing algorithms against synthetic and real-world data streams, by considering different performance measures, e.g., detection delay, false positive, and false negative.

To address this research problem, we introduce three kinds of (sliding window-based) drift detection methods, which utilize Hoeffding's and McDiarmid's inequalities (Hoeffding, 1963; McDiarmid, 1989). We represent the Fast Hoeffding Drift Detection Method (FHDDM), as the first variant of our methods, in Section 3.2. Then, we describe the Stacking Fast Hoeffding Drift Detection Method (FHDDMS) and its Additive version

(FHDDMS$_{\text{add}}$), as two extensions of FHDDM, in Section 3.3. Finally, the McDiarmid Drift Detection Methods (MDDMs) are introduced in Section 3.4. The reader should note that we separately assess the above-mentioned methods against the state-of-the-art in each section, to discuss the main idea behind the invention of each. We experimentally show our methods detect the drift points with fewer false positives, fewer false negatives, and shorter detection delays compared to the existing algorithms.

3.2 Fast Hoeffding Drift Detection Method

The Fast Hoeffding Drift Detection Method (FHDDM) is a *window-based* drift detection approach that operates as follows. The FHDDM algorithm slides a window with a size of n on the prediction results. That is, the algorithm inserts a 1 into the window if the prediction result is *correct*; otherwise, it inserts a 0. As inputs are processed, it calculates the mean of elements, i.e., μ^t, in the sliding window at time t, and also keeps the maximum mean observed so far, i.e., μ^m. Equation (3.1) shows if the value of μ^t is greater than the value of μ^m, the value of μ^m will be updated.

$$\text{if} \quad \mu^m < \mu^t \Rightarrow \mu^m = \mu^t \tag{3.1}$$

Recall that, on the basis of the probably approximately correct (PAC) learning model (Mitchell, 1997), the accuracy of classification would increase or stay steady as the number of instances increases; otherwise, the possibility of facing concept drift increases (Gama et al., 2004). Thus, the value of μ^m should increase or remain steady as we process instances. In other words, the possibility of facing concept drift increases if μ^m does not change and μ^t decreases over time. Finally, as in Equation (3.2), a significant difference between μ^m and μ^t indicates the occurrence of concept drift in the stream.

$$\Delta\mu = \mu^m - \mu^t \geq \varepsilon_d \Rightarrow \text{Drift} := \text{True} \tag{3.2}$$

We use the Hoeffding inequality to define the value of ε_d, Equation (3.4). Hoeffding's inequality has a very attractive property that it is independent of the probability distribution generating the data. (Hoeffding, 1963; Domingos and Hulten, 2000; Frías-Blanco et al., 2015). That is, it assigns an upper bound for the deviation between the mean of n random variables and its expected value.

Theorem I: Hoeffding's Inequality – Let $X_1, X_2, ..., X_n$ be n independent random variables, as $X_i \in [0, 1]$, then with probability at most δ, the difference between the empirical mean $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ and the true mean $E[\overline{X}]$ is at least ε_H, i.e., $Pr(|\overline{X} - E[\overline{X}]| \geq \varepsilon_H) \leq \delta$, where:

$$\varepsilon_H = \sqrt{\frac{1}{2n} \ln \frac{2}{\delta}} \tag{3.3}$$

Corollary I: FHDDM test[2] – In a streaming setting, assume μ^t is the mean of a sequence of n random entries, each entry is in $\{0, 1\}$, at time t, and μ^m is the maximum mean observed so far. Let $\Delta\mu = \mu^m - \mu^t \geq 0$ be the difference between the two means. Given the desired δ, i.e., the probability of error allowed, the Hoeffding inequality guarantees a drift has happened if $\Delta\mu \geq \varepsilon_d$, where:

$$\varepsilon_d = \sqrt{\frac{1}{2n} \ln \frac{1}{\delta}} \tag{3.4}$$

Figure 3.1 depicts an illustrative example of the FHDDM algorithm. In this example, n and δ are set to 10 and 0.2, respectively. Using Equation (3.4), the value of ε_d is equal to 0.28. In this example, a drift occurs right after the 12$^{\text{th}}$ instance. The values of μ^t and μ^m are null and zero until 10 elements are inserted into the window. We have seven 1s in the window after reading the first 10 elements, and so μ^t is equal to 0.7. The value of μ^m is set to 0.7 too. The 1$^{\text{st}}$ element is dropped out from the window before the 11$^{\text{th}}$ prediction status is inserted. Since the value of prediction status is 0, the value of μ^t decreases to 0.6. The value of μ^m stays the same because it is greater than the current μ^t. This progress is continued until the 18$^{\text{th}}$ element is inserted. At this moment, the difference between μ^m and μ^t becomes greater than the value of ε_d. So, the FHDDM algorithm alarms for concept drift at this point.

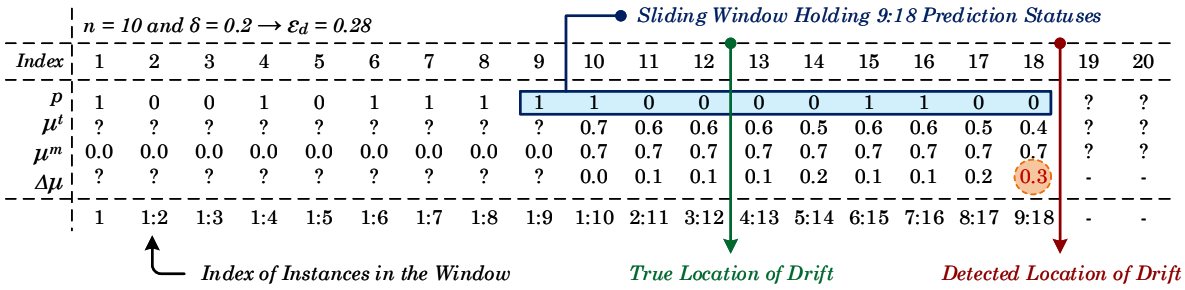


Fig. 3.1. FHDDM: Illustrative Example

We present the pseudocode of FHDDM in Algorithm 3.1. First, we need to instantiate an object from FHDDM and then call its DETECT function. The result of the prediction is sent to the DETECT function as an input to determine whether concept drift has occurred, in line 8. The oldest element is dropped out from the sliding window if it is full; then, a new element is pushed into it, as shown in lines 9 to 11. The algorithm returns *False* in the case of having not enough elements in the window, as depicted in lines 12 and 13. Next, the values of μ^t, μ^m, and $\Delta\mu$ are calculated or updated (lines 15 to 17). Once $\Delta\mu \geq \varepsilon_d$, it resets the parameters and alarms for concept drift by returning *True*.

2 Theoretical proofs are available in Section C.1.

Algorithm 3.1 Fast Hoeffding Drift Detection Method (FHDDM)

```
1: function INITIALIZE(windowSize, delta)
2:     (n, δ) ← (windowSize, delta)
3:     ε_d = sqrt((1/(2n)) ln(1/δ))
4:     RESET()

5: function RESET()
6:     w = []                                  ▷ Creating an empty sliding window.
7:     μ^m = 0

8: function DETECT(p)                          ▷ The p is 1 for correct predictions, 0 otherwise.
9:     if w.size() = n then
10:        w.tail.drop()                       ▷ Dropping an element from the tail.
11:    w.push(p)                               ▷ Pushing an element into the head.
12:    if w.size() < n then
13:        return False
14:    else
15:        μ^t = w.count(1)/w.size()           ▷ Calculate the current mean.
16:        if μ^m < μ^t then
17:            μ^m = μ^t
18:        Δμ = μ^m − μ^t
19:        if Δμ ≥ ε_d then
20:            RESET()                         ▷ Resetting parameters.
21:            return True                     ▷ Signaling for an alarm.
22:        else
23:            return False
```

3.2.1 Sensitivity of Parameters

In this section, we investigate the impact of the change in parameters δ and n on the value of ε_d. We also study the effects of varying these values on the detection delay, the false positive as well as false negative rates, the memory usage, and the runtime. To this end, we conducted a number of experiments with the values of n in $\{25, 100, 200, 300, 400, 500\}$ and δ in $\{0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001\}$.

All the results that are shown are against the SINE1 synthetic data stream, which is susceptible to abrupt drift. Recall that the classification is the classic $y = sin(x)$ function, and the classes are reversed when concept drift happens. The reader may refer to Section 2.4.1 for more information. Our explorative results are summarized in Table 3.1. Note that the runtimes reported are the average of execution runtimes over five contexts. Recall that the context refers to the interval between two consecutive concept drifts.

Table 3.1 (a) demonstrates that, as the value of n increases, the value of ε_d decreases. This implies that, since we have more observations, a more optimistic error bound may be used. For a constant n, there is an inverse relationship between δ and ε_d. That is, as the value of δ decreases the ε_d value increases (i.e., the bound becomes more conservative). Further, Table 3.1 (b) illustrates that, for a constant $\delta = 10^{-6}$, the detection delay increases

as we increase the value of n. Intuitively, memory usage and runtime also increase as the window size grows. Table 3.1 (c) lists the results for a constant $n = 100$. The table shows that, as we decrease the value of δ, the detection delay increases but the false positive rate decreases.

Table 3.1. FHDDM and Sensitivity of Parameters

(a) Values ε_d of FHDDM for Different n and δ

		δ					
		0.01	0.001	0.0001	0.00001	0.000001	0.0000001
n	25	0.30349	0.37169	0.42919	0.47985	0.52565	0.56777
	100	0.15174	0.18585	0.21460	0.23993	0.26283	0.28388
	200	0.10730	0.13141	0.15174	0.16965	0.18585	0.20074
	300	0.08761	0.10730	0.12390	0.13852	0.15174	0.16390
	400	0.07587	0.09292	0.10730	0.11996	0.13141	0.14194
	500	0.06786	0.08311	0.09597	0.10730	0.11754	0.12696

(b) Behavior of FHDDM for Different Values of n and $\delta = 10^{-6}$

		Delay	TP	FP	FN	Memory	Runtime	Accuracy
n	25	41.13	4.00	0.00	0.00	232.00	11.98	93.31
	100	49.08	4.00	0.00	0.00	528.00	17.84	93.29
	200	57.94	4.00	0.00	0.00	928.00	23.65	93.26
	300	64.06	4.00	0.00	0.00	1328.00	30.47	93.24
	400	70.03	4.00	0.00	0.00	1728.00	37.34	93.22
	500	74.89	4.00	0.00	0.00	2128.00	42.88	93.20

(c) Behavior of FHDDM for $n = 25$ and Different Values of δ

		Delay	TP	FP	FN	Memory	Runtime	Accuracy
δ	0.01	25.17	4.00	3.66	0.00	232.00	11.68	93.41
	0.001	30.56	4.00	0.06	0.00	232.00	11.50	93.34
	0.0001	33.19	4.00	0.01	0.00	232.00	11.62	93.33
	0.00001	35.93	4.00	0.00	0.00	232.00	11.40	93.32
	0.000001	41.13	4.00	0.00	0.00	232.00	11.34	93.31
	0.0000001	43.37	4.00	0.00	0.00	232.00	11.51	93.30

3.2.2 Experimental Evaluation

This section evaluates the FHDDM algorithm against the state-of-the-art methods[3]. We used Hoeffding Tree (HT) and Naive Bayes (NB) as the incremental learners[4]. As discussed in Section 2.2.5.6, we employed the Hoeffding Tree and Naive Bayes algorithms because they are often considered as base learners in the literature for showing high accuracy, e.g., in (Huang et al., 2014; Barros et al., 2017; Duong et al., 2018a,b). We ran both classifiers with each drift detector for 100 times. We then averaged the results of the detection delay, true positive, false positive, false negative, memory usage (in bytes), detection runtime (in millisecond) of drift detectors as well as the accuracies of classifiers.

3 We explained the state-of-the-art methods in Section 2.3.4.
4 We described Hoeffding Tree and Naive Bayes in Section 2.2.5.

We set the *acceptable delay length* Δ to 250 for SINE1 and MIXED and to 1000 for the CIRCLES and LED data streams. We considered a greater Δ for CIRCLES and LED because these data streams contain *gradual* concept drift. Preliminary experiments confirmed that a longer acceptable delay length should be used against gradual drift; otherwise, the false negative number would increase.

Parameter Settings – We ran FHDDM with two window sizes of 25 and 100. We observed that a shorter window may alarm for an abrupt drift point with a shorter delay, whereas a longer window may detect gradual drift points with less false negative rates. We provide more information regarding this observation in the discussion part of our experiments against abrupt and gradual concept drifts. Our preliminary inspections against benchmark data streams helped us to adjust the window size n and confidence interval δ resulting in shorter detection delay, fewer false positives, and fewer false negatives. Since the sliding windows are small, the confidence interval δ should be set to a small value to guarantee that the ε_d is big enough. Our initial experiments indicated that 10^{-6} is an appropriate choice for δ. Other drift detectors were run with the default parameters as recommended by their creators and set in MOA.

The experiments were evaluated *prequentially* which means that an instance is first tested and then trained (Gama et al., 2004; Bifet and Kirkby, 2009). The reader may find more information regarding *prequential learning* in Section 2.5.1.2. Finally, experiments were performed on an Intel Core i5 @ 2.8 GHz with 16 GB of RAM running Apple OS X Yosemite.

We organize our experiments and discussion as follows:

- Drift detection methods against *abrupt* change: We summarize the performance of FHDDM and the existing drift detection methods against the SINE1 and MIXED data streams, in which *abrupt* change is experienced, with Hoeffding Tree and Naive Bayes in Tables 3.2 and 3.3, respectively. We show FHDDM may detect abrupt concept drift faster with a shorter sliding window.
- Drift detection methods against *gradual* change: We contrast FHDDM with the state-of-the-art methods against the CIRCLES and LED data streams, which contain *gradual* concept drift, with Hoeffding Tree and Naive Bayes in Tables 3.4 and 3.5, respectively. We represent FHDDM may introduce no false negative against gradual concept drift when its sliding window is longer.

3.2.2.1 Experiments against Abrupt Concept Drift

As mentioned earlier, we summarized the results of experiments for Hoeffding Tree (HT) and Naive Bayes (NB) against *abrupt* concept drift in Tables 3.2 and 3.3, respectively. We discuss the results for each classifier and the paired drift detectors as follows.

Hoeffding Tree against SINE1 and MIXED – We show the results of the experiments with Hoeffding Tree for abrupt concept drift in Table 3.2. $\text{HDDM}_{\text{W-test}}$ and FHDDM_{25} led to the shortest detection delays whereas EDDM, PageHinkley, and SeqDrift2 had the longest detection delays. EDDM showed the longest detection delay for not alarming for concept drift in an acceptable delay length of 250. We experienced the lowest false positive numbers with FHDDM_{25} and CUSUM which suggest that they had higher precisions. On the contrary, EDDM, ADWIN, and SeqDrift2 falsely alarmed for concept drift frequently, and so, they had the highest false positive numbers. As to the false negative results, EDDM, PageHinkely, and DDM suffered from not alarming for concept drift in the acceptable delay length, and it led them to the highest false negative numbers. SeqDrift2, ADWIN, and RDDM were expensive in terms of memory usage while FHDDM_{25}, EDDM, and CUSUM had the fastest execution runtimes. Finally, we achieved the highest classification accuracies by FHDDM_{25} and $\text{HDDM}_{\text{W-test}}$ against SINE1, and by FHDDM_{25} and FHDDM_{100} against MIXED. The reader should notice that FHDDM_{25} had shorter detection delays compared to FHDDM_{100}. Indeed, FHDDM_{25}'s μ^t drops immediately once an abrupt change happens, since its (shorter) window keeps fewer entries compared to FHDDM_{100}'s.

Table 3.2. Hoeffding Tree and FHDDM against Data Streams with Abrupt Drift

(a) HT - SINE1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDM_{25}	**40.65 ± 3.15**	4.00	**0.10 ± 0.33**	0.00	232.00	**11.40 ± 3.50**	**87.07 ± 0.16**
FHDDM_{100}	48.09 ± 2.82	4.00	0.43 ± 0.78	0.00	528.00	16.89 ± 3.93	87.05 ± 0.15
CUSUM	86.89 ± 4.47	4.00	**0.24 ± 0.47**	0.00	128.00	15.60 ± 3.95	86.94 ± 0.15
PageHinkley	229.24 ± 13.20	2.30 ± 1.07	1.71 ± 1.08	1.70 ± 1.07	136.00	12.26 ± 3.75	86.46 ± 0.17
DDM	163.11 ± 22.73	3.36 ± 0.77	3.30 ± 2.20	0.64 ± 0.77	136.00	15.50 ± 3.46	86.06 ± 1.34
EDDM	243.83 ± 14.25	0.22 ± 0.44	33.90 ± 11.61	3.78 ± 0.44	136.00	**11.30 ± 3.33**	84.71 ± 0.55
RDDM	93.63 ± 7.57	4.00	4.72 ± 3.58	0.00	7224.00	17.90 ± 4.66	86.79 ± 0.18
ADWIN_{32}	63.84 ± 1.12	4.00	7.31 ± 3.18	0.00	> 2060	> 124.40	86.67 ± 0.21
SeqDrift2	200.00	4.00	4.83 ± 1.16	0.00	> 82270	40.12 ± 5.72	86.53 ± 0.15
$\text{HDDM}_{\text{A-test}}$	57.62 ± 11.81	4.00	0.71 ± 0.89	0.00	160.00	34.77 ± 5.82	87.01 ± 0.16
$\text{HDDM}_{\text{W-test}}$	**35.70 ± 2.95**	4.00	0.46 ± 0.68	0.00	160.00	36.55 ± 5.88	**87.07 ± 0.15**

(b) HT - MIXED Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDM_{25}	**40.55 ± 3.70**	4.00	**0.65 ± 0.94**	0.00	232.00	**11.85 ± 3.81**	**83.39 ± 0.10**
FHDDM_{100}	49.19 ± 8.38	3.98 ± 0.14	1.82 ± 1.46	0.02 ± 0.14	528.00	15.64 ± 4.16	**83.32 ± 0.13**
CUSUM	90.90 ± 6.13	4.00	**0.32 ± 0.58**	0.00	128.00	14.25 ± 3.66	83.27 ± 0.12
PageHinkley	229.91 ± 13.27	2.26 ± 0.98	1.74 ± 0.98	1.74 ± 0.98	136.00	12.75 ± 3.64	82.88 ± 0.11
DDM	195.73 ± 22.12	2.76 ± 1.01	2.91 ± 1.96	1.24 ± 1.01	136.00	15.81 ± 4.41	81.78 ± 2.06
EDDM	248.46 ± 7.69	0.05 ± 0.22	21.51 ± 7.70	3.95 ± 0.22	136.00	**11.82 ± 3.37**	80.65 ± 0.82
RDDM	106.68 ± 11.26	3.99 ± 0.10	3.49 ± 2.47	0.01 ± 0.10	7224.00	17.12 ± 4.08	83.16 ± 0.12
ADWIN_{32}	64.72 ± 2.79	4.00	4.84 ± 2.44	0.00	> 2065	> 124.75	83.25 ± 0.12
SeqDrift2	200.00	4.00	4.98 ± 1.20	0.00	> 81470	39.36 ± 5.74	82.91 ± 0.11
$\text{HDDM}_{\text{A-test}}$	69.42 ± 15.51	4.00	1.28 ± 1.09	0.00	160.00	35.64 ± 5.44	83.31 ± 0.11
$\text{HDDM}_{\text{W-test}}$	**35.56 ± 3.50**	4.00	3.23 ± 1.95	0.00	160.00	34.79 ± 5.55	83.27 ± 0.12

Naive Bayes against SINE1 and MIXED – We present the results of the experiments with Naive Bayes and drift detection methods against SINE1 and MIXED in Table 3.3. $FHDDM_{25}$ and $HDDM_{W\text{-test}}$ found concept drifts with the shortest delays, followed by $FHDDM_{100}$ and ADWIN. On the other hand, EDDM, SeqDrift2, DDM, and PageHinkley detected drift points with lengthy delays. The reader may notice that EDDM had the longest delays as well as the highest false negative numbers for not alarming for concept drift during the acceptable intervals of 250. Regarding the false positive measure, we obtained the best results with $FHDDM_{25}$ and $FHDDM_{100}$ for SINE1, and with $FHDDM_{25}$ and CUSUM for MIXED. DDM showed the second highest false negative numbers after EDDM. SeqDrif2, RDDM, and ADWIN were costly in terms of memory usage; and ADWIN had the slowest execution runtimes. Finally, we got the highest classification accuracies by $FHDDM_{25}$, $FHDDM_{100}$, and $HDDM_{W\text{-test}}$ for both data streams. Similar to the previous experiment, $FHDDM_{25}$ detected drift points faster than $FHDDM_{100}$ because its shorter window reflects the abrupt change quickly.

Table 3.3. Naive Bayes and FHDDM against Data Streams with Abrupt Drift

(a) NB - SINE1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	**40.48 ± 3.37**	4.00	**0.04 ± 0.20**	0.00	232.00	**12.11 ± 3.63**	**86.08 ± 0.25**
$FHDDM_{100}$	48.19 ± 5.54	3.99 ± 0.10	**0.13 ± 0.34**	0.01 ± 0.10	528.00	17.46 ± 4.58	86.06 ± 0.25
CUSUM	83.27 ± 6.96	3.99 ± 0.10	0.71 ± 0.86	0.01 ± 0.10	128.00	14.41 ± 3.57	85.96 ± 0.25
PageHinkley	175.07 ± 24.72	3.71 ± 0.50	0.35 ± 0.54	0.29 ± 0.50	136.00	**11.53 ± 2.99**	85.69 ± 0.27
DDM	179.18 ± 26.83	2.87 ± 0.84	3.09 ± 1.88	1.13 ± 0.84	136.00	15.83 ± 3.87	82.39 ± 4.32
EDDM	234.28 ± 22.22	0.57 ± 0.64	33.53 ± 11.50	3.43 ± 0.64	136.00	12.52 ± 4.04	83.44 ± 2.87
RDDM	89.72 ± 16.45	3.99 ± 0.10	3.93 ± 2.91	0.01 ± 0.10	7224.00	17.08 ± 4.35	85.98 ± 0.27
$ADWIN_{32}$	63.92 ± 0.80	4.00	3.86 ± 1.09	0.00	> 2070	> 126.30	85.93 ± 0.23
SeqDrift2	200.00	4.00	4.26 ± 0.58	0.00	> 83665	39.35 ± 6.06	85.59 ± 0.25
$HDDM_{A\text{-test}}$	88.03 ± 25.73	3.97 ± 0.17	0.35 ± 0.55	0.03 ± 0.17	160.00	35.68 ± 6.06	85.95 ± 0.25
$HDDM_{W\text{-test}}$	**35.52 ± 3.10**	4.00	0.41 ± 0.58	0.00	160.00	35.62 ± 6.25	**86.09 ± 0.25**

(b) NB - MIXED Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	**40.56 ± 3.72**	4.00	**0.25 ± 0.48**	0.00	232.00	**11.54 ± 3.57**	**83.38 ± 0.08**
$FHDDM_{100}$	48.77 ± 6.86	3.99 ± 0.10	0.58 ± 0.79	0.01 ± 0.10	528.00	15.61 ± 3.95	**83.36 ± 0.09**
CUSUM	88.23 ± 8.97	3.99 ± 0.10	**0.35 ± 0.54**	0.01 ± 0.10	128.00	14.49 ± 3.82	83.27 ± 0.08
PageHinkley	198.79 ± 18.72	3.56 ± 0.65	0.44 ± 0.65	0.44 ± 0.65	136.00	**11.62 ± 3.35**	82.97 ± 0.10
DDM	192.99 ± 23.82	2.78 ± 1.00	2.41 ± 1.44	1.22 ± 1.00	136.00	15.82 ± 4.17	80.28 ± 4.11
EDDM	247.47 ± 8.60	0.11 ± 0.31	20.22 ± 7.66	3.89 ± 0.31	136.00	11.79 ± 3.25	80.30 ± 2.32
RDDM	104.97 ± 12.06	3.99 ± 0.10	1.86 ± 1.65	0.01 ± 0.10	7224.00	18.11 ± 4.13	83.24 ± 0.09
$ADWIN_{32}$	64.48 ± 1.90	4.00	3.47 ± 1.42	0.00	> 2070	> 124.45	83.28 ± 0.08
SeqDrift2	200.00	4.00	4.39 ± 0.79	0.00	> 83520	38.91 ± 5.71	82.91 ± 0.08
$HDDM_{A\text{-test}}$	83.71 ± 19.46	3.96 ± 0.20	0.48 ± 0.64	0.04 ± 0.20	160.00	35.57 ± 5.59	83.28 ± 0.09
$HDDM_{W\text{-test}}$	**35.75 ± 3.94**	4.00	1.77 ± 1.39	0.00	160.00	34.92 ± 6.01	**83.36 ± 0.09**

3.2.2.2 Experiments against Gradual Concept Drift

Tables 3.4 and 3.5 summarize the experiments with Naive Bayes (NB) and Hoeffding Tree (HT), respectively, against CIRCLES and LED in which *gradual* concept drift happens at every $25,000$ instances, with a transition length ζ of 500. Recall that we set the acceptable detection delay length Δ to $1,000$ for gradual drift.

Hoeffding Tree against CIRCLES and LED – As Table 3.4 represents the results of the experiments with Hoeffding Tree, $FHDDM_{100}$ detected the gradual drift points with the shortest delays, i.e., 20 time units faster than $HDDM_{W\text{-test}}$ in average. On the contrary, EDDM and PageHinkely showed longer delays for not alarming during the acceptable delay length; consequently, they had higher false negative numbers. $FHDDM_{100}$ and CUSUM had the least false alarms for CIRCLES. For LED, CUSUM, FHDDMs, PageHinkley, and $HDDM_{W\text{-test}}$ had the fewest false positives, while EDDM, ADWIN, and SeqDrift2 caused false alarms frequently which suggests that their tests should become more conservative by altering the default values of their parameters. $FHDDM_{100}$, CUSUM, RDDM, $HDDM_{A\text{-test}}$, and $HDDM_{W\text{-test}}$ showed the lowest false negative numbers for both data streams; moreover, PageHinkley, DDM, and $FHDDM_{25}$ caused a small number of false negatives against LED. As to the runtime results, EDDM, $FHDDM_{25}$, and PageHinkley were faster than others. $FHDDM_{100}$ and $HDDM_{W\text{-test}}$ resulted in the highest classification accuracies for CIRCLES; and RDDM, HDDMs, and $FHDDM_{100}$ for LED. The reader may notice that ADWIN and SeqDrift2 led to lower accuracies against LED since they frequently issued false alarms, and subsequently, both caused incomplete learning. Finally, we observed that $FHDDM_{100}$ detected gradual drift points faster with lower false positive and false negative numbers than $FHDDM_{25}$. This observation confirms a longer window can keep more information from a gradual transition and its μ^t reflects this kind of change adequately.

Table 3.4. Hoeffding Tree and FHDDM against Data Streams with Gradual Drift

(a) HT - CIRCLES Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	243.97 ± 122.26	2.71 ± 0.45	0.37 ± 0.56	0.29 ± 0.45	232.00	**11.39 ± 3.03**	86.44 ± 0.26
$FHDDM_{100}$	**79.28 ± 20.64**	3.00	**0.17 ± 0.40**	**0.00**	528.00	16.25 ± 4.18	**86.58 ± 0.13**
CUSUM	220.07 ± 31.79	2.99 ± 0.10	**0.04 ± 0.20**	**0.01 ± 0.10**	128.00	14.95 ± 3.59	86.51 ± 0.13
PageHinkley	855.37 ± 56.27	1.79 ± 0.45	1.24 ± 0.47	1.21 ± 0.45	136.00	12.58 ± 3.49	85.96 ± 0.15
DDM	487.97 ± 82.24	2.78 ± 0.52	1.41 ± 1.24	0.22 ± 0.52	136.00	16.86 ± 4.80	86.21 ± 0.47
EDDM	987.61 ± 54.35	0.07 ± 0.26	24.61 ± 14.48	2.93 ± 0.26	136.00	**11.35 ± 3.13**	84.89 ± 0.29
RDDM	293.80 ± 38.52	2.98 ± 0.14	0.79 ± 1.25	**0.02 ± 0.14**	7224.00	18.89 ± 4.79	86.46 ± 0.16
$ADWIN_{32}$	236.48 ± 130.94	2.67 ± 0.47	9.74 ± 3.05	0.33 ± 0.47	> 2145	> 125.89	85.62 ± 0.19
SeqDrift2	202.67 ± 16.11	3.00	3.08 ± 0.90	**0.00**	> 104370	41.23 ± 6.30	86.47 ± 0.14
$HDDM_{A\text{-test}}$	111.96 ± 68.22	2.96 ± 0.20	0.65 ± 0.92	**0.04 ± 0.20**	160.00	35.02 ± 6.61	86.52 ± 0.20
$HDDM_{W\text{-test}}$	**94.03 ± 57.61**	2.98 ± 0.14	0.73 ± 0.87	**0.02 ± 0.14**	160.00	36.48 ± 5.98	**86.53 ± 0.18**

(b) HT - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	346.36 ± 97.35	2.94 ± 0.28	**0.06 ± 0.28**	**0.06 ± 0.28**	232.00	12.60 ± 3.86	89.53 ± 0.04
$FHDDM_{100}$	**220.40 ± 76.00**	2.97 ± 0.22	**0.03 ± 0.22**	**0.03 ± 0.22**	528.00	17.39 ± 4.52	**89.57 ± 0.04**
CUSUM	300.68 ± 50.30	3.00	**0.00**	**0.00**	128.00	12.65 ± 3.50	**89.56 ± 0.03**
PageHinkley	560.30 ± 79.43	2.95 ± 0.26	**0.04 ± 0.24**	**0.05 ± 0.26**	136.00	**10.19 ± 3.23**	89.35 ± 0.04
DDM	444.13 ± 79.82	2.97 ± 0.17	0.32 ± 0.58	**0.03 ± 0.17**	136.00	14.04 ± 3.66	89.47 ± 0.56
EDDM	954.97 ± 62.98	0.66 ± 0.71	5.97 ± 1.69	2.34 ± 0.71	136.00	**9.69 ± 3.14**	88.33 ± 0.50
RDDM	321.88 ± 50.94	2.98 ± 0.14	0.61 ± 0.96	**0.02 ± 0.14**	7224.00	16.54 ± 3.68	**89.63 ± 0.04**
$ADWIN_{32}$	521.47 ± 239.71	2.40 ± 0.77	474.95 ± 14.12	0.60 ± 0.77	> 1401	97.69 ± 9.50	72.25 ± 0.49
SeqDrift2	426.00 ± 173.31	2.78 ± 0.44	277.06 ± 47.48	0.22 ± 0.44	> 15265	39.36 ± 5.56	76.51 ± 2.28
$HDDM_{A\text{-test}}$	295.03 ± 85.29	2.98 ± 0.20	0.16 ± 0.44	**0.02 ± 0.20**	160.00	31.54 ± 5.82	**89.58 ± 0.05**
$HDDM_{W\text{-test}}$	**259.18 ± 87.25**	2.95 ± 0.26	**0.08 ± 0.31**	**0.05 ± 0.26**	160.00	32.60 ± 6.57	**89.56 ± 0.04**

Naive Bayes against Circles and LED – Table 3.5 lists the results of the experiments for Naive Bayes (NB) and the paired drift detectors against Circles and LED which witness gradual concept at every 25,000 instances. FHDDM$_{100}$ detected gradual concept drifts with the shortest delays for both data streams, and it is followed by ADWIN for Circles, and by HDDM$_{\text{W-test}}$ for LED. Similar to the previous experiments, we observed EDDM, PageHinkley, and DDM had the longest delays. FHDDMs, CUSUM, PageHinkley, and HDDMs had the fewest false alarms, whereas, EDDM and ADWIN had the highest false positive numbers for Circles (Table 3.4 (a)), and ADWIN and SeqDrift2 for LED (Table 3.4 (b)). This observation suggests EDDM, ADWIN, and SeqDrift2 should become more restrictive to increase their precision. FHDDM$_{100}$, CUSUM, RDDM, and HDDM$_{\text{A-test}}$ showed the minimum false negative numbers, while EDDM had the highest which confirms it is not an accurate method. FHDDM$_{25}$, EDDM, and PageHinkley were among the fastest methods in terms of execution runtime. We obtained the highest classification accuracies by FHDDM$_{100}$ and SeqDrift2 against the Circles data stream, and by RDDM, HDDM$_{\text{A-test}}$, FHDDM$_{100}$, and CUSUM against LED. The reader may again notice that Naive Bayes had the lowest accuracies with ADWIN and SeqDrift2; as they repeatedly and falsely alarmed for concept drift. Finally, FHDDM$_{100}$ outperformed FHDDM$_{25}$, concerning the detection delay, false positive, and false negative measures for keeping more information in its window against gradual concept drifts; which led to detect them accurately.

Table 3.5. Naive Bayes and FHDDM against Data Streams with Gradual Drifts

(a) NB - Circles Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDM$_{25}$	432.30 ± 88.98	2.16 ± 0.37	**0.40 ± 0.66**	0.84 ± 0.37	232.00	**10.83 ± 3.14**	83.96 ± 0.15
FHDDM$_{100}$	**166.13 ± 83.84**	2.96 ± 0.20	**0.43 ± 0.60**	**0.04 ± 0.20**	528.00	16.51 ± 4.09	**84.14 ± 0.13**
CUSUM	299.78 ± 52.29	3.00	**0.40 ± 0.62**	**0.00**	128.00	14.92 ± 4.30	84.08 ± 0.12
PageHinkley	677.32 ± 76.30	2.11 ± 0.55	0.93 ± 0.53	0.89 ± 0.55	136.00	12.35 ± 3.94	83.94 ± 0.13
DDM	703.59 ± 122.67	1.92 ± 0.72	2.33 ± 1.49	1.08 ± 0.72	136.00	15.87 ± 4.03	83.18 ± 1.61
EDDM	938.27 ± 106.60	0.35 ± 0.50	31.09 ± 18.14	2.65 ± 0.50	136.00	**11.77 ± 3.29**	83.12 ± 0.40
RDDM	406.50 ± 69.40	2.99 ± 0.10	2.15 ± 1.94	**0.01 ± 0.10**	7224.00	17.94 ± 4.35	84.05 ± 0.11
ADWIN$_{32}$	**222.61 ± 57.00**	2.99 ± 0.10	5.56 ± 2.57	**0.01 ± 0.10**	> 2157	> 128	84.12 ± 0.11
SeqDrift2	276.67 ± 91.10	2.92 ± 0.27	2.49 ± 0.97	0.08 ± 0.27	> 106125	38.55 ± 6.02	**84.13 ± 0.14**
HDDM$_{\text{A-test}}$	306.91 ± 107.78	2.91 ± 0.29	**0.49 ± 0.69**	0.09 ± 0.29	160.00	35.06 ± 6.18	84.09 ± 0.12
HDDM$_{\text{W-test}}$	242.43 ± 134.19	2.73 ± 0.44	1.59 ± 1.00	0.27 ± 0.44	160.00	34.88 ± 5.56	84.11 ± 0.13

(b) NB - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDM$_{25}$	347.53 ± 101.36	2.93 ± 0.32	**0.02 ± 0.14**	**0.07 ± 0.32**	232.00	12.80 ± 3.66	89.54 ± 0.05
FHDDM$_{100}$	**220.40 ± 76.00**	2.97 ± 0.22	**0.03 ± 0.22**	**0.03 ± 0.22**	528.00	17.52 ± 4.40	**89.57 ± 0.04**
CUSUM	300.61 ± 50.30	3.00	**0.00**	**0.00**	128.00	12.81 ± 3.51	**89.57 ± 0.03**
PageHinkley	559.27 ± 78.99	2.95 ± 0.26	**0.04 ± 0.24**	**0.05 ± 0.26**	136.00	**10.19 ± 2.99**	89.36 ± 0.04
DDM	446.23 ± 82.12	2.96 ± 0.20	0.33 ± 0.58	**0.04 ± 0.20**	136.00	14.64 ± 3.61	89.29 ± 1.15
EDDM	949.61 ± 68.94	0.70 ± 0.73	6.33 ± 1.96	2.30 ± 0.73	136.00	**10.05 ± 3.07**	88.32 ± 0.53
RDDM	321.80 ± 50.94	2.98 ± 0.14	0.61 ± 0.96	**0.02 ± 0.14**	7224.00	16.27 ± 3.65	**89.63 ± 0.04**
ADWIN$_{32}$	521.36 ± 238.56	2.36 ± 0.76	465.41 ± 12.53	0.64 ± 0.76	> 1406	94.45 ± 10.1	72.73 ± 0.44
SeqDrift2	445.33 ± 192.27	2.75 ± 0.46	278.82 ± 47.50	0.25 ± 0.46	> 14799	38.94 ± 6.05	76.54 ± 2.25
HDDM$_{\text{A-test}}$	295.85 ± 83.23	2.98 ± 0.20	0.17 ± 0.47	**0.02 ± 0.20**	160.00	31.83 ± 5.41	**89.58 ± 0.05**
HDDM$_{\text{W-test}}$	**259.17 ± 87.21**	2.95 ± 0.26	**0.05 ± 0.22**	**0.05 ± 0.26**	160.00	33.26 ± 5.24	**89.56 ± 0.03**

3.2.2.3 Discussion

We conducted experiments to compare FHDDM with the state-of-the-art methods against both *abrupt* and *gradual* concept drifts. Overall, we saw the FHDDM and HDDM variants detected drift points faster with lower false positive and false negative numbers suggesting they were more accurate for the task of drift detection. Hoeffding Tree and Naive Bayes paired with FHDDM and HDDM also led to the highest classification accuracies. On the contrary, EDDM and PageHinkley had lengthy detection delays and the highest false negative numbers for not alarming for concept drift during the acceptable delay intervals. Moreover, we experienced many false alarms with EDDM, ADWIN, and SeqDrift2 which may be cured by changing default parameters to make their tests more conservative. RDDM, recently introduced by Barros et al. (2017), had better performance compared to its predecessor DDM regarding the detection delay, the detection false negative number, and the classification accuracy, albeit for the cost of a higher false positive number.

We ran FHDDM with two different window sizes (i.e., 25 and 100) for the experiments. We make the following conclusions about the window size of FHDDM:

- **Shorter Window for Abrupt Drift:** FHDDM_{25} detected *abrupt* changes with shorter delays than FHDDM_{100} because FHDDM_{25}'s μ^t quickly drops once a sudden change happens as its (shorter) window keeps fewer elements compared to a longer window. On the contrary, a shorter window may not work promptly against gradual concept drift, and subsequently, it may result in a higher false negative number, as it does not hold enough entries for indicating a gradual transition.

- **Longer Window for Gradual Drift:** We observed that FHDDM_{100} alarmed for *gradual* drifts faster, with fewer false positives and fewer false negatives, compared to FHDDM_{25}. That is, a longer window can keep more information regarding a gradual transition, and so, its μ^t reflects this kind of change accurately. Nonetheless, FHDDM_{100} may lead to a longer detection delay against *abrupt* concept drift for keeping information from a previous context.

Our conclusion illustrates the size of the sliding window is a crucial parameter for the task of drift detection. Furthermore, in the real-world, abrupt and gradual concept drifts may appear together within the same stream. Consequently, sliding only either a long or a short window may not be a beneficial option. To address this challenge, we introduced the Stacking Fast Hoeffding Drift Detection Methods in (Pesaranghader et al., 2018a) where a long window and a short window are run together for concept drift detection. We devote the next section to describing our stacking methods and evaluating them against FHDDM.

3.3 Stacking Fast Hoeffding Drift Detection Methods

We introduced two new members of the Fast Hoeffding Drift Detection Method (FHDDM) family in (Pesaranghader et al., 2018a), the so-called the Stacking Fast Hoeffding Drift Detection Method (FHDDMS) and the Additive Stacking Fast Hoeffding Drift Detection Method ($\text{FHDDMS}_{\text{add}}$). They are explained in Section 3.3.1 and 3.3.2, respectively.

3.3.1 Stacking Fast Hoeffding Drift Detection Method

The Stacking Hoeffding Drift Detection Method (FHDDMS)[5], extends the idea of FHDDM by maintaining windows of different sizes. That is, a short window and a long window are superimposed, as shown in Fig. 3.2. The rationale behind this approach is to reduce the detection delay and false negative rate. Intuitively, a short window may react to abrupt drift faster, and a long window may lead to a lower false negative rate. Similar to FHDDM, the FHDDMS algorithm inserts a 1 into both the short and the long windows when the prediction result is *correct*, whereas a 0 is inserted otherwise. As Fig. 3.2 illustrates the sizes of the long and the short sliding windows, i.e., $|W_l|$, and $|W_s|$, are set to 20 and 5, respectively.

Fig. 3.2. Stacking Fast Hoeffding Drift Detection Method (FHDDMS)

FHDDMS calculates the means of the elements inside the long and the short sliding windows at time t, i.e., μ_l^t and μ_s^t, as the stream is processed. We define μ_l^m and μ_s^m as the maximum means observed so far for the long and the short window, respectively:

$$\begin{aligned} \mu_l^m < \mu_l^t \Rightarrow \mu_l^m = \mu_l^t \\ \mu_s^m < \mu_s^t \Rightarrow \mu_s^m = \mu_s^t \end{aligned} \tag{3.5}$$

Recall that the classification accuracy increases or remains constant as the number of instances increases; otherwise, the possibility of facing concept drift increases (Gama et al., 2004; Pesaranghader and Viktor, 2016). So, the values of both μ_l^m and μ_s^m must increase or remain constant as we process instances. Alternatively, probability of the concept drift occurrence increases if the values of μ_l^m and μ_s^m do not change and the values of μ_s^l and μ_s^t decrease over time. As shown in Equation (3.6), a significant difference between the current means and their maximums indicates the occurrence of a change in the stream.

5 We added the 'S' at the end of FHDDMS to have it behind FHDDM in an *alphabetical* order.

$$\begin{gathered}\Delta\mu_l = \mu_l^m - \mu_l^t \geq \varepsilon_l \Rightarrow \psi_l = True \\ \Delta\mu_s = \mu_s^m - \mu_s^t \geq \varepsilon_s \Rightarrow \psi_s = True \\ \text{if } (\psi_l = True) \text{ or } (\psi_s = True) \Rightarrow \text{alarm for a drift}\end{gathered} \tag{3.6}$$

Here ψ_l and ψ_s denote status of the concept drift detection as observed by the long and short windows, respectively. We use the previously introduced Hoeffding's inequality (Hoeffding, 1963) (i.e., Theorem I) and FHDDM test (i.e., Corollary I) to define the values of ε_l and ε_s as follows:

$$\varepsilon_l = \sqrt{\frac{1}{2n_l} \ln \frac{1}{\delta}}, \text{ and } \varepsilon_s = \sqrt{\frac{1}{2n_s} \ln \frac{1}{\delta}} \tag{3.7}$$

where n_l and n_s are the sizes of the long and the short windows, respectively.

The pseudocode for the FHDDMS algorithm is presented in Algorithm 3.2. In summary, the INITIALIZE function initializes the parameters for the stacking windows. The DETECT function analyzes the prediction results to determine if a concept drift has occurred (lines 19-21). A drift is either detected when $(\mu_l^m - \mu_l^t) \geq \varepsilon_l$ or when $(\mu_s^m - \mu_s^t) \geq \varepsilon_s$.

Algorithm 3.2 Stacking Fast Hoeffding Drift Detection Method (FHDDMS)

```
1: function INITIALIZE(|W_l|, |W_s|, delta)
2:     (n_l, n_s, δ) ← (|W_l|, |W_s|, delta)
3:     ε_l = sqrt((1/(2n_l)) ln(1/δ)), ε_s = sqrt((1/(2n_s)) ln(1/δ))
4:     RESET()

5: function RESET()
6:     w = []                          ▷ Creating an empty sliding window for stacking.
7:     μ_l^m, μ_s^m = 0

8: function DETECT(p)                  ▷ The p is 1 for correct predictions, 0 otherwise.
9:     if w is full then
10:        drop an element from tail
11:    insert p into w
12:    calculate μ_l^t and μ_s^t
13:    if μ_l^m < μ_l^t then
14:        μ_l^m = μ_l^t
15:    if μ_s^m < μ_s^t then
16:        μ_s^m = μ_s^t
17:    if (μ_l^m − μ_l^t) ≥ ε_l or (μ_s^m − μ_s^t) ≥ ε_s then
18:        RESET()                     ▷ Resetting parameters.
19:        return True                 ▷ Signaling for an alarm.
20:    return False
```

3.3.2 Additive FHDDMS (FHDDMS$_{\text{add}}$)

Additive FHDDMS, denoted as FHDDMS$_{\text{add}}$, substitutes binary entries with their summations. Fig. 3.3 shows the short and the long windows are characterized by the sum of their respective most recent prediction results. The short window holds a single summation of the 5 most recent bits, while the long window holds four summations for the 20 most recent bits seen so far. For each window, the maximum means observed so far, which are μ_s^m and μ_l^m, are updated as required. In this example, the values of μ_l^t and μ_s^t are 0.8 and 0.6, respectively. Similar to the FHDDMS algorithm, FHDDMS$_{\text{add}}$ alarms for concept drift if the difference between the current means and the maximum means exceed a certain threshold, as determined by Equation (3.7) based on Hoeffding's inequality.

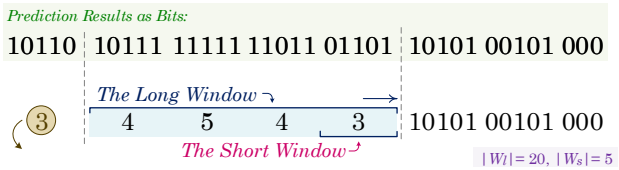


Fig. 3.3. Additive FHDDMS Approach (FHDDMS$_{\text{add}}$)

Fig. 3.4 shows a toy example which illustrates how FHDDMS and FHDDMS$_{\text{add}}$ operate. In this toy example, we have $n_s = 5$, $n_l = 20$, and $\delta = 0.002$. Using Equation (3.7), we have $\varepsilon_l = 0.394$, and $\varepsilon_s = 0.788$. Recall that FHDDMS considers the predictions bit-by-bit, whereas, FHDDMS$_{\text{add}}$ holds the summation of every 5 entries. Throughout the learning process, the values of μ_l, μ_l^m, $\Delta\mu_l$, μ_s, μ_s^m, $\Delta\mu_s$ are continuously updated, as indicated on the right-side of the figure. The reader should notice that both algorithms alarm for concept drift when $\Delta\mu_s$ exceeds 0.8, and it has a value greater than ε_s.

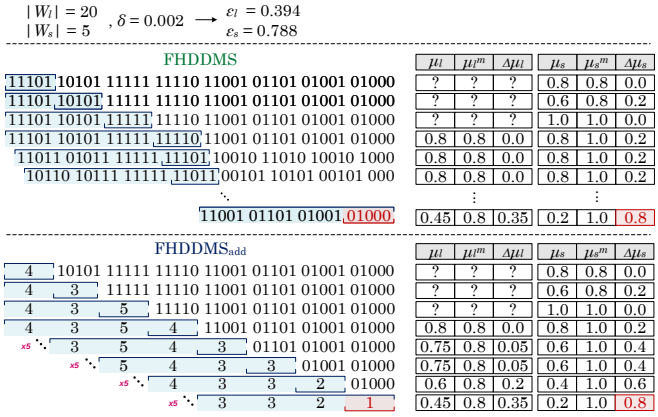


Fig. 3.4. FHDDMS and FHDDMS$_{\text{add}}$: Illustration Example

Intuitively, FHDDMS$_{\text{add}}$ should require less memory and exhibit a faster execution time, when compared to FHDDMS, because its data structure is more concise. Nevertheless, this may lead to a longer drift detection delay, since the algorithm must ensure that the short window has accumulated n_s new entries after dropping every element from the long window's tail. This is further confirmed by our experimental results in Section 3.3.4.

3.3.3 Sensitivity of Parameters

The sensitivity of parameters for FHDDMS and FHDDMS$_{\text{add}}$ is analogous to that of FHDDM which discussed in Section 3.2.1. It means the values of ε_s and ε_l increase by decreasing the value of δ; that leads to a more restrictive test. On the other hand, they decrease if the values of n_l and n_s increase; that implies the test is less conservative as more information is available. Further, increasing the value of δ may also lead to longer detection delay and fewer false positive number as it makes the test strict. A longer window size, i.e., either for n_s or n_l, may eventuate longer delay for waiting for more information, regarding a transition, to detect concept drift.

3.3.4 Experimental Evaluation

Recall that in Section 3.2.2, we evaluated FHDDM against the existing methods. We also observed that CUSUM, HDDM$_{\text{A-test}}$, and HDDM$_{\text{W-test}}$ had better performances compared to other methods, including but not limited to DDM, ADWIN, and SeqDrift2. Therefore, in this section, we only concentrate on the comparison of FHDDMS and FHDDMS$_{\text{add}}$ with FHDDM, CUSUM, and HDDMs.

Similar to the previous experiments, we used Hoeffding Tree (HT) and Naive Bayes (NB) as the incremental learners, and we ran them with each drift detector for 100 times. We then averaged the detection delays, true positives, false positives, false negatives, memory usage (in bytes), and detection runtimes (in milliseconds) of drift detectors as well as the accuracies of classifiers. We set the *acceptable delay length* Δ to 250 for SINE1 and MIXED and to $1,000$ for the CIRCLES and LED data streams. Recall that a greater Δ should be used for *gradual* drift to avoid a possible increase in the false negative number.

Parameter Settings – We ran the FHDDM algorithm with two different window sizes of 25 and 100, as in Section 3.2.2. Recall that a short window reacts to abrupt concept drift faster, whereas a long window detects the gradual drift points with fewer false negatives. Hence, we introduced FHDDMS and FHDDMS$_{\text{add}}$ to settle the choice of the window size; by stacking a short and a long windows on top of each. For a fair comparison with FHDDM$_{25}$ and FHDDM$_{100}$, FHDDMS and FHDDMS$_{\text{add}}$ similarly slide two windows with sizes of 25 and 100 over their entries. Similar to previous experiments,

we set δ to 10^{-6} to guarantee the values of ε_s and ε_l are big enough. CUSUM and HDDMs were run with the default parameters as set by their creators in MOA.

The experiments were evaluated *prequentially* on an Intel Core i5 @ 2.8 GHz with 16 GB of RAM running Apple OS X Yosemite. Finally, our experiments are organized as follows.

- FHDDMS variants against *abrupt* change: We present the results of our experiments for FHDDMS and $\text{FHDDMS}_{\text{add}}$ against the SINE1 and MIXED data streams, in which abrupt concept drift happens at every $20,000$ instances, in Tables 3.6 and 3.7. The purpose of the experiments is to show that FHDDMS detects *abrupt* drift faster than FHDDM_{100} while its false negative number is comparable to that of FHDDM_{25}. We further compare $\text{FHDDMS}_{\text{add}}$ against FHDDM, CUSUM, and HDDMs.
- FHDDMS variants against *gradual* change: Tables 3.8 and 3.9 show the performance of FHDDMS and $\text{FHDDMS}_{\text{add}}$ as well as FHDDM, CUSUM and HDDMs against *gradual* concept drift in CIRCLES and LED. We show that FHDDMS detects gradual concept drift faster than FHDDM_{100}, with fewer false negative numbers compared to FHDDM_{25}. We also discuss the comparable performance of $\text{FHDDMS}_{\text{add}}$ to that of existing methods.

3.3.4.1 Experiments against Abrupt Concept Drift

As mentioned earlier, Tables 3.6 and 3.7 compare the results of experiments for FHDDMS and $\text{FHDDMS}_{\text{add}}$, against other detection methods, with Hoeffding Tree (HT) and Naive Bayes (NB), respectively. We discuss the results for FHDDMS and $\text{FHDDMS}_{\text{add}}$ separately as follows.

FHDDMS – As shown in Tables 3.6 and 3.7, FHDDMS alarmed for concept drift faster than FHDDM_{100}, CUSUM, and $\text{HDDM}_{\text{A-test}}$. We also see the false negative numbers are comparable. FHDDMS represented higher false positive numbers than FHDDMs because its short and long windows may individually trigger an alarm for concept drift. Although FHDDMS and FHDDM_{100} consume the same level of memory, FHDDMS is slower than FHDDM_{100} regarding execution runtime due to more computations. Finally, FHDDMS, FHDDMs, and $\text{HDDM}_{\text{W-test}}$ resulted in higher classification accuracies; whereas, CUSUM led to the lowest ones.

$\text{FHDDMS}_{\text{add}}$ – $\text{FHDDMS}_{\text{add}}$ led to shorter detection delays compared to $\text{HDDM}_{\text{A-test}}$ and CUSUM; but lengthier than FHDDMs, FHDDMS, and $\text{HDDM}_{\text{W-test}}$. It alarmed for concept drift with longer delays compared FHDDMS and FHDDM_{25} because of waiting for its short window to be filled with every 25 instances. $\text{FHDDMS}_{\text{add}}$ represented lower false positive numbers than FHDDMS which suggest the summation of bits may decrease the chance of false alarm. Meanwhile, it showed similar false negative numbers in comparison with others. $\text{FHDDMS}_{\text{add}}$ had the second lowest memory usage, following CUSUM, for its summarization solution.

Table 3.6. Hoeffding Tree with FHDDMS Variants against Data Streams with Abrupt Drift

(a) HT - Sine1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**40.33 ± 3.18**	4.00	0.46 ± 0.83	0.00	528.00	29.24 ± 5.04	**87.06 ± 0.16**
$\text{FHDDMS}_{\text{add}}$	52.31 ± 4.21	4.00	**0.18 ± 0.43**	0.00	**152.00**	**13.77 ± 3.96**	87.04 ± 0.15
FHDDM_{25}	**40.65 ± 3.15**	4.00	**0.10 ± 0.33**	0.00	232.00	**11.40 ± 3.50**	**87.07 ± 0.16**
FHDDM_{100}	48.09 ± 2.82	4.00	0.43 ± 0.78	0.00	528.00	16.89 ± 3.93	87.05 ± 0.15
CUSUM	86.89 ± 4.47	4.00	0.24 ± 0.47	0.00	**128.00**	**15.60 ± 3.95**	86.94 ± 0.15
$\text{HDDM}_{\text{A-test}}$	57.62 ± 11.81	4.00	0.71 ± 0.89	0.00	160.00	34.77 ± 5.82	87.01 ± 0.16
$\text{HDDM}_{\text{W-test}}$	**35.70 ± 2.95**	4.00	0.46 ± 0.68	0.00	160.00	36.55 ± 5.88	**87.07 ± 0.15**

(b) HT - Mixed Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**40.37 ± 6.85**	3.99 ± 0.10	2.25 ± 1.73	0.01 ± 0.10	528.00	29.57 ± 5.42	83.31 ± 0.13
$\text{FHDDMS}_{\text{add}}$	52.25 ± 6.82	3.99 ± 0.10	**0.73 ± 0.90**	0.01 ± 0.10	**152.00**	14.80 ± 3.46	**83.37 ± 0.11**
FHDDM_{25}	**40.55 ± 3.70**	4.00	**0.65 ± 0.94**	0.00	232.00	**11.85 ± 3.81**	**83.39 ± 0.10**
FHDDM_{100}	49.19 ± 8.38	3.98 ± 0.14	1.82 ± 1.46	0.02 ± 0.14	528.00	15.64 ± 4.16	83.32 ± 0.13
CUSUM	90.90 ± 6.13	4.00	**0.32 ± 0.58**	0.00	**128.00**	14.25 ± 3.66	83.27 ± 0.12
$\text{HDDM}_{\text{A-test}}$	69.42 ± 15.51	4.00	1.28 ± 1.09	0.00	160.00	35.64 ± 5.44	83.31 ± 0.11
$\text{HDDM}_{\text{W-test}}$	**35.56 ± 3.50**	4.00	3.23 ± 1.95	0.00	160.00	34.79 ± 5.55	83.27 ± 0.12

Table 3.7. Naive Bayes with FHDDMS Variants against Data Streams with Abrupt Drift

(a) NB - Sine1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**40.62 ± 5.73**	3.99 ± 0.10	0.17 ± 0.38	0.01 ± 0.10	528.00	30.18 ± 5.68	**86.08 ± 0.25**
$\text{FHDDMS}_{\text{add}}$	51.94 ± 4.21	4.00	**0.02 ± 0.14**	0.00	**152.00**	**14.64 ± 4.01**	86.05 ± 0.25
FHDDM_{25}	**40.48 ± 3.37**	4.00	**0.04 ± 0.20**	0.00	232.00	**12.11 ± 3.63**	**86.08 ± 0.25**
FHDDM_{100}	48.19 ± 5.54	3.99 ± 0.10	0.13 ± 0.34	0.01 ± 0.10	528.00	17.46 ± 4.58	86.06 ± 0.25
CUSUM	83.27 ± 6.96	3.99 ± 0.10	0.71 ± 0.86	0.01 ± 0.10	**128.00**	**14.41 ± 3.57**	85.96 ± 0.25
$\text{HDDM}_{\text{A-test}}$	88.03 ± 25.73	3.97 ± 0.17	0.35 ± 0.55	0.03 ± 0.17	160.00	35.68 ± 6.06	85.95 ± 0.25
$\text{HDDM}_{\text{W-test}}$	**35.52 ± 3.10**	4.00	0.41 ± 0.58	0.00	160.00	35.62 ± 6.25	**86.09 ± 0.25**

(b) NB - Mixed Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**40.58 ± 6.92**	3.99 ± 0.10	0.78 ± 0.88	0.01 ± 0.10	528.00	30.23 ± 5.62	**83.37 ± 0.08**
$\text{FHDDMS}_{\text{add}}$	52.25 ± 6.88	3.99 ± 0.10	**0.16 ± 0.39**	0.01 ± 0.10	**152.00**	**14.01 ± 3.49**	**83.37 ± 0.08**
FHDDM_{25}	**40.56 ± 3.72**	4.00	**0.25 ± 0.48**	0.00	232.00	**11.54 ± 3.57**	**83.38 ± 0.08**
FHDDM_{100}	48.77 ± 6.86	3.99 ± 0.10	0.58 ± 0.79	0.01 ± 0.10	528.00	15.61 ± 3.95	83.36 ± 0.09
CUSUM	88.23 ± 8.97	3.99 ± 0.10	0.35 ± 0.54	0.01 ± 0.10	**128.00**	**14.49 ± 3.82**	83.27 ± 0.08
$\text{HDDM}_{\text{A-test}}$	83.71 ± 19.46	3.96 ± 0.20	0.48 ± 0.64	0.04 ± 0.20	160.00	35.57 ± 5.59	83.28 ± 0.09
$\text{HDDM}_{\text{W-test}}$	**35.75 ± 3.94**	4.00	1.77 ± 1.39	0.00	160.00	34.92 ± 6.01	**83.36 ± 0.09**

3.3.4.2 Experiments against Gradual Concept Drift

We summarized the performance of FHDDMS, $\text{FHDDMS}_{\text{add}}$, and the other methods with Hoeffding Tree (HT) and Naive Bayes (NB) against CIRCLES and LED in Tables 3.8 and 3.9, respectively. Recall that gradual concept drift is simulated at every $33,333$ instances of CIRCLES and LED, with a transition length of 500. We analyze results of the experiments for FHDDMS and $\text{FHDDMS}_{\text{add}}$ as follows.

FHDDMS – FHDDMS detected gradual concept drifts with the shortest delays compared to other methods. Although FHDDMS resulted in lower false negative numbers compared to FHDDM_{25}, it showed higher false positive numbers than FHDDM_{100}. The reader may notice that FHDDMS and FHDDM_{100} had the same level of memory usage. The runtimes of FHDDMS were comparable to those of $\text{HDDM}_{\text{A-test}}$ and $\text{HDDM}_{\text{W-test}}$. Furthermore, the classification accuracy is similar for FHDDMS, FHDDMs and HDDMs.

$\text{FHDDMS}_{\text{add}}$ – As to the detection delay, $\text{FHDDMS}_{\text{add}}$ was slower than FHDDMS and FHDDM_{100} to trigger an alarm for concept drift; but faster than FHDDM_{25}. $\text{FHDDMS}_{\text{add}}$ also represented comparable, or even shorter, detection delays compared to other methods. Concerning the false positive measure, $\text{FHDDMS}_{\text{add}}$ performed better than FHDDM_{25}, FHDDM_{100}, and HDDMs; that shows the use of the summation of entries is beneficial. $\text{FHDDMS}_{\text{add}}$ also indicated the fastest execution runtime and less memory consumption in comparison with FHDDMs and HDDMs for its concise data summarization technique. Finally, we obtained comparable classification accuracies by $\text{FHDDMS}_{\text{add}}$ to the rest of methods.

Table 3.8. Hoeffding Tree with FHDDMS Variants against Data Streams with Gradual Drift

(a) HT - CIRCLES Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**75.29 ± 25.40**	3.00	0.31 ± 0.59	0.00	528.00	30.61 ± 5.38	**86.58 ± 0.14**
$\text{FHDDMS}_{\text{add}}$	96.75 ± 33.35	3.00	**0.08 ± 0.27**	0.00	**152.00**	**14.13 ± 3.57**	**86.59 ± 0.14**
FHDDM_{25}	243.97 ± 122.26	2.71 ± 0.45	0.37 ± 0.56	0.29 ± 0.45	232.00	**11.39 ± 3.03**	86.44 ± 0.26
FHDDM_{100}	**79.28 ± 20.64**	3.00	0.17 ± 0.40	0.00	528.00	16.25 ± 4.18	**86.58 ± 0.13**
CUSUM	220.07 ± 31.79	2.99 ± 0.10	**0.04 ± 0.20**	0.01 ± 0.10	**128.00**	**14.95 ± 3.59**	86.51 ± 0.13
$\text{HDDM}_{\text{A-test}}$	111.96 ± 68.22	2.96 ± 0.20	0.65 ± 0.92	0.04 ± 0.20	160.00	35.02 ± 6.61	86.52 ± 0.20
$\text{HDDM}_{\text{W-test}}$	94.03 ± 57.61	2.98 ± 0.14	0.73 ± 0.87	0.02 ± 0.14	160.00	36.48 ± 5.98	86.53 ± 0.18

(b) HT - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**220.17 ± 76.03**	2.97 ± 0.22	0.04 ± 0.24	0.03 ± 0.22	528.00	27.56 ± 5.26	89.56 ± 0.04
$\text{FHDDMS}_{\text{add}}$	249.75 ± 77.20	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	**152.00**	**11.90 ± 3.30**	89.56 ± 0.04
FHDDM_{25}	346.36 ± 97.35	2.94 ± 0.28	0.06 ± 0.28	0.06 ± 0.28	232.00	**12.60 ± 3.86**	89.53 ± 0.04
FHDDM_{100}	**220.40 ± 76.00**	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	528.00	17.39 ± 4.52	89.57 ± 0.04
CUSUM	300.68 ± 50.30	3.00	0.00	**0.00**	**128.00**	**12.65 ± 3.50**	89.56 ± 0.03
$\text{HDDM}_{\text{A-test}}$	295.03 ± 85.29	2.98 ± 0.20	0.16 ± 0.44	0.02 ± 0.20	160.00	31.54 ± 5.82	89.58 ± 0.05
$\text{HDDM}_{\text{W-test}}$	259.18 ± 87.25	2.95 ± 0.26	0.08 ± 0.31	0.05 ± 0.26	160.00	32.60 ± 6.57	89.56 ± 0.04

Table 3.9. Naive Bayes with FHDDMS Variants against Data Streams with Gradual Drift

(a) NB - Circles Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**164.87 ± 86.81**	**2.95 ± 0.22**	0.60 ± 0.77	**0.05 ± 0.22**	528.00	28.95 ± 5.73	**84.14 ± 0.12**
$\text{FHDDMS}_{\text{add}}$	260.42 ± 120.28	2.76 ± 0.43	**0.20 ± 0.49**	0.24 ± 0.43	**152.00**	**14.38 ± 3.76**	84.08 ± 0.14
FHDDM_{25}	432.30 ± 88.98	2.16 ± 0.37	0.40 ± 0.66	0.84 ± 0.37	232.00	**10.83 ± 3.14**	83.96 ± 0.15
FHDDM_{100}	**166.13 ± 83.84**	**2.96 ± 0.20**	0.43 ± 0.60	**0.04 ± 0.20**	528.00	16.51 ± 4.09	**84.14 ± 0.13**
CUSUM	299.78 ± 52.29	**3.00**	0.40 ± 0.62	**0.00**	**128.00**	**14.92 ± 4.30**	84.08 ± 0.12
$\text{HDDM}_{\text{A-test}}$	306.91 ± 107.78	2.91 ± 0.29	0.49 ± 0.6	0.09 ± 0.29	160.00	35.06 ± 6.18	84.09 ± 0.12
$\text{HDDM}_{\text{W-test}}$	242.43 ± 134.19	2.73 ± 0.44	1.59 ± 1.00	0.27 ± 0.44	160.00	34.88 ± 5.56	84.11 ± 0.13

(b) NB - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
FHDDMS	**220.17 ± 76.03**	2.97 ± 0.22	0.04 ± 0.24	0.03 ± 0.22	528.00	27.56 ± 5.83	89.57 ± 0.04
$\text{FHDDMS}_{\text{add}}$	249.92 ± 78.45	2.97 ± 0.22	0.02 ± 0.14	0.03 ± 0.22	**152.00**	**13.07 ± 3.88**	89.57 ± 0.04
FHDDM_{25}	347.53 ± 101.36	2.93 ± 0.32	0.02 ± 0.14	0.07 ± 0.32	232.00	**12.80 ± 3.66**	89.54 ± 0.05
FHDDM_{100}	**220.40 ± 76.00**	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	528.00	17.52 ± 4.40	89.57 ± 0.04
CUSUM	300.61 ± 50.30	3.00	0.00	0.00	**128.00**	**12.81 ± 3.51**	89.57 ± 0.03
$\text{HDDM}_{\text{A-test}}$	295.85 ± 83.23	2.98 ± 0.20	0.17 ± 0.47	0.02 ± 0.20	160.00	31.83 ± 5.41	89.58 ± 0.05
$\text{HDDM}_{\text{W-test}}$	259.17 ± 87.21	2.95 ± 0.26	0.05 ± 0.22	0.05 ± 0.26	160.00	33.26 ± 5.24	89.56 ± 0.03

3.3.4.3 Discussion

In Section 3.2, the Fast Hoeffding Drift Detection Method (FHDDM) was introduced for detecting concept drift in evolving data streams. We discussed that the FHDDM algorithm needs a shorter window to detect *abrupt* drifts faster; while it needs a longer window to detect *gradual* drifts with lower false negative numbers. Nonetheless, abrupt and gradual concept drifts may happen together in real-world applications; and consequently, a fixed size window, either short or long, may not be advantageous. For this issue, we presented the Stacking Fast Hoeffding Drift Detection Method (FHDDMS), and its additive version ($\text{FHDDMS}_{\text{add}}$), which slide a short and a long windows together over their entries to detect abrupt and gradual drifts. Our evaluation showed FHDDMS and $\text{FHDDMS}_{\text{add}}$ had comparable performance to FHDDM_{25}, FHDDM_{100}, CUSUM, and HDDMs for both *abrupt* and *gradual* concept drifts. Our findings are summarized as follows.

- **FHDDMS** – FHDDMS alarmed faster than FHDDM_{100}, CUSUM, and $\text{HDDM}_{\text{A-test}}$ for abrupt drift; and also faster than $\text{HDDM}_{\text{W-test}}$ for gradual drift. FHDDMS had comparable false negative numbers to FHDDM_{100}; however, it represented more false positives because its short and long windows individually triggered alarms for concept drift.
- **$\text{FHDDMS}_{\text{add}}$** – $\text{FHDDMS}_{\text{add}}$ detected concept drift with longer delays than FHDDMS and FHDDMs, because of waiting for its short window to be filled for every 25 instances. Nevertheless, it led to shorter detection delays compared to $\text{HDDM}_{\text{A-test}}$ and CUSUM. $\text{FHDDMS}_{\text{add}}$ represented fewer false positives, which indicates that the summation of bits may decrease the chance of false alarm. Meanwhile, $\text{FHDDMS}_{\text{add}}$

showed analogous false negative numbers to others. Finally, it consumed less memory than FHDDMS and FHDDMs for its summarization technique.

- **Classification Accuracy** – FHDDMS and $\text{FHDDMS}_{\text{add}}$ led to the similar classification accuracies, with both Hoeffding Tree (HT) and Naive Bayes (NB), compared to the state-of-the-art methods.

So far we assumed that the entries into the sliding windows of the FHDDM variants share the same weights. However, in a streaming environment, the earlier examples may become less important as instances are processed (Gama et al., 2014). This observation may also be extended to the prediction results as we are interested to know how a classifier behaves against the most recent examples. Thus, for the task of drift detection, assigning higher weights to the recent entries may lead to *faster* detect concept drift in a data stream. In the next section, we introduce the McDiarmid Drift Detection Methods (MDDMs) which associate the entries of their windows with weights, where $w_i < w_{i+1}$ meaning that the most recent elements have higher weights, to detect concept drift faster than FHDDM.

3.4 McDiarmid Drift Detection Methods

Incremental learners should rely on the most recent examples for training, as they reflect the current situation more adequately (Gama et al., 2014). Fading or weighting approaches are typically used by online learning algorithms to increase the weight attributed to the most recent instances (Gama et al., 2014). This is important from an adaptive learning perspective, especially when a transition between two contexts is occurring. For example, Klinkenberg (2004) relied on an exponential weighting scheme $w_\lambda(\mathrm{x}_i) = exp(-\lambda i)$, where λ is a parameter, and i is the entry index, to assign lower weights to old examples. Based on this observation, assigning higher weights to the recent predictions may potentially result in faster detection of concept drift. In this section, we introduce the McDiarmid Drift Detection Methods (MDDMs) which utilizes a weighting scheme to ponderate the elements of its sliding window for faster detection of concept drift. We also discuss three variants of MDDM as well as the sensitivity of their parameters.

3.4.1 McDiarmid Drift Detection Methods (MDDMs)

The McDiarmid Drift Detection Method (MDDM) applies McDiarmid's inequality (McDiarmid, 1989) to detect concept drift in evolving data streams. The MDDM algorithm slides a window of size n over the prediction results. It inserts a 1 into the window if the prediction result is *correct*; and 0 otherwise. Each element in the window is associated with a weight, as illustrated in Fig. 3.5, where $w_i < w_{i+1}$. While inputs are processed, the

weighted average of the elements inside the sliding window is calculated, i.e., μ_w^t, as well as the maximum weighted mean observed so far, i.e., μ_w^m, as indicated in Equation (3.8).

$$\text{if} \quad \mu_w^m < \mu_w^t \Rightarrow \mu_w^m = \mu_w^t \tag{3.8}$$

Recall that MDDM relies on the assumption that, by weighting the prediction results in the sliding window, and by putting more emphasis on the most recent elements, concept drift could potentially be detected faster and more efficiently. Given the rule $w_i < w_{i+1}$, the elements located at the head of the window have higher weights than those located at the tail. Different weighting schemes have been considered including the *arithmetic* and the *geometric* schemes. The arithmetic scheme is given by $w_i = 1 + (i-1) * d$, 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. Also, we employ the Euler scheme which is defined by $r = e^{\lambda}$ where $\lambda \geq 0$. We have implemented three weighted drift detection methods based on these three schemes: MDDM-A (A for arithmetic), MDDM-G (G for geometric), and MDDM-E (E for Euler)[6]. Each algorithm works as follows. As the entries are processed one-by-one, the algorithm calculates the weighted average of the elements inside the sliding window and simultaneously updates two variables μ_w^t (i.e., the current weighted average) and μ_w^m (i.e., the maximum weighted average observed so far). A significant difference between μ_w^m and μ_w^t implies the occurrence of concept drift.

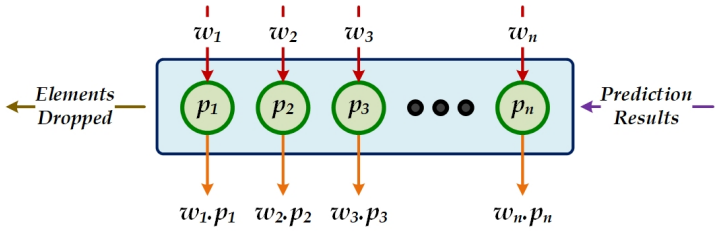


Fig. 3.5. McDiarmid Drift Detection Method (MDDM)

Recall that on the basis of the PAC learning model (Mitchell, 1997), the classification accuracy should increase or stay constant as the number of instances increases; otherwise, the possibility of facing drift increases (Gama et al., 2004; Sebastião et al., 2017). Thus, the value of μ_w^m should increase or remain constant as more instances are processed. That is, the possibility of facing concept drift increases if μ_w^m does not change and μ_w^t decreases over time. Finally, as shown in Equation (3.9), a significant difference in between μ_w^m and μ_w^t indicates the occurrence of concept drift:

$$\Delta\mu_w = \mu_w^m - \mu_w^t \geq \varepsilon_w \Rightarrow \text{Drift} := \text{True} \tag{3.9}$$

6 The source codes for MOA are available at `https://www.github.com/alipsgh/`.

The McDiarmid inequality (McDiarmid, 1989) is employed to determine if the difference is deemed to be significant.

Theorem II: McDiarmid's Inequality – Let $X_1, X_2, ..., X_n$ be n independent random variables all taking values in the set χ. Further, let $f : \chi^n \mapsto \mathbb{R}$ be a function of $X_1, ..., X_n$ that satisfies $\forall i, \forall x_1, ..., x_n, x_i' \in \chi$,

$$|f(x_1, ..., x_i, ..., x_n) - f(x_1, ..., x_i', ..., x_n)| \leq c_i.$$

This implies that replacing the i^{th} coordinate x_i by some arbitrary value changes the function f by at most c_i. Then, for all $\varepsilon_M > 0$, we have:

$$\Pr\{E[f] - f \geq \varepsilon_M\} \leq \exp\left(-\frac{2\varepsilon_M^2}{\sum_{i=1}^n c_i^2}\right) \tag{3.10}$$

Consequently, for a given confidence level δ_M, the value of ε_M is obtained by:

$$\varepsilon_M = \sqrt{\frac{\sum_{i=1}^n c_i^2}{2} \ln \frac{1}{\delta_M}} \tag{3.11}$$

For MDDM, we denote the function f by f_w and define it as the weighted mean of variables (or entries), i.e., μ_w, by:

$$\mu_w = f_w(p_1, ..., p_i, ..., p_n) = \frac{\sum_{i=1}^n w_i \times p_i}{\sum_{i=1}^n w_i} \tag{3.12}$$

where p_i is the i^{th} variable (or entry), and $p_i \in \{0, 1\}$. Thus, f_w changes by $v_i = w_i / \sum_{i=1}^n w_i$ if p_i changes from 0 to 1, and vice versa. Following this observation, we define Corollary II, where $\mu_w^t = f_w(p_1, ..., p_i, ..., p_n)^t$ at time t.

Corollary II: MDDM test – In a streaming context, assume μ_w^t is the weighted mean of a sequence of n random entries, at time t, and μ_w^m is the maximum weighted mean observed so far. Recall that each entry p_i is in $\{0, 1\}$ and has a weight of w_i. Let $\Delta\mu_w = \mu_w^m - \mu_w^t \geq 0$ be the difference between these two weighted means. Given the confidence level δ_w, the McDiarmid inequality detects a drift if $\Delta\mu_w \geq \varepsilon_w$, where

$$\varepsilon_w = \sqrt{\frac{\sum_{i=1}^n v_i^2}{2} \ln \frac{1}{\delta_w}} \tag{3.13}$$

and where v_i is given by

$$v_i = \frac{w_i}{\sum_{i=1}^n w_i} \tag{3.14}$$

All three of our MDDM approaches apply Corollary II in order to detect concept drift. The reader should note that the McDiarmid inequality allows for comparing the expectation of a function of the random variables, such as the maximum, with the function "per se". This stands in contrast with, for instance the Hoeffding inequality (Hoeffding, 1963), in which the comparison is restricted to the expectation of the random variables with their empirical mean.

The pseudocode for the MDDM algorithm appears in Algorithm 3.3. The INITIALIZE function initializes the parameters, e.g., window size n, δ_w, ε_w. As the data stream examples are processed, the prediction results are pushed into the window (lines 8-14). The algorithm updates the variables μ_w^t and μ_w^m over time (lines 15-17). Finally, concept drift is detected if $(\mu_w^m - \mu_w^t) \geq \varepsilon_w$ (lines 18-21). Recall that we have $w_i = 1 + (i-1) * d$ for MDDM-A, $w_i = r^{(i-1)}$ for MDDM-G, and $w_i = e^{\lambda(i-1)}$ for MDDM-E.

Algorithm 3.3 McDiarmid Drift Detection Method (MDDM)

```
1:  function INITIALIZE(windowSize, delta)
2:      (n, δ_w) ← (windowSize, delta)
3:      ε_w = CALCULATEEPSILON()
4:      RESET()
5:  function RESET()
6:      win = []                                  ▷ Creating an empty sliding window.
7:      μ_w^m = 0
8:  function DETECT(p)                            ▷ The p is 1 for correct predictions, 0 otherwise.
9:      if win.size() = n then
10:         win.tail.drop()                       ▷ Dropping an element from the tail.
11:     win.push(p)                               ▷ Pushing an element into the head.
12:     if win.size() < n then
13:         return False
14:     else
15:         μ_w^t = GETWEIGHTEDMEAN()
16:         if μ_w^m < μ_w^t then
17:             μ_w^m = μ_w^t
18:         Δμ_w = μ_w^m − μ_w^t
19:         if Δμ_w ≥ ε_w then
20:             RESET()                           ▷ Resetting parameters.
21:             return True                       ▷ Signaling for an alarm.
22:         else
23:             return False
24: function CALCULATEEPSILON()
25:     S = Σ_{i=1}^{n} v_i^2                     ▷ v_i = w_i / Σ_{i=1}^{n} w_i
26:     return sqrt( (S/2) ln (1/δ_w) )
27: function GETWEIGHTEDMEAN()
28:     return Σ_{i=1}^{n} (p_i × w_i) / Σ_{i=1}^{n} w_i
```

3.4.2 Discussion on MDDM Variants

Recall that the MDDM-A approach employs an arithmetic scheme $w_i = 1+(i-1)*d$, where $w_{i+1} - w_i = d$ meaning that the weights increase *linearly*. On the other hand, MDDM-G applies the geometric scheme $w_i = r^{(i-1)}$, indicating that the weights increase *exponentially* with $w_{i+1}/w_i = r$ (Note that $r = e^{\lambda}$ for MDDM-E). The linear or exponential nature of the weighting scheme affects the detection delay and the false positive rate. That is, the exponential weighting scheme often results in a faster concept drift detection, but at the expense of a higher false positive rate if compared to the linear weighting scheme. These statements are supported by experimental results in Section 3.4.4.

3.4.3 Sensitivity of Parameters

The parameters n and δ_w are inversely proportional with respect to ε_w. That is, as the value of n increases, the value of ε_w decreases. This implies that, as more observations become available, a more optimistic error bound should be applied. On the other hand, as the value of δ_w decreases, the values of ε_w increases (i.e., the bound becomes more conservative). The parameter d in MDDM-A controls the scale of the weights assigned to the sliding window elements. The value of ε_w increases, as the value of d increases. A larger value of d leads to a faster drift detection, since higher weights are assigned to the element located at the head of the window; however, the false positive rate may increase. MDDM-G and MDDM-E behave similarly; the scale of their weight is determined by their parameters r and λ, respectively. That is, a higher r or λ leads to a shorter detection delay, but at the expense of a higher false positive rate. In order to set the default values of these parameters, we conducted a number of experiments against various (benchmark) synthetic data streams. We gradually increased the values of these parameters to find the optimal values: $\delta_w = 10^{-6}$, $d = 0.01$, $r = 1.01$, and $\lambda = 0.01$.

3.4.4 Experimental Evaluation

For our experiment, we compare the performance of MDDMs with that of FHDDM as well as CUSUM, $\text{HDDM}_{\text{A-tes}}$, and $\text{HDDM}_{\text{W-test}}$ which showed promising performance compared to other state-of-the-art methods in Section 3.2.2.

Similar to the previous experiments, we employed Hoeffding Tree (HT) and Naive Bayes (NB) as the base learner. We ran them with every drift detector for 100 times and averaged the results of detection delay, true positive, false positive, false negative, detection runtime (in millisecond), memory usage (in bytes) of drift detectors as well as the accuracies of classifiers. We set the *acceptable delay length* Δ to 250 for SINE1 and MIXED and to $1,000$ for the CIRCLES and LED data streams. Recall that a greater Δ is used for gradual drift to avoid an increase in the false negative number.

Parameter Settings – We ran FHDDM with two window sizes of 25 and 100 in Section 3.2.2, and discussed that a short window reacts to an abrupt concept drift faster; whereas, a long window detects gradual concept drifts with fewer false negatives (and so shorter delay). Hence, for a fair comparison, we also ran the MDDM algorithms with two window sizes of 25 and 100 against data streams with *abrupt* and *gradual* concept drifts, respectively. As before, we set δ and δ_w to 10^{-6} to guarantee a big enough ε_d and ε_w. The reader may refer back to Section 3.2.2 for further details. CUSUM and HDDMs were run with the default parameters as set in MOA.

The experiments were evaluated *prequentially* on an Intel Core i5 @ 2.8 GHz with 16 GB of RAM running Apple OS X Yosemite. We, finally, organized our experiments as follows.

- MDDM variants against *abrupt* change: We evaluate MDDMs against the SINE1 and MIXED data streams, in which *abrupt* change is experienced at every $20,000$ instances, in Tables 3.10 and 3.11 for Hoeffding Tree and Naive Bayes, respectively.
- MDDM variants against *gradual* change: We compare MDDMs, FHDDM, CUSUM, and HDDMs against CIRCLES and LED, in which *gradual* concept drift happens at every $33,333$ instances, with the Hoeffding Tree and Naive Bayes classifiers in Tables 3.12 and 3.13, respectively.

The reader should note that the primary goal of our experiments for the MDDM variants is to show they are capable of detecting concept drift *faster* than their predecessor FHDDM regarding drift detection delay, while their false positive and false negative numbers are comparable to other methods.

3.4.4.1 Experiments against Abrupt Concept Drift

Considering Tables 3.10 and 3.11, MDDM-A_{25}, MDDM-G_{25}, and MDDM-E_{25} detected the drift points with shorter delay compared to FHDDM_{25}; however, they slightly had higher false positive numbers. Although the MDDM_{25} variants were slower than $\text{HDDM}_{\text{W-test}}$ concerning detection delay, their false positive numbers were less than half of $\text{HDDM}_{\text{W-test}}$'s. This observation confirms that the MDDM_{25} variants had higher precision. The MDDM_{25} algorithms also detected concept drifts faster than CUSUM, and $\text{HDDM}_{\text{A-test}}$ with fewer false alarms. MDDMs had longer execution runtime compared to FHDDM in average, for conducting more calculations. In addition, MDDMs and FHDDM had the same memory usage. As shown in Table 3.10, for the Hoeffding Tree classifier, the highest classification accuracies were obtained with MDDMs and FHDDM, since they detected drifts with the shortest delays and the lowest false positive rates. Similar observations apply to the Naive Bayes classifier in Table 3.11. Considering both tables, we may notice that the false positive rate is lower for Naive Bayes. This observation suggests that Naive Bayes represented fewer fluctuations against the noisy SINE1 and MIXED data streams.

Table 3.10. Hoeffding Tree and MDDM Variants against Data Streams with Abrupt Drift

(a) HT - Sine1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
MDDM-A_{25}	**38.60 ± 3.38**	4.00	0.21 ± 0.43	0.00	232.00	19.86 ± 4.23	87.07 ± 0.16
MDDM-G_{25}	**38.56 ± 3.36**	4.00	0.20 ± 0.42	0.00	232.00	15.47 ± 4.04	87.07 ± 0.16
MDDM-E_{25}	**38.56 ± 3.36**	4.00	0.20 ± 0.42	0.00	232.00	35.78 ± 5.72	87.07 ± 0.16
$FHDDM_{25}$	40.65 ± 3.15	4.00	0.10 ± 0.33	0.00	232.00	11.40 ± 3.50	87.07 ± 0.16
CUSUM	86.89 ± 4.47	4.00	0.24 ± 0.47	0.00	128.00	15.60 ± 3.95	86.94 ± 0.15
$HDDM_{A\text{-test}}$	57.62 ± 11.81	4.00	0.71 ± 0.89	0.00	160.00	34.77 ± 5.82	87.01 ± 0.16
$HDDM_{W\text{-test}}$	**35.70 ± 2.95**	4.00	0.46 ± 0.68	0.00	160.00	36.55 ± 5.88	87.07 ± 0.15

(b) HT - Mixed Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
MDDM-A_{25}	**38.38 ± 3.66**	4.00	1.11 ± 1.15	0.00	232.00	20.47 ± 4.49	83.36 ± 0.11
MDDM-G_{25}	**38.28 ± 3.64**	4.00	1.19 ± 1.21	0.00	232.00	15.31 ± 3.79	83.36 ± 0.11
MDDM-E_{25}	**38.28 ± 3.64**	4.00	1.19 ± 1.21	0.00	232.00	35.84 ± 6.61	83.36 ± 0.11
$FHDDM_{25}$	40.55 ± 3.70	4.00	0.65 ± 0.94	0.00	232.00	11.85 ± 3.81	83.39 ± 0.10
CUSUM	90.90 ± 6.13	4.00	0.32 ± 0.58	0.00	128.00	14.25 ± 3.66	83.27 ± 0.12
$HDDM_{A\text{-test}}$	69.42 ± 15.51	4.00	1.28 ± 1.09	0.00	160.00	35.64 ± 5.44	83.31 ± 0.11
$HDDM_{W\text{-test}}$	**35.56 ± 3.50**	4.00	3.23 ± 1.95	0.00	160.00	34.79 ± 5.55	83.27 ± 0.12

Table 3.11. Naive Bayes and MDDM Variants against Data Streams with Abrupt Drift

(a) NB - Sine1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
MDDM-A_{25}	**38.55 ± 3.35**	4.00	0.13 ± 0.34	0.00	232.00	20.54 ± 4.08	86.08 ± 0.25
MDDM-G_{25}	**38.47 ± 3.35**	4.00	0.14 ± 0.35	0.00	232.00	15.10 ± 3.48	86.08 ± 0.25
MDDM-E_{25}	**38.46 ± 3.35**	4.00	0.14 ± 0.35	0.00	232.00	35.98 ± 6.75	86.08 ± 0.25
$FHDDM_{25}$	40.48 ± 3.37	4.00	0.04 ± 0.20	0.00	232.00	12.11 ± 3.63	86.08 ± 0.25
CUSUM	83.27 ± 6.96	3.99 ± 0.10	0.71 ± 0.86	0.01 ± 0.10	128.00	14.41 ± 3.57	85.96 ± 0.25
$HDDM_{A\text{-test}}$	88.03 ± 25.73	3.97 ± 0.17	0.35 ± 0.55	0.03 ± 0.17	160.00	35.68 ± 6.06	85.95 ± 0.25
$HDDM_{W\text{-test}}$	**35.52 ± 3.10**	4.00	0.41 ± 0.58	0.00	160.00	35.62 ± 6.25	86.09 ± 0.25

(b) NB - Mixed Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
MDDM-A_{25}	**38.52 ± 3.81**	4.00	0.69 ± 0.89	0.00	232.00	20.76 ± 4.69	83.37 ± 0.09
MDDM-G_{25}	**38.41 ± 3.81**	4.00	0.70 ± 0.89	0.00	232.00	16.44 ± 4.08	83.37 ± 0.09
MDDM-E_{25}	**38.41 ± 3.81**	4.00	0.70 ± 0.89	0.00	232.00	36.58 ± 5.77	83.37 ± 0.09
$FHDDM_{25}$	**40.56 ± 3.72**	4.00	0.25 ± 0.48	0.00	232.00	11.54 ± 3.57	83.38 ± 0.08
CUSUM	88.23 ± 8.97	3.99 ± 0.10	0.35 ± 0.54	0.01 ± 0.10	128.00	14.49 ± 3.82	83.27 ± 0.08
$HDDM_{A\text{-test}}$	83.71 ± 19.46	3.96 ± 0.20	0.48 ± 0.64	0.04 ± 0.20	160.00	35.57 ± 5.59	83.28 ± 0.09
$HDDM_{W\text{-test}}$	**35.75 ± 3.94**	4.00	1.77 ± 1.39	0.00	160.00	34.92 ± 6.01	83.36 ± 0.09

3.4.4.2 Experiments against Gradual Concept Drift

Tables 3.12 and 3.13 represent the results of our evaluations for the Hoeffding Tree and Naive Bayes classifiers against the CIRCLES and LED data streams. The $MDDM_{100}$ variants resulted in the shortest concept drift detection delays, followed by $FHDDM_{100}$ and $HDDM_{W\text{-test}}$. In contrast to $FHDDM_{100}$, the $MDDM_{100}$ variants detected drifts faster because of their weighting schemes which favor the most recent elements. On the other hand, CUSUM led to the longest detection delays, which implies that CUSUM is less sensitive to the transitions among concepts. MDDMs showed fewer false positives compared to HDDMs, which indicates their higher precision for avoiding false alarms. Further, $HDDM_{A\text{-test}}$ and $HDDM_{W\text{-test}}$ showed higher false negative numbers; the difference is not considerable though. In general, the $MDDM_{100}$ variants led to the higher accuracies with both classifiers.

Table 3.12. Hoeffding Tree and MDDMs against Data Streams with Gradual Drift

(a) HT - CIRCLES Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$MDDM\text{-}A_{100}$	**71.98 ± 22.19**	3.00	0.27 ± 0.51	0.00	528.00	41.82 ± 5.91	86.58 ± 0.16
$MDDM\text{-}G_{100}$	**69.42 ± 22.09**	3.00	0.36 ± 0.61	0.00	528.00	32.80 ± 5.89	86.58 ± 0.17
$MDDM\text{-}E_{100}$	**69.52 ± 22.12**	3.00	0.37 ± 0.61	0.00	528.00	51.70 ± 6.90	86.57 ± 0.17
$FHDDM_{100}$	79.28 ± 20.64	3.00	0.17 ± 0.40	0.00	528.00	16.25 ± 4.18	86.58 ± 0.13
CUSUM	220.07 ± 31.79	2.99 ± 0.10	0.04 ± 0.20	0.01 ± 0.10	128.00	14.95 ± 3.59	86.51 ± 0.13
$HDDM_{A\text{-test}}$	111.96 ± 68.22	2.96 ± 0.20	0.65 ± 0.92	0.04 ± 0.20	160.00	35.02 ± 6.61	86.52 ± 0.20
$HDDM_{W\text{-test}}$	94.03 ± 57.61	2.98 ± 0.14	0.73 ± 0.87	0.02 ± 0.14	160.00	36.48 ± 5.98	86.53 ± 0.18

(b) HT - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$MDDM\text{-}A_{100}$	**210.31 ± 73.05**	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	48.18 ± 8.94	89.56 ± 0.04
$MDDM\text{-}G_{100}$	**208.65 ± 73.05**	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	34.55 ± 6.39	89.56 ± 0.04
$MDDM\text{-}E_{100}$	**208.61 ± 73.05**	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	53.15 ± 7.07	89.56 ± 0.04
$FHDDM_{100}$	220.40 ± 76.00	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	528.00	17.39 ± 4.52	89.57 ± 0.04
CUSUM	300.68 ± 50.30	3.00	0.00	0.00	128.00	12.65 ± 3.50	89.56 ± 0.03
$HDDM_{A\text{-test}}$	295.03 ± 85.29	2.98 ± 0.20	0.16 ± 0.44	0.02 ± 0.20	160.00	31.54 ± 5.82	89.58 ± 0.05
$HDDM_{W\text{-test}}$	259.18 ± 87.25	2.95 ± 0.26	0.08 ± 0.31	0.05 ± 0.26	160.00	32.60 ± 6.57	89.56 ± 0.04

3.4.4.3 Discussion

We introduced the McDiarmid Drift Detection Methods (MDDMs) for *faster* detection of concept drift by weighting entries into their sliding windows; where greater weights were assigned to the most recent instances. We evaluated MDDMs against FHDDM, CUSUM, and HDDMs for both abrupt and gradual concept drifts in Sections 3.4.4.1 and 3.4.4.2, respectively. We summarize and discuss our findings below:

Table 3.13. Naive Bayes and MDDMs against Data Streams with Gradual Drift

(a) NB - CIRCLES Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
MDDM-A_{100}	**161.25 ± 87.26**	2.95 ± 0.22	0.63 ± 0.70	0.05 ± 0.22	528.00	44.79 ± 7.24	84.14 ± 0.12
MDDM-G_{100}	**161.73 ± 89.49**	2.94 ± 0.24	0.80 ± 0.73	0.06 ± 0.24	528.00	33.98 ± 5.43	84.14 ± 0.12
MDDM-E_{100}	**161.74 ± 89.49**	2.94 ± 0.24	0.81 ± 0.73	0.06 ± 0.24	528.00	54.70 ± 7.28	84.14 ± 0.12
$FHDDM_{100}$	166.13 ± 83.84	2.96 ± 0.20	0.43 ± 0.60	0.04 ± 0.20	528.00	16.51 ± 4.09	84.14 ± 0.13
CUSUM	299.78 ± 52.29	3.00	0.40 ± 0.62	0.00	128.00	14.92 ± 4.30	84.08 ± 0.12
$HDDM_{A\text{-test}}$	306.91 ± 107.78	2.91 ± 0.29	0.49 ± 0.69	0.09 ± 0.29	160.00	35.06 ± 6.18	84.09 ± 0.12
$HDDM_{W\text{-test}}$	242.43 ± 134.19	2.73 ± 0.44	1.59 ± 1.00	0.27 ± 0.44	160.00	34.88 ± 5.56	84.11 ± 0.13

(b) NB - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
MDDM-A_{100}	**210.31 ± 73.05**	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	52.24 ± 8.31	89.57 ± 0.04
MDDM-G_{100}	**208.65 ± 73.05**	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	34.05 ± 6.14	89.57 ± 0.04
MDDM-E_{100}	**208.61 ± 73.05**	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	53.35 ± 7.59	89.57 ± 0.04
$FHDDM_{100}$	220.40 ± 76.00	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	528.00	17.52 ± 4.40	89.57 ± 0.04
CUSUM	300.61 ± 50.30	3.00	0.00	0.00	128.00	12.81 ± 3.51	89.57 ± 0.03
$HDDM_{A\text{-test}}$	295.85 ± 83.23	2.98 ± 0.20	0.17 ± 0.47	0.02 ± 0.20	160.00	31.83 ± 5.41	89.58 ± 0.05
$HDDM_{W\text{-test}}$	259.17 ± 87.21	2.95 ± 0.26	0.05 ± 0.22	0.05 ± 0.26	160.00	33.26 ± 5.24	89.56 ± 0.03

- **MDDM vs. CUSUM and HDDMs** – MDDMs detected abrupt and gradual drifts faster than $HDDM_{A\text{-test}}$ and CUSUM. MDDMs also had shorter detection delay compared to $HDDM_{W\text{-test}}$ for *gradual* drift, while they showed comparative results for *abrupt* drift. Fewer false alarms experienced by MDDMs than by HDDMs, which indicates MDDMs are more precise. We also achieved higher accuracies by MDDMs.

- **MDDM vs. FHDDM** – We observed that the MDDM algorithms detected drifts swiftly compared to FHDDM in terms of drift detection delay because of assigning greater weights to the most recent entries; however, the weighting approach makes the MDDM variants more sensitive to the performance of classifiers (and indirectly to noise). As a result, we comparably experienced more false alarms with MDDMs than with FHDDMs. MDDMs and FHDDM showed the same memory usage whereas MDDMs were slower regarding execution runtime due to more computations.

- **MDDM Variants** – MDDM-G and MDDM-E had shorter drift detection delays than MDDM-A. The reason is to be found in the fact that they both utilize an exponential weighting scheme (i.e., greater weights are put on the most recent entries which are the ones required for faster detection) as opposed to MDDM-A which has a linear one. The reader may notice that the false positive rates of these two variants against the two streams with gradual change, namely CIRCLES and LED, were higher than those of MDDM-A. This is a consequence of the fact that MDDM-A put more emphasis on the older entries in the window, which, in these cases are beneficial to the learning process. In general, one may observe that an exponential-like scheme is beneficial in scenarios when faster detection is required. It follows that the optimal shape for the weighting function is data, context and application dependent.

We introduced the Fast Hoeffding Drift Detection Method (FHDDM) in Section 3.2; and then represented its significant performance against CUSUM, PageHinkely, DDM, EDDM, RDDM, ADWIN, SeqDrift2, $\text{HDDM}_{\text{A-test}}$, and $\text{HDDM}_{\text{W-test}}$ for both *abrupt* and *gradual* concept drifts in Section 3.2.2. We also indicated the FHDDM window size is a crucial parameter for detecting concept drift with a shorter delay and a lower false negative rate. Furthermore, abrupt and gradual concept drifts may coexist within the same stream in real-world applications. As a result, sliding only either a long window or a short window may not be adequate. Therefore, we devised the Stacking Fast Hoeffding Drift Detection Methods (FHDDMS), in Section 3.3. The FHDDMS algorithm slides a long window and a short window together for concept drift detection. In Section 3.3.4, the FHDDMS variants showed promising results to FHDDM_{25} and FHDDM_{100} against data streams with abrupt and gradual drifts. Thereupon, we introduced the McDiarmid Drift Detection Methods (MDDMs) which put more emphasis on recent entries for detecting concept drifts rapidly, when it comes to comparison with FHDDM. All our evaluations were against *synthetic* data streams of SINE1 and MIXED for abrupt drift as well as CIRCLES and LED for gradual drift. One may refer to Appendix B, where we put all experimental results for all drift detection methods together in Tables B.1 to B.8. The tables indicate that using drift detectors is beneficial compared to the *no detection* setting regarding the classification accuracy. In the next section, we extend our experiments to compare our methods with the state-of-the-art against the synthetic streams of bits, in which elements are *independent and identically distributed* (i.e., i.i.d), and the real-world data streams.

3.5 Complementary Evaluations

3.5.1 Evaluation against Synthetic Streams of Bits

We further investigate the performance of the drift detection methods against synthetic data streams of $50,000$ bits. Similar experiments exist in (Bifet and Gavalda, 2007; Huang et al., 2015). Each stream contains 5 segments where each segment holds $10,000$ bits; and bits are independent and identically distributed. That means every bit b_t is independent of previous bits (for the independent condition) and all bits could be either 0 or 1 with the same probability for each (for the identically distributed condition), before and after a drift point. We simulated abrupt and gradual concept drifts at the *center* of each segment. That is, there exist 5 drifts at location steps of $5,000$, $15,000$, $25,000$, $35,000$, and $45,000$. The probability of observing a 1 as a bit before and after a drift is 80% and 20%, respectively. We considered two values of 0.005 and 0.001 for the *change ratio* r to simulate *gradual* drift. Change ratio defines the speed of shift from one context to another. The acceptable delay length Δ was set to $5,000$ for measuring detection delay as well as false positive and false negative numbers. We conducted a set of preliminary experiments to find the impartial values for the r and Δ parameters. Tables 3.14 and 3.15 summarize the results

of experiments only in terms of average detection delay, false positive, and false negative results for the *abrupt* and *gradual* concept drift, respectively. The reader should also note that we had 100 streams for each experiment, and we averaged the results for each method. We discuss the results as follows.

Abrupt Concept Drift – Table 3.14 shows the MDDM_{25} variants, FHDDMS, FHDDM_{25}, and $\text{HDDM}_{\text{W-test}}$ had the shortest detection delays, whereas, PageHinkley and EDDM resulted in the longest detection delays. PageHinkley, $\text{FHDDMS}_{\text{add}}$, $\text{HDDM}_{\text{W-test}}$, and FHDDM_{100} led to the lowest false positive numbers. $\text{HDDM}_{\text{W-test}}$ had a higher false positive number compared to the FHDDM, FHDDMS and MDDM variants. EDDM showed the highest number of false positive, followed by ADWIN. Although PageHinkley showed a small false positive number, it had the greatest false negative number amongst others.

Table 3.14. Experiments against Synthetic Streams of Bits with Abrupt Drift

	Abrupt		
Detector	Delay	FP	FN
FHDDM_{25}	**11.49 ± 1.52**	**0.99 ± 0.93**	0.00
FHDDM_{100}	24.04 ± 3.08	**0.53 ± 0.83**	0.00
FHDDMS	**11.60 ± 1.99**	**1.38 ± 1.16**	0.00
$\text{FHDDMS}_{\text{add}}$	22.15 ± 0.85	**0.12 ± 0.35**	0.00
MDDM-A_{25}	**9.91 ± 1.58**	**1.73 ± 1.20**	0.00
MDDM-A_{100}	**18.46 ± 2.65**	**0.73 ± 0.93**	0.00
MDDM-G_{25}	**9.92 ± 1.59**	**1.85 ± 1.26**	0.00
MDDM-G_{100}	**16.24 ± 2.54**	**0.91 ± 1.11**	0.00
MDDM-E_{25}	**9.92 ± 1.59**	**1.85 ± 1.26**	0.00
MDDM-E_{100}	**16.22 ± 2.55**	**0.91 ± 1.11**	0.00
CUSUM	153.98 ± 97.32	**0.26 ± 0.52**	0.01 ± 0.10
PageHinkley	3066.16 ± 29.13	0.01 ± 0.10	1.01 ± 0.10
DDM	301.08 ± 38.78	**0.61 ± 0.96**	0.00
EDDM	1074.73 ± 76.60	33.57 ± 11.78	0.00
RDDM	160.95 ± 17.93	3.66 ± 1.82	0.00
ADWIN	42.95 ± 4.73	19.78 ± 2.01	0.00
SeqDrift2	197.00	4.10 ± 0.33	0.00
$\text{HDDM}_{\text{A-test}}$	36.31 ± 12.58	**0.45 ± 0.59**	0.00
$\text{HDDM}_{\text{W-test}}$	**18.61 ± 99.49**	2.69 ± 1.62	0.01 ± 0.10

Gradual Concept Drift – As in Table 3.15, we experienced the shortest detection delays with the FHDDM_{25}, FHDDMS, MDDM_{25} variants, and $\text{HDDM}_{\text{W-test}}$ when $r = 0.005$, and with the FHDDM_{100}, FHDDMS, MDDM_{100} variants, and $\text{HDDM}_{\text{W-test}}$ when $r = 0.001$. Recall that r defines the change ratio of concept drift, and a smaller r results in a slower change or a shift. In all cases, $\text{HDDM}_{\text{W-test}}$ led to higher false alarms compared to the variants of FHDDM and MDDM. $\text{FHDDMS}_{\text{add}}$ and FHDDM_{100} resulted in fewer false positives when $r = 0.005$; whereas, CUSUM and DDM were more accurate when $r = 0.001$. Similar to the case of abrupt drift, EDDM and PageHinkley had the highest false positive and false negative numbers, respectively. They also had lengthy detection delays. Further, the reader may notice that false positives increased by decreasing the change ratio from 0.005 to 0.001. Indeed, a slower shift makes it harder for all methods to detect drifts accurately. Consequently, we experienced that abundance in the false positives.

Table 3.15. Experiments against Synthetic Streams of Bits with Gradual Drift

	Gradual ($r = 0.005$)			**Gradual** ($r = 0.001$)		
Detector	Delay	FP	FN	Delay	FP	FN
$FHDDM_{25}$	**67.82 ± 6.99**	1.91 ± 1.30	0.00	217.73 ± 27.24	6.34 ± 1.23	0.00
$FHDDM_{100}$	76.14 ± 5.50	**0.62 ± 0.86**	0.00	**188.90 ± 16.61**	**5.96 ± 1.06**	0.00
FHDDMS	**65.99 ± 7.02**	1.73 ± 1.26	0.00	**181.99 ± 17.93**	6.97 ± 1.30	0.00
$FHDDMS_{add}$	83.90 ± 6.88	**0.21 ± 0.45**	0.00	216.60 ± 18.39	**5.21 ± 0.45**	0.00
$MDDM\text{-}A_{25}$	**64.99 ± 7.19**	2.81 ± 1.43	0.00	208.35 ± 25.99	7.40 ± 1.34	0.00
$MDDM\text{-}A_{100}$	**69.81 ± 5.47**	**0.85 ± 0.97**	0.00	**179.81 ± 18.75**	**6.26 ± 1.16**	0.00
$MDDM\text{-}G_{25}$	**64.76 ± 7.29**	2.96 ± 1.50	0.00	206.38 ± 26.41	7.57 ± 1.42	0.00
$MDDM\text{-}G_{100}$	**67.11 ± 5.83**	**1.02 ± 1.09**	0.00	**176.48 ± 19.38**	**6.46 ± 1.28**	0.00
$MDDM\text{-}E_{25}$	**64.74 ± 7.29**	2.96 ± 1.50	0.00	206.38 ± 26.41	7.57 ± 1.42	0.00
$MDDM\text{-}E_{100}$	**67.09 ± 5.83**	**1.02 ± 1.09**	0.00	**176.41 ± 19.44**	**6.46 ± 1.28**	0.00
CUSUM	220.52 ± 11.18	**0.32 ± 0.60**	0.00	515.51 ± 21.10	**1.32 ± 0.72**	0.00
PageHinkley	3091.74 ± 29.32	0.01 ± 0.10	1.02 ± 0.14	2987.62 ± 288.01	0.65 ± 0.52	0.78 ± 0.41
DDM	375.48 ± 39.00	**0.62 ± 1.01**	0.00	677.84 ± 45.56	**1.24 ± 1.26**	0.00
EDDM	1124.73 ± 84.79	32.83 ± 11.96	0.00	1311.78 ± 105.75	35.28 ± 14.14	0.00
RDDM	234.37 ± 20.89	3.78 ± 1.87	0.00	519.56 ± 31.67	**6.08 ± 2.30**	0.00
ADWIN	105.22 ± 6.70	27.21 ± 2.13	0.00	257.35 ± 15.81	55.70 ± 3.16	0.00
SeqDrift2	197.00	8.73 ± 0.61	0.00	333.00 ± 42.33	11.83 ± 1.45	0.00
$HDDM_{A\text{-test}}$	97.46 ± 14.13	**0.77 ± 0.83**	0.00	249.45 ± 25.32	**5.60 ± 1.04**	0.00
$HDDM_{W\text{-test}}$	**61.03 ± 6.10**	3.71 ± 1.91	0.00	**182.58 ± 23.37**	9.01 ± 1.72	0.00

Conclusion – The FHDDM and MDDM variants, as well as $HDDM_{W\text{-test}}$, alarmed for concept drift faster than other methods; whereas, EDDM and PageHinkley represented the longest detection delay. Furthermore, the FHDDM and MDDM variants showed fewer false alarms compared to HDDMs, which indicates that our methods are more accurate. Although $FHDDMS_{add}$ had fewer false alarms compared to FHDDM and FHDDMS, it had longer detection delays due to its data summarization approach. In these experiments, we again observed the that MDDM variants detected concept drifts with shorter delays compared to FHDDM for the cost of slightly greater false positive numbers. EDDM and ADWIN frequently caused false alarms which indicate their default tests should become more restrictive. We also observed that the false positive numbers increased as we decreased the change ratio r. In other words, drift detection methods may be deceived by very slow changes, and repeatedly alarm for concept drift.

3.5.2 Evaluation against Real-world Data Streams

There is a consensus among researchers that the locations and/or the presence of concept drift in the ELECTRICITY, FOREST COVERTYPE, and POKER HAND data streams are not known (Frías-Blanco et al., 2015; Pesaranghader and Viktor, 2016; Huang et al., 2015; Bifet et al., 2009; Losing et al., 2018). This implies, in turn, that the drift detection delay as well as the false positive and false negative rates cannot be determined since the knowledge of the drift locations is necessary in order to evaluate these quantities. Consequently, our evaluation is based on the overall classification accuracy and the number of alarms for concept drift issued by each drift detector.

We have also considered *blind adaptation* and *no detection* approaches as benchmarks for our experiments. In the *blind adaptation*, the classifier is retrained ab initio at every 100 instances. The classifiers are trained without drift detectors in the case of *no detection*.

Table 3.16 represents the results for the ELECTRICITY, FOREST COVERTYPE, and POKER HAND data streams with the Hoeffding Tree (HT) and Naive Bayes (NB) classifiers. Firstly, the Hoeffding Tree classifier showed higher classification accuracies compared to Naive Bayes when executed without drift detector. This suggests that the Hoeffding Tree classifier could branch out and adequately reflect the new patterns. Secondly, both classifiers achieved higher classification accuracies by using drift detection. Although this observation indicates that using drift detection methods is beneficial compared to the *no detection* case, it does not necessarily mean that a drift detector outperforms the others. Indeed, in a recent study by Bifet (2017), it was found that blind detection had the highest classification accuracies, against the ELECTRICITY and FOREST COVERTYPE data streams. Based on multiple experiments, Bifet (2017) concluded that this behavior may be explained by the *temporal dependencies* between the instances of the streams.

As shown in Table 3.16, a drift detection method with a higher number of alarms usually led to a higher classification accuracy. In such a case, a classifier learns from a small portion of the data stream where almost all instances are labelled with a common label (this refers to temporal dependencies among examples as stated by Bifet (2017)). To support this observation, as mentioned earlier, we considered a blind adaptation as a benchmark. As shown in the same table, the blind adaptation led to the highest or the second highest classification accuracies. We further extended our experiments by running the FHDDM, FHDDMS and MDDM variants with higher confidence values, i.e., δ. Recall that a higher confidence implies that the drift detection technique is less conservative. As indicated in the table, as the FHDDM, FHDDMS and MDDM variants became less conservative, the number of alarms as well as the classification accuracies increased. Therefore, because of *temporal* dependencies, both classifiers repeatedly learned from instances presenting the same labels between two consecutive alarms.

Conclusion – We conclude that using drift detection methods against the real-world data streams is beneficial. Nevertheless, we are not in a position to make a strong statement based solely on the accuracy since (1) the location of the drift is unknown, and (2) the temporal dependencies exist between instances of the streams (Bifet, 2017). The FHDDM, FHDDMS and MDDM variants consistently led to high classification accuracies. Especially, they achieved the highest classification accuracies in all cases when their value of confidence, i.e., δ, increased from 10^{-6} to 0.001 and 0.01.

Table 3.16. Hoeffding Tree (HT) and Naive Bayes (NB) against Real-world Data Streams

		Electricity				Forest Covertype				Poker hand			
		HT		NB		HT		NB		HT		NB	
	Detector	Alarms	Acc.	Alarms	Acc.	Alarms	Acc.	Alarms	Acc.	Alarms	Acc.	Alarms	Acc.
	FHDDM$_{25}$	90	84.59	109	83.13	1794	85.08	1850	85.09	1876	76.72	1928	76.68
	FHDDM$_{100}$	62	84.09	80	81.86	1110	83.92	1172	83.68	1338	75.74	1423	75.60
	FHDDMS	91	84.42	106	83.14	1650	84.88	1716	84.80	1823	76.60	1906	76.54
	FHDDMS$_{\text{add}}$	73	83.44	86	81.59	1263	83.86	1329	83.68	1326	75.57	1419	75.37
	MDDM-A$_{25}$	105	84.60	126	83.47	1963	85.33	2022	85.38	2036	76.89	2145	76.83
	MDDM-A$_{100}$	65	84.19	86	82.32	1230	84.40	1319	84.09	1409	76.02	1490	75.85
	MDDM-G$_{25}$	105	84.60	126	83.47	1966	85.35	2025	85.39	2034	76.89	2149	76.83
	MDDM-G$_{100}$	70	84.68	91	82.59	1333	84.35	1398	84.22	1466	76.14	1527	76.02
	MDDM-E$_{25}$	105	84.60	126	83.47	1966	85.35	2025	85.39	2034	76.89	2149	76.83
	MDDM-E$_{100}$	68	84.78	91	82.59	1324	84.37	1398	84.21	1467	76.14	1527	76.01
	CUSUM	22	81.71	28	79.21	226	83.01	286	81.55	617	72.85	659	72.54
	PageHinkley	6	81.95	10	78.04	90	81.65	117	80.06	403	71.30	489	70.67
	DDM	169	**85.41**	143	81.18	4301	**87.35**	4634	**88.03**	1046	72.74	433	61.97
	EDDM	191	84.91	203	**84.83**	2466	86.00	2416	86.08	4806	**77.30**	4863	**77.48**
	RDDM	143	85.18	164	84.19	2671	86.42	2733	86.86	2512	76.70	2579	76.67
	ADWIN	65	83.23	88	81.03	1062	83.36	1151	83.24	1358	73.84	1388	73.69
	SeqDrift2	59	82.83	60	79.68	710	82.85	757	82.44	1322	72.51	1395	72.25
	HDDM$_{\text{A-test}}$	210	**85.71**	211	**84.92**	3695	87.24	3284	87.42	2565	76.40	2615	76.48
	HDDM$_{\text{W-test}}$	117	85.06	132	84.09	2342	85.97	2383	86.22	2211	77.11	2312	77.11
	Blind$_{\lvert w\rvert = 100}$	453	84.82	453	84.26	5810	**87.70**	5810	**87.24**	8292	**78.18**	8292	**77.96**
	No Detection	—	73.36	—	79.20	—	60.52	—	80.31	—	59.55	—	76.07
$\delta = 0.001$	FHDDM$_{25}$	170	**85.67**	193	**84.83**	3067	**86.74**	3027	**86.99**	4010	**77.73**	4082	**77.87**
	FHDDM$_{100}$	98	84.65	119	83.42	1686	84.90	1752	84.90	2004	76.48	2065	76.43
	FHDDMS	151	85.20	170	84.16	2462	86.17	2487	86.39	3180	77.28	3277	77.37
	FHDDMS$_{\text{add}}$	124	84.25	139	83.50	2285	85.05	2296	85.11	2474	76.58	2567	76.52
	MDDM-A$_{25}$	180	**85.79**	208	**85.00**	3253	**87.03**	3221	**87.27**	4320	**77.82**	4378	**78.03**
	MDDM-A$_{100}$	107	84.86	128	83.69	1849	85.22	1916	85.25	2102	76.68	2165	76.70
	MDDM-G$_{25}$	182	**85.78**	209	**85.01**	3270	**87.05**	3231	**87.29**	4370	**77.83**	4425	**78.05**
	MDDM-G$_{100}$	112	84.90	128	83.96	1947	85.39	2012	85.49	2208	76.76	2276	76.85
	MDDM-E$_{25}$	182	**85.78**	209	**85.01**	3270	**87.06**	3234	**87.29**	4369	**77.84**	4427	**78.06**
	MDDM-E$_{100}$	112	84.93	128	83.97	1947	85.39	2013	85.49	2209	76.76	2276	76.86
$\delta = 0.01$	FHDDM$_{25}$	251	**85.86**	261	**85.65**	3869	**87.63**	3780	**87.95**	6035	**78.16**	6092	**78.49**
	FHDDM$_{100}$	117	84.86	131	83.99	1956	85.37	2019	85.40	2390	76.71	2457	76.67
	FHDDMS	204	85.13	206	84.85	2873	86.73	2870	87.06	4086	77.48	4125	77.69
	FHDDMS$_{\text{add}}$	176	84.42	189	84.00	2901	85.68	2839	85.89	3559	77.00	3652	76.94
	MDDM-A$_{25}$	256	**85.86**	265	**85.60**	3884	**87.63**	3791	**87.95**	6075	**78.18**	6099	**78.51**
	MDDM-A$_{100}$	133	84.83	146	83.96	2206	85.72	2231	85.91	2600	77.00	2700	76.99
	MDDM-G$_{25}$	252	**85.98**	265	**85.78**	3969	**87.73**	3856	**88.05**	6095	**78.21**	6118	**78.54**
	MDDM-G$_{100}$	136	85.05	150	84.26	2339	85.89	2369	86.17	2752	77.08	2845	77.11
	MDDM-E$_{25}$	252	**85.98**	266	**85.77**	3980	**87.73**	3864	**88.05**	6091	**78.22**	6116	**78.54**
	MDDM-E$_{100}$	136	85.06	150	84.26	2335	85.91	2369	86.18	2758	77.09	2844	77.12

3.6 Summary

We introduced the Fast Hoeffding Drift Detection Methods as well as the McDiarmid Drift Detection Methods for adaptive learning from evolving data streams, in this chapter. We also compared them with the state-of-the-art methods.

In Section 3.2, we represented the Fast Hoeffding Drift Detection Method (FHDDM) which works on the basis of the PAC learning model. This is, the accuracy of a classifier should increase or stay steady as more instances arrive; otherwise, it implies the existence of concept drift in the stream. FHDDM slides a window with a size of n on the prediction results and measures the μ^t, i.e., the average of correct predictions in the most recent n instances at time t. It also updates the value of μ^m that holds the maximum average observed so far. A significant difference, bounded by Hoeffding's inequality (Hoeffding, 1963), between μ^t and μ^m suggests the occurrence of concept drift. In Section 3.2.2, our experiments showed that FHDDM detected concept drifts with shorter detection delay, fewer false positives, and fewer false negatives when compared to the state-of-the-art. We further discussed that the FHDDM window size is a crucial parameter for detecting concept drift with a shorter delay and a lower false negative rate; particularly, in real-world applications, where abrupt and gradual concept drift may coexist in the same streaming data. Hence, sliding only either a long window or a short window may not be adequate.

To settle the issue described above, the Stacking Fast Hoeffding Drift Detection Method (FHDDMS) and Additive Stacking Fast Hoeffding Drift Detection Method ($\text{FHDDMS}_{\text{add}}$) were presented in Section 3.3. FHDDMS is an extension of FHDDM that slides a long window and a short window on the prediction results to reduce the detection delays and false negative rates. Intuitively, a short window should detect abrupt drifts faster, while a long window should detect gradual drifts with a lower false negative rate. $\text{FHDDMS}_{\text{add}}$, as an extension of FHDDMS, substitutes binary entries with their summations to save memory and CPU. Nonetheless, it may compromise the delay of detection for saving resources. Our experimental evaluation indicated that the FHDDMS variants obtained promising results compared to individual FHDDM_{25} and FHDDM_{100} against streams of abrupt and gradual concept drifts, respectively.

In most streaming settings, earlier examples may be considered less informative while instances are processed over time(Gama et al., 2014). This observation also applies to the prediction results as we are interested in monitoring how accurate a classifier behave against the most recent examples. Hence, we may detect concept drift instantly by assigning greater weights to the recent prediction results. In Section 3.4, we introduced the McDiarmid Drift Detection Methods (MDDMs) which associates the elements at the head of its sliding window with greater weights for the instant detection of concept drift. By our evaluations in Section 3.4.4, we concluded that the MDDM variants could detect concept drift with shorter delay compared to FHDDMs; albeit, for the cost of a higher false positive rate. Further, MDDMs showed promising results compared to the existent methods.

The next chapter describes the notion of adaptive multi-strategy learning in dynamic environments. We also introduce the TORNADO framework which we developed for adaptive multi-strategy learning from evolving data streams where concept drift is inevitable. Also, we describe two abstract performance measures to evaluate the learner-detector pairs over time, and recommend the optimal pair, i.e., the most efficient one, to the user.

Chapter 4

Adaptive Multi-Strategy Learning

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

- *"A Framework for Classification in Data Streams using Multi-strategy Learning"*, presented at International Conference on Discovery Science (DS 2016) and published in the proceedings (Pesaranghader et al., 2016). In this paper, we defined the EMR measure for ranking the incremental learners while instances from a data stream are processed one-by-one. Further, we introduced the TORNADO framework for the first time in the paper. We describe the EMR measure and the TORNADO framework in Sections 4.3.2 and 4.4, respectively.
- *"Reservoir of Diverse Adaptive Learners and Stacking Fast Hoeffding Drift Detection Methods for Evolving Data Streams"*, published in the Machine Learning Journal, Springer (Pesaranghader et al., 2018a). In this paper, we expanded our TORNADO framework for adaptive multi-strategy learning and introduced the CAR measure for ranking the (classifier, detector) pairs. We explain the CAR measure and the TORNADO framework in Sections 4.3.3 and 4.4, respectively.

4.1 Problem Statement

Research Problem I – Recall that the most suitable type of learner may vary in evolving and dynamic environments, due to the (distributional) changes. For example, a learner with the lowest error-rate at the current time may show a very higher error-rate in the future while learning from evolving data streams. By viewing a data stream as a sequence of batches of data, we cannot claim a particular learner outperforms others all the time, just because it outperforms them for the current batch.

Fig. 4.1 shows a toy example to represent the fact that a change in the distribution of data, i.e., concept drift, may end up with an increase or a decrease in the error-rates

of different kinds of classifiers. In this toy example, we simulated some drift points and showed that, after a concept drift, the classifier with the lowest error-rate changes.

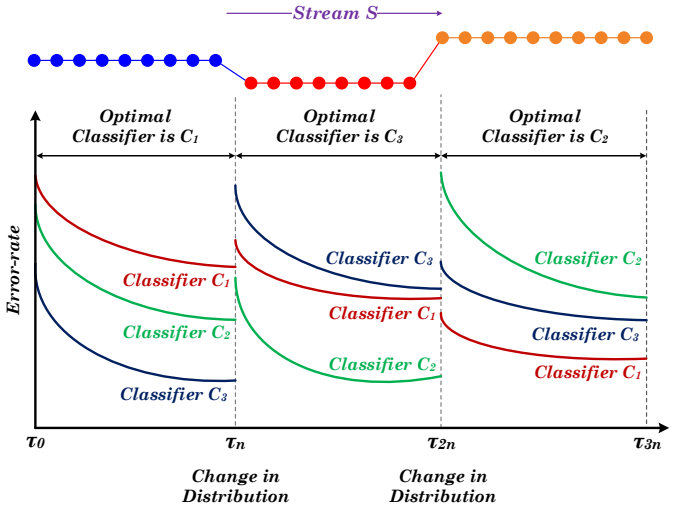


Fig. 4.1. Effect of Distributional Change on Performance of Different Classifiers

Subsequently, the question *"How can we have an optimal learner at hand at the current time against dynamic data streams?"* may be raised. To address this question, we design a reservoir of adaptive learning algorithms by applying the notion of multi-strategy learning from evolving data streams. The reservoir runs various kinds of learners *simultaneously* and keeps track of the one with the highest performance at the current time (or interval). To the best of our knowledge, this challenge was not addressed so far, and available frameworks, e.g., MOA (Bifet et al., 2010a) and WEKA (Garner et al., 1995; Hall et al., 2009), run only one learner against a data stream. Recall that we expressed this research problem and its corresponding research objective in Section 1.2.2 as follows:

> **Research Problem 2.1** is that *a current optimal learning algorithm may not be optimal in the very near future due to the dynamic nature of evolving environments.* We express such a challenge as follows: "*how can we guarantee that an optimal model is available at every time step against evolving data?*"
>
> **Research Objective 2.1** is *to develop a reservoir of various adaptive learning algorithms, for multi-strategy learning against evolving data streams*, able to run different kinds of learning algorithms in *parallel* continuously and to recommend the optimal model, at the current moment or interval, to the user.

In Section 4.4, we introduce the TORNADO framework to address this research problem. The framework implements a reservoir of various learning algorithms and drift detection

methods. All the (classifier, detector) pairs proceed in parallel to construct their models against data streams. At any time, the TORNADO framework provides the user with a pair that currently yields the highest performance. Our experiments confirm that the current '*best*' (classifier, detector) pair is not only heavily dependent on the characteristics of the stream but also such choice evolves as the stream flows.

Research Problem II – The notion of *optimality* is subjective and depends on the domain of application. We consider the most efficient learning algorithm regarding the domain of application as the optimal solution. One may consider the accuracy of classification, or any other performance measure, to define an optimal learner. Another person may probe two or more performance measures and set a compromise among them. For instance, consider learner L_1 with an accuracy of 90.5% and a runtime of $2,000$ milliseconds per instance, and learner L_2 with an accuracy of 91.8% and a runtime of $10,000$ milliseconds per instance. One may compromise accuracy for runtime for a (near) real-time decision-making application, e.g., emergency response and security (Bolton and Hand, 2002; Thuraisingham, 2006), and considers learner L_1. Thus, we need comprehensive measures able to summarize all individual performance measures together and set trade-offs among them. This issue forms another research problem and research objective as follows:

> **Research Problem 2.2** is that *the classification error-rate does not cover all scenarios when it comes to evaluating the optimality of learning algorithms against evolving data streams.* For that, we need to go beyond the classification error-rate to be able to recommend an optimal solution to the user, considering the current situation.
>
> **Research Objective 2.2** is *to define a comprehensive measure which not only combines classification error-rate with adaptation and resource consumption measures* but also *lets us compromise specific individual measures for others*, for ranking and recommending the classification algorithms.

To address this problem, we define the EMR and CAR performance measures in Sections 4.3.2 and 4.3.3, respectively.

The reader should note that we first study the second research problem in Section 4.3, and then introduce the TORNADO framework in Section 4.4. We follow this order because the TORNADO framework uses the EMR and CAR measures for ranking predictive algorithms while they learn from evolving data streams. Since we founded our work on the notion of *multi-strategy* learning, we explain that notion as well as how it contributes to address Research Problem 2.1 in the next section.

4.2 Multi-strategy Learning

We differentiate *multi-strategy learning* from *mono-strategy learning* as well as *decision model combination* in this section.

Mono-strategy learning train a 'single' decision model, e.g., a decision tree or a neural network, for a particular problem. This learning strategy can be very effective and useful if learning problems are narrowly defined (Michalski, 1993). In evolving environments, a mono-strategy learning system may not appropriately learn actual decision boundaries since the distribution of data could evolve.

On the other hand, decision model combination tends to keep the accuracy of learning high by engaging more than a single learning algorithm in the induction process. Ensemble learning and meta-learning are two kinds of decision model combination. The former one exploits variability in the given data and combines multiple copies of a single learning algorithm applied to different subsets of data. Bagging, boosting, and random forests are examples of these methods (Breiman, 1996; Freund et al., 1999; Breiman, 2001). The latter one exploits variability among learning algorithms. Stacking and cascade generalization are examples of meta-learning (Brazdil, 2012). Although in the classical ensemble learning, the classification models are assumed to be trained by a specific learning algorithm, in the hybrid ensemble learning various kinds of learning algorithms participate in decision making (Hsu, 2017). For example, Salgado et al. (2006) applied an ensemble of Artificial Neural Networks and Support Vector Machines to predict daily electricity load. Min and Cho (2011) used Naive Bayes and Support Vector Machines for activity recognition. The reader should note that the final output of combination methods is a predicted label for a new example. The model combination methods are promising for higher accuracy, even in non-stationary environments; however, they rarely consider the two factors of *memory* and *time* while learning from evolving data streams. That is, for instance, a user may want to compromise accuracy for memory usage because affording enough memory may not be possible at the moment. In the evolving environments, memory and runtime must also be considered as decisive measures because their availability may vary as does the distribution of data.

To address this issue, we present a reservoir of (adaptive) learning algorithms, in Section 4.4, where various kinds of predictive models are trained simultaneously, and the optimal one (which is chosen based on the indicators of classification, adaptation and resource consumption) is recommended to the user. We use the '*multi-strategy learning*' terminology to refer to this kind of learning. The reader should note that the primary goal of this approach is to provide an optimal *learner*, i.e., the most efficient learner, based on not only accuracy but also the adaptation and resource consumption measures, for the current situation to the user. Thus, the reader should differentiate multi-strategy learning from ensemble learning. An ensemble algorithm, however, may be used as an individual learner amongst others in a multi-strategy learning setting.

4.3 Performance Measures

We review the performance measures for the evaluation of adaptive learning algorithms in this section. We further define two measures of EMR and CAR which we consider to rank the learning algorithms in a multi-strategy learning setting.

4.3.1 Single-purpose Measures

Classification – We earlier noted that error-rate, or accuracy, is often considered as the defining measure of the classification performance for evaluating learning algorithms in most streaming research works (Gama et al., 2004, 2006; Bifet and Gavalda, 2007; Huang et al., 2015; Baena-Garcıa et al., 2006). Nonetheless, the interplay between error-rate and other factors, such as memory usage and runtime, has received fewer attentions. For example, Bifet et al. (2009) applied the memory, time and accuracy measures separately, to evaluate the performance of ensembles of classifiers. Bifet et al. (2010b) further introduced the RAM-Hour measure, where every RAM-Hour equals to 1 GB of RAM occupied for one hour, to compare the performance of three versions of Perceptron-based Hoeffding Trees.

Drift Detection and Adaptation – Recall that in Chapter 2, we introduced an approach to count the true positive (TP), false positive (FP), and false negative (FN) numbers of drift detection, for the evaluation of drift detectors. We defined the acceptable delay length notion as a threshold that determines how far a detected drift could be from the real location of drift to be considered as a true positive. This approach is originally published in (Pesaranghader and Viktor, 2016). Žliobaitė et al. (2015) introduced the return on investment (ROI) measure to determine whether the *adaptation* of a learning algorithm is beneficial. They concluded that adaptation should take place only if the expected gain in performance, measured by accuracy, exceeds the cost of the resources (e.g., memory and CPU) required for adaptation. Please note that the ROI measure does not comprise the adaption measures, i.e., the drift detection delay, the false positive rate, and the false negative rate, which are critical in many applications. Olorunnimbe et al. (2017) extended the ROI measure to dynamically adapt the number of base learners in online bagging ensembles.

As explained above, the performance measures for evaluating either classification or adaptation have often been used *separately* and *individually* to assess adaptive learning algorithms. To the best of our knowledge, no holistic measure that considers classification, adaptation, and resource consumption together has been developed. Such a measure would allow one to assess a bigger picture regarding the costs and benefits of a particular learning and adaptation strategy. To address this challenge, we introduce the EMR and CAR measures in the following sections.

4.3.2 The EMR Measure

The '*error-rate-only*' evaluation approach is not beneficial in all settings. For example, consider an emergency response setting where the response time, i.e., the time it takes to present a model to the users, may be the most important criterion. That is, users may sacrifice accuracy for speed and partial information. Pocket data mining is another example, which has much application in many areas such as defense and environmental impact assessment (Gaber et al., 2014b). In such applications, the memory resources may be limited, and also reducing the memory footprint is of importance due to connection bottlenecks. As a toy example, assume two classifiers C_1 and C_2 which are used for classification over stream S. Suppose that the model constructed by classifier C_1 has the error-rate of 8.0%, memory usage of 20 MB and runtime of 100 Seconds. On the contrary, the model built by classifier C_2 causes the same degree of error-rate but has the memory usage of 1.5 MB and the runtime of $10,000$ Seconds. It follows that these two classifiers have the same level of error-rate; nevertheless, in a pocket data mining setting, the memory usage of the first classifier may terminate the program when running on a sensor or mobile device (Gaber et al., 2014b). Further, the second classifier may not be suitable for an emergency response setting where the goal is improving the *just-in-time* decision making task.

Based on the foregoing observations, we introduced the Error-Memory-Runtime (EMR) measure in (Pesaranghader et al., 2016) as follows:

$$EMR_C = \frac{w_e \times E_C + w_m \times M_C + w_r \times R_C}{w_e + w_m + w_r} \tag{4.1}$$

where w_e is the error-rate cost weight, E_C is the normalized[1] error-rate of classifier C, w_m is to the memory usage cost weight, M_C is the normalized memory usage of classifier C, w_r is the runtime cost weight, and finally R_C is the normalized runtime of classifier C.

Domain-dependent *cost weights* of w_e, w_m, w_r are associated with the error-rate, memory usage, and runtime to emphasize their importance when we evaluate the classifiers while they process instances of a data stream online. In the real world, the values of the three weights, i.e., w_e, w_m, and w_r, are set in a way to reflect the domain of application. For using Equation (4.1), we have to normalize the values of error-rate, memory usage and runtime. Normalization is separately done for each measure, considering the values obtained for each measure by all classifiers. A high EMR value means that the classifier has resulted in a high error-rate, high memory usage, and/or long runtime. Therefore, a classifier with the lowest EMR is preferred.

The EMR measure can be used to score classifiers in a multi-strategy learning setting. The score of classifier C is calculated by Equation (4.2). The reader should note that the deduction of the lowest EMR from 1 results in the highest score, which is preferred.

1 We used the '*min-max*' scaling to normalize the values of measures for all classifiers.

$$Score_C = 1 - EMR_C \tag{4.2}$$

We used the EMR measure to rank predictive models in a multi-strategy learning setting in (Pesaranghader et al., 2016); however, it does not apply a few key measures in evaluating predictive algorithms against evolving data streams. For example, the EMR measure does not adequately reflect the performance of learning methods in an environment where concept drift is inevitable. Consider medical applications where adaptive algorithms must handle concept drift very rapidly; or autopilot applications where adaptive learning algorithms with many false alarms for concept drift are not considered suitable. In those applications, the EMR measure does not effectively represent the overall performance of adaptive learning algorithms since it does not take the detection delay, as well as the false alarm rate, into consideration. Hence, we need to integrate the performances of the classifiers, their adaptability as well as the resources allocated into a single measure. To address this issue, we defined the CAR measure in (Pesaranghader et al., 2018a).

4.3.3 The CAR Measure

The CAR measure takes not only the classification error-rate, memory usage, and runtime but also the drift detection delay, false positive, and false negative rates into consideration for evaluating the adaptive algorithms in an evolving setting. The CAR measure consists of three core components namely *Classification*, *Adaptation*, and *Resource Consumption*. The classification part holds the error-rate ($\mathrm{E_C}$) of classifier C, the adaptation part keeps the detection delay ($\mathrm{D_D}$), false positive ($\mathrm{FP_D}$), and false negative ($\mathrm{FN_D}$) of drift detector D, and the resource consumption part includes the memory use ($\mathrm{M_{(C,D)}}$) and runtime ($\mathrm{R_{(C,D)}}$) of the (classifier, detector) pair. Note that the symbolic integration of the aforesaid components, using $\diamond$ and $\oplus$ symbols, into the CAR measure is given by Equation (4.3). We provide the mathematical calculation for the CAR measure shortly.

$$\begin{aligned} \mathrm{CAR}_{(C,D)} &:= \mathrm{Classification}_C \diamond \mathrm{Adaptation}_D \diamond \mathrm{Resourse\ Use}_{(C,D)} \\ &:= E_C \diamond (D_D \oplus FP_D \oplus FN_D) \diamond (M_{(C,D)} \oplus R_{(C,D)}) \end{aligned} \tag{4.3}$$

The score associated with the pair (C, D) is obtained from Equation (4.4). The equation implies that a pair with a low CAR has a high score.

$$\mathrm{Score}_{(C,D)} := 1 - \mathrm{CAR}_{(C,D)} \tag{4.4}$$

To compute the CARs of the adaptive algorithms, a matrix containing all the values for the classification, adaptation, and resource consumption measures is created every time an instance is processed. That is, a new matrix is created for a new instance, and the old

matrix is discarded. Assume n classifiers and m drift detectors are available; which means $n \times m$ pairs exist. The measures associated with each pair for the t^{th} instance are placed, row-by-row, into a matrix M^t. This matrix is defined as follows:

$$M^t_{n*m\times 6} = \begin{bmatrix} E^t_{C_1} & D^t_{D_1} & FP^t_{D_1} & FN^t_{D_1} & (M^t_{C_1} + M^t_{D_1}) & (R^t_{C_1} + R^t_{D_1}) \\ E^t_{C_1} & D^t_{D_2} & FP^t_{D_2} & FN^t_{D_2} & (M^t_{C_1} + M^t_{D_2}) & (R^t_{C_1} + R^t_{D_2}) \\ E^t_{C_1} & D^t_{D_3} & FP^t_{D_3} & FN^t_{D_3} & (M^t_{C_1} + M^t_{D_3}) & (R^t_{C_1} + R^t_{D_3}) \\ & & \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & \\ & & \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & \\ E^t_{C_n} & D^t_{D_m} & FP^t_{D_m} & FN^t_{D_m} & (M^t_{C_n} + M^t_{D_m}) & (R^t_{C_n} + R^t_{D_m}) \end{bmatrix} \tag{4.5}$$

Afterwards, the elements of the matrix are normalized, column-by-column, using the '*min-max*' scaling approach. The normalized matrix, i.e., $\overline{M^t}$, is defined in Equation (4.6):

$$\overline{M^t}_{n*m\times 6} = \begin{bmatrix} \overline{E^t_{C_1}} & \overline{D^t_{D_1}} & \overline{FP^t_{D_1}} & \overline{FN^t_{D_1}} & \overline{(M^t_{C_1} + M^t_{D_1})} & \overline{(R^t_{C_1} + R^t_{D_1})} \\ \overline{E^t_{C_1}} & \overline{D^t_{D_2}} & \overline{FP^t_{D_2}} & \overline{FN^t_{D_2}} & \overline{(M^t_{C_1} + M^t_{D_2})} & \overline{(R^t_{C_1} + R^t_{D_2})} \\ \overline{E^t_{C_1}} & \overline{D^t_{D_3}} & \overline{FP^t_{D_3}} & \overline{FN^t_{D_3}} & \overline{(M^t_{C_1} + M^t_{D_3})} & \overline{(R^t_{C_1} + R^t_{D_3})} \\ & & \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & \\ & & \cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & \\ \overline{E^t_{C_n}} & \overline{D^t_{D_m}} & \overline{FP^t_{D_m}} & \overline{FN^t_{D_m}} & \overline{(M^t_{C_n} + M^t_{D_m})} & \overline{(R^t_{C_n} + R^t_{D_m})} \end{bmatrix} \tag{4.6}$$

We associated each measure with a weight for calculating CAR. The weights are placed into a weight vector $\overrightarrow{w}$ as in Equation (4.7). The elements of the vector $\overrightarrow{w}$ are the weights associated in order with the classification error-rate, the detection delay, the false positive rate, the false negative rate, the memory usage, and the runtime. Each weight emphasizes the importance of a particular measure in the evaluation process.

$$\overrightarrow{w}_{6\times 1} = \begin{bmatrix} w_e & w_d & w_{fp} & w_{fn} & w_m & w_r \end{bmatrix}^T \tag{4.7}$$

The CARs and scores for the $n \times m$ pairs are evaluated with Equations (4.8) and (4.9), respectively. Please note that $J_{n*m\times 1}$ is a vector of ones, i.e., all elements are equal to one.

$$\mathrm{CAR}^t_{n*m\times 1} = \frac{\overline{M^t}_{n*m\times 6} \times \overrightarrow{w}_{6\times 1}}{J^T_{6\times 1} \times \overrightarrow{w}_{6\times 1}} \tag{4.8}$$

$$\mathrm{Score}^t_{n*m\times 1} = J_{n*m\times 1} - \mathrm{CAR}^t_{n*m\times 1} \tag{4.9}$$

The index of the recommended classifier at time t, i.e., $Index_{opt}$, is defined by Equation (4.10). The $imax$ is a function that finds the index of the pair presenting the highest score.

$$\mathrm{Index}_{opt} = imax(\mathrm{Score}^t_{n*m\times 1}) \tag{4.10}$$

In summary, the CAR values and scores are calculated as follow:

1. Initialize the matrix M, and the weight vector $\overrightarrow{w}$,
2. *Replace* the elements of matrix M row-by-row (i.e., for every pair) as a new instance arrives,
3. Normalize the elements of matrix M column-by-column using the '*min-max*' scaling,
4. Calculate the CAR measures and the scores of pairs with Equations (4.8) and (4.9),
5. Rank the pairs based on their scores,
6. Recommend the pair with the highest score to the user,
7. Repeat steps 2 to 6.

The reader should recall that the weights are domain dependent. For example, if the memory resources are limited, like in the pocket data mining applications (Gaber et al., 2014b), the weight w_m should be set to a higher value. On the other hand, if memory is abundant, but accuracy and speed of model construction are important, the w_m value may be decreased (or even set to zero). In medical applications, where a rapid reaction to concept drift is critical, w_d may be the dominant weight.

As mentioned earlier in Section 4.3.2, the EMR measure does not adequately reflect the overall performance of classification models in an evolving environment where concept drift happens, whereas, the CAR measure is capable of combining classification, adaptation, and resource consumption measures together to estimate the performance of classifiers in a multi-strategy learning setting. Hence, we consider the CAR measure to rank the predictive models in our experiments in Section 4.5.

4.4 TORNADO: Reservoir of Multi-Strategy Learning

We explain the TORNADO framework for multi-strategy learning, as well as mining evolving data streams, in this section. The framework is implemented in Python, and its source code is publicly available[2]. As mentioned earlier in this chapter, we introduced this framework for executing a number of *distinct* (classifier, drift detector) pairs in *parallel*[3] and offering the optimal pair to the user over time, against data streams.

As presented in Fig. 4.2, the main components of the framework are STREAM READER, CLASSIFIERS, DETECTORS, CLASSIFIER-DETECTOR PAIRS, and CAR CALCULATOR. The input is constituted of a Stream, a list of (classifier, detector) pairs, and a weight vector. The reader should note that C_n and D_m represent the n^{th} classifier and the m^{th} detector, respectively. The TORNADO framework follows the *prequential* approach where instances are first tested and then used for training (Gama et al., 2013, 2014).

2 The TORNADO Framework: `https://github.com/alipsgh/tornado`
3 In other words, the (classifier, drift detector) pairs are synched per instance.

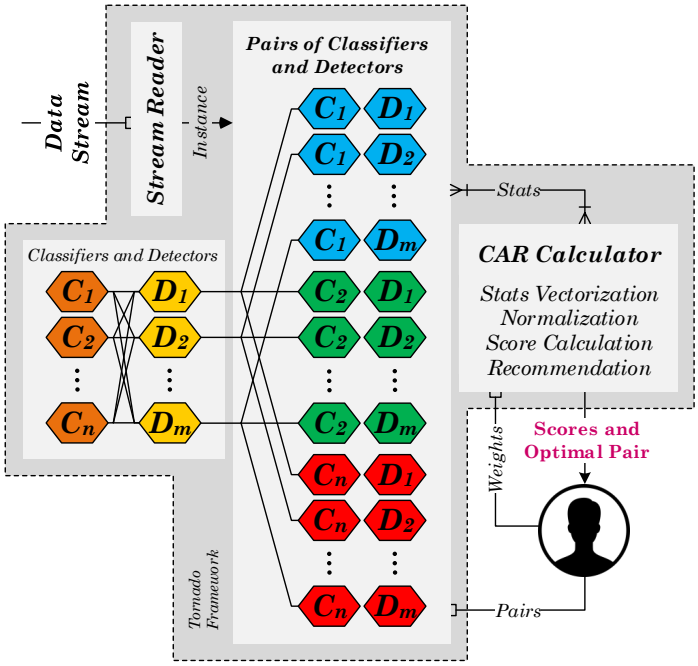


Fig. 4.2. The TORNADO Framework

The (classifier, detector) pairs are constructed as shown in Fig. 4.2, prior to the learning process. The STREAM READER reads instances from the stream and sends them one-by-one to the (classifier, detector) pairs for model construction. Each learner incrementally and prequentially builds a model. That is, each instance is first used for testing and then for training. Simultaneously, CLASSIFIERS send their statistics, e.g., error-rates or the current prediction results, to their corresponding DRIFT DETECTORS to detect concept drift. Then, the CAR CALCULATOR determines the score of each (classifier, detector) pair by considering the classification error-rate, detection delay, detection false positive, detection false negative, total memory usage, and runtime. Eventually, the model with the highest score is presented to the user. The model with the highest score may change as a result of incremental learning and the concept drift phenomenon. This process continues until either a predefined condition is met or all instances are processed.

Fig. 4.3 illustrates that, while the various pairs are executed concurrently, the one with the highest score is recommended at each time interval. Recall that an interval is a time period between two consecutive concept drifts. As illustrated in the figure, during the interval τ_0 to τ_n, the pair (C_1, D_3) has the highest score. Suddenly, at time τ_n, the data distribution is altered resulting in the pair (C_3, D_2) being recommended to the user. In this illustrative example, another drift occurs at τ_{2n} resulting in the pair (C_2, D_1) having the highest score.

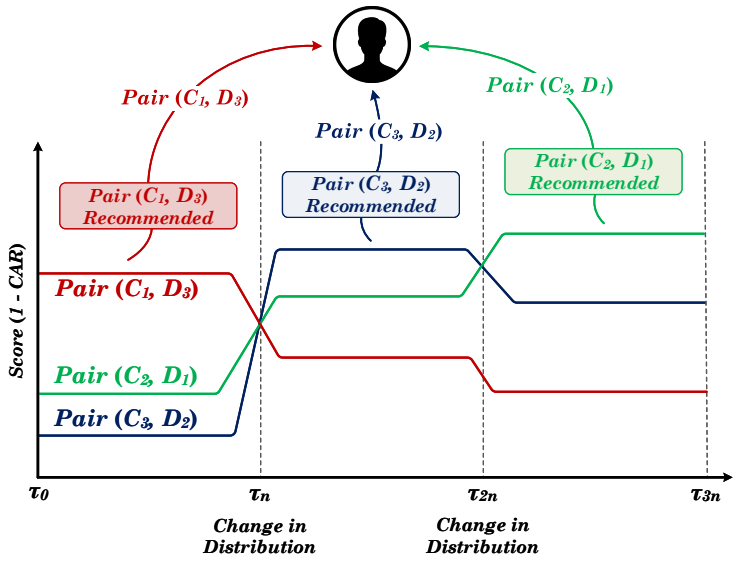


Fig. 4.3. Recommendation of (Classifier, Detector) Pairs over Time

The following observation is noteworthy. The continuous outputs of our TORNADO framework are the currently best performing (classifier, detector) pair, which may change over time. Therefore, our work should not be confused with a hybrid ensemble of classifiers setting (Hsu, 2017; Min and Cho, 2011; Verikas et al., 2010; Salgado et al., 2006). Typically, a hybrid ensemble contains a number of diverse classifiers that form a committee, which aims at increasing the prediction accuracy by utilizing the diversity of the members of the ensemble. In contrast, within the TORNADO framework, the individual learners proceed independently to construct their models. The rationale behind our framework is that we aim to utilize diverse learning strategies that potentially behave in evolving environments differently.

Available Classifiers and Detectors – As of today, Naive Bayes (NB), Decision Stump (DS), Hoeffding Tree (HT) (Domingos and Hulten, 2000), Perceptron (PR) (Bifet et al., 2010b), and K-Nearest Neighbours (K-NN) are available as (online) learning algorithms in TORNADO. Also, we developed various drift detection methods including Cumulative Sum (CUSUM) and its Page-Hinkley (PH) variant (Page, 1954), Drift Detection Method (DDM) (Gama et al., 2004), Early Drift Detection Method (EDDM) (Baena-Garcıa et al., 2006), Reactive Drift Detection Method (RDDM) (Barros et al., 2017), Hoeffding's bound based Drift Detection Methods ($\text{HDDM}_{\text{A-test}}$ and $\text{HDDM}_{\text{W-test}}$) (Frías-Blanco et al., 2015), Adaptive Windowing (ADWIN) (Bifet and Gavalda, 2007), SeqDrift2 (Pears et al., 2014), Fast Hoeffding Drift Detection Method (FHDDM) (Pesaranghader and Viktor, 2016), Stacking Fast Hoeffding Drift Detection Methods (FHDDMS) (Pesaranghader et al., 2018a), and McDiarmid Drift Detection Methods (Pesaranghader et al., 2018b).

4.5 Experimental Study of TORNADO Framework

The experimental results for the TORNADO framework against the synthetic and semi-real-world data streams are presented in this section. Five incremental classifiers were employed, namely Naive Bayes (NB), Decision Stump (DS), Hoeffding Tree (HT), Perceptron (PR), and 5 Nearest Neighbours (5-NN). Please note that preliminary experiments indicated that $K = 5$ for the K-NN learner achieved higher accuracy results and reasonable runtimes than the other values of K. Furthermore, 15 drift detectors were applied, namely FHDDM_{25}, FHDDM_{100}, $\text{FHDDMS}_{\text{add}}$, FHDDMS, MDDM-$\text{G}_{25}$, MDDM-$\text{G}_{100}$, CUSUM, PageHinkley, DDM, EDDM, RDDM, ADWIN, SeqDrift2, $\text{HDDM}_{\text{A-test}}$, and $\text{HDDM}_{\text{W-test}}$[4]. As a result, we have a total of 75 pairs of (classifier, detector) for our experiments. The scores associated with the (classifier, detector) pairs are calculated by Equation (4.9), using the CAR measure, while instances are prequentially processed over time. The experimental results and the recommended (classifier, detector) pairs against various data streams are presented in this section. The reader should consider that the main goal of our experimental study is the demonstration of how TORNADO works for a multi-strategy setting.

Diagram Interpretation – For representing recommended pairs over time, we created a diagram which illustrates a recommended pair by a specific colour in a foursquare form of time unit; for example, as illustrated in Fig. 4.5. In our diagrams, the timeline begins at the top-left corner and is then unfolded line by line, from left to right. Note that the pairs that use members of FHDDM and FHDDMS as drift detectors are indicated in shades of *purple* and *blue*, respectively, while the pairs of the MDDM variants are displayed in shades of *teal*. The other drift detector pairs are displayed in shades of *gold* (for CUSUM), *yellow* (for PageHinkley), *orange* (for DDM, EDDM, and RDDM), *red* (for ADWIN), *pink* (for SeqDrift2), and *green* (for the HDDM variants). The full pair-color map list for 75 pairs of (classifier, detector) is available in Fig. 4.4. The reader should note that we mark the locations of concept drifts by the symbol in the figures.

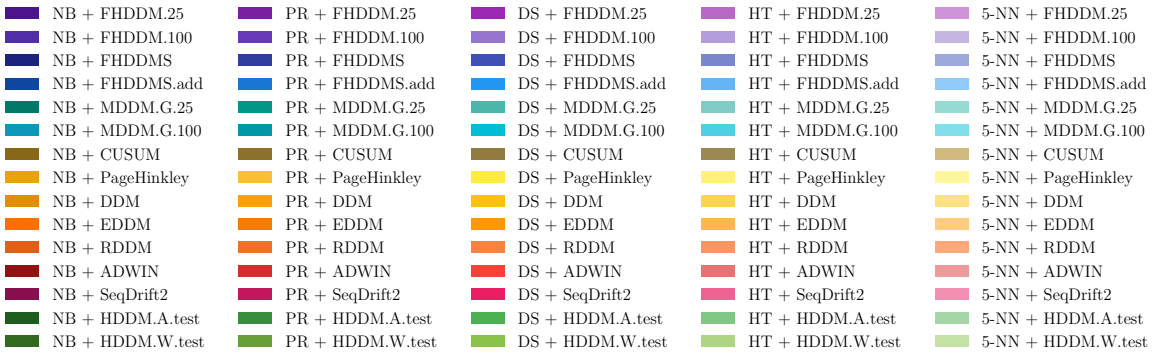


Fig. 4.4. Pair-Color Map

4 We considered MDDM-G as the representative of the MDDM variants for our experiments.

4.5.1 Synthetic Data Streams

This subsection summarizes our experiment for the synthetic data streams, for which all the (classifier, detector) pairs are run in parallel, and are ranked over time, using the TORNADO framework. We first present the top 20 pairs (out of 75 pairs), ranked by the highest *average* scores, in Tables 4.1 and 4.2. In addition, complementary information, e.g., averaged error-rates and averaged drift detection delays, are available in Tables B.9 to B.16. Then, we show the recommended (classifier, detector) pairs in Figs. 4.5 to 4.8.

4.5.1.1 Top 20 (Classifier, Detector) Pairs

We list the top 20 pairs, with the highest *average* scores, for the SINE1, SINE2, MIXED, and STAGGER data streams in Table 4.1, and for the CIRCLES and LED data streams in Table 4.2. In this set of experimental examples, the weight vector $\overrightarrow{w}$ is a vector of one, i.e., $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]^T$, for SINE1, SINE2, MIXED, and STAGGER as well as CIRCLES and LED (as shown in Tables 4.1 and 4.2). We further considered other weight vectors, in which the elements are not necessarily identical, for CIRCLES and LED (as shown in Table 4.2). We conclude our observation as follows.

Table 4.1 shows that Naive Bayes and Perceptron paired with FHDDM, FHDDMS, MDDM, and HDDM variants are often ranked among the top 10 pairs by obtaining the highest average scores. That is due to the fact that they resulted in lower classification error-rates, shorter drift detection delays, and reasonable resource usage than other pairs.

Table 4.1. Top 20 Pairs with Highest Average Scores against Data Streams with Abrupt Concept Drift

$\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$							
SINE1		SINE2		MIXED		STAGGER	
Pair	Score	Pair	Score	Pair	Score	Pair	Score
PR + MDDM.G.25	0.919	NB + HDDM.W.test	0.902	NB + FHDDMS	0.905	NB + HDDM.W.test	0.871
PR + FHDDM.25	0.919	NB + MDDM.G.25	0.901	NB + FHDDM.25	0.903	NB + MDDM.G.25	0.871
PR + FHDDMS	0.919	NB + FHDDMS	0.901	NB + MDDM.G.25	0.903	PR + HDDM.W.test	0.871
PR + HDDM.W.test	0.918	NB + MDDM.G.100	0.899	NB + MDDM.G.100	0.903	PR + MDDM.G.25	0.871
PR + HDDM.A.test	0.917	NB + FHDDM.25	0.897	NB + FHDDM.100	0.902	PR + FHDDM.25	0.869
PR + MDDM.G.100	0.914	NB + FHDDM.100	0.897	NB + FHDDMS.add	0.901	NB + FHDDMS	0.869
PR + FHDDM.100	0.912	NB + FHDDMS.add	0.892	NB + HDDM.W.test	0.897	PR + FHDDMS	0.868
PR + FHDDMS.add	0.910	NB + ADWIN	0.880	PR + FHDDMS.add	0.884	NB + FHDDM.25	0.866
NB + HDDM.A.test	0.901	NB + CUSUM	0.879	PR + FHDDM.25	0.883	NB + MDDM.G.100	0.863
NB + FHDDMS	0.899	NB + HDDM.A.test	0.875	PR + FHDDM.100	0.880	NB + FHDDM.100	0.861
NB + FHDDM.25	0.895	NB + RDDM	0.858	PR + FHDDMS	0.879	PR + FHDDM.100	0.861
NB + MDDM.G.100	0.893	5-NN + MDDM.G.25	0.827	PR + MDDM.G.25	0.877	PR + MDDM.G.100	0.861
NB + HDDM.W.test	0.893	NB + DDM	0.819	NB + HDDM.A.test	0.875	NB + FHDDMS.add	0.859
NB + MDDM.G.25	0.892	5-NN + FHDDMS.add	0.812	PR + MDDM.G.100	0.875	NB + HDDM.A.test	0.857
NB + FHDDM.100	0.892	5-NN + FHDDM.25	0.811	NB + CUSUM	0.874	PR + FHDDMS.add	0.857
NB + FHDDMS.add	0.891	5-NN + ADWIN	0.810	PR + HDDM.W.test	0.874	NB + ADWIN	0.853
PR + CUSUM	0.891	5-NN + MDDM.G.100	0.806	PR + ADWIN	0.864	PR + ADWIN	0.852
NB + ADWIN	0.878	5-NN + FHDDM.100	0.796	PR + CUSUM	0.863	PR + HDDM.A.test	0.851
PR + RDDM	0.872	5-NN + HDDM.W.test	0.792	NB + DDM	0.836	HT + MDDM.G.25	0.839
NB + CUSUM	0.871	HT + FHDDM.25	0.792	5-NN + FHDDMS	0.832	HT + FHDDM.25	0.839

For the CIRCLES data stream, as represented in Table 4.2, we observe Naive Bayes (NB) and Hoeffding Tree (HT) paired with FHDDM, MDDM, and HDDM variants are among the top 20 pairs by considering the average scores. In fact, Perceptron is not an optimal choice for classification since the decision boundary for CIRCLES is not linear (as illustrated in Fig. 2.9). Interestingly, the pairs of Hoeffding Tree (HT) move up when we use the weight vector of $\overrightarrow{w} = [2\ 4\ 3\ 1\ 0.5\ 0]^T$. In fact, by applying that weight vector, we emphasize the classification error-rate, drift detection delay, and false positive number, while resource consumption measures are ignored, for the (classifier, pair) recommendation task. As a result, the pairs of Hoeffding Tree move up as their abundant resource consumptions are ignored by setting w_m and w_r to 0.5 and 0, respectively.

Regarding the LED data stream, we again experience that the pairs of the Perceptron and Naive Bayes classifiers with FHDDM, MDDM, and HDDM are among the top 20 pairs, as represented in Table 4.2. In this experiment, we observed that the pairs of Perceptron commonly outperform Naive Bayes when $\overrightarrow{w} = [1\ 1\ 2\ 3\ 4\ 4]^T$. As a consequence of assigning higher weights to memory usage and runtime, Perceptron pairs become dominant for having faster runtime and less memory usage.

Table 4.2. Top 20 Pairs with Highest Average Scores against CIRCLES and LED Data Streams with Gradual Concept Drift

CIRCLES				LED			
$\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$		$\vec{w} = [2\ 4\ 3\ 1\ 0.5\ 0]$		$\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$		$\vec{w} = [1\ 1\ 2\ 3\ 4\ 4]$	
Pair	Score	Pair	Score	Pair	Score	Pair	Score
NB + FHDDM.100	0.894	NB + FHDDMS	0.861	PR + FHDDM.100	0.880	PR + FHDDM.100	0.902
NB + FHDDMS	0.893	NB + FHDDM.100	0.861	PR + FHDDMS	0.879	PR + FHDDMS	0.902
NB + FHDDMS.add	0.889	HT + HDDM.W.test	0.851	PR + MDDM.G.100	0.876	PR + HDDM.A.test	0.900
NB + HDDM.W.test	0.882	HT + MDDM.G.100	0.850	NB + FHDDM.100	0.875	PR + MDDM.G.100	0.900
NB + MDDM.G.100	0.880	HT + FHDDMS	0.849	NB + MDDM.G.100	0.875	PR + FHDDMS.add	0.899
NB + CUSUM	0.876	NB + FHDDMS.add	0.849	NB + FHDDMS	0.875	PR + CUSUM	0.897
NB + HDDM.A.test	0.876	HT + FHDDM.100	0.848	PR + HDDM.A.test	0.873	PR + HDDM.W.test	0.895
NB + MDDM.G.25	0.871	HT + FHDDMS.add	0.842	NB + FHDDMS.add	0.871	PR + MDDM.G.25	0.889
NB + FHDDM.25	0.865	NB + HDDM.W.test	0.836	PR + FHDDMS.add	0.871	PR + DDM	0.887
NB + ADWIN	0.862	NB + MDDM.G.100	0.835	PR + CUSUM	0.865	NB + FHDDM.100	0.874
NB + RDDM	0.855	HT + MDDM.G.25	0.832	PR + HDDM.W.test	0.863	PR + RDDM	0.872
NB + DDM	0.846	HT + HDDM.A.test	0.828	NB + CUSUM	0.856	NB + FHDDMS	0.872
NB + SeqDrift2	0.829	HT + FHDDM.25	0.828	NB + HDDM.A.test	0.856	NB + MDDM.G.100	0.872
HT + HDDM.W.test	0.824	NB + HDDM.A.test	0.822	HT + MDDM.G.100	0.854	NB + FHDDMS.add	0.871
HT + FHDDM.100	0.824	NB + CUSUM	0.819	HT + FHDDM.100	0.854	PR + FHDDM.25	0.870
HT + FHDDMS	0.824	HT + CUSUM	0.811	NB + HDDM.W.test	0.854	NB + HDDM.A.test	0.865
HT + MDDM.G.100	0.823	NB + MDDM.G.25	0.810	HT + FHDDMS	0.854	NB + CUSUM	0.865
HT + FHDDMS.add	0.822	HT + ADWIN	0.808	HT + FHDDMS.add	0.851	NB + HDDM.W.test	0.863
DS + CUSUM	0.816	NB + FHDDM.25	0.804	NB + MDDM.G.25	0.850	NB + MDDM.G.25	0.862
HT + MDDM.G.25	0.816	NB + ADWIN	0.802	PR + MDDM.G.25	0.849	NB + DDM	0.859

In summary, we saw the pairs of Naive Bayes and Perceptron with the FHDDM, FHDDMS, MDDM, and HDDM detectors were among the top 20 pairs for being accurate for both classification and drift detection. Nonetheless, in the case of the CIRCLES data stream, Perceptron did not perform accurately since the classification boundary is not linearly separable. We provide graphical illustrations of the (classifier, detector) pair recommendation in the next section.

4.5.1.2 Illustration of Pair Recommendation

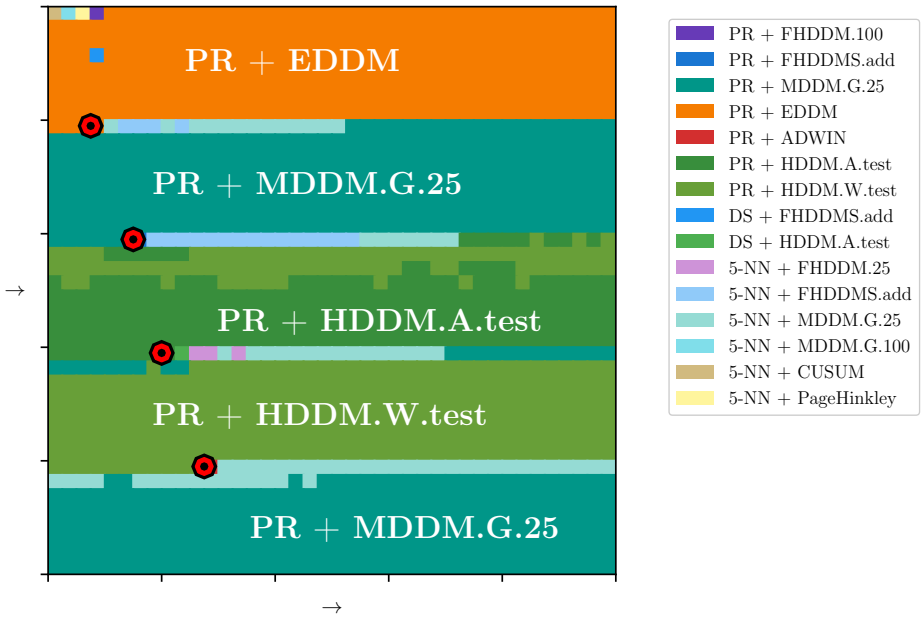
We illustrate the (classifier, detector) pairs recommended for our experiments against the synthetic data streams in the last section by the TORNADO framework over time in Figs. 4.5 to 4.8. Recall that we set all the wights to 1, i.e., $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]^T$, with which we assumed that corresponding quantities of measures are of equal importance. We also had experiments with weight vectors of $\vec{w} = [2\ 4\ 3\ 1\ 0.5\ 0]^T$ and $\vec{w} = [1\ 1\ 2\ 3\ 4\ 4]^T$ for the CIRCLES and LED data streams, respectively. The reader should recall that Tables 4.1 and 4.2 held the top 20 pairs with the highest average scores for the data streams. We discuss our findings by considering the illustrations, i.e., Figs. 4.5 to 4.8, as follows.

The recommended (classifier, detector) pairs against the SINE1, SINE2, MIXED, and STAGGER are shown in Figs. 4.5 and 4.6. The reader may recall that we mark the locations of concept drifts by the symbol in the figures. As illustrated in the figures, PR + EDDM, NB + EDDM, NB + PageHinkley, and PR + PageHinkley are the recommended pairs prior to the first concept drift. This observation is due to the fact that EDDM and PageHinkley often have faster runtime and lower memory usage compared to the other methods; consequently, their pairs are offered. On the other hand, the pairs of Naive Bayes (NB) and Perceptron (PR) with MDDM and HDDM variants become superior after the first drift. Indeed, the MDDM and HDDM variants, of course, paired with NB and PR, are able to detect subsequent concept drifts with shorter delays, fewer false positives, and fewer false negatives. Therefore, NB + MDDMs and NB + HDDMs are often recommended after the first context.

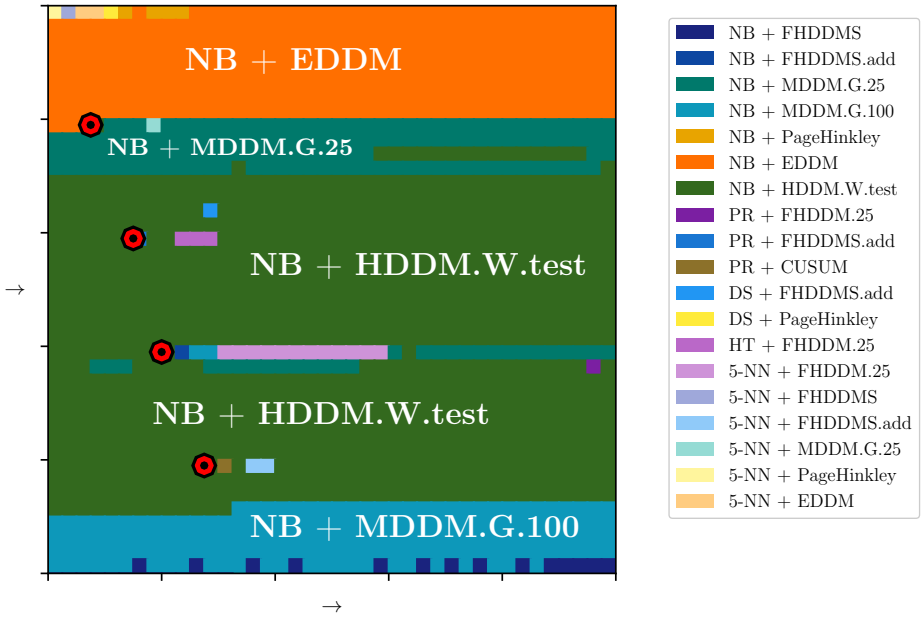
Figs. 4.7 and 4.8 illustrate the recommended pairs against the CIRCLES and LED data streams. In these figures, we also represent the impact of the weight vector in the figures. For CIRCLES, PR + $\text{HDDM}_{\text{W-test}}$ is often offered to the user when $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]^T$. In contrast, when $\overrightarrow{w} = [2\ 4\ 3\ 1\ 0.5\ 0]^T$, the NB + MDDM-G_{100} pair generally outperforms others. That is, the classification error-rate, drift detection delay, and false positive rate are assumed to be more important, and the resulting best pair reflects this change. Although Hoeffding Tree typically imposes higher memory usage and runtime than Naive Bayes, HT + MDDM-G_{100} and HT + $\text{HDDM}_{\text{W-test}}$ are also recommended since the weights of memory usage and runtime, i.e., w_m and w_r, were set to 0.5 and 0, respectively. As to the LED data stream, the NB + MDDM-G_{100} pair is recommended when $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]^T$, whereas, PR + MDDM-G_{100} is preferred over others when $\vec{w} = [1\ 1\ 2\ 3\ 4\ 4]^T$. In fact, as w_m and w_r have more contributions in the final score, the Perceptron classifier becomes a suitable choice for being lighter and faster than the Naive Bayes classifier.

Conclusion – The illustrations confirm that no pair outperforms others; and that no drift detector or classifier dominate. The pairs with the FHDDM, MDDM, and HDDM drift detectors were frequently recommended for having higher scores. Further, as shown in the figures, the Naive Bayes (NB) and Perceptron (PR) classifiers were often preferred, since they were light, fast, and accurate, particularly when the weights were equal.

Pair Recommendation over Time against
Sine1 and Sine2 Data Stream with Abrupt Concept Drift
(*from top-left corner to bottom-right corner*)



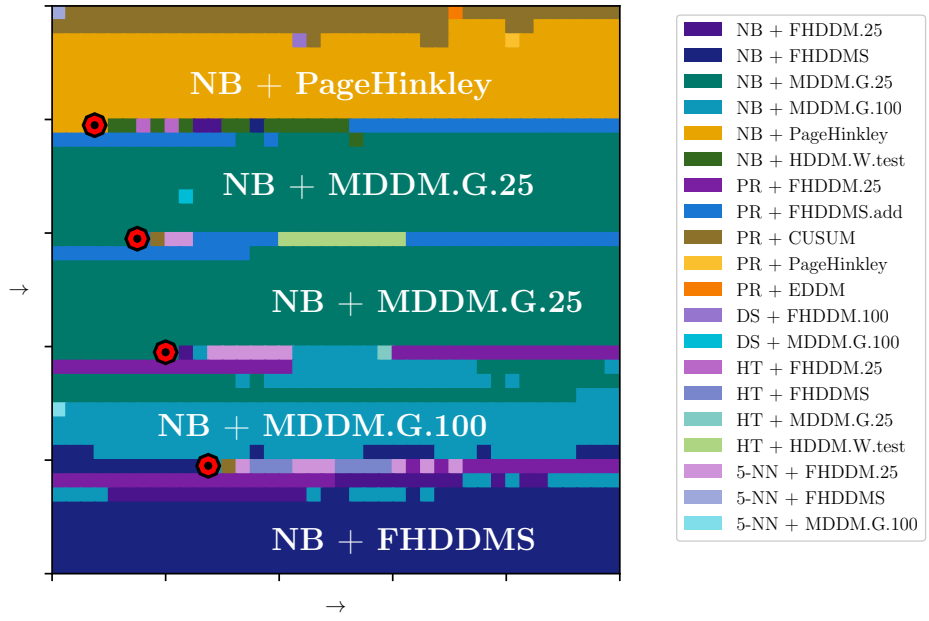
(a) The Sine1 Data Stream



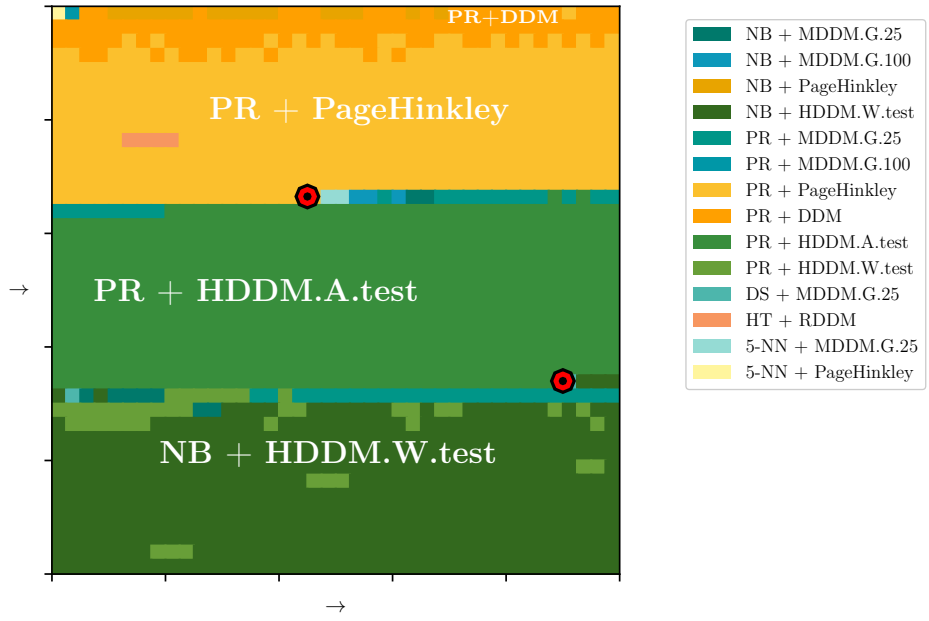
(b) The Sine2 Data Stream

Fig. 4.5. Recommended Classifier+Detector Pairs out of 75 Pairs against Sine1 and Sine2 Data Streams with Abrupt Concept Drifts

Pair Recommendation over Time against
Mixed and Stagger Data Stream with Abrupt Concept Drift
(*from top-left corner to bottom-right corner*)



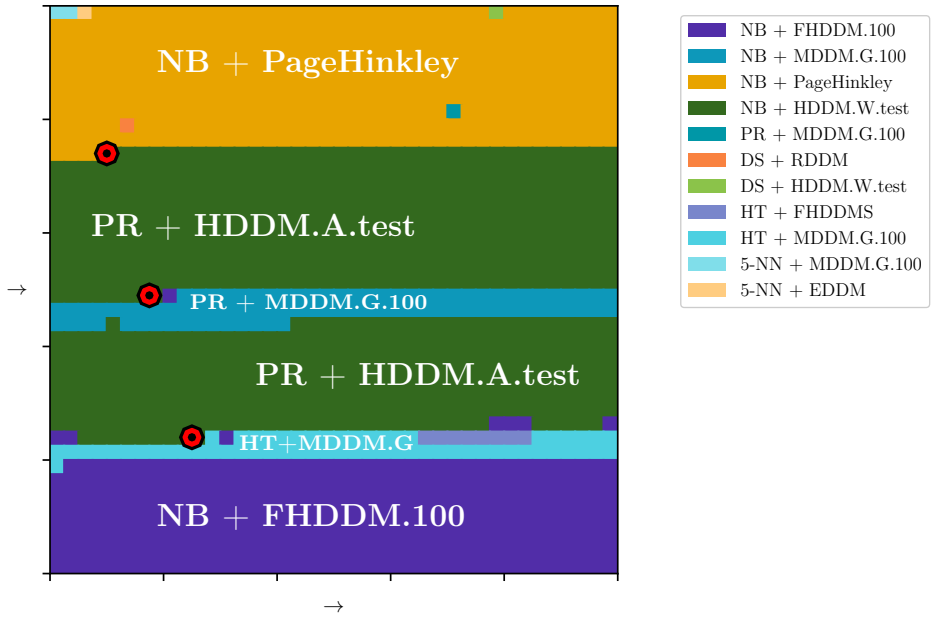
(a) The Mixed Data Stream



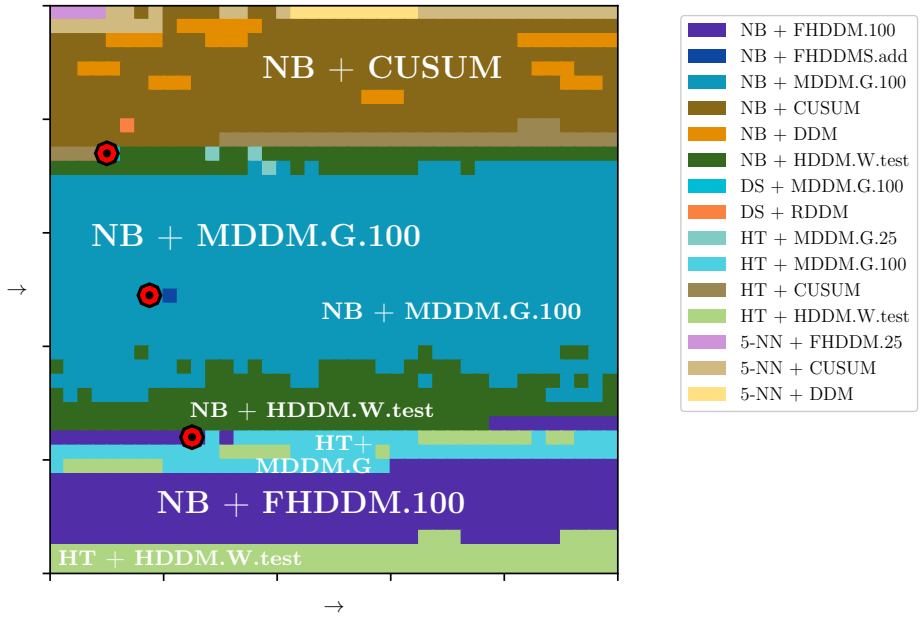
(b) The Stagger Data Stream

Fig. 4.6. Recommended Classifier+Detector Pairs out of 75 Pairs against Mixed and Stagger Data Streams with Abrupt Concept Drifts

Pair Recommendation over Time against
CIRCLES Data Stream with Gradual Concept Drift
(*from top-left corner to bottom-right corner*)

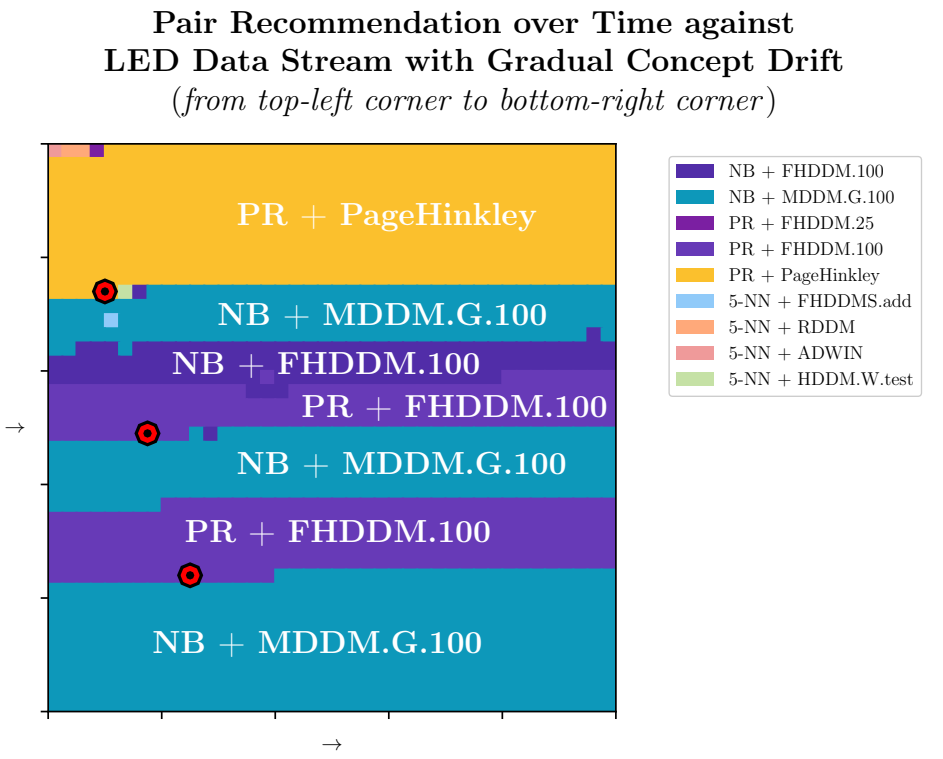


(a) The CIRCLES Data Stream and $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$

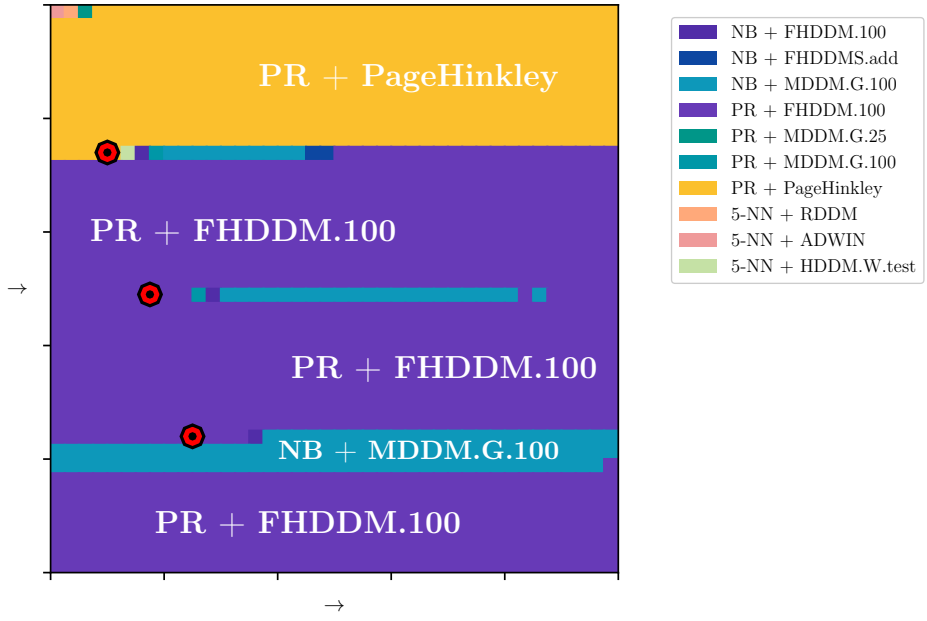


(b) The CIRCLES Data Stream and $\vec{w} = [2\ 4\ 3\ 1\ 0.5\ 0]$

Fig. 4.7. Recommended Classifier+Detector Pairs out of 75 Pairs against CIRCLES Data Stream with Gradual Concept Drifts



(a) The LED Data Stream and $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$



(b) The LED Data Stream and $\vec{w} = [1\ 1\ 2\ 3\ 4\ 4]$

Fig. 4.8. Recommended Classifier+Detector Pairs out of 75 Pairs against LED Data Stream with Gradual Concept Drifts

4.5.2 Semi-real-world Data Streams

Our experimental results for the semi-real-world data streams are reported in Table 4.3 and Figs. 4.9 to 4.11. Table 4.3 lists the top 20 pairs with the highest average scores against the data streams, while Figs. 4.9 to 4.11 illustrate the recommended pairs to the user. We also provide complementary information, e.g., averaged error-rates, in Tables B.17 to B.22.

4.5.2.1 Top 20 (Classifier, Detector) Pairs

As mentioned above, Table 4.3 represents the top 20 pairs with the highest average scores against the ADULT, NURSERY, and SHUTTLE semi-real-world streams. Our observation for each data stream is provided below:

- **ADULT** – The Naive Bayes (NB) and Decision Stump (DS) classifiers may be considered optimal for this data stream for both weight vectors. Further, the pairs of FHDDM, MDDM, and HDDM drift detectors with Naive Bayes are among the top 10 pairs. The reader may notice that the rank of pairs may change when the weight vector $\vec{w} = [2\ 1.5\ 3\ 2\ 1.5\ 1]^T$ is used. In fact, in this case, the pairs with low classification error-rates as well as fewer false positives and false negatives are preferred. (Please note that the recommended pairs are illustrated in Fig. 4.9.)

- **NURSERY** – The pairs of FHDDM, MDDM, and HDDM with Naive Bayes are ranked on top; followed by their Perceptron pairs when $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$. On the other hand, when $\vec{w} = [3\ 2\ 2\ 1\ 0\ 0.5]$, the Hoeffding Tree (HT) pairs with the variants of FHDDM, MDDM, and HDDM become the preferred choices. Although Hoeffding Tree is expensive regarding memory usage and runtime, that experience is due to the less importance of the memory usage and runtime measures in this example, as their corresponding weights, i.e., w_m and w_r, are set to 0 and 0.5. (The reader may refer to Fig. 4.10 for the illustration of the recommended pairs.)

- **SHUTTLE** – The pairs of Decision Stump and Naive Bayes with ADWIN, FHDDM, MDDM, and HDDM are among the outperforming pairs when $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$. The ranking of pairs may undergo a change when we use $\vec{w} = [3\ 2\ 1\ 2\ 1\ 1]$. The latter puts more stress on the classification error-rate, drift detection delay, and false negative rate. The pairs mentioned above are recommended, since they had lower error-rate, and shorter drift detection delay and fewer false negatives when compared to others. (Fig. 4.11 is the illustration of the pair recommendation for this experiment.)

In summary, Naive Bayes was often considered as an optimal classifier when it was paired with the FHDDM, MDDM, and HDDM drift detectors. Our experiments justify the importance of the weights of individual measures for ranking the pairs before offering the optimal one to the user.

Table 4.3. Top 20 Pairs with Highest Average Scores against CIRCLES and LED Data Streams with Gradual Concept Drift

ADULT				NURSERY				SHUTTLE			
$\vec{w}$ = [1 1 1 1 1 1]		$\vec{w}$ = [2 1.5 3 2 1.5 1]		$\vec{w}$ = [1 1 1 1 1 1]		$\vec{w}$ = [3 2 2 1 0 0.5]		$\vec{w}$ = [1 1 1 1 1 1]		$\vec{w}$ = [3 2 1 2 1 1]	
Pair	Score	Pair	Score	Pair	Score	Pair	Score	Pair	Score	Pair	Score
NB + MDDM.G.100	0.879	NB + FHDDMS.add	0.887	NB + HDDM.W.test	0.885	HT + HDDM.A.test	0.892	DS + ADWIN	0.863	NB + ADWIN	0.849
NB + HDDM.W.test	0.876	NB + MDDM.G.100	0.886	NB + FHDDM.100	0.884	HT + MDDM.G.100	0.876	DS + FHDDM.100	0.861	DS + ADWIN	0.847
NB + FHDDMS.add	0.876	NB + FHDDM.25	0.885	NB + HDDM.A.test	0.884	HT + FHDDM.100	0.875	DS + MDDM.G.100	0.861	DS + MDDM.G.100	0.843
NB + FHDDM.100	0.872	NB + MDDM.G.25	0.883	NB + FHDDMS	0.883	HT + FHDDMS	0.875	DS + FHDDMS	0.861	5-NN + ADWIN	0.840
NB + FHDDMS	0.872	NB + CUSUM	0.883	NB + MDDM.G.100	0.883	HT + HDDM.W.test	0.875	DS + FHDDMS.add	0.859	DS + FHDDM.100	0.839
NB + MDDM.G.25	0.871	NB + HDDM.A.test	0.883	NB + ADWIN	0.883	HT + RDDM	0.874	NB + ADWIN	0.859	DS + FHDDMS	0.838
NB + FHDDM.25	0.871	NB + ADWIN	0.879	NB + FHDDMS.add	0.882	HT + FHDDMS.add	0.873	DS + HDDM.A.test	0.858	NB + MDDM.G.100	0.838
NB + HDDM.A.test	0.869	NB + HDDM.W.test	0.879	PR + MDDM.G.100	0.879	HT + SeqDrift2	0.870	NB + FHDDM.100	0.858	NB + FHDDMS	0.838
NB + ADWIN	0.869	NB + FHDDM.100	0.877	PR + FHDDM.100	0.878	HT + FHDDM.25	0.868	NB + FHDDMS	0.858	NB + HDDM.W.test	0.837
NB + CUSUM	0.868	NB + FHDDMS	0.877	PR + FHDDMS	0.877	HT + MDDM.G.25	0.867	DS + HDDM.W.test	0.858	DS + HDDM.A.test	0.837
NB + RDDM	0.854	NB + RDDM	0.872	PR + FHDDMS.add	0.877	HT + CUSUM	0.867	NB + MDDM.G.100	0.857	NB + FHDDM.100	0.836
NB + SeqDrift2	0.852	NB + DDM	0.869	NB + CUSUM	0.876	NB + ADWIN	0.867	NB + HDDM.W.test	0.856	DS + FHDDMS.add	0.836
NB + DDM	0.850	NB + SeqDrift2	0.864	PR + HDDM.A.test	0.876	NB + HDDM.W.test	0.865	DS + MDDM.G.25	0.856	DS + HDDM.W.test	0.836
NB + PageHinkley	0.840	NB + PageHinkley	0.861	NB + MDDM.G.25	0.874	HT + DDM	0.864	DS + CUSUM	0.856	NB + MDDM.G.25	0.835
DS + CUSUM	0.836	DS + CUSUM	0.849	PR + HDDM.W.test	0.874	NB + HDDM.A.test	0.862	NB + FHDDMS.add	0.855	NB + FHDDMS.add	0.834
DS + FHDDM.100	0.835	DS + FHDDMS.add	0.843	PR + MDDM.G.25	0.874	NB + MDDM.G.100	0.861	NB + MDDM.G.25	0.854	DS + MDDM.G.25	0.834
DS + FHDDMS.add	0.833	DS + FHDDM.100	0.839	PR + FHDDM.25	0.872	NB + FHDDM.100	0.861	NB + HDDM.A.test	0.853	DS + CUSUM	0.833
DS + MDDM.G.100	0.830	DS + HDDM.A.test	0.831	NB + DDM	0.871	NB + FHDDMS	0.860	NB + CUSUM	0.852	DS + RDDM	0.831
DS + FHDDM.25	0.829	DS + ADWIN	0.827	NB + FHDDM.25	0.871	NB + FHDDMS.add	0.858	DS + RDDM	0.851	NB + HDDM.A.test	0.830
DS + HDDM.A.test	0.824	DS + MDDM.G.100	0.827	PR + CUSUM	0.869	NB + SeqDrift2	0.856	DS + FHDDM.25	0.849	NB + RDDM	0.830

4.5.2.2 Illustration of Pair Recommendation

Figs. 4.9 to 4.11 illustrate the (classifier, detector) pairs recommended to the user against the ADULT, NURSERY, and STAGGER data streams, respectively. Recall that the pairs with the variants of FHDDM and FHDDMS as drift detectors are indicated in shades of *purple* and *blue* in our figures, respectively. Further, the pairs of the MDDM variants are displayed in shades of *teal*. The other drift detector pairs are displayed in shades of *gold* (for CUSUM), *yellow* (for PageHinkley), *orange* (for DDM, EDDM, and RDDM), *red* (for ADWIN), *pink* (for SeqDrift2), and *green* (for the HDDM variants). The full pair-color map list for the 75 pairs of (classifier, detector) is available in Fig. 4.4. We also indicate the locations of concept drifts by the symbol in the figures. We explain our observations for the data streams as follows.

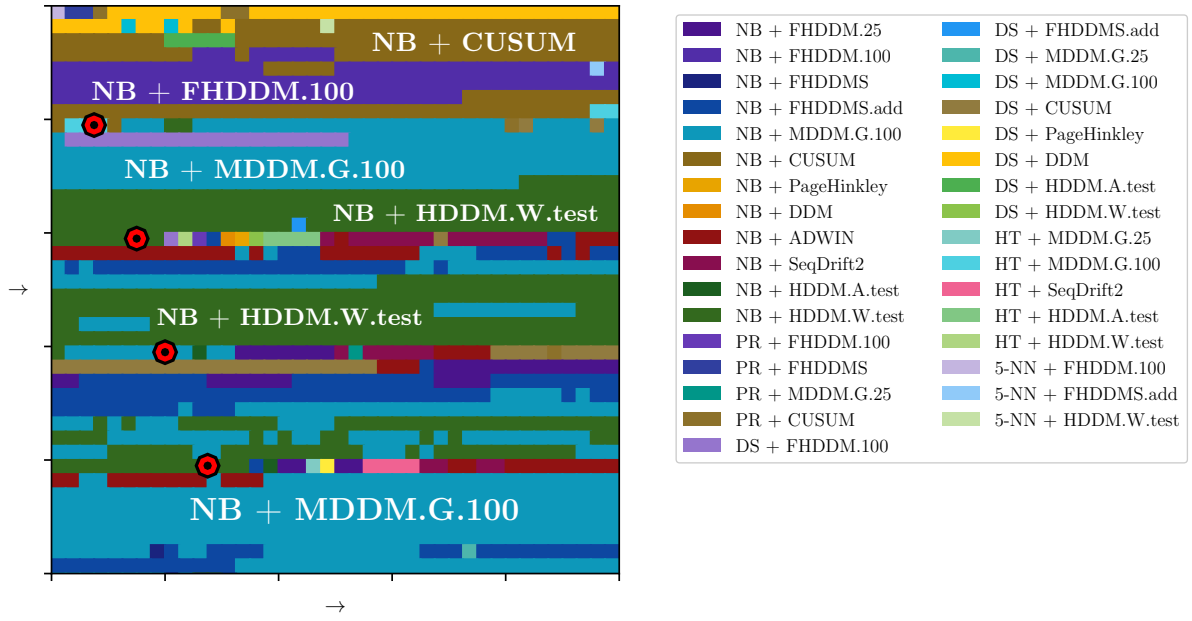
- **ADULT** – Naive Bayes (NB) paired with MDDM-G_{100} and $HDDM_{W\text{-test}}$ are often recommended as shown in Fig. 4.9; however, $FHDDMS_{add}$ outperforms $HDDM_{W\text{-test}}$ when we apply $\vec{w} = [2\ 1.5\ 3\ 2\ 1.5\ 1]^T$. As we set w_{fp} to 3, NB + $FHDDMS_{add}$ may be preferred to NB + $HDDM_{W\text{-test}}$, for introducing fewer false positives.

- **NURSERY** – Fig. 4.10 explicitly depicts the pairs of Naive Bayes (NB) with EDDM, ADWIN, $FHDDM_{100}$, HDDMs, and MDDM-G_{100} are offered when $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$. On the other hand, when $\vec{w} = [3\ 2\ 2\ 1\ 0\ 0.5]$, the pairs of Hoeffding Tree (HT) with EDDM and $HDDM_{A\text{-test}}$ are recommended. In the latter case, the Hoeffding Tree classifier is preferable than Naive Bayes, because we consider the classification error-rate as a decisive measure than memory usage or runtime for pair recommendation; as we set w_e, w_m, and w_r to 3, 0, and 0.5, respectively. Further, $HDDM_{A\text{-test}}$ becomes superior to FHDDM, MDDM, and $HDDM_{W\text{-test}}$, for having shorter detection delay and few false positive in that experiment.

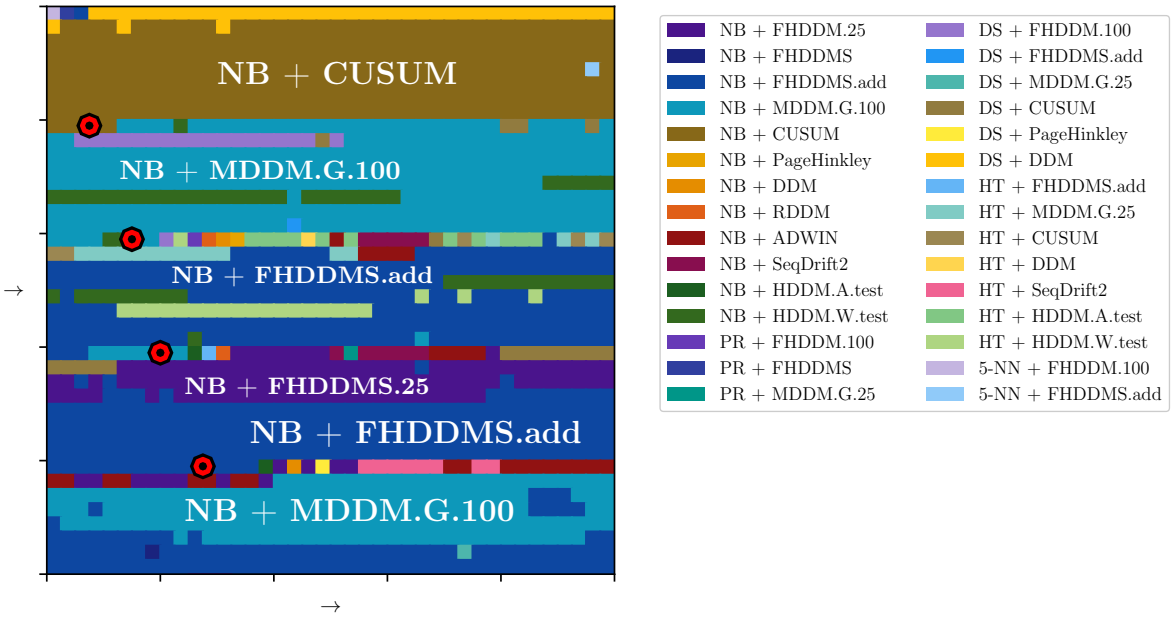
- **SHUTTLE** – In Fig. 4.10, we see the pairs of Naive Bayes (NB) and Decision Stump (DS) with EDDM, DDM, $FHDDM_{100}$, FHDDMS, and MDDM-G_{100} are preferred. It is seen that NB + $FHDDM_{100}$, NB + FHDDMS, and NB + MDDM-G_{100} are often preferred over others when $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$. Differently, in the case of $\vec{w} = [3\ 2\ 1\ 2\ 1\ 1]$, the NB + ADWIN and 5-NN + ADWIN pairs are recommended, for having lower classification error-rate and shorter drift detection delay, as higher weights are assigned to error-rate and detection delay measures for this use case.

To conclude our experiments, we used the TORNADO framework to process various kinds of classifier-detector pairs against the synthetic and semi-real-world data streams. We observed that the pairs employing FHDDM, MDDM, and HDDM variants, as drift detection methods, often outscored others against both groups of data streams. Our results confirmed that the *best* (classifier, detector) pairs vary as the stream evolves; and that choice is domain dependent. Finally, the most efficient pair is highly dependent on the weights, especially when we vary the weights of memory and runtime versus error-rate.

Pair Recommendation over Time against
ADULT Data Stream with Gradual Concept Drift
(from top-left corner to bottom-right corner)



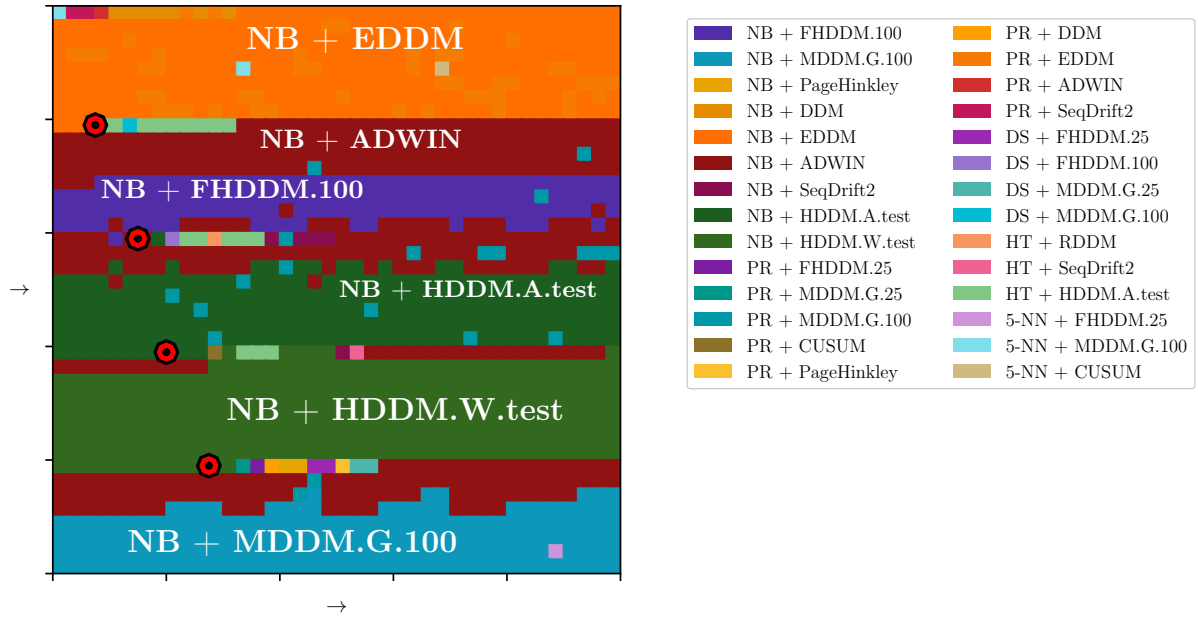
(a) The ADULT Data Stream and $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$



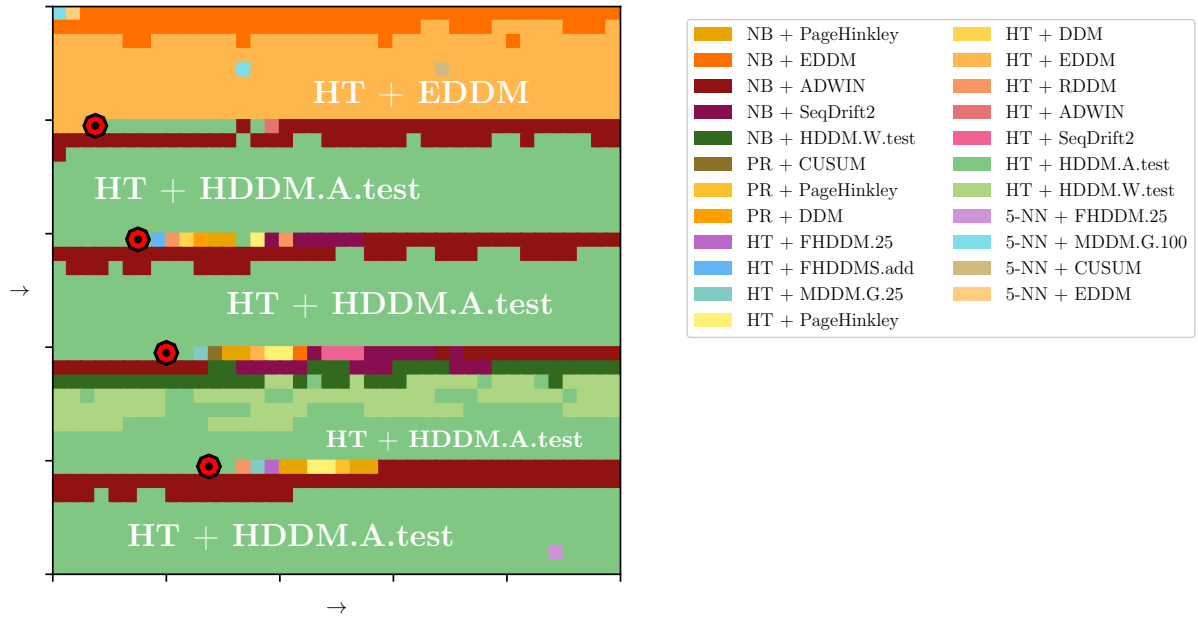
(b) The ADULT Data Stream and $\vec{w} = [2\ 1.5\ 3\ 2\ 1.5\ 1]$

Fig. 4.9. Recommended Classifier+Detector Pairs out of 75 Pairs against ADULT Data Stream with Gradual Concept Drifts

Pair Recommendation over Time against
NURSERY Data Stream with Gradual Concept Drift
(*from top-left corner to bottom-right corner*)



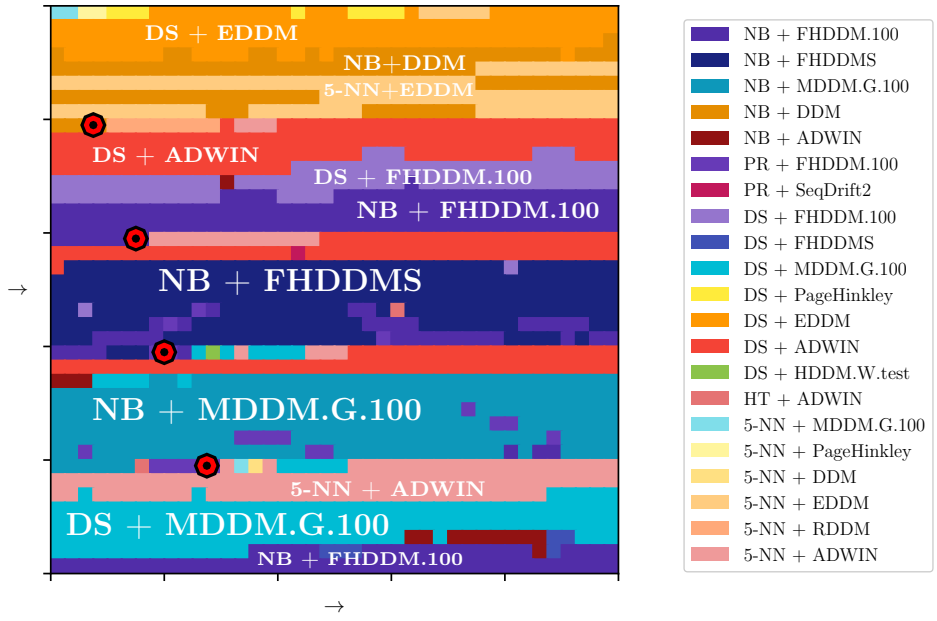
(a) The NURSERY Data Stream and $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$



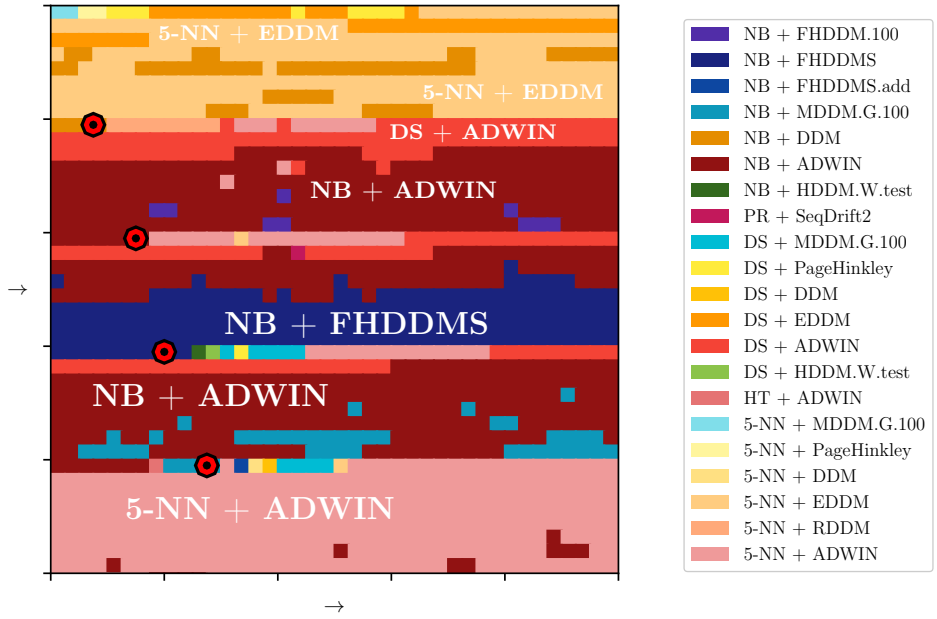
(b) The NURSERY Data Stream and $\vec{w} = [3\ 2\ 2\ 1\ 0\ 0.5]$

Fig. 4.10. Recommended Classifier+Detector Pairs out of 75 Pairs against NURSERY Data Stream with Gradual Concept Drifts

Pair Recommendation over Time against
SHUTTLE Data Stream with Gradual Concept Drift
(from top-left corner to bottom-right corner)



(a) The SHUTTLE Data Stream and $\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$



(b) The SHUTTLE Data Stream and $\vec{w} = [3\ 2\ 1\ 2\ 1\ 1]$

Fig. 4.11. Recommended Classifier+Detector Pairs out of 75 Pairs against SHUTTLE Data Stream with Gradual Concept Drifts

4.5.3 Discussion

We conducted experiments using the TORNADO framework against the synthetic and semi-real-world data streams in sections 4.5.1 and 4.5.2, respectively. Our experimental study consisted of 75 diverse (classifier, detector) pairs, which were evaluated in parallel against various kinds of data streams. The experimental results clearly showed that, as expected, no specific pair dominates others in all cases. In many cases, however, the pairs that contain the Naive Bayes (NB) or Perceptron (PR) classifiers yielded the best results, when all the weights are identical. That is, they are able to lead to a high accuracy while their resource consumptions are low. They stand against to Hoeffding Tree (HT), Decision Stump (DS), and K-Nearest Neighbours (K-NN). The Hoeffding Tree algorithm generally is expensive, in terms of memory consumption as the tree grows. Also, its runtime may increase when branching decisions become difficult. The K-NN algorithm is known as a lazy learner, meaning that it is voracious for both memory and runtime as training instance are stored for decision making. Therefore, these two methods are more suitable when runtime and memory are not critical measures.

Overall, as illustrated in Tables 4.1 to 4.3 and Figs. 4.5 to 4.11, the members of FHDDM, MDDM, and HDDM are often considered as competent drift detection methods, because of faster detection of drift points with fewer false positives and false negatives. Consequently, their pairs are offered to the user. It is worth mentioning that the pairs with other drift detectors ranked lower, for causing longer drift detection delays and higher false positive and false negative rates.

4.6 Summary

Increasingly, there is a need for adaptive learning algorithms to explore evolving data streams. Such algorithms should provide decision makers with realistic, just-in-time models for short-term and mid-term decision making against today's vast streams of data. These models should be not only timely but also accurate and able to adapt to changes in the data distribution swiftly. Intuitively, no adaptive learner outperforms others in all settings. Similarly, the effectiveness of drift detection methods is determined by the data characteristics, the types of drifts, as well as the rates of false positives and false negatives, amongst others.

Based on this observation, we defined the EMR and CAR measures in Sections 4.3.2 and 4.3.3, to consider classification, adaptation, and resource consumption measures together for the evaluation of adaptive learners against evolving data streams. Moreover, we introduced the TORNADO framework, as a reservoir of diverse adaptive learners and drift detection algorithms, in Section 4.4. The TORNADO framework is able to run various (classifier, detector) pairs simultaneously, and build their classification models incrementally in

parallel. Subsequently, the framework selects the most efficient model, regarding various performance measures, and recommends it to the user. Our experimental results against the synthetic and semi-real-world data streams confirm that the best (classifier, detector) pairs may vary in a dynamic and evolving setting over time. Furthermore, the experiments showed that the pairs with the FHDDM, MDDM, and HDDM detectors outperformed others with respect to the CAR measure.

The reader should notice that the primary goal behind introducing the CAR measure and the Tornado framework was to execute various kinds of adaptive learners simultaneously, to assess their performance based on not only the classification error-rate but also the adaptation measures, and to provide an optimal (classifier, detector) pair to the user while instances are processed over time. Hence, the purpose of this study is different from the research work by Žliobaitė et al. (2015) in which the ROI measure was used to investigate whether the adaptation is beneficial. Also, recall that the ROI measure does not comprise adaptation measures, e.g., the drift detection delay, and the false positive as well as false negative rates which may be critical in many applications, including self-driving cars and emergency settings.

We encountered several interesting avenues of future work. The incorporation of the ensemble learning into the Tornado framework needs our consideration. For this thesis, we implemented our Tornado framework on a single machine. We tend to design a hybrid environment where we use the Cloud services together with mobile devices, such as tablets. This research is motivated by our observation that, in many settings, such as environmental impact studies and emergency response, domain experts would require the ability to not only receive up-to-date models on their mobile devices but also to be able to build their models locally. In this case, we foresee that the heavy 'bulk' analytical tasks would be performed in the Cloud, while the mobile devices would contain personalized, lightweight algorithms. Domain experts are often eager to include their expertise, and thus an active learning component might prove useful.

Part IV

Epilogue

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Classic learning algorithms assume that they can load all the data in the memory, and also that they are given an unlimited amount of time for decision making. These assumptions do not hold in data stream mining, where the size of the data is exponentially larger than the size of available memory, and decision models should be able to project an output in real-time, and at any time. Hence, incremental/online learning algorithms are developed to make learning from data streams possible. They train their predictive models incrementally by processing instances, one-by-one, in real-time. The online learning algorithms must comply with the requirements of data stream mining, as explained in Section 2.2.3. To this end, they have to perform the task of learning by using a limited amount of memory in real-time. Moreover, they must be able to provide an output at any point in time.

Data streams often arrive from evolving and dynamic environments. That means the distribution of data may undergo changes over time, and consequently, causing a classification model to be outdated. This phenomenon of distributional change in data is known as *concept drift*. Adaptive learning algorithms usually utilize *drift detection methods* to find the drift points in data. Adaptive algorithms have to detect concept drift with the minimum false positive and false negative rates as well as the shortest delay of detection, as discussed in Section 2.3.3. To address this problem, as our first contribution, we introduced new drift detection methods to find concept drift with the least delay as well as the minimum false positive and false negative rates. Furthermore, since a data stream evolves, learning algorithms may show different performances over time. As a result, tracking the best predictive models, for the current distribution, is highly important. As our second contribution, we developed a framework for running various kinds of adaptive algorithms together to offer the most efficient algorithm (i.e., classifier-detector pair) to the user. We conclude our contributions as follows:

Our contribution in adaptive learning – In Chapter 3, we presented our new drift detection methods which are originally introduced in (Pesaranghader and Viktor, 2016), (Pesaranghader et al., 2018b), and (Pesaranghader et al., 2018a). The Fast Hoeffding Drift Detection Method (FHDDM), as the first generation, was presented in Section 3.2. FHDDM works based on the PAC learning model that the accuracy of a classifier should increase or stay steady as more instances arrive; otherwise, it implies facing concept drift. FHDDM slides a window with a size of n on the prediction results. The average of elements inside the sliding window is measured and is stored in the μ^t variable. It also stores the maximum average experienced so far in the μ^m variable. The FHDDM algorithm alarms for concept drift once a significant difference, bounded by Hoeffding's inequality (Hoeffding, 1963), between μ^m and μ^t is observed.

We represented the Stacking Fast Hoeffding Drift Detection Method (FHDDMS) and its Additive variant ($\text{FHDDMS}_{\text{add}}$) in Section 3.3. They extend the idea of FHDDM by stacking two sliding windows on the top of each other for achieving shorter detection delay as well as lower false negative rate. FHDDMS slides a large and a short sliding window on the prediction results to reduce the detection delay and false negative rate. Intuitively, a short window should detect abrupt drift faster, while a long window should detect gradual drift with a lower false negative rate. FHDDMS follows a similar procedure taken by FHDDM for detecting concept drift. $\text{FHDDMS}_{\text{add}}$, as an extension of FHDDMS, substitutes the summation of entries for binary indicators to save memory and CPU. Although, it may compromise the delay of detection for resources.

Then, we introduced the McDiarmid Drift Detection Methods (MDDMs) in Section 3.4. In contrast to the FHDDM solutions, MDDMs associated the entries into their sliding windows with weights for a faster detection of concept drift. That is, greater weights are assigned to the most recent instances, i.e., $w_i < w_{i+1}$. The MDDM variants weight the elements in the sliding window in different ways. For detecting concept drift, an MDDM algorithm calculates the weighted mean of elements in the sliding window, and stores it in the μ_w^t variable. It also keeps the maximum weighted mean observed so far in the μ_w^m variable. Finally, a significant difference, bounded by McDiarmid's inequality (McDiarmid, 1989), between μ_w^t and μ_w^m suggests the occurrence of concept drift.

Experimental evaluation, against the synthetic and the real-world data streams, indicated that FHDDM, the FHDDMS members, and the MDDM variants detected the drift points with shorter delays, fewer false positives, fewer false negatives, using an affordable amount of resources, compared to the state-of-the-art methods. Our experiments showed that the idea of stacking window works effectively. We also observed that the MDDM variants alarmed for concept drift with shorter delay compared to FHDDM. The predictive models also benefited from our drift detection methods for keeping the accuracy of classification persistently high.

Our contribution in adaptive multi-strategy learning – As mentioned earlier, a data stream typically arrives from an evolving and dynamic environment, where the underlying distribution of data may change over time. Accordingly, once concept drift is detected, adaptive learning algorithms retrain their decision models to keep the classification accuracy high. However, this does not guarantee that the most efficient learner, at the current moment, would stay efficient forever due to the distributional changes. In Chapter 4, we introduced our TORNADO framework able to simultaneously run diverse kinds of learning algorithms against a data stream. In the framework, each learning algorithm builds its (classifier, detector) pair independently. All pairs participate in decision making by providing the predicted class labels. The TORNADO framework ranks the (classifier, detector) pairs based on their performances. It then recommends the most efficient one to the user. For monitoring the performance of the (classifier, detector) pairs, we defined the EMR and CAR measures which integrate individual measures, e.g., error-rate, drift detection delay, and memory usage, into a holistic one. The reader may find more information in Section 4.3. The EMR and CAR measures let one prioritize some specific measures to others. We experimental study confirmed that the most efficient (classifier, detector) pair varies over time.

5.2 Future Work

As for future research, plenty of interesting avenues exist that we describe as follow.

Stacking McDiarmid Drift Detection Methods: We defined the notion of *stacking windows* to extend our Fast Hoeffding Drift Detection Method (FHDDM) in Section 3.3. Recall that, in our stacking windows-based methods, i.e., FHDDMS, multiple windows are simultaneously slid over the entries, where each window may react to *abrupt* and *gradual* drift patterns differently. That is a short window detects abrupt concept drift faster, while a long window monitors gradual concept drift accurately. We will apply this idea to our McDiarmid Drift Detection Methods (MDDMs) as well, and study their behaviours against abrupt and gradual concept drift patterns.

Dealing with Time-Series Data: In this thesis, we focused on cross-sectional, i.e., non-time-series, data against which we performed our experiments to evaluate our drift detection methods as well as the TORNADO framework. As for future work, we are eager to conduct experiments on (evolving) time-series data streams and to study the behaviour of our drift detection methods and the TORNADO framework. In addition, we may deploy the learning algorithms that are specifically designed to deal with time-series data.

Applications in the Real-world: Žliobaitė (2010) surveyed several real-world applications where concept drift is inevitable, and she categorized the applications into general groups of *monitoring and controlling systems*, *personal assistance and information systems*, *decision-making systems*, and *artificial intelligence and robotics*. It is feasible to utilize our drift detection methods for adaptive learning in such domains. For example, they may be used to detect adversary behaviours, which cause concept drift, in the intrusion detection systems. In the customer profiling application, adaptation to changes in the customer's preferences is necessary to keep the accuracy of system high, for example, for the item recommendation task.

Dealing with the Class Imbalance Problem: Class imbalance is a recognized problem in machine learning where the minority class has a very low prior probability compared to the majority class. We may experience this problem, for example, in the fraud detection and credit scoring systems where suspicious records appear rarely. As to this problem, detecting any drift with respect to the minority class may be challenging. Our drift detection methods may accompany the SCUT-DS algorithms (Olaitan, 2018) for adaptive learning from imbalanced data streams.

Addressing High Arrival Data Rates: Although we mentioned that the online learners should process instances as fast as they arrive in Section 2.2.3, it may not be achievable in some cases such as traffic or network systems where high congestion of data, and so high arriving rate, may be experienced at some peak hours. To avoid loss of any information when data arrive so fast, we must intelligently pick the most informative examples, e.g., using unsupervised methods, for training. To this end, while instances are filtered online, some of those examples are used for training, and the rest are stored into a (limited) buffer. The classifiers may be updated on the buffered examples, in an *offline* mode, during off-peak hours. We will implement an intelligent instance extraction filter for the TORNADO framework in the future.

Enabling Anytime Recommendation in Tornado: We assumed that we have enough time for the prediction and the evaluation tasks in the Tornado framework while the input instances are processed one-by-one for an optimal classifier-detector pair recommendation. The assumption may introduce some exhaustion in terms of the waiting time. For example, lazy classifiers, e.g., K-NN, which take a longer time to project an output, may cause a delay in the ranking of pairs. That is, a faster classifier, e.g., Naive Bayes, must wait for a slower classifier until it projects an output. To address this scenario, all classifiers may be upper-bounded by utilizing the amount of time for prediction, and to be ready for anytime prediction before passing the upper-bound. That clause and restriction will ensure us that anytime pair recommendation is possible by the TORNADO framework.

Ensemble Learning Algorithms for Tornado: In classical machine learning, ensemble algorithms often lead to higher accuracy compared to individual learners (Han et al., 2011). In Chapter 4, we introduced the notion of *multi-strategy learning* and separate it from ensemble learning. Nonetheless, as discussed in that chapter, an ensemble algorithm may

be paired with a drift detection method for a multi-strategy learning task. Therefore, the implementation of ensemble learners, for the TORNADO framework, can be considered as an avenue of future work. Besides, we plan to develop *hybrid* ensemble learning solutions for supervised learning.

Integrating Semi-supervised and Active Learning into Tornado: Although the scope of this thesis was (adaptive) learning from *fully* labelled data, handling unlabelled instances, missing labels, and late-arriving labels are amongst the challenges of data stream mining. The reader should note that manual labelling of all instances are not only (rather) impractical but also costly due to the fundamental characteristics of data stream mining, i.e., learning in real-time and using a limited amount of memory. As future work, the online semi-supervised techniques will be implemented for our TORNADO framework to deal with partially labelled data streams. Furthermore, we may consider the active learning solutions to minimize the reinforcement of error in the semi-supervised learning settings. In fact, we may use the knowledge of an expert in semi-supervised learning.

Upscaling Tornado and Launching in the Cloud: For this thesis work, we deployed the TORNADO framework on a single machine. We will extend our framework by designing a hybrid architecture which allows the users to utilize the Cloud services and their devices, e.g., smart-phones, tablets, and notebooks, for decision making. We are motivated by the fact that in many applications, including environmental impact studies and emergency response, the end users would like to require the ability to not only receive up-to-date models from the Cloud but also to be able to build the predictive models locally on their devices. That is, the system lets the users perform the massive analytical and reasoning tasks in the Cloud, while they can contain personalized (lightweight) learning algorithms on their own devices.

Part V

APPENDICES

Appendix A

Pseudocodes of Online Learning Algorithms

A.1 Naive Bayes

Algorithm A.1 Naive Bayes (NB) Pseudocode

```
1: function INITIALIZE(attributes, labels)
2:     (A, Y) ← (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 ∈ 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(x|c) = ∏_{k=1}^{|x|} P(x_k|c)
19:        Calculate relative-likelihood Rel(c|x) = P(x|c) · P(c)
20:    return the class with the maximum relative-likelihood
```

A.2 Decision Stump

Algorithm A.2 Decision Stump (DS) Pseudocode

```
1: function INITIALIZE(attributes, labels)
2:     (A, Y) ← (attributes, labels)
3:     foreach class c in C do
4:         N(c) = 0
5:         foreach value v of a ∈ 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 ∈ A that holds the value of x_i ∈ x do
12:        Increase number N(x_i|y) by 1
13:    foreach class c in Y do
14:        Entropy_Total += −(N(c)/n) · log_2(N(c)/n)
15:    foreach attribute a ∈ 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) · log_2 P(c|v)
20:            Entropy_A += (N(v)/n) × 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
```

A.3 Hoeffding Tree

Algorithm A.3 Hoeffding Tree (HT) Pseudocode
(Domingos and Hulten, 2000; Hulten et al., 2005)

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, δ, τ) $\leftarrow$ (*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 $X_i \in$ **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 $X_i \in$ 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 $X_i \in X_l$
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** $X_a \neq X_\emptyset$ **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 $X_m = X - \{X_a\}$
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

A.4 Perceptron

Algorithm A.4 Perceptron (PR) Pseudocode

function INITIALIZE([learning rate], labels)
    $Y \leftarrow$ labels
    $\eta \leftarrow$ [learning rate]
    Let w be randomly initialized

function TRAIN($\mathbf{x}$,y)
    foreach class c in Y **do**
        if $c = y$ **then**
            $\delta = (1 - \tilde{f}_{\mathbf{w}_i}(\mathbf{x})) \cdot \tilde{f}_{\mathbf{w}_i}(\mathbf{x}) \cdot (1 - \tilde{f}_{\mathbf{w}_i}(\mathbf{x}))$
        else
            $\delta = (0 - \tilde{f}_{\mathbf{w}_i}(\mathbf{x})) \cdot \tilde{f}_{\mathbf{w}_i}(\mathbf{x}) \cdot (1 - \tilde{f}_{\mathbf{w}_i}(\mathbf{x}))$
        $\mathbf{w}_c = \mathbf{w}_c + \eta \cdot \delta \cdot \mathbf{x}$

function TEST($\mathbf{x}$)
    return $argmax_c \tilde{f}_{\mathbf{w}_c}(\mathbf{x})$

A.5 K-Nearest Neighbours

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

function INITIALIZE([num. of neighbours], [window size])
    K $\leftarrow$ [num. of neighbours]
    $n \leftarrow$ [window size]
    $W \leftarrow$ Initialize a window with size of n

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

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

Appendix B

Complementary Tables

In Tables B.1 to B.8, the results are the averaged drift detection delays, the averaged drift detection true positive numbers, the averaged drift detection false positive numbers, the averaged drift detection false negative numbers, the averaged memory usage of the detector (in bytes), the averaged runtime of the detector (in milliseconds), and the averaged classification accuracy from left to right.

Tables B.9 to B.22 list the classification error-rates, the averaged drift detection delays, the drift detection false positive numbers, the drift detection false negative numbers, the memory usage of pairs (in kilobytes), and the runtime of pairs (in milliseconds), from left to right.

Table B.1. Hoeffding Tree and DDMs against SINE1 Data Stream with Abrupt Drift

HT - SINE1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	40.65 ± 3.15	4.00	0.10 ± 0.33	0.00	232.00	11.40 ± 3.50	87.07 ± 0.16
$FHDDM_{100}$	48.09 ± 2.82	4.00	0.43 ± 0.78	0.00	528.00	16.89 ± 3.93	87.05 ± 0.15
FHDDMS	40.33 ± 3.18	4.00	0.46 ± 0.83	0.00	528.00	29.24 ± 5.04	87.06 ± 0.16
$FHDDMS_{add}$	52.31 ± 4.21	4.00	0.18 ± 0.43	0.00	152.00	13.77 ± 3.96	87.04 ± 0.15
$MDDM\text{-}A_{25}$	38.60 ± 3.38	4.00	0.21 ± 0.43	0.00	232.00	19.86 ± 4.23	87.07 ± 0.16
$MDDM\text{-}A_{100}$	42.98 ± 2.84	4.00	0.41 ± 0.75	0.00	528.00	40.73 ± 5.18	87.05 ± 0.15
$MDDM\text{-}G_{25}$	38.56 ± 3.36	4.00	0.20 ± 0.42	0.00	232.00	15.47 ± 4.04	87.07 ± 0.16
$MDDM\text{-}G_{100}$	41.02 ± 2.86	4.00	0.48 ± 0.73	0.00	528.00	31.76 ± 5.54	87.05 ± 0.15
$MDDM\text{-}E_{25}$	38.56 ± 3.36	4.00	0.20 ± 0.42	0.00	232.00	35.78 ± 5.72	87.07 ± 0.16
$MDDM\text{-}E_{100}$	41.01 ± 2.88	4.00	0.48 ± 0.71	0.00	528.00	51.61 ± 6.27	87.05 ± 0.15
CUSUM	86.89 ± 4.47	4.00	0.24 ± 0.47	0.00	128.00	15.60 ± 3.95	86.94 ± 0.15
PageHinkley	229.24 ± 13.20	2.30 ± 1.07	1.71 ± 1.08	1.70 ± 1.07	136.00	12.26 ± 3.75	86.46 ± 0.17
DDM	163.11 ± 22.73	3.36 ± 0.77	3.30 ± 2.20	0.64 ± 0.77	136.00	15.50 ± 3.46	86.06 ± 1.34
EDDM	243.83 ± 14.25	0.22 ± 0.44	33.90 ± 11.61	3.78 ± 0.44	136.00	11.30 ± 3.33	84.71 ± 0.55
RDDM	93.63 ± 7.57	4.00	4.72 ± 3.58	0.00	7224.00	17.90 ± 4.66	86.79 ± 0.18
$ADWIN_{32}$	63.84 ± 1.12	4.00	7.31 ± 3.18	0.00	> 2060	> 124.40	86.67 ± 0.21
SeqDrift2	200.00	4.00	4.83 ± 1.16	0.00	> 82270	40.12 ± 5.72	86.53 ± 0.15
$HDDM_{A\text{-test}}$	57.62 ± 11.81	4.00	0.71 ± 0.89	0.00	160.00	34.77 ± 5.82	87.01 ± 0.16
$HDDM_{W\text{-test}}$	35.70 ± 2.95	4.00	0.46 ± 0.68	0.00	160.00	36.55 ± 5.88	87.07 ± 0.15
No Detection	—	—	—	—	—	—	57.65 ± 0.81

Table B.2. Hoeffding Tree and DDMs against MIXED Data Stream with Abrupt Drift

HT - MIXED Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	40.55 ± 3.70	4.00	0.65 ± 0.94	0.00	232.00	11.85 ± 3.81	83.39 ± 0.10
$FHDDM_{100}$	49.19 ± 8.38	3.98 ± 0.14	1.82 ± 1.46	0.02 ± 0.14	528.00	15.64 ± 4.16	83.32 ± 0.13
FHDDMS	40.37 ± 6.85	3.99 ± 0.10	2.25 ± 1.73	0.01 ± 0.10	528.00	29.57 ± 5.42	83.31 ± 0.13
$FHDDMS_{add}$	52.25 ± 6.82	3.99 ± 0.10	0.73 ± 0.90	0.01 ± 0.10	152.00	14.80 ± 3.46	83.37 ± 0.11
$MDDM\text{-}A_{25}$	38.38 ± 3.66	4.00	1.11 ± 1.15	0.00	232.00	20.47 ± 4.49	83.36 ± 0.11
$MDDM\text{-}A_{100}$	45.04 ± 9.73	3.97 ± 0.17	2.17 ± 1.63	0.03 ± 0.17	528.00	40.65 ± 6.02	83.32 ± 0.13
$MDDM\text{-}G_{25}$	38.28 ± 3.64	4.00	1.19 ± 1.21	0.00	232.00	15.31 ± 3.79	83.36 ± 0.11
$MDDM\text{-}G_{100}$	42.54 ± 8.29	3.98 ± 0.14	2.32 ± 1.77	0.02 ± 0.14	528.00	31.88 ± 5.47	83.30 ± 0.13
$MDDM\text{-}E_{25}$	38.28 ± 3.64	4.00	1.19 ± 1.21	0.00	232.00	35.84 ± 6.61	83.36 ± 0.11
$MDDM\text{-}E_{100}$	42.53 ± 8.30	3.98 ± 0.14	2.32 ± 1.78	0.02 ± 0.14	528.00	50.93 ± 7.24	83.30 ± 0.13
CUSUM	90.90 ± 6.13	4.00	0.32 ± 0.58	0.00	128.00	14.25 ± 3.66	83.27 ± 0.12
PageHinkley	229.91 ± 13.27	2.26 ± 0.98	1.74 ± 0.98	1.74 ± 0.98	136.00	12.75 ± 3.64	82.88 ± 0.11
DDM	195.73 ± 22.12	2.76 ± 1.01	2.91 ± 1.96	1.24 ± 1.01	136.00	15.81 ± 4.41	81.78 ± 2.06
EDDM	248.46 ± 7.69	0.05 ± 0.22	21.51 ± 7.70	3.95 ± 0.22	136.00	11.82 ± 3.37	80.65 ± 0.82
RDDM	106.68 ± 11.26	3.99 ± 0.10	3.49 ± 2.47	0.01 ± 0.10	7224.00	17.12 ± 4.08	83.16 ± 0.12
$ADWIN_{32}$	64.72 ± 2.79	4.00	4.84 ± 2.44	0.00	> 2065	> 124.75	83.25 ± 0.12
SeqDrift2	200.00	4.00	4.98 ± 1.20	0.00	> 81470	39.36 ± 5.74	82.91 ± 0.11
$HDDM_{A\text{-test}}$	69.42 ± 15.51	4.00	1.28 ± 1.09	0.00	160.00	35.64 ± 5.44	83.31 ± 0.11
$HDDM_{W\text{-test}}$	35.56 ± 3.50	4.00	3.23 ± 1.95	0.00	160.00	34.79 ± 5.55	83.27 ± 0.12
No Detection	—	—	—	—	—	—	58.11 ± 1.09

Table B.3. Hoeffding Tree and DDMs against Circles Data Stream with Gradual Drift

HT - Circles Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	243.97 ± 122.26	2.71 ± 0.45	0.37 ± 0.56	0.29 ± 0.45	232.00	11.39 ± 3.03	86.44 ± 0.26
$FHDDM_{100}$	79.28 ± 20.64	3.00	0.17 ± 0.40	0.00	528.00	16.25 ± 4.18	86.58 ± 0.13
FHDDMS	75.29 ± 25.40	3.00	0.31 ± 0.59	0.00	528.00	30.61 ± 5.38	86.58 ± 0.14
$FHDDMS_{add}$	96.75 ± 33.35	3.00	0.08 ± 0.27	0.00	152.00	14.13 ± 3.57	86.59 ± 0.14
$MDDM\text{-}A_{25}$	221.43 ± 117.64	2.82 ± 0.38	0.45 ± 0.68	0.18 ± 0.38	232.00	19.77 ± 4.07	86.47 ± 0.22
$MDDM\text{-}A_{100}$	71.98 ± 22.19	3.00	0.27 ± 0.51	0.00	528.00	41.82 ± 5.91	86.58 ± 0.16
$MDDM\text{-}G_{25}$	216.46 ± 118.24	2.83 ± 0.38	0.46 ± 0.71	0.17 ± 0.38	232.00	15.32 ± 3.96	86.48 ± 0.22
$MDDM\text{-}G_{100}$	69.42 ± 22.09	3.00	0.36 ± 0.61	0.00	528.00	32.80 ± 5.89	86.58 ± 0.17
$MDDM\text{-}E_{25}$	216.10 ± 118.47	2.83 ± 0.38	0.46 ± 0.71	0.17 ± 0.38	232.00	36.74 ± 5.20	86.48 ± 0.22
$MDDM\text{-}E_{100}$	69.52 ± 22.12	3.00	0.37 ± 0.61	0.00	528.00	51.70 ± 6.90	86.57 ± 0.17
CUSUM	220.07 ± 31.79	2.99 ± 0.10	0.04 ± 0.20	0.01 ± 0.10	128.00	14.95 ± 3.59	86.51 ± 0.13
PageHinkley	855.37 ± 56.27	1.79 ± 0.45	1.24 ± 0.47	1.21 ± 0.45	136.00	12.58 ± 3.49	85.96 ± 0.15
DDM	487.97 ± 82.24	2.78 ± 0.52	1.41 ± 1.24	0.22 ± 0.52	136.00	16.86 ± 4.80	86.21 ± 0.47
EDDM	987.61 ± 54.35	0.07 ± 0.26	24.61 ± 14.48	2.93 ± 0.26	136.00	11.35 ± 3.13	84.89 ± 0.29
RDDM	293.80 ± 38.52	2.98 ± 0.14	0.79 ± 1.25	0.02 ± 0.14	7224.00	18.89 ± 4.79	86.46 ± 0.16
$ADWIN_{32}$	236.48 ± 130.94	2.67 ± 0.47	9.74 ± 3.05	0.33 ± 0.47	> 2145	> 125.89	85.62 ± 0.19
SeqDrift2	202.67 ± 16.11	3.00	3.08 ± 0.90	0.00	> 104370	41.23 ± 6.30	86.47 ± 0.14
$HDDM_{A\text{-test}}$	111.96 ± 68.22	2.96 ± 0.20	0.65 ± 0.92	0.04 ± 0.20	160.00	35.02 ± 6.61	86.52 ± 0.20
$HDDM_{W\text{-test}}$	94.03 ± 57.61	2.98 ± 0.14	0.73 ± 0.87	0.02 ± 0.14	160.00	36.48 ± 5.98	86.53 ± 0.18
No Detection	—	—	—	—	—	—	82.03 ± 0.69

Table B.4. Hoeffding Tree and DDMs against LED Data Stream with Gradual Drift

HT - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	346.36 ± 97.35	2.94 ± 0.28	0.06 ± 0.28	0.06 ± 0.28	232.00	12.60 ± 3.86	89.53 ± 0.04
$FHDDM_{100}$	220.40 ± 76.00	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	528.00	17.39 ± 4.52	89.57 ± 0.04
FHDDMS	220.17 ± 76.03	2.97 ± 0.22	0.04 ± 0.24	0.03 ± 0.22	528.00	27.56 ± 5.26	89.56 ± 0.04
$FHDDMS_{add}$	249.75 ± 77.20	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	152.00	11.90 ± 3.30	89.56 ± 0.04
$MDDM\text{-}A_{25}$	333.62 ± 95.16	2.94 ± 0.28	0.06 ± 0.28	0.06 ± 0.28	232.00	21.70 ± 4.50	89.54 ± 0.04
$MDDM\text{-}A_{100}$	210.31 ± 73.05	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	48.18 ± 8.94	89.56 ± 0.04
$MDDM\text{-}G_{25}$	332.70 ± 95.31	2.94 ± 0.28	0.06 ± 0.28	0.06 ± 0.28	232.00	17.37 ± 3.93	89.54 ± 0.04
$MDDM\text{-}G_{100}$	208.65 ± 73.05	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	34.55 ± 6.39	89.56 ± 0.04
$MDDM\text{-}E_{25}$	332.70 ± 95.31	2.94 ± 0.28	0.06 ± 0.28	0.06 ± 0.28	232.00	38.11 ± 5.77	89.54 ± 0.04
$MDDM\text{-}E_{100}$	208.61 ± 73.05	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	53.15 ± 7.07	89.56 ± 0.04
CUSUM	300.68 ± 50.30	3.00	0.00	0.00	128.00	12.65 ± 3.50	89.56 ± 0.03
PageHinkley	560.30 ± 79.43	2.95 ± 0.26	0.04 ± 0.24	0.05 ± 0.26	136.00	10.19 ± 3.23	89.35 ± 0.04
DDM	444.13 ± 79.82	2.97 ± 0.17	0.32 ± 0.58	0.03 ± 0.17	136.00	14.04 ± 3.66	89.47 ± 0.56
EDDM	954.97 ± 62.98	0.66 ± 0.71	5.97 ± 1.69	2.34 ± 0.71	136.00	9.69 ± 3.14	88.33 ± 0.50
RDDM	321.88 ± 50.94	2.98 ± 0.14	0.61 ± 0.96	0.02 ± 0.14	7224.00	16.54 ± 3.68	89.63 ± 0.04
$ADWIN_{32}$	521.47 ± 239.71	2.40 ± 0.77	474.95 ± 14.12	0.60 ± 0.77	> 1401	97.69 ± 9.50	72.25 ± 0.49
SeqDrift2	426.00 ± 173.31	2.78 ± 0.44	277.06 ± 47.48	0.22 ± 0.44	> 15265	39.36 ± 5.56	76.51 ± 2.28
$HDDM_{A\text{-test}}$	295.03 ± 85.29	2.98 ± 0.20	0.16 ± 0.44	0.02 ± 0.20	160.00	31.54 ± 5.82	89.58 ± 0.05
$HDDM_{W\text{-test}}$	259.18 ± 87.25	2.95 ± 0.26	0.08 ± 0.31	0.05 ± 0.26	160.00	32.60 ± 6.57	89.56 ± 0.04
No Detection	—	—	—	—	—	—	83.86 ± 1.89

Table B.5. Naive Bayes and DDMs against SINE1 Data Stream with Abrupt Drift

NB - SINE1 Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	40.48 ± 3.37	4.00	0.04 ± 0.20	0.00	232.00	12.11 ± 3.63	86.08 ± 0.25
$FHDDM_{100}$	48.19 ± 5.54	3.99 ± 0.10	0.13 ± 0.34	0.01 ± 0.10	528.00	17.46 ± 4.58	86.06 ± 0.25
FHDDMS	40.62 ± 5.73	3.99 ± 0.10	0.17 ± 0.38	0.01 ± 0.10	528.00	30.18 ± 5.68	86.08 ± 0.25
$FHDDMS_{add}$	51.94 ± 4.21	4.00	0.02 ± 0.14	0.00	152.00	14.64 ± 4.01	86.05 ± 0.25
$MDDM\text{-}A_{25}$	38.55 ± 3.35	4.00	0.13 ± 0.34	0.00	232.00	20.54 ± 4.08	86.08 ± 0.25
$MDDM\text{-}A_{100}$	43.12 ± 5.65	3.99 ± 0.10	0.26 ± 0.46	0.01 ± 0.10	528.00	42.02 ± 7.12	86.07 ± 0.25
$MDDM\text{-}G_{25}$	38.47 ± 3.35	4.00	0.14 ± 0.35	0.00	232.00	15.10 ± 3.48	86.08 ± 0.25
$MDDM\text{-}G_{100}$	41.21 ± 5.66	3.99 ± 0.10	0.28 ± 0.49	0.01 ± 0.10	528.00	32.84 ± 5.65	86.07 ± 0.25
$MDDM\text{-}E_{25}$	38.46 ± 3.35	4.00	0.14 ± 0.35	0.00	232.00	35.98 ± 6.75	86.08 ± 0.25
$MDDM\text{-}E_{100}$	41.16 ± 5.66	3.99 ± 0.10	0.28 ± 0.49	0.01 ± 0.10	528.00	53.67 ± 6.08	86.07 ± 0.25
CUSUM	83.27 ± 6.96	3.99 ± 0.10	0.71 ± 0.86	0.01 ± 0.10	128.00	14.41 ± 3.57	85.96 ± 0.25
PageHinkley	175.07 ± 24.72	3.71 ± 0.50	0.35 ± 0.54	0.29 ± 0.50	136.00	11.53 ± 2.99	85.69 ± 0.27
DDM	179.18 ± 26.83	2.87 ± 0.84	3.09 ± 1.88	1.13 ± 0.84	136.00	15.83 ± 3.87	82.39 ± 4.32
EDDM	234.28 ± 22.22	0.57 ± 0.64	33.53 ± 11.50	3.43 ± 0.64	136.00	12.52 ± 4.04	83.44 ± 2.87
RDDM	89.72 ± 16.45	3.99 ± 0.10	3.93 ± 2.91	0.01 ± 0.10	7224.00	17.08 ± 4.35	85.98 ± 0.27
$ADWIN_{32}$	63.92 ± 0.80	4.00	3.86 ± 1.09	0.00	> 2070	> 126.30	85.93 ± 0.23
SeqDrift2	200.00	4.00	4.26 ± 0.58	0.00	> 83665	39.35 ± 6.06	85.59 ± 0.25
$HDDM_{A\text{-test}}$	88.03 ± 25.73	3.97 ± 0.17	0.35 ± 0.55	0.03 ± 0.17	160.00	35.68 ± 6.06	85.95 ± 0.25
$HDDM_{W\text{-test}}$	35.52 ± 3.10	4.00	0.41 ± 0.58	0.00	160.00	35.62 ± 6.25	86.09 ± 0.25
No Detection	—	—	—	—	—	—	57.25 ± 0.17

Table B.6. Naive Bayes and DDMs against MIXED Data Stream with Abrupt Drift

NB - MIXED Data Stream ($\zeta = 50$, $\Delta = 250$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	40.56 ± 3.72	4.00	0.25 ± 0.48	0.00	232.00	11.54 ± 3.57	83.38 ± 0.08
$FHDDM_{100}$	48.77 ± 6.86	3.99 ± 0.10	0.58 ± 0.79	0.01 ± 0.10	528.00	15.61 ± 3.95	83.36 ± 0.09
FHDDMS	40.58 ± 6.92	3.99 ± 0.10	0.78 ± 0.88	0.01 ± 0.10	528.00	30.23 ± 5.62	83.37 ± 0.08
$FHDDMS_{add}$	52.25 ± 6.88	3.99 ± 0.10	0.16 ± 0.39	0.01 ± 0.10	152.00	14.01 ± 3.49	83.37 ± 0.08
$MDDM\text{-}A_{25}$	38.52 ± 3.81	4.00	0.69 ± 0.89	0.00	232.00	20.76 ± 4.69	83.37 ± 0.09
$MDDM\text{-}A_{100}$	44.34 ± 8.41	3.98 ± 0.14	0.92 ± 0.93	0.02 ± 0.14	528.00	45.28 ± 7.44	83.36 ± 0.09
$MDDM\text{-}G_{25}$	38.41 ± 3.81	4.00	0.70 ± 0.89	0.00	232.00	16.44 ± 4.08	83.37 ± 0.09
$MDDM\text{-}G_{100}$	42.01 ± 6.69	3.99 ± 0.10	0.98 ± 1.00	0.01 ± 0.10	528.00	33.13 ± 5.66	83.37 ± 0.09
$MDDM\text{-}E_{25}$	38.41 ± 3.81	4.00	0.70 ± 0.89	0.00	232.00	36.58 ± 5.77	83.37 ± 0.09
$MDDM\text{-}E_{100}$	41.99 ± 6.69	3.99 ± 0.10	0.98 ± 1.00	0.01 ± 0.10	528.00	53.87 ± 7.35	83.37 ± 0.09
CUSUM	88.23 ± 8.97	3.99 ± 0.10	0.35 ± 0.54	0.01 ± 0.10	128.00	14.49 ± 3.82	83.27 ± 0.08
PageHinkley	198.79 ± 18.72	3.56 ± 0.65	0.44 ± 0.65	0.44 ± 0.65	136.00	11.62 ± 3.35	82.97 ± 0.10
DDM	192.99 ± 23.82	2.78 ± 1.00	2.41 ± 1.44	1.22 ± 1.00	136.00	15.82 ± 4.17	80.28 ± 4.11
EDDM	247.47 ± 8.60	0.11 ± 0.31	20.22 ± 7.66	3.89 ± 0.31	136.00	11.79 ± 3.25	80.30 ± 2.32
RDDM	104.97 ± 12.06	3.99 ± 0.10	1.86 ± 1.65	0.01 ± 0.10	7224.00	18.11 ± 4.13	83.24 ± 0.09
$ADWIN_{32}$	64.48 ± 1.90	4.00	3.47 ± 1.42	0.00	> 2070	> 124.45	83.28 ± 0.08
SeqDrift2	200.00	4.00	4.39 ± 0.79	0.00	> 83520	38.91 ± 5.71	82.91 ± 0.08
$HDDM_{A\text{-test}}$	83.71 ± 19.46	3.96 ± 0.20	0.48 ± 0.64	0.04 ± 0.20	160.00	35.57 ± 5.59	83.28 ± 0.09
$HDDM_{W\text{-test}}$	35.75 ± 3.94	4.00	1.77 ± 1.39	0.00	160.00	34.92 ± 6.01	83.36 ± 0.09
No Detection	—	—	—	—	—	—	56.62 ± 0.12

Table B.7. Naive Bayes and DDMs against CIRCLES Data Stream with Gradual Drift

NB - CIRCLES Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	432.30 ± 88.98	2.16 ± 0.37	0.40 ± 0.66	0.84 ± 0.37	232.00	10.83 ± 3.14	83.96 ± 0.15
$FHDDM_{100}$	166.13 ± 83.84	2.96 ± 0.20	0.43 ± 0.60	0.04 ± 0.20	528.00	16.51 ± 4.09	84.14 ± 0.13
FHDDMS	164.87 ± 86.81	2.95 ± 0.22	0.60 ± 0.77	0.05 ± 0.22	528.00	28.95 ± 5.73	84.14 ± 0.12
$FHDDMS_{add}$	260.42 ± 120.28	2.76 ± 0.43	0.20 ± 0.49	0.24 ± 0.43	152.00	14.38 ± 3.76	84.08 ± 0.14
$MDDM\text{-}A_{25}$	384.56 ± 114.42	2.28 ± 0.45	0.64 ± 0.78	0.72 ± 0.45	232.00	19.97 ± 4.78	84.00 ± 0.16
$MDDM\text{-}A_{100}$	161.25 ± 87.26	2.95 ± 0.22	0.63 ± 0.70	0.05 ± 0.22	528.00	44.79 ± 7.24	84.14 ± 0.12
$MDDM\text{-}G_{25}$	383.31 ± 114.93	2.28 ± 0.45	0.65 ± 0.78	0.72 ± 0.45	232.00	15.12 ± 3.48	84.00 ± 0.16
$MDDM\text{-}G_{100}$	161.73 ± 89.49	2.94 ± 0.24	0.80 ± 0.73	0.06 ± 0.24	528.00	33.98 ± 5.43	84.14 ± 0.12
$MDDM\text{-}E_{25}$	383.31 ± 114.93	2.28 ± 0.45	0.65 ± 0.78	0.72 ± 0.45	232.00	36.13 ± 5.45	84.00 ± 0.16
$MDDM\text{-}E_{100}$	161.74 ± 89.49	2.94 ± 0.24	0.81 ± 0.73	0.06 ± 0.24	528.00	54.70 ± 7.28	84.14 ± 0.12
CUSUM	299.78 ± 52.29	3.00	0.40 ± 0.62	0.00	128.00	14.92 ± 4.30	84.08 ± 0.12
PageHinkley	677.32 ± 76.30	2.11 ± 0.55	0.93 ± 0.53	0.89 ± 0.55	136.00	12.35 ± 3.94	83.94 ± 0.13
DDM	703.59 ± 122.67	1.92 ± 0.72	2.33 ± 1.49	1.08 ± 0.72	136.00	15.87 ± 4.03	83.18 ± 1.61
EDDM	938.27 ± 106.60	0.35 ± 0.50	31.09 ± 18.14	2.65 ± 0.50	136.00	11.77 ± 3.29	83.12 ± 0.40
RDDM	406.50 ± 69.40	2.99 ± 0.10	2.15 ± 1.94	0.01 ± 0.10	7224.00	17.94 ± 4.35	84.05 ± 0.11
$ADWIN_{32}$	222.61 ± 57.00	2.99 ± 0.10	5.56 ± 2.57	0.01 ± 0.10	> 2157	> 128	84.12 ± 0.11
SeqDrift2	276.67 ± 91.10	2.92 ± 0.27	2.49 ± 0.97	0.08 ± 0.27	> 106125	38.55 ± 6.02	84.13 ± 0.14
$HDDM_{A\text{-test}}$	306.91 ± 107.78	2.91 ± 0.29	0.49 ± 0.69	0.09 ± 0.29	160.00	35.06 ± 6.18	84.09 ± 0.12
$HDDM_{W\text{-test}}$	242.43 ± 134.19	2.73 ± 0.44	1.59 ± 1.00	0.27 ± 0.44	160.00	34.88 ± 5.56	84.11 ± 0.13
No Detection	—	—	—	—	—	—	71.81 ± 0.17

Table B.8. Naive Bayes and DDMs against LED Data Stream with Gradual Drift

NB - LED Data Stream ($\zeta = 500$, $\Delta = 1000$)

Detector	Delay	TP	FP	FN	Memory	Runtime	Accuracy
$FHDDM_{25}$	347.53 ± 101.36	2.93 ± 0.32	0.02 ± 0.14	0.07 ± 0.32	232.00	12.80 ± 3.66	89.54 ± 0.05
$FHDDM_{100}$	220.40 ± 76.00	2.97 ± 0.22	0.03 ± 0.22	0.03 ± 0.22	528.00	17.52 ± 4.40	89.57 ± 0.04
FHDDMS	220.17 ± 76.03	2.97 ± 0.22	0.04 ± 0.24	0.03 ± 0.22	528.00	27.56 ± 5.83	89.57 ± 0.04
$FHDDMS_{add}$	249.92 ± 78.45	2.97 ± 0.22	0.02 ± 0.14	0.03 ± 0.22	152.00	13.07 ± 3.88	89.57 ± 0.04
$MDDM\text{-}A_{25}$	334.81 ± 99.48	2.93 ± 0.32	0.02 ± 0.14	0.07 ± 0.32	232.00	23.30 ± 4.77	89.54 ± 0.05
$MDDM\text{-}A_{100}$	210.31 ± 73.05	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	52.24 ± 8.31	89.57 ± 0.04
$MDDM\text{-}G_{25}$	333.88 ± 99.64	2.93 ± 0.32	0.02 ± 0.14	0.07 ± 0.32	232.00	17.87 ± 3.63	89.54 ± 0.05
$MDDM\text{-}G_{100}$	208.65 ± 73.05	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	34.05 ± 6.14	89.57 ± 0.04
$MDDM\text{-}E_{25}$	333.88 ± 99.64	2.93 ± 0.32	0.02 ± 0.14	0.07 ± 0.32	232.00	38.81 ± 6.01	89.54 ± 0.05
$MDDM\text{-}E_{100}$	208.61 ± 73.05	2.98 ± 0.14	0.03 ± 0.17	0.02 ± 0.14	528.00	53.35 ± 7.59	89.57 ± 0.04
CUSUM	300.61 ± 50.30	3.00	0.00	0.00	128.00	12.81 ± 3.51	89.57 ± 0.03
PageHinkley	559.27 ± 78.99	2.95 ± 0.26	0.04 ± 0.24	0.05 ± 0.26	136.00	10.19 ± 2.99	89.36 ± 0.04
DDM	446.23 ± 82.12	2.96 ± 0.20	0.33 ± 0.58	0.04 ± 0.20	136.00	14.64 ± 3.61	89.29 ± 1.15
EDDM	949.61 ± 68.94	0.70 ± 0.73	6.33 ± 1.96	2.30 ± 0.73	136.00	10.05 ± 3.07	88.32 ± 0.53
RDDM	321.80 ± 50.94	2.98 ± 0.14	0.61 ± 0.96	0.02 ± 0.14	7224.00	16.27 ± 3.65	89.63 ± 0.04
$ADWIN_{32}$	521.36 ± 238.56	2.36 ± 0.76	465.41 ± 12.53	0.64 ± 0.76	> 1406	94.45 ± 10.1	72.73 ± 0.44
SeqDrift2	445.33 ± 192.27	2.75 ± 0.46	278.82 ± 47.50	0.25 ± 0.46	> 14799	38.94 ± 6.05	76.54 ± 2.25
$HDDM_{A\text{-test}}$	295.85 ± 83.23	2.98 ± 0.20	0.17 ± 0.47	0.02 ± 0.20	160.00	31.83 ± 5.41	89.58 ± 0.05
$HDDM_{W\text{-test}}$	259.17 ± 87.21	2.95 ± 0.26	0.05 ± 0.22	0.05 ± 0.26	160.00	33.26 ± 5.24	89.56 ± 0.03
No Detection	—	—	—	—	—	—	72.60 ± 4.41

Table B.9. Top 20 Pairs with Highest Average Scores against SINE1 Data Stream with Abrupt Concept Drift

SINE1							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
PR+MDDM.G.25	14.30	34.00	0	0	8.77	276.97	0.919
PR+FHDDM.25	14.31	34.75	0	0	8.73	251.19	0.919
PR+FHDDMS	14.30	34.75	0	0	9.43	286.74	0.919
PR+HDDM.W.test	14.45	31.50	1	0	10.09	292.87	0.918
PR+HDDM.A.test	14.26	44.75	0	0	8.78	288.54	0.917
PR+MDDM.G.100	14.33	37.00	0	0	9.42	433.28	0.914
PR+FHDDM.100	14.28	45.00	0	0	9.38	221.04	0.912
PR+FHDDMS.add	14.25	51.50	0	0	8.73	213.10	0.910
NB+HDDM.A.test	15.47	43.25	0	0	33.70	396.51	0.901
NB+FHDDMS	15.51	35.75	0	0	34.34	410.32	0.899
NB+FHDDM.25	15.51	35.75	0	0	33.64	668.33	0.895
NB+MDDM.G.100	15.45	40.25	0	0	34.33	557.93	0.893
NB+HDDM.W.test	15.63	33.50	1	0	35.00	363.09	0.893
NB+MDDM.G.25	15.68	34.75	1	0	33.68	364.26	0.892
NB+FHDDM.100	15.40	47.50	0	0	34.29	382.19	0.892
NB+FHDDMS.add	15.37	51.50	0	0	33.64	340.88	0.891
PR+CUSUM	14.30	78.50	0	0	8.38	214.14	0.891
NB+ADWIN	15.44	65.50	0	0	42.13	530.82	0.878
PR+RDDM	14.19	92.75	0	0	71.40	238.89	0.872
NB+CUSUM	15.40	83.00	0	0	33.29	331.80	0.871

Table B.10. Top 20 Pairs with Highest Average Scores against SINE2 Data Stream with Abrupt Concept Drift

SINE2							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+HDDM.W.test	16.53	37.75	1	0	34.99	382.56	0.902
NB+MDDM.G.25	16.47	41.00	1	0	33.67	367.94	0.901
NB+FHDDMS	16.45	42.50	0	0	34.33	425.84	0.901
NB+MDDM.G.100	16.42	40.50	0	0	34.33	572.09	0.899
NB+FHDDM.25	16.45	42.50	0	0	33.64	581.64	0.897
NB+FHDDM.100	16.52	48.75	0	0	34.29	368.37	0.897
NB+FHDDMS.add	16.46	57.75	0	0	33.64	344.40	0.892
NB+ADWIN	16.48	65.50	0	0	42.14	545.16	0.880
NB+CUSUM	16.42	82.25	0	0	33.29	342.49	0.879
NB+HDDM.A.test	16.42	86.25	0	0	33.69	417.07	0.875
NB+RDDM	16.57	91.75	1	0	96.30	339.89	0.858
5-NN+MDDM.G.25	18.25	32.75	4	0	22.12	1930.62	0.827
NB+DDM	16.30	171.50	0	0	33.30	348.14	0.819
5-NN+FHDDMS.add	18.29	53.25	2	0	22.12	2589.02	0.812
5-NN+FHDDM.25	18.29	37.25	2	0	22.10	3163.99	0.811
5-NN+ADWIN	18.08	66.50	1	0	30.46	2740.17	0.810
5-NN+MDDM.G.100	18.34	39.25	6	0	22.71	2170.61	0.806
5-NN+FHDDM.100	18.30	46.75	7	0	22.71	1957.01	0.796
5-NN+HDDM.W.test	18.41	32.50	10	0	23.35	1442.68	0.792
HT+FHDDM.25	15.79	36.75	0	0	904.69	615.26	0.792

Table B.11. Top 20 Pairs with Highest Average Scores against MIXED Data Stream with Abrupt Concept Drift

MIXED							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+FHDDMS	14.73	41.25	0	0	41.08	527.97	0.905
NB+FHDDM.25	14.73	41.25	0	0	40.39	661.67	0.903
NB+MDDM.G.25	15.03	40.00	1	0	40.41	456.23	0.903
NB+MDDM.G.100	14.77	40.75	0	0	41.08	666.49	0.903
NB+FHDDM.100	14.72	47.25	0	0	41.04	453.32	0.902
NB+FHDDMS.add	14.76	51.50	0	0	40.39	437.32	0.901
NB+HDDM.W.test	15.19	37.00	2	0	41.73	426.86	0.897
PR+FHDDMS.add	18.37	51.50	0	0	10.52	266.60	0.884
PR+FHDDM.25	18.50	40.00	1	0	10.52	287.66	0.883
PR+FHDDM.100	18.55	50.25	1	0	11.17	253.27	0.880
PR+FHDDMS	18.63	40.00	2	0	11.22	298.76	0.879
PR+MDDM.G.25	18.53	39.25	2	0	10.56	283.84	0.877
NB+HDDM.A.test	15.05	63.00	1	0	40.43	457.17	0.875
PR+MDDM.G.100	18.60	42.75	2	0	11.21	426.90	0.875
NB+CUSUM	14.89	89.00	0	0	40.04	440.24	0.874
PR+HDDM.W.test	18.63	38.50	3	0	11.89	289.25	0.874
PR+ADWIN	18.38	73.50	0	0	19.02	483.42	0.864
PR+CUSUM	18.38	91.00	0	0	10.17	267.60	0.863
NB+DDM	14.71	152.50	0	0	40.05	452.67	0.836
5-NN+FHDDMS	18.38	39.75	4	0	30.78	2839.45	0.832

Table B.12. Top 20 Pairs with Highest Average Scores against STAGGER Data Stream with Abrupt Concept Drift

STAGGER							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+HDDM.W.test	5.22	28.50	0	0	20.90	530.86	0.871
NB+MDDM.G.25	5.18	30.50	0	0	19.57	525.04	0.871
PR+HDDM.W.test	5.37	28.50	0	0	12.04	511.91	0.871
PR+MDDM.G.25	5.36	30.50	0	0	10.70	493.85	0.871
PR+FHDDM.25	5.38	33.00	0	0	10.67	450.22	0.869
NB+FHDDMS	5.17	33.00	0	0	20.24	586.40	0.869
PR+FHDDMS	5.38	33.00	0	0	11.37	503.04	0.868
NB+FHDDM.25	5.17	33.00	0	0	19.54	933.63	0.866
NB+MDDM.G.100	5.13	45.50	0	0	20.24	778.57	0.863
NB+FHDDM.100	5.17	54.00	0	0	20.18	491.67	0.861
PR+FHDDM.100	5.36	54.00	0	0	11.31	386.37	0.861
PR+MDDM.G.100	5.38	45.50	0	0	11.37	747.91	0.861
NB+FHDDMS.add	5.16	63.50	0	0	19.53	400.10	0.859
NB+HDDM.A.test	5.21	89.50	0	0	19.59	514.64	0.857
PR+FHDDMS.add	5.37	63.50	0	0	10.67	385.41	0.857
NB+ADWIN	5.15	49.00	0	0	28.50	742.12	0.853
PR+ADWIN	5.35	49.00	0	0	19.62	711.07	0.852
PR+HDDM.A.test	5.39	109.50	0	0	10.73	488.28	0.851
HT+MDDM.G.25	5.20	34.00	0	0	110.01	911.10	0.839
HT+FHDDM.25	5.20	34.50	0	0	109.98	928.03	0.839

Table B.13. Top 20 Pairs with Highest Average Scores against CIRCLES Data Stream with Gradual Concept Drift (1)

CIRCLES							
$\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+FHDDM.100	14.76	93.33	0	0	34.26	483.57	0.894
NB+FHDDMS	14.76	93.33	0	0	34.30	544.76	0.893
NB+FHDDMS.add	14.70	176.00	0	0	33.61	444.63	0.889
NB+HDDM.W.test	14.76	351.67	1	0	34.96	515.06	0.882
NB+MDDM.G.100	14.81	353.33	1	0	34.28	689.64	0.880
NB+CUSUM	14.65	235.33	0	0	33.26	435.03	0.876
NB+HDDM.A.test	14.65	246.33	0	0	33.66	527.05	0.876
NB+MDDM.G.25	14.70	402.00	0	0	33.64	516.15	0.871
NB+FHDDM.25	14.68	420.67	0	0	33.60	763.01	0.865
NB+ADWIN	14.48	169.33	4	0	41.92	555.30	0.862
NB+RDDM	14.60	332.33	3	0	96.27	434.09	0.855
NB+DDM	14.47	563.67	0	0	33.27	444.58	0.846
NB+SeqDrift2	14.66	267.67	0	0	450.04	521.11	0.829
HT+HDDM.W.test	14.25	59.00	0	0	707.58	873.04	0.824
HT+FHDDM.100	14.16	69.00	0	0	706.87	789.33	0.824
HT+FHDDMS	14.15	66.67	0	0	706.92	862.48	0.824
HT+MDDM.G.100	14.17	56.67	1	0	699.13	1020.66	0.823
HT+FHDDMS.add	14.23	84.33	0	0	706.26	759.07	0.822
DS+CUSUM	24.40	145.67	1	0	21.57	628.94	0.816
HT+MDDM.G.25	14.20	245.00	0	0	707.36	845.90	0.816

Table B.14. Top 20 Pairs with Highest Average Scores against CIRCLES Data Stream with Gradual Concept Drift (2)

CIRCLES							
$\vec{w} = [2\ 4\ 3\ 1\ 0.5\ 0]$							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+FHDDM.100	14.76	93.33	0	0	34.26	483.57	0.861
NB+FHDDMS	14.76	93.33	0	0	34.30	544.76	0.861
HT+HDDM.W.test	14.25	59.00	0	0	707.58	873.04	0.851
HT+MDDM.G.100	14.17	56.67	1	0	699.13	1020.66	0.850
HT+FHDDMS	14.15	66.67	0	0	706.92	862.48	0.849
NB+FHDDMS.add	14.70	176.00	0	0	33.61	444.63	0.849
HT+FHDDM.100	14.16	69.00	0	0	706.87	789.33	0.848
HT+FHDDMS.add	14.23	84.33	0	0	706.26	759.07	0.842
NB+HDDM.W.test	14.76	351.67	1	0	34.96	515.06	0.836
NB+MDDM.G.100	14.81	353.33	1	0	34.28	689.64	0.835
HT+MDDM.G.25	14.20	245.00	0	0	707.36	845.90	0.832
HT+HDDM.A.test	14.21	133.67	0	0	724.25	831.57	0.828
HT+FHDDM.25	14.23	260.67	0	0	707.05	852.26	0.828
NB+HDDM.A.test	14.65	246.33	0	0	33.66	527.05	0.822
NB+CUSUM	14.65	235.33	0	0	33.26	435.03	0.819
HT+CUSUM	14.24	177.00	0	0	723.04	747.54	0.811
NB+MDDM.G.25	14.70	402.00	0	0	33.64	516.15	0.810
HT+ADWIN	14.69	115.33	6	0	552.21	664.04	0.808
NB+FHDDM.25	14.68	420.67	0	0	33.60	763.01	0.804
NB+ADWIN	14.48	169.33	4	0	41.92	555.30	0.802

Table B.15. Top 20 Pairs with Highest Average Scores against LED Data Stream with Gradual Concept Drift (1)

LED							
$\vec{w}$ = [1 1 1 1 1 1]							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
PR+FHDDM.100	15.12	292.67	0	0	74.96	3584.89	0.880
PR+FHDDMS	15.20	292.67	0	0	74.98	3650.60	0.879
PR+MDDM.G.100	15.08	294.67	0	0	74.96	3781.53	0.876
NB+FHDDM.100	11.66	197.33	0	0	171.90	4848.60	0.875
NB+MDDM.G.100	11.68	193.33	0	0	171.95	5169.65	0.875
NB+FHDDMS	11.66	197.33	0	0	171.95	5049.60	0.875
PR+HDDM.A.test	15.84	311.67	1	0	74.33	3345.66	0.873
NB+FHDDMS.add	11.65	242.67	0	0	171.22	5031.88	0.871
PR+FHDDMS.add	15.08	334.33	0	0	74.27	3560.79	0.871
PR+CUSUM	14.91	387.33	0	0	73.89	3525.70	0.865
PR+HDDM.W.test	15.01	474.67	0	0	75.63	3656.77	0.863
NB+CUSUM	11.47	310.67	0	0	170.86	5149.56	0.856
NB+HDDM.A.test	11.60	316.00	1	0	170.73	4649.49	0.856
HT+MDDM.G.100	11.68	193.33	0	0	177.64	10840.10	0.854
HT+FHDDM.100	11.66	197.33	0	0	177.60	10669.40	0.854
NB+HDDM.W.test	11.53	316.67	0	0	172.59	5023.58	0.854
HT+FHDDMS	11.66	197.33	0	0	177.64	10717.33	0.854
HT+FHDDMS.add	11.65	242.67	0	0	176.91	10613.79	0.851
NB+MDDM.G.25	11.49	337.67	0	0	171.28	5006.51	0.850
PR+MDDM.G.25	14.74	562.00	0	0	74.28	3620.82	0.849

Table B.16. Top 20 Pairs with Highest Average Scores against LED Data Stream with Gradual Concept Drift (2)

LED							
$\vec{w}$ = [1 1 2 3 4 4]							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
PR+FHDDM.100	15.12	292.67	0	0	74.96	3584.89	0.902
PR+FHDDMS	15.20	292.67	0	0	74.98	3650.60	0.902
PR+HDDM.A.test	15.84	311.67	1	0	74.33	3345.66	0.900
PR+MDDM.G.100	15.08	294.67	0	0	74.96	3781.53	0.900
PR+FHDDMS.add	15.08	334.33	0	0	74.27	3560.79	0.899
PR+CUSUM	14.91	387.33	0	0	73.89	3525.70	0.897
PR+HDDM.W.test	15.01	474.67	0	0	75.63	3656.77	0.895
PR+MDDM.G.25	14.74	562.00	0	0	74.28	3620.82	0.889
PR+DDM	14.79	568.67	0	0	73.94	3521.77	0.887
NB+FHDDM.100	11.66	197.33	0	0	171.90	4848.60	0.874
PR+RDDM	14.90	445.33	3	0	136.92	3588.59	0.872
NB+FHDDMS	11.66	197.33	0	0	171.95	5049.60	0.872
NB+MDDM.G.100	11.68	193.33	0	0	171.95	5169.65	0.872
NB+FHDDMS.add	11.65	242.67	0	0	171.22	5031.88	0.871
PR+FHDDM.25	13.95	581.33	0	1	74.29	5394.66	0.870
NB+HDDM.A.test	11.60	316.00	1	0	170.73	4649.49	0.865
NB+CUSUM	11.47	310.67	0	0	170.86	5149.56	0.865
NB+HDDM.W.test	11.53	316.67	0	0	172.59	5023.58	0.863
NB+MDDM.G.25	11.49	337.67	0	0	171.28	5006.51	0.862
NB+DDM	11.26	418.33	0	0	170.87	4938.92	0.859

Table B.17. Top 20 Pairs with Highest Average Scores against ADULT Data Stream with Gradual Concept Drift (1)

ADULT							
$\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+MDDM.G.100	21.93	121.25	5	0	193.45	1769.83	0.879
NB+HDDM.W.test	21.75	139.75	7	0	193.64	1099.17	0.876
NB+FHDDMS.add	22.06	170.25	1	0	193.00	1807.25	0.876
NB+FHDDM.100	22.50	136.25	4	0	193.33	1506.83	0.872
NB+FHDDMS	22.52	134.50	4	0	193.38	1565.01	0.872
NB+MDDM.G.25	21.69	217.00	2	0	192.95	1776.04	0.871
NB+FHDDM.25	21.63	220.75	0	0	193.01	2074.97	0.871
NB+HDDM.A.test	21.75	214.50	1	0	193.00	1907.63	0.869
NB+ADWIN	20.78	219.00	4	0	201.24	1991.38	0.869
NB+CUSUM	21.31	263.00	0	0	192.66	1800.52	0.868
NB+RDDM	21.01	317.75	0	0	255.66	1802.01	0.854
NB+SeqDrift2	20.82	203.00	4	0	514.29	1874.31	0.852
NB+DDM	20.49	412.25	0	0	192.67	1837.19	0.850
NB+PageHinkley	20.43	489.00	0	0	192.72	1878.34	0.840
DS+CUSUM	25.50	279.00	0	0	107.95	2997.54	0.836
DS+FHDDM.100	25.82	192.25	5	0	108.52	2283.50	0.835
DS+FHDDMS.add	25.67	239.75	1	0	108.24	2913.34	0.833
DS+MDDM.G.100	26.12	180.50	7	0	108.30	1917.94	0.830
DS+FHDDM.25	25.71	216.50	9	0	107.55	1865.93	0.829
DS+HDDM.A.test	25.38	279.75	2	0	108.16	2763.26	0.824

Table B.18. Top 20 Pairs with Highest Average Scores against ADULT Data Stream with Gradual Concept Drift (2)

ADULT							
$\vec{w} = [2\ 1.5\ 3\ 2\ 1.5\ 1]$							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+FHDDMS.add	22.06	170.25	1	0	193.00	1807.25	0.887
NB+MDDM.G.100	21.93	121.25	5	0	193.45	1769.83	0.886
NB+FHDDM.25	21.63	220.75	0	0	193.01	2074.97	0.885
NB+MDDM.G.25	21.69	217.00	2	0	192.95	1776.04	0.883
NB+CUSUM	21.31	263.00	0	0	192.66	1800.52	0.883
NB+HDDM.A.test	21.75	214.50	1	0	193.00	1907.63	0.883
NB+ADWIN	20.78	219.00	4	0	201.24	1991.38	0.879
NB+HDDM.W.test	21.75	139.75	7	0	193.64	1099.17	0.879
NB+FHDDM.100	22.50	136.25	4	0	193.33	1506.83	0.877
NB+FHDDMS	22.52	134.50	4	0	193.38	1565.01	0.877
NB+RDDM	21.01	317.75	0	0	255.66	1802.01	0.872
NB+DDM	20.49	412.25	0	0	192.67	1837.19	0.869
NB+SeqDrift2	20.82	203.00	4	0	514.29	1874.31	0.864
NB+PageHinkley	20.43	489.00	0	0	192.72	1878.34	0.861
DS+CUSUM	25.50	279.00	0	0	107.95	2997.54	0.849
DS+FHDDMS.add	25.67	239.75	1	0	108.24	2913.34	0.843
DS+FHDDM.100	25.82	192.25	5	0	108.52	2283.50	0.839
DS+HDDM.A.test	25.38	279.75	2	0	108.16	2763.26	0.831
DS+ADWIN	25.21	291.75	6	0	116.36	2843.82	0.827
DS+MDDM.G.100	26.12	180.50	7	0	108.30	1917.94	0.827

Table B.19. Top 20 Pairs with Highest Average Scores against NURSERY Data Stream with Gradual Concept Drift (1)

NURSERY							
$\vec{w}$ = [1 1 1 1 1 1]							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+HDDM.W.test	11.43	114.00	1	0	54.07	803.03	0.885
NB+FHDDM.100	11.86	124.50	0	0	53.35	742.64	0.884
NB+HDDM.A.test	11.57	143.25	0	0	52.74	784.58	0.884
NB+FHDDMS	11.86	124.50	0	0	53.40	810.99	0.883
NB+MDDM.G.100	11.95	112.50	0	0	53.39	926.60	0.883
NB+ADWIN	10.03	118.50	9	0	60.52	717.20	0.883
NB+FHDDMS.add	11.64	145.25	0	0	52.70	730.63	0.882
PR+MDDM.G.100	14.34	112.00	0	0	19.83	681.21	0.879
PR+FHDDM.100	14.21	130.50	0	0	19.79	467.98	0.878
PR+FHDDMS	14.22	130.50	0	0	19.83	543.92	0.877
PR+FHDDMS.add	14.10	145.25	0	0	19.14	457.64	0.877
NB+CUSUM	10.79	216.50	0	0	52.36	710.51	0.876
PR+HDDM.A.test	14.18	160.50	1	0	19.18	530.28	0.876
NB+MDDM.G.25	10.90	214.25	0	0	52.75	790.91	0.874
PR+HDDM.W.test	14.06	145.75	0	0	20.51	565.36	0.874
PR+MDDM.G.25	13.92	167.00	0	0	19.18	529.83	0.874
PR+FHDDM.25	13.71	196.75	0	0	19.15	521.31	0.872
NB+DDM	10.26	271.25	0	0	52.38	702.35	0.871
NB+FHDDM.25	10.91	216.50	0	0	52.72	1057.87	0.871
PR+CUSUM	13.50	226.75	0	0	18.80	455.76	0.869

Table B.20. Top 20 Pairs with Highest Average Scores against NURSERY Data Stream with Gradual Concept Drift (2)

NURSERY							
$\vec{w}$ = [3 2 2 1 0 0.5]							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
HT+HDDM.A.test	10.40	75.00	0	0	809.15	1428.57	0.892
HT+MDDM.G.100	10.37	121.50	0	0	810.61	1552.94	0.876
HT+FHDDM.100	10.39	123.75	0	0	809.71	1369.79	0.875
HT+FHDDMS	10.39	123.75	0	0	809.76	1436.77	0.875
HT+HDDM.W.test	9.93	135.50	1	0	814.05	1418.22	0.875
HT+RDDM	9.35	180.25	2	0	872.59	1342.95	0.874
HT+FHDDMS.add	10.22	139.00	0	0	809.90	1339.59	0.873
HT+SeqDrift2	8.65	204.00	5	0	1082.34	1293.68	0.870
HT+FHDDM.25	9.45	222.50	0	0	817.49	1421.29	0.868
HT+MDDM.G.25	9.74	199.75	0	0	816.19	1369.16	0.867
HT+CUSUM	9.54	213.00	0	0	817.08	1346.46	0.867
NB+ADWIN	10.03	118.50	9	0	60.52	717.20	0.867
NB+HDDM.W.test	11.43	114.00	1	0	54.07	803.03	0.865
HT+DDM	9.26	241.50	0	0	819.48	1313.47	0.864
NB+HDDM.A.test	11.57	143.25	0	0	52.74	784.58	0.862
NB+MDDM.G.100	11.95	112.50	0	0	53.39	926.60	0.861
NB+FHDDM.100	11.86	124.50	0	0	53.35	742.64	0.861
NB+FHDDMS	11.86	124.50	0	0	53.40	810.99	0.860
NB+FHDDMS.add	11.64	145.25	0	0	52.70	730.63	0.858
NB+SeqDrift2	9.66	204.75	7	0	366.30	743.70	0.856

Table B.21. Top 20 Pairs with Highest Average Scores against SHUTTLE Data Stream with Gradual Concept Drift (1)

SHUTTLE							
$\vec{w} = [1\ 1\ 1\ 1\ 1\ 1]$							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
DS+ADWIN	8.66	131.00	4	0	80.65	1845.22	0.863
DS+FHDDM.100	10.62	129.75	0	0	73.26	1782.27	0.861
DS+MDDM.G.100	9.85	117.25	2	0	73.21	1953.67	0.861
DS+FHDDMS	10.62	129.75	0	0	73.31	1803.92	0.861
DS+FHDDMS.add	10.39	151.50	0	0	72.59	1749.63	0.859
NB+ADWIN	7.91	124.00	5	0	136.91	1718.73	0.859
DS+HDDM.A.test	10.00	153.50	1	0	72.57	1795.26	0.858
NB+FHDDM.100	10.68	136.75	0	0	129.56	1585.81	0.858
NB+FHDDMS	10.14	136.75	1	0	129.54	1654.46	0.858
DS+HDDM.W.test	9.94	189.00	0	0	73.92	1802.22	0.858
NB+MDDM.G.100	10.19	123.50	1	0	129.57	1800.45	0.857
NB+HDDM.W.test	9.44	190.25	1	0	130.20	1620.01	0.856
DS+MDDM.G.25	9.95	200.00	0	0	72.58	1795.46	0.856
DS+CUSUM	9.89	204.50	0	0	72.21	1730.89	0.856
NB+FHDDMS.add	10.38	164.00	0	0	128.89	1631.30	0.855
NB+MDDM.G.25	9.19	209.00	1	0	128.85	1641.12	0.854
NB+HDDM.A.test	10.31	166.25	0	0	128.92	1617.18	0.853
NB+CUSUM	9.75	215.25	0	0	128.52	1564.18	0.852
DS+RDDM	9.99	195.75	0	0	135.20	1752.44	0.851
DS+FHDDM.25	9.65	227.25	0	0	72.52	1815.13	0.849

Table B.22. Top 20 Pairs with Highest Average Scores against SHUTTLE Data Stream with Gradual Concept Drift (2)

SHUTTLE							
$\vec{w} = [3\ 2\ 1\ 2\ 1\ 1]$							
Pair	Error-rate	Delay	FP	FN	Memory	Runtime	Score
NB+ADWIN	7.91	124.00	5	0	136.91	1718.73	0.849
DS+ADWIN	8.66	131.00	4	0	80.65	1845.22	0.847
DS+MDDM.G.100	9.85	117.25	2	0	73.21	1953.67	0.843
5-NN+ADWIN	3.47	91.00	4	0	56.47	6684.94	0.840
DS+FHDDM.100	10.62	129.75	0	0	73.26	1782.27	0.839
DS+FHDDMS	10.62	129.75	0	0	73.31	1803.92	0.838
NB+MDDM.G.100	10.19	123.50	1	0	129.57	1800.45	0.838
NB+FHDDMS	10.14	136.75	1	0	129.54	1654.46	0.838
NB+HDDM.W.test	9.44	190.25	1	0	130.20	1620.01	0.837
DS+HDDM.A.test	10.00	153.50	1	0	72.57	1795.26	0.837
NB+FHDDM.100	10.68	136.75	0	0	129.56	1585.81	0.836
DS+FHDDMS.add	10.39	151.50	0	0	72.59	1749.63	0.836
DS+HDDM.W.test	9.94	189.00	0	0	73.92	1802.22	0.836
NB+MDDM.G.25	9.19	209.00	1	0	128.85	1641.12	0.835
NB+FHDDMS.add	10.38	164.00	0	0	128.89	1631.30	0.834
DS+MDDM.G.25	9.95	200.00	0	0	72.58	1795.46	0.834
DS+CUSUM	9.89	204.50	0	0	72.21	1730.89	0.833
DS+RDDM	9.99	195.75	0	0	135.20	1752.44	0.831
NB+HDDM.A.test	10.31	166.25	0	0	128.92	1617.18	0.830
NB+RDDM	9.63	199.75	1	0	191.51	1570.90	0.830

Appendix C

Theoretical Proofs

C.1 FHDDM Bounds

This section provides the theoretical proofs on the false positive and the false negative bounds for the Fast Hoeffding Drift Detection Method (FHDDM).

Assumptions

Assume the sliding window W with a size n at time t, which is represented by $W^t \equiv [p_1^t, \ldots, p_n^t]$ where p_i^t is the i^{th} input. For this sliding window we have:

- The empirical mean of elements inside the sliding window at time t:

$$\mu^t \equiv \hat{\mu}^t = \frac{1}{n} \sum_{k=1}^{n} p_k^t$$

- The maximum mean observed so far:

$$\mu^m \equiv \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{t}\right) = \max\left(\hat{\mu}^1, \ldots, \hat{\mu}^t\right)$$

Please note that we use the notations of $\hat{\mu}$ and $\max\left(\{\hat{\mu}^i\}_{i=1}^{t}\right)$ in our proofs of the bounds on False Positive and False Negative for the Fast Hoeffding Drift Detection Methods (FHDDMs).

C.1.1 False Positive Bound

We prove there is an upper bound, of at most δ, for the False Positive of Hoeffding Drift Detection Method (FHDDM). Our demonstration is inspired by the work of Bifet and Gavalda (2007).

Consequently, we must demonstrate that

$$\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{t}\right)\right| \geq \varepsilon_d\right) \leq \delta.$$

Proof. By the probability approximately correct (PAC) learning model (Valiant, 1984), we obtain:

$$\lim_{t\to\infty} \Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{t}\right)\right| \geq \varepsilon_d\right) \to \Pr\left(\left|\hat{\mu} - E\left(\hat{\mu}\right)\right| \geq \varepsilon_d\right),$$

as the theorem implies that the maximum of the empirical means tends toward its expectation. If we apply Hoeffding's inequality to the right member of the previous equation we have:

$$\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| \geq \varepsilon_d\right) \leq 2\exp\left(-2{\varepsilon_d}^2 n\right).$$

In order to have:

$$\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| \geq \varepsilon_d\right) \leq \delta,$$

we must have:

$$2\exp\left(-2{\varepsilon_d}^2 n\right) \leq \delta \Rightarrow \varepsilon_d = \sqrt{\frac{1}{2n}\ln\frac{2}{\delta}} \equiv \sqrt{\frac{1}{2n}\ln\frac{1}{\delta'}} \therefore \delta' \triangleq \frac{\delta}{2} < \delta.$$

The last result is exact. If we do not make use of the absolute value, as in Section 5, we have:

$$\exp\left(-2{\varepsilon_d}^2 n\right) \leq \delta \Rightarrow \varepsilon_d = \sqrt{\frac{1}{2n}\ln\frac{1}{\delta}}.$$

In practice, both results are the same since they both depend on the value of the parameter delta.

□

The demonstration is similar for the long and the short sliding windows of the Stacking Fast Hoeffding Drift Detection Method (FHDDMS).

C.1.2 False Negative Bound

We prove there is an upper bound, of at most δ, for the False Negative of Hoeffding Drift Detection Method (FHDDM). Our demonstration is inspired by the work of Pears et al. (2014).

As in (Pears et al., 2014), we want to demonstrate that:

$$\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| \geq \varepsilon_d\right) > 1 - \delta,$$

which is equivalent to proof that the alternative hypothesis:

$$\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| < \varepsilon_d\right) > 1 - \delta$$

is false.

Proof. By considering the PAC learning model and the probability rule of $\Pr(Z < z) = 1 - \Pr(Z \geq z)$, we have:

$$\begin{aligned}\lim_{t\to\infty} \Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{t}\right)\right| < \varepsilon_d\right) &\to \Pr\left(\left|\hat{\mu} - E\left(\hat{\mu}\right)\right| < \varepsilon_d\right) \\ &= 1 - \Pr\left(\left|\hat{\mu} - E\left(\hat{\mu}\right)\right| \geq \varepsilon_d\right).\end{aligned}$$

If we apply Hoeffding's inequality to the right member of the previous equation we obtain:

$$\begin{aligned}&\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| < \varepsilon_d\right) \leq 1 - 2\exp\left(-2{\varepsilon_d}^2 n\right) \\ &\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| < \varepsilon_d\right) \leq 1 - \delta.\end{aligned}$$

This contradicts the alternative hypothesis above, which means our assumption is false. Consequently, the alternative assumption of:

$$\Pr\left(\left|\hat{\mu} - \max\left(\left\{\hat{\mu}^i\right\}_{i=1}^{\infty}\right)\right| \geq \varepsilon_d\right) > 1 - \delta$$

is true. This implies in turn that the probability of false negative is $< \delta$ as stated in (Pears et al., 2014).

□

The demonstration is analogous for the long and the short sliding windows of the Stacking Fast Hoeffding Drift Detection Method (FHDDMS), as well as for the McDiarmid Drift Detection Methods (MDDMs).

References

Davide Anguita. Smart adaptive systems: State of the art and future directions of research. In *Proceedings of the European Symposium on Intelligent Technologies, Hybrid Systems and their Implementation on Smart Adaptive Systems-EUNITE*, pages 1–4, 2001.

Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013.

Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. Early drift detection method. In *4th International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86, 2006.

Roberto SM Barros, Danilo RL Cabral, Silas GTC Santos, et al. Rddm: Reactive drift detection method. *Expert Systems with Applications*, 2017.

Michèle Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

Sergei Bernstein. The theory of probabilities, 1946.

Albert Bifet. Adaptive learning and mining for data streams and frequent patterns. *ACM SIGKDD Explorations Newsletter*, 11(1):55–56, 2009.

Albert Bifet. Classifier concept drift detection and the illusion of progress. In *International Conference on Artificial Intelligence and Soft Computing*, pages 715–725. Springer, 2017.

Albert Bifet and Eibe Frank. Sentiment knowledge discovery in twitter streaming data. In *International Conference on Discovery Science*, pages 1–15. Springer, 2010.

Albert Bifet and Ricard Gavalda. Kalman filters and adaptive windows for learning in data streams. In *International Conference on Discovery Science*, pages 29–40. Springer, 2006.

Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, volume 7, page 2007. SIAM, 2007.

Albert Bifet and Richard Kirkby. Data stream mining a practical approach. 2009.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148. ACM, 2009.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010a.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. *Advances in knowledge discovery and data mining*, pages 299–310, 2010b.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Data stream mining a practical approach. 2011.

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

Christopher M Bishop. *Pattern recognition and machine learning*. Springer-Verlag New York, 2006.

Michaela Black and Ray Hickey. Classification of customer call data in the presence of concept drift and noise. In *Soft-Ware 2002: Computing in an Imperfect World*, pages 74–87. Springer, 2002.

Michaela Black and Ray Hickey. Detecting and adapting to concept drift in bioinformatics. In *Knowledge Exploration in Life Science Informatics*, pages 161–168. Springer, 2004.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

David M Blei and John D Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pages 113–120. ACM, 2006.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Richard J Bolton and David J Hand. Statistical fraud detection: A review. *Statistical science*, pages 235–249, 2002.

Pavel B Brazdil. Multistrategy learning. In *Encyclopedia of the Sciences of Learning*, pages 2396–2399. Springer, 2012.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Jason Catlett. *Megainduction: machine learning on very large databases*. Basser Department of Computer Science, University of Sydney, 1991.

Jason Catlett. Statlog (shuttle) data set, 2002.

Marion G Ceruti. Data management challenges and development for military information systems. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1059–1068, 2003.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

Darryl Charles, A Kerr, M McNeill, M McAlister, M Black, J Kcklich, A Moore, and K Stringer. Player-centred game design: Player modelling and adaptive digital games. In *Proceedings of the digital games research conference*, volume 285, page 00100, 2005.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Paulo Cortez and Aníbal de Jesus Raimundo Morais. A data mining approach to predict forest fires using meteorological data. 2007.

Fernando Crespo and Richard Weber. A methodology for dynamic data mining based on fuzzy clustering. *Fuzzy Sets and Systems*, 150(2):267–284, 2005.

Roberto Souto Maior de Barros and Silas Garrido T de Carvalho Santos. A large-scale comparison of concept drift detectors. *Information Sciences*, 2018.

Roberto Souto Maior de Barros, Juan Isidro González Hidalgo, and Danilo Rafael de Lima Cabral. Wilcoxon rank sum test drift detector. *Neurocomputing*, 275:1954–1963, 2018.

Paul De Bra, Ad Aerts, Bart Berden, Barend De Lange, Brendan Rousseau, Tomi Santic, David Smits, and Natalia Stash. AHA! the adaptive hypermedia architecture. In *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, pages 81–84. ACM, 2003.

Danilo Rafael de Lima Cabral and Roberto Souto Maior de Barros. Concept drift detection based on fisher's exact test. *Information Sciences*, 442:220–234, 2018.

Andrea De Mauro, Marco Greco, and Michele Grimaldi. A formal definition of big data based on its essential features. *Library Review*, 65(3):122–135, 2016.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM, 2000.

Pedro M Domingos and Geoff Hulten. Catching up with the data: Research issues in mining data streams. In *DMKD*, 2001.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg. Applying temporal dependence to detect changes in streaming data. *Applied Intelligence*, pages 1–19, 2018a.

Quang-Huy Duong, Heri Ramampiaro, Kjetil Nørvåg, Philippe Fournier-Viger, and Thu-Lan Dam. High utility drift detection in quantitative data streams. *Knowledge-Based Systems*, 2018b.

Florentino Fdez-Riverola, Eva Lorenzo Iglesias, Fernando Díaz, José Ramon Méndez, and Juan M Corchado. Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications*, 33(1):36–48, 2007.

Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to Boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 209–217. ACM, 1998.

Yoav Freund, Robert Schapire, and N Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé 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.

Mohamed Medhat Gaber, João Gama, Shonali Krishnaswamy, João Bártolo Gomes, and Frederic Stahl. Data stream mining in ubiquitous environments: state-of-the-art and current directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(2):116–138, 2014a.

Mohamed Medhat Gaber, João Bártolo Gomes, and Frederic Stahl. Pocket data mining. *Big Data on Small Devices. Series: Studies in Big Data*, 2014b.

Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer, 2004.

João Gama, Ricardo Fernandes, and Ricardo Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1):23–45, 2006.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM, 2009.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine learning*, 90(3):317–346, 2013.

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

John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007:1–16, 2012.

Jing Gao, Wei Fan, Jiawei Han, and Philip 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*, pages 3–14. SIAM, 2007.

Stephen R Garner et al. Weka: The Waikato environment for knowledge analysis. In *Proceedings of the New Zealand computer science research students conference*, pages 57–64. Citeseer, 1995.

Raffaella Giacomini and Barbara Rossi. Detecting and predicting forecast breakdowns. *The Review of Economic Studies*, 76(2):669–705, 2009.

Christophe Giraud-Carrier. A note on the utility of incremental learning. *AI Communications*, 13(4):215–223, 2000.

Joao Bártolo Gomes, Clifton Phua, and Shonali Krishnaswamy. Where will you go? mobile data mining for next place prediction. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 146–158. Springer, 2013.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The Weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

Richard W Hamming. Error detecting and error correcting codes. *Bell Labs Technical Journal*, 29(2):147–160, 1950.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Michael Bonnell Harries, Claude Sammut, and Kim Horn. Extracting hidden context. *Machine learning*, 32(2):101–126, 1998.

Constantinos S Hilas. Designing an expert system for fraud detection in private telecommunications networks. *Expert Systems with applications*, 36(9):11559–11569, 2009.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Karen L Hollis. Adaptation and learning. In *Encyclopedia of the Sciences of Learning*, pages 95–98. Springer, 2012.

Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

Kuo-Wei Hsu. A theoretical analysis of why hybrid ensembles work. *Computational intelligence and neuroscience*, 2017:1–12, 2017.

David Tse Jung Huang, Yun Sing Koh, Gillian Dobbie, and Russel Pears. Detecting volatility shift in data streams. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 863–868. IEEE, 2014.

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

Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106. ACM, 2001.

Geoff Hulten, Pedro Domingos, and Laurie Spencer. Mining massive data streams. *The Journal of Machine Learning Research*, 2005.

Wayne Iba and Pat Langley. Induction of one-level decision trees. In *Proceedings of the 9th international conference on machine learning*, pages 233–240, 1992.

Bilal Idiri and Aldo Napoli. The automatic identification system of maritime accident risk using rule-based reasoning. In *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pages 125–130. IEEE, 2012.

Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data mining and knowledge discovery*, 23(1):128–168, 2011.

Chris Jermaine. Data mining for multiple antibiotic resistance. 2008.

Mark G Kelly, David J Hand, and Niall M Adams. The impact of changing populations on classifier performance. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 367–371. ACM, 1999.

Jungwon Kim, Peter J Bentley, Uwe Aickelin, Julie Greensmith, Gianni Tedesco, and Jamie Twycross. Immune system approaches to intrusion detection–a review. *Natural computing*, 6 (4):413–466, 2007.

Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

Ralf Klinkenberg. Meta-learning, model selection, and example selection in machine learning domains with concept drift. In *LWA*, volume 2005, pages 164–171, 2005.

Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *ICML*, pages 487–494, 2000.

Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning drifting concepts. In *Proceedings of AAAI-98/ICML-98 Workshop Learning for Text Categorization*, pages 33–40, 1998.

Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 202–207, 1996.

Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53 (4):89–97, 2010.

Georg Krempl, Indre Žliobaite, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, Sonja Sievi, Myra Spiliopoulou, et al. Open challenges for data stream mining research. *ACM SIGKDD explorations newsletter*, 16(1):1–10, 2014.

Miroslav Kubat and Gerhard Widmer. Adapting to drift in continuous domains. In *European Conference on Machine Learning*, pages 307–310. Springer, 1995.

P Ravi Kumar and Vadlamani Ravi. Bankruptcy prediction in banks and firms via statistical and intelligent techniques–a review. *European Journal of Operational Research*, 180(1):1–28, 2007.

Ludmila I Kuncheva. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In *2nd Workshop SUEMA*, volume 2008, pages 5–10, 2008.

Carsten Lanquillon. *Enhancing text classification to improve information filtering*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, Universitätsbibliothek, 2001.

Neal Lathia, Stephen Hailes, and Licia Capra. knn cf: a temporal social network. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, pages 227–234. ACM, 2008.

Andreas D Lattner, Andrea Miene, Ubbo Visser, and Otthein Herzog. Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In *Robot Soccer World Cup*, pages 118–129. Springer, 2005.

Mihai M Lazarescu, Svetha Venkatesh, and Hung H Bui. Using multiple windows to track concept drift. *Intelligent data analysis*, 8(1):29–59, 2004.

Lin Liao, Donald J Patterson, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, 2007.

Viktor Losing, Barbara Hammer, and Heiko Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.

Jie Luo, Andrzej Pronobis, Barbara Caputo, and Patric Jensfelt. Incremental learning for place recognition in dynamic environments. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 721–728. IEEE, 2007.

Bruno Iran Ferreira Maciel, Silas Garrido Teixeira Carvalho Santos, and Roberto Souto Maior Barros. A lightweight concept drift detection ensemble. In *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*, pages 1061–1068. IEEE, 2015.

Azhar Mahmood, Ke Shi, Shaheen Khatoon, and Mi Xiao. Data mining techniques for wireless sensor networks: A survey. *International Journal of Distributed Sensor Networks*, 2013, 2013.

Mohammad Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. A multi-partition multi-chunk ensemble technique to classify concept-drifting data streams. *Advances in Knowledge Discovery and Data Mining*, pages 363–375, 2009.

Oleksiy Mazhelis and Seppo Puuronen. Comparing classifier combining techniques for mobile-masquerader detection. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 465–472. IEEE, 2007.

Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1): 148–188, 1989.

Ryszard S Michalski. *Multistrategy Learning: A Special Issue of Machine Learning*, volume 240. Kluwer Academic Publisher, 1993.

Jun-Ki Min and Sung-Bae Cho. Activity recognition based on wearable sensors using selection/-fusion hybrid ensemble. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 1319–1324. IEEE, 2011.

Tom M Mitchell. Machine learning, 1997.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

Joao Pedro Carvalho Leal Mendes Moreira. *Travel time prediction for the planning of mass transit companies: a machine learning approach*. PhD thesis, Universidade do Porto, 2008.

Fernando Mourao, Leonardo Rocha, Renata Araújo, Thierson Couto, Marcos Gonçalves, and Wagner Meira Jr. Understanding temporal aspects in document classification. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 159–170. ACM, 2008.

Kyriakos Mouratidis and Dimitris Papadias. Continuous nearest neighbor queries over sliding windows. *IEEE Transactions on Knowledge and Data Engineering*, 19(6):789–803, 2007.

Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and information systems*, 45(3):535–569, 2015.

Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *International conference on discovery science*, pages 264–269. Springer, 2007.

Olubukola Olaitan. Scut-ds: Methodologies for learning in imbalanced data streams. Master's thesis, Université d'Ottawa/University of Ottawa, 2018.

M Kehinde Olorunnimbe, Herna L Viktor, and Eric Paquet. Intelligent adaptive ensembles for data stream mining: a high return on investment approach. In *International Workshop on New Frontiers in Mining Complex Patterns*, pages 61–75. Springer, 2015.

M Kehinde Olorunnimbe, Herna L Viktor, and Eric Paquet. Dynamic adaptation of online ensembles for drifting data streams. *Journal of Intelligent Information Systems*, pages 1–23, 2017.

Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

ES Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470, 2007.

Russel Pears, Sripirakas Sakthithasan, and Yun Sing Koh. Detecting concept change in dynamic data streams. *Machine Learning*, 97(3):259–293, 2014.

Ali Pesaranghader and Herna L Viktor. Fast hoeffding drift detection method for evolving data streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 96–111. Springer, 2016.

Ali Pesaranghader, Herna L Viktor, and Eric Paquet. A framework for classification in data streams using multi-strategy learning. In *International Conference on Discovery Science*, pages 341–355. Springer, 2016.

Ali Pesaranghader, Herna Viktor, and Eric Paquet. Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams. *Machine Learning*, 2018a. ISSN 1573-0565. doi: 10.1007/s10994-018-5719-z.

Ali Pesaranghader, Herna L Viktor, and Eric Paquet. Mcdiarmid drift detection methods for evolving data streams. In *International Joint Conference on Neural Networks*. IEEE, 2018b.

Nhan Duc Phung, Mohamed Medhat Gaber, and Uwe Rohm. Resource-aware online data mining in wireless sensor networks. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 139–146. IEEE, 2007.

Norman Poh, Rita Wong, Josef Kittler, and Fabio Roli. Challenges and research directions for adaptive biometric recognition systems. In *International Conference on Biometrics*, pages 753–764. Springer, 2009.

Michael J Procopio, Jane Mulligan, and Greg Grudic. Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments. *Journal of Field Robotics*, 26(2):145–175, 2009.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Parisa Rashidi and Diane 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.

SW Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1): 97–101, 2000.

Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

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

Antonin Rozsypal and Miroslav Kubat. Association mining in time-varying domains. *Intelligent Data Analysis*, 9(3):273–288, 2005.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

Reza Sadoddin and Ali A Ghorbani. Real-time alert correlation using stream data mining techniques. In *AAAI*, pages 1731–1737, 2008.

Reza Sadoddin and Ali A Ghorbani. An incremental frequent structure mining framework for real-time alert correlation. *computers & security*, 28(3-4):153–173, 2009.

Sripirakas Sakthithasan, Russel Pears, and Yun Sing Koh. One pass concept change detection for data streams. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 461–472. Springer, 2013.

Ricardo Menezes Salgado, Joaquim JF Pereira, Takaaki Ohishi, Rosangela Ballini, CAM Lima, and Fernando J Von Zuben. A hybrid ensemble model applied to the short-term load forecasting problem. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 2627–2634. IEEE, 2006.

Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, 1997.

Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Carlos Roberto Sanquetta, Jaime Wojciechowski, Ana Paula Dalla Corte, Aurélio Lourenço Rodrigues, and Greyce Charllyne Benedet Maas. On the use of data mining for estimating carbon storage in the trees. *Carbon balance and management*, 8(1):1–9, 2013.

Jeffrey C Schlimmer and Richard H Granger Jr. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

Raquel Sebastião, João Gama, and Teresa Mendonça. Fading histograms in detecting distribution and concept changes. *International Journal of Data Science and Analytics*, pages 1–30, 2017.

Jeffrey W Seifert. Data mining and homeland security: An overview. DTIC Document, 2007.

BW Sillverman, MC Jones, E Fix, and JL Hodges. An important contribution to nonparametric discriminant analysis and density estimation. *International Statistical Review*, 57(3):233–247, 1951.

Marie-Odette St-Hilaire and Melita Hadzagic. Information mining technologies to enable discovery of actionable intelligence to facilitate maritime situational awareness: I-mine. Technical report, DTIC Document, 2013.

Internet World Stats. World internet users statistics and 2018 world population statistics. `http://internetworldstats.com/stats.htm`, 2018. (Last access 15-May-2018).

W Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382. ACM, 2001.

Tae Kyung Sung, Namsik Chang, and Gunhee Lee. Dynamics of modeling in data mining: interpretive approach to bankruptcy prediction. *Journal of management information systems*, 16 (1):63–85, 1999.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

Bhavani M Thuraisingham. Data mining for security applications. In *Intelligence and security informatics*, pages 1–3. Springer, 2006.

Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. Dynamic integration of classifiers for handling concept drift. *Information fusion*, 9(1):56–68, 2008.

Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Antanas Verikas, Zivile Kalsyte, Marija Bacauskiene, and Adas Gelzinis. Hybrid and ensemble-based soft computing techniques in bankruptcy prediction: a survey. *Soft Computing*, 14(9): 995–1010, 2010.

Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

Chong Wang, David Blei, and David Heckerman. Continuous time dynamic topic models. *arXiv preprint arXiv:1206.3298*, 2012.

Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *Machine learning: ECML-93*, pages 227–243. Springer, 1993.

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

Janusz Wojtusiak. Rule learning. In *Encyclopedia of the Sciences of Learning*, pages 2082–2083. Springer, 2012.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Gil Zeira, Oded Maimon, Mark Last, and Lior Rokach. Change detection in classification models induced from time series data. *Data mining in time series databases*, 57:101, 2004.

Jun Zhou, Li Cheng, Walter F Bischof, et al. Prediction and change detection in sequential data for interactive applications. In *AAAI*, pages 805–810, 2008.

S.K. Zhou, H. Greenspan, and D. Shen. *Deep Learning for Medical Image Analysis*. Elsevier and MICCAI Society book series. Elsevier Science & Technology Books, 2017. ISBN 9780128104088.

Indrė Žliobaitė. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010.

Indrė Žliobaitė. How good is the electricity benchmark for evaluating concept drift adaptation. *arXiv preprint arXiv:1301.3524*, 2013.

Indrė Žliobaitė, Albert Bifet, Mohamed Gaber, Bogdan Gabrys, Joao Gama, Leandro Minku, and Katarzyna Musial. Next challenges for adaptive learning systems. *ACM SIGKDD Explorations Newsletter*, 14(1):48–55, 2012.

Indrė Žliobaitė, Marcin Budka, and Frederic Stahl. Towards cost-sensitive adaptation: when is it worth updating your predictive model? *Neurocomputing*, 150:240–249, 2015.

Blaz Zupan, Marko Bohanec, Ivan Bratko, and Janez Demsar. Machine learning by function decomposition. In *ICML*, pages 421–429, 1997.