
Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

# MULTIVARIATE NON-PARAMETRIC STATISTICAL TESTS TO REUSE CLASSIFIERS IN RECURRING CONCEPT DRIFTING ENVIRONMENTS

Paulo Mauricio Gonçalves Júnior

TESE DE DOUTORADO

Recife
April 23, 2013

Universidade Federal de Pernambuco
Centro de Informática

Paulo Mauricio Gonçalves Júnior

# MULTIVARIATE NON-PARAMETRIC STATISTICAL TESTS TO REUSE CLASSIFIERS IN RECURRING CONCEPT DRIFTING ENVIRONMENTS

*Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.*

Orientador: *Prof. Dr. Roberto Souto Maior de Barros*

Recife
April 23, 2013

**Catalogação na fonte**
**Bibliotecária Jane Souto Maior, CRB4-571**

**Gonçalves Júnior, Paulo Mauricio**
**Multivariate non-parametric statistical tests to reuse classifiers in recurring concept drifting environments / Paulo Mauricio Gonçalves Júnior. - Recife: O Autor, 2013.**
**xxvii, 127 f. : il., fig., tab.**

**Orientador: Roberto Souto Maior de Barros.**
**Tese (doutorado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2013.**

**Inclui bibliografia e apêndice.**

**1. Inteligência artificial. 2. Mineração de dados. I. Barros, Roberto Souto Maior de (orientador). II. Título.**

**006.3 CDD (23. ed.) MEI2013 – 099**

Tese de Doutorado apresentada por **Paulo Mauricio Gonçalves Júnior** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Testes Estatísticos Não-paramétricos Multivariados para Reutilização de Classificadores em Ambientes com Mudanças de Conceito Recorrentes**" orientada pelo **Prof. Roberto Souto Maior de Barros** e aprovada pela Banca Examinadora formada pelos professores:

_____
Prof. Paulo Jorge Leitão Adeodato
Centro de Informática / UFPE

_____
Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

_____
Prof. Sergio Ricardo de Melo Queiroz
Centro de Informática / UFPE

_____
Prof. José Alfredo Ferreira Costa
Departamento de Engenharia Elétrica / UFRN

_____
Prof. Marco Antonio de Oliveira Domingues
Instituto Federal de Pernambuco – Campus Recife

Visto e permitida a impressão.
Recife, 23 de abril de 2013

_____
**Profa. Edna Natividade da Silva Barros**
Vice-Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

*I dedicate this thesis to my family and my girlfriend. Thanks for your unconditional support during these hard times.*

# ACKNOWLEDGEMENTS

Firstly, I would like to express my sincerest gratitude to my supervisor, Prof. Roberto S. M. Barros, for his availability, advices, and all the support he has given me in completing this thesis.

I thank Dariusz Brzeziński, for donating his implementations of the Accuracy Weighted Ensemble and Accuracy Updated Ensemble classifiers to the community (used in this thesis). I also thank him for giving invaluable help in the implementation of the Learn++.NSE classifier.

I also thank Prof. Paulo Adeodato and Prof. George Darmiton for contributing to my academic training and for always requiring a high quality work, which made me look for more challenging topics.

I would also like to thank all my friends at the Centro de Informática (CIn–UFPE) for the mutual support and entertaining moments shared. Finally, I thank Prof. André Carvalho, for introducing me to the data streams research area, which turned to be the beginning of the research work described in this thesis and all the staff of the Centro de Informática (CIn–UFPE), for their efficiency and availability for solving problems.

# RESUMO

Fluxos de dados são um modelo de processamento de dados recente, onde os dados chegam continuamente, em grandes quantidades, a altas velocidades, de modo que eles devem ser processados em tempo real. Além disso, várias instituições públicas e privadas armazenam grandes quantidades de dados que também devem ser processadas. Classificadores tradicionais não são adequados para lidar com grandes quantidades de dados por basicamente duas razões. Primeiro, eles costumam ler os dados disponíveis várias vezes até convergirem, o que é impraticável neste cenário. Em segundo lugar, eles assumem que o contexto representado por dados é estável no tempo, o que pode não ser verdadeiro. Na verdade, a mudança de contexto é uma situação comum em fluxos de dados, e é chamado de mudança de conceito.

Esta tese apresenta o RCD, uma estrutura que oferece uma abordagem alternativa para lidar com os fluxos de dados que sofrem de mudanças de conceito recorrentes. Ele cria um novo classificador para cada contexto encontrado e armazena uma amostra dos dados usados para construí-lo. Quando uma nova mudança de conceito ocorre, RCD compara o novo contexto com os antigos, utilizando um teste estatístico não paramétrico multivariado para verificar se ambos os contextos provêm da mesma distribuição. Se assim for, o classificador correspondente é reutilizado. Se não, um novo classificador é gerado e armazenado.

Três tipos de testes foram realizados. Um compara o RCD com vários algoritmos adaptativos (entre as abordagens individuais e de agrupamento) em conjuntos de dados artificiais e reais, entre os mais utilizados na área de pesquisa de mudança de conceito, com mudanças bruscas e graduais. É observada a capacidade dos classificadores em representar cada contexto, como eles lidam com as mudanças de conceito e os tempos de treinamento e teste necessários para avaliar os conjuntos de dados. Os resultados indicam que RCD teve resultados estatísticos semelhantes ou melhores, em comparação com os outros classificadores. Nos conjuntos de dados do mundo real, RCD apresentou precisões próximas do melhor classificador em cada conjunto de dados.

Outro teste compara dois testes estatísticos (KNN e Cramer) em suas capacidades de representar e identificar contextos. Os testes foram realizados utilizando classificadores

tradicionais e adaptativos como base do RCD, em conjuntos de dados artificiais e do mundo real, com várias taxas de variação. Os resultados indicam que, em média, KNN obteve melhores resultados em comparação com o teste de Cramer, além de ser mais rápido. Independentemente do critério utilizado, RCD apresentou valores mais elevados de precisão em comparação com seus respectivos classificadores base.

Também é apresentada uma melhoria do RCD onde os testes estatísticos são executadas em paralelo por meio do uso de um pool de threads. Os testes foram realizados em três processadores com diferentes números de núcleos. Melhores resultados foram obtidos quando houve um elevado número de mudanças de conceito detectadas, o tamanho das amostras utilizadas para representar cada distribuição de dados era grande, e havia uma alta freqüência de testes. Mesmo que nenhuma destas condições se aplicam, a execução paralela e seqüencial ainda têm performances muito semelhantes.

Finalmente, uma comparação entre seis diferentes métodos de detecção de mudança de conceito também foi realizada, comparando a precisão, os tempos de avaliação, manipulação das mudanças de conceito, incluindo as taxas de falsos positivos e negativos, bem como a média da distância ao ponto de mudança e o seu desvio padrão.

**Palavras-chave:** Fluxos de dados, mudanças de conceito, teste estatístico não-paramétrico multivariado, contextos recorrentes, aprendizado em tempo real

# ABSTRACT

Data streams are a recent processing model where data arrive continuously, in large quantities, at high speeds, so that they must be processed on-line. Besides that, several private and public institutions store large amounts of data that also must be processed. Traditional batch classifiers are not well suited to handle huge amounts of data for basically two reasons. First, they usually read the available data several times until convergence, which is impractical in this scenario. Second, they imply that the context represented by data is stable in time, which may not be true. In fact, the context change is a common situation in data streams, and is named concept drift.

This thesis presents RCD, a framework that offers an alternative approach to handle data streams that suffer from recurring concept drifts. It creates a new classifier to each context found and stores a sample of the data used to build it. When a new concept drift occurs, RCD compares the new context to old ones using a non-parametric multivariate statistical test to verify if both contexts come from the same distribution. If so, the corresponding classifier is reused. If not, a new classifier is generated and stored.

Three kinds of tests were performed. One compares the RCD framework with several adaptive algorithms (among single and ensemble approaches) in artificial and real data sets, among the most used in the concept drift research area, with abrupt and gradual concept drifts. It is observed the ability of the classifiers in representing each context, how they handle concept drift, and training and testing times needed to evaluate the data sets. Results indicate that RCD had similar or better statistical results compared to the other classifiers. In the real-world data sets, RCD presented accuracies close to the best classifier in each data set.

Another test compares two statistical tests (KNN and Cramer) in their capability in representing and identifying contexts. Tests were performed using adaptive and batch classifiers as base learners of RCD, in artificial and real-world data sets, with several rates-of-change. Results indicate that, in average, KNN had better results compared to the Cramer test, and was also faster. Independently of the test used, RCD had higher accuracy values compared to their respective base learners.

It is also presented an improvement in the RCD framework where the statistical tests

are performed in parallel through the use of a thread pool. Tests were performed in three processors with different numbers of cores. Better results were obtained when there was a high number of detected concept drifts, the buffer size used to represent each data distribution was large, and there was a high test frequency. Even if none of these conditions apply, parallel and sequential execution still have very similar performances.

Finally, a comparison between six different drift detection methods was also performed, comparing the predictive accuracies, evaluation times, and drift handling, including false alarm and miss detection rates, as well as the average distance to the drift point and its standard deviation.

**Keywords:** Data streams, concept drifts, multivariate non-parametric statistical test, recurring contexts, on-line learning

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

| | |
|---|---|
| ADWIN | Adaptive Windowing. |
| AUE | Accuracy Updated Ensemble. |
| AWE | Accuracy Weighted Ensemble. |
| CCP | Conceptual Clustering and Prediction. |
| CD | Critical Difference. |
| CVFDT | Concept-adapting Very Fast Decision Tree. |
| DBMS | Database Management Systems. |
| DDD | Diversity for Dealing with Drifts. |
| DDM | Drift Detection Method. |
| DSMS | Data Stream Management Systems. |
| DWAA | Dynamic Weight Assignment and Adjustment. |
| DWM | Dynamic Weighted Majority. |
| EB | Ensemble Building. |
| ECDD | EWMA for Concept Drift Detection. |
| EDDM | Early Drift Detection Method. |

| | |
|---|---|
| EWMA | Exponentially Weighted Moving Average. |
| HT | Hoeffding Tree. |
| HTNB | Hoeffding Tree with Naive Bayes. |
| KNN | k-Nearest Neighbors. |
| LNSE | Learn++.NSE. |
| MLP | Multilayer Perceptron. |
| MOA | Massive Online Analysis. |
| OcVFDT | One-class Very Fast Decision Tree. |
| PECS | Prediction Error Context Switching. |
| PHT | Page-Hinkley Test. |
| PL | Paired Learners. |
| RCD | Recurring Concept Drifts. |
| SEA | Streaming Ensemble Algorithm. |
| SRMTDS | Semi-Random Multiple Decision Trees for Data Streams. |
| STEPD | Statistical Test of Equal Proportions. |

VFDT Very Fast Decision Tree.

WMA Weighted Majority Algorithm.

# PUBLISHED PAPERS

As a result of the research developed in this thesis, some papers have been published.

**RCD: A Recurring Concept Drift Framework** (Gonçalves and Barros, 2013a). This paper, that describes the RCD framework and compares it to several single and ensemble classifiers, is based on the findings of Chapter 4, and was published in the *Pattern Recognition Letters* journal.

**A Comparison on How Statistical Tests Deal with Concept Drifts** (Gonçalves and Barros, 2012). This paper, comparing the k-Nearest Neighbors (KNN) and Cramer multivariate non-parametric statistical tests, is based on Chapter 5 and was published in *The 2012 International Conference on Artificial Intelligence*.

**Speeding Up Statistical Tests to Detect Recurring Concept Drifts** (Gonçalves and Barros, 2013b). This paper was published as a book chapter of the *Studies in Computational Intelligence* series by Springer, based on the study performed at Chapter 6, which analyzes parallelism in the RCD framework.

CHAPTER 1

# INTRODUCTION

Recent years have witnessed an increase in the amount of applications that have to deal with information that occur in the form of a continuous flow of data. This situation is named *data streams*. In this processing model, data arrive continuously, in large quantities and quickly, making it impossible to store data for later analysis, except in small amounts. Thus, data must be processed on-line, i.e., as they arrive.

Several types of applications in computer networks (Cranor et al., 2003), stock market analysis (Zhu and Shasha, 2003), climate monitoring (Lee et al., 2007), among others, are part of this data processing model. Applications that implement this data model are called Data Stream Management Systems (DSMS).

Besides that, many institutions like financial ones, manufacturing facilities, government departments, etc., all store large amounts of historical data that must also be processed. In all these situations, millions of instances must be analyzed, on-line or off-line. Traditional batch classifiers are not well suited to handle huge amounts of data. They usually read the available data several times until convergence, which can be a laborious and time consuming activity considering millions and millions of instances. In this situation, usually a sample of data is used to train the batch classifier. This is not an ideal solution because available data are not fully used, possibly configuring a waste of important information.

Also, as described by Widmer and Kubat (1996) and Tsymbal (2004), it is not very common for data distributions and concepts to keep stable over a long period of time. Maloof (2005) describes that "traditional approaches to data mining are based on an assumption that the process that generated or is generating a data stream is static". Brzeziński (2010) agrees with that statement when he says that "traditional classification techniques give great results in static environments, however, they fail to successfully process data streams because of two factors: their overwhelming volume and their distinctive feature – concept drift".

What is a concept drift? Concept drift is a common situation in data stream environments according to Gaber et al. (2005). Wang et al. (2011) describe that, in machine learning, "the term concept refers to the quantity that a learning model is trying to pre-

dict, i.e., the variable. Concept drift is the situation in which the statistical properties of the target concept change over time." Any application that tries to model human behavior is subject to concept drift. Examples of applications where concept drift is present are intrusion detection (Lane and Brodley, 1998), credit card fraud detection (Wang et al., 2003), and spam filtering (Delany et al., 2005).

Several approaches exist to handle concept drifts. One is to modify batch classifiers to make them suited to deal with drifts like the proposals of Hulten et al. (2001) to decision trees and of Ferrer-Troyano et al. (2005a,b) to decision rules. Another approach is to identify when a concept drift occurs and, then, take an action like creating a new classifier to better represent the new concept (Gama et al., 2004a; Baena-García et al., 2006; Ross et al., 2012). A common approach to handle concept drifts is by the use of ensemble classifiers. Here, several classifiers are trained and weights are assigned to each one, usually indicating how well it represents the current concept (Street and Kim, 2001; Wang et al., 2003; Kolter and Maloof, 2007; Brzeziński and Stefanowski, 2011). The final classification is usually based on the best learners, indicated by their weights.

A common situation concerning concept drifts is context recurrence. It occurs when a previously seen context reappears. According to Harries et al. (1998), domains where it can happen include "financial prediction, dynamic control and other commercial data mining applications". They also claim that "recurring contexts may be due to cyclic phenomena, such as seasons of the year, or may be associated with irregular phenomena, such as inflation rates or market mood". With the occurrence of many concept drifts, algorithms tend to better represent the last observed concepts, forgetting previously learned concepts.

One approach commonly used to treat recurring concept drifts is to store information about the contexts, and continuously identify if the current context is similar to old contexts. If it is so, obtain stored information and use it again as it is expected that it also represents the current context. Examples of algorithms that use this approach to deal with context recurrence are FLORA (Widmer and Kubat, 1996), Prediction Error Context Switching (PECS) (Salganicoff, 1997), and SPLICE (Harries et al., 1998). It is important to point out that these proposals have a weakness concerning how they deal with recurring concept drifts: they only manipulate categoric data. More recent approaches do not suffer from this restriction, for example, the ones proposed by Ramamurthy and Bhatnagar (2007) and Elwell and Polikar (2011).

On-line classifiers must fit some specific requirements due to the nature of data they handle, as described by Bifet et al. (2010a). These requirements are:

R1. The next available instance from the stream is passed to the algorithm.

R2. The instance is processed by the algorithm, where it updates its internal data structures. The algorithm must process it without exceeding the memory bounds set on it.

R3. The algorithm must process the data instances as fast as it can.

R4. The algorithm is ready to accept the next instance. On request it must be able to predict the class of unseen instances.

In this thesis it is proposed a framework to better deal with recurring concept drifts. The RCD framework reuses previously generated classifiers, associating, to each classifier, a sample of data used to build it. It applies a multivariate non-parametric statistical test to recent data and stored samples to identify which of the stored classifiers best represents the the current context.

To verify how RCD deals with concept drifts, several tests have been performed on data sets with different types of concept drift (abrupt and gradual), various levels of noise, among artificial and real-world, comparing this proposal to several other algorithms, varying from single classifiers, drift detection methods, and ensemble classifiers.

## 1.1 CONTRIBUTIONS

There are several contributions of this thesis that can be highlighted. The first one is the use of *multivariate non-parametric statistical tests* to select the best classifiers based on data distribution, both in the training and testing phases. In the training phase, it is used to create a classifier to each different data distribution available. If a previously seen data distribution is detected, reflecting a recurring context, the classifier trained on examples associated with that data distribution is reused.

Another contribution is related to the testing phase. Current ensemble classifiers add, remove, and change learners weight in the training phase. In the testing phase, they use the stored learners with their associated weight to classify instances. Thus, no modifications in the ensemble are performed. RCD works differently. In the testing phase it also does not modify the ensemble, but, like in the training phase, it allows the best classifier to be dynamically selected based on the the current data distribution.

Differently than FLORA, PECS, and SPLICE, which are also proposals to deal with recurring concept drifts, as mentioned in the previous section and detailed in the next chapter, that only deal with categorical data, RCD has the ability to also deal with numeric

data. In addition, they also implement the window concept, where learners are adapted to hidden changes in context by updating the current concept to match a window of recent instances. RCD, in its turn, proposes storing a data sample of configurable size to each different data distribution identified in the data stream together with the classifier trained on that data sample.

Another contribution of this thesis is to propose the execution of the statistical tests in parallel. Performing the tests in parallel through the use of a thread pool improved the evaluation time of the RCD framework considerably.

## 1.2 ORGANIZATION

This thesis is organized as follows:

Chapter 2 presents the problem of concept drift, some definitions, types of concept drift that can occur (abrupt and gradual), types of algorithms used to deal with it and how they work, like single classifiers, drift detection methods and ensemble classifiers, and the data sets used in the experiments of this thesis, including artificial and real-world data sets.

Chapter 3 presents the framework proposed in this thesis, the algorithm, the parameters used to fine-tune its use, and how it works.

Chapter 4 presents the analysis of the results obtained by the algorithms in the selected data sets, showing accuracy, time spent in the training and testing phases, and how the algorithms deal with concept drifts.

Chapter 5 presents a comparison between two statistical tests, KNN and Cramer, on how they can distinguish between new and old contexts. Test were performed with five different base learners, among batch and dynamic classifiers. Results show in which conditions each statistical test offers best performance, both in terms of accuracy and evaluation time.

Chapter 6 presents a comparison between several thread pool sizes to execute the statistical tests in parallel. Tests were performed with three RCD configurations in three computer systems with one, two, and four active cores and helps to identify in which circumstances the usage of parallelism offers best results, and the best number of active cores to use in each kind of system.

Chapter 7 analyzes the predictive accuracies, evaluation times, false alarms rates, miss detection rates, and mean distance and standard deviation to the drift point of six different drift detection methods, in four artificial data sets, two of them suffering from abrupt concept drifts and two from gradual concept drifts. Each data set was tested

using two different configurations, varying the training size or the speed of change. Four real-world data sets were also used in the experiments. Based on these information, it was possible to identify the best drift detection method to use inside RCD based on the data set characteristics.

Finally, Chapter 8 presents the conclusions of this thesis and proposes some future work.

CHAPTER 2

# BACKGROUND

This chapter describes the data stream model, its main characteristics and applications built to manage them. Concept drifts, a problem that usually occurs in the data stream model is also described, besides with techniques and algorithms used to handle them, and how they work. The data sets used in the experimental chapters are also presented, including artificial and real-world ones.

## 2.1 DATA STREAMS

Database Management Systems (DBMS) have been used in applications that demand complex querying and persistent data storage for many years. In this data model, queries occur more frequently than insertions, updates, and deletions. Queries are executed when demanded and answers reflect the current state of the database. However, as described by Golab and Özsu (2003), "the past few years have witnessed an emergence of applications that do not fit this data model and querying paradigm. Instead, information naturally occurs in the form of a sequence (stream) of data values". Babcock et al. (2002) goes in the same direction when he says that "traditional DBMS's are not designed for rapid and continuous loading of individual data items". To handle the kind of applications that must deal with a continuous flow of data, Data Stream Management Systems (DSMS) were developed. Their main characteristics are:

- *Continuous queries*: "Queries which are issued once and then logically run continuously over the database" (Babu and Widom, 2001; Arasu et al., 2006). It means that as new data arrive, the DSMS must reexecute stored queries.

- *Approximation* (Manku and Motwani, 2002): As data are always flowing, summary data structures are needed to describe information while data are arriving, keeping a balance between storage size and precision in the results.

- *Adaptivity* (Babu and Widom, 2004): It is the capability of the system to adapt when it cannot handle data because of its high volume. Load shedding (Chi et al.,

2005; Tatbul et al., 2007) is a common technique. It selectively drops unprocessed tuples to reduce system load.

To facilitate the manipulation of this model, various applications have been developed. Examples include general purpose DSMS for monitoring applications like AURORA (Abadi et al., 2003; Balakrishnan et al., 2004), MAIDS (Mining Alarming Incidents from Data Streams) (Cai et al., 2004), STREAM (Arasu et al., 2004), and SMM (Stream Mill Miner) (Thakkar et al., 2008).

MEDUSA (Cherniack et al., 2003) extends AURORA to operate in a distributed fashion, and BOREALIS (Abadi et al., 2005) extends both AURORA and MEDUSA to support dynamic revision of query results, dynamic query modification, QoS (Quality of Service) model, among other improvements.

Other proposed DSMS's focus on specific problems. For example, VEDAS (Vehicle Data Stream Mining) (Kargupta et al., 2004) is a mobile and distributed data stream mining system that allows real time vehicle-health monitoring and driver characterization. GIGASCOPE (Cranor et al., 2003), on its turn, is a stream database for network applications offering several functionalities like traffic analysis, intrusion detection, router configuration analysis, network research, network monitoring, and performance monitoring and debugging.

Mining in such environments using traditional batch classifiers is not simple. Batch learners commonly iterate over the available instances until convergence. This is usually a time consuming activity, limiting the learner to deal with a fixed set of examples with small to medium sizes. Therefore, to better cope with huge amounts of data, other types of classifiers are needed.

A common approach to perform classification in these kind of environments is to adapt batch classifiers. Many adaptations have been proposed to allow classifiers to deal with streaming data. One classifier created to deal with data streams was Very Fast Decision Tree (VFDT) (Domingos and Hulten, 2000). VFDT is a decision tree, which can deal with huge amounts of data using few computational resources; it presents a performance comparable to a batch decision tree, given enough examples. It reads each example only once and needs a small amount of time to process it, allowing the creation of a classifier based on huge data sets. To identify the best attribute to be tested at a given node, it is sufficient to consider only a small subset of the training examples that pass that node. Therefore, the first examples are used to choose the root test; once the attribute is identified, the next examples are passed to the corresponding leaves and used to choose the appropriate attributes, and so on.

To solve the problem of identifying exactly how many examples are needed to each node, a statistical result known as Hoeffding bound (Hoeffding, 1963; Maron and Moore, 1994) is used. Consider a real random variable $r$ whose interval is $R$ (e.g., to a probability, the interval is one, and to the information gain, the interval is $\log c$, where $c$ is the number of classes). Suppose we have $n$ independent observations of this variable, and computed its average $\overline{r}$, the Hoeffding bound informs that, with probability $1 - \delta$, the variable average is at least $\overline{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{2.1}$$

The Hoeffding bound possesses the property of being independent of the probability distribution that generated the observations.

Gama et al. (2003) proposed VFDTc, which extends VFDT with the ability to deal with continuous attributes and the use of naive Bayes classifiers at tree leaves.

Another extension to VFDT was proposed by Jin and Agrawal (2003). They make two contributions: (1) a numerical interval pruning (NIP) approach for efficiently processing numerical attributes (showing an average of 39% reduction in execution times); and (2) exploiting the properties of the gain function entropy (and Gini coefficient) to reduce the sample size required for obtaining a given bound on the accuracy (showing a 37% reduction in the number of data instances required).

Hu et al. (2007) proposed an ensemble of decision trees for mining data streams named Semi-Random Multiple Decision Trees for Data Streams (SRMTDS). It "uses the inequality of Hoeffding bounds to choose the minimum number of split-examples, a heuristic method to compute the information gain for obtaining the split thresholds of numerical attributes, and a Naive Bayes classifier to estimate the class labels of tree leaves" (Hu et al., 2007). Experimental tests showed that it outperformed VFDTc on the memory usage, the anti-noise capability, and accuracy.

Li et al. (2009) proposed One-class Very Fast Decision Tree (OcVFDT), an algorithm to extend VFDT to deal with one-class problems, based on the fact that fully labeled data streams are expensive to obtain. It only deals with discrete attribute values. Experiments on both synthetic and real-world data shown that even when 80% of the samples in the stream were unlabeled, the classification performance of OcVFDT was still very close to that of VFDT which was trained on fully labeled data stream.

Besides decision trees, other proposals to handle data streams using neural networks (Gama and Rodrigues, 2007) and decision rules (Ferrer-Troyano et al., 2004) were also described.

## 2.2 CONCEPT DRIFTS

A common problem when learning in real-world domains is that the concept of interest may depend on a context that is not explicitly stated in the form of predictive features. Usually, changes in the context can induce changes in the target concept, which is generally known as concept drift (Widmer and Kubat, 1996).

An informal definition of concept drift was stated by Kolter and Maloof (2007): "concept drift occurs when a set of examples has legitimate class labels at one time and has different legitimate labels at another time." This is also described as *real* concept drifts. Another type is *virtual* concept drift. It occurs when "the target concepts remain the same but the data distribution changes" (Delany et al., 2005). This difference concerns the reason of change. In practice, as Tsymbal et al. (2008) describe, "virtual concept drift and real concept drift often occur together". Salganicoff (1997) refers to virtual concept drift as sampling shift, and real concept drift is referred to as concept shift.

Concept drifts can also be classified by the speed of change. Abrupt concept drifts happen when the transition from an old to a new concept occurs suddenly. Gradual concept drifts occur when the transition is smooth. For example, "someone graduating from college might suddenly have completely different monetary concerns, whereas a slowly wearing piece of factory equipment might cause a gradual change in the quality of output parts" (Stanley, 2003). Stanley (2003) even divides gradual concept drifts in moderate and slow concept drifts, depending on the speed of change.

Another difficulty concerning dealing with concept drifts is that algorithms must differentiate between noise and drifting concepts.

### 2.2.1 Single classifiers

Many proposals have been made to deal with concept drifts. The first system developed to specifically handle concept drift was STAGGER (Schlimmer and Granger, 1986). It uses a distributed concept description consisting of class nodes, corresponding to features and class labels. Each class node is linked to attribute-value nodes by probabilistic arcs, representing the strength of association between class labels and features. Two probabilities associated with each arc represent logical necessity and logical sufficiency. As STAGGER processes new examples, it updates probabilities, and it may add nodes and arcs, corresponding to new classes and features. Over time, STAGGER modifies its probabilities to better handle concept drifts, forgetting concepts that no longer hold. Empirical results on a proposed synthetic data set showed that STAGGER represented the

three concepts of the data set used in the tests.

Concept-adapting Very Fast Decision Tree (CVFDT) is a classifier proposed by Hulten et al. (2001) as an extension to VFDT (Domingos and Hulten, 2000). As described by Hulten et al. (2001), CVFDT "is an extension to VFDT which maintains VFDT's speed and accuracy advantages but adds the ability to detect and respond to changes in the example-generating process". It tries to keep its model up-to-date by the use of a sliding window of examples. For each new arriving example, it recomputes the statistics, reducing the influence of the oldest examples. If the concept starts to change, alternative attributes will increase its information gain, making the Hoeffding test on the split no longer pass. At this moment, an alternative tree begins to grow with the new best attribute set at its root. If this subtree becomes more accurate than the old one on new data, it is substituted.

### 2.2.2 Drift Detection Methods

Another common approach to deal with concept drifts is to identify when it occurs and create a new classifier. Therefore, only classifiers trained on a current concept are maintained. Algorithms that follow this approach work in the following way: each arriving training instance is first evaluated by the base classifier. Internal statistics are updated with the results and two thresholds are computed: a warning level and an error level. As the base classifier makes mistakes, the warning level is reached and instances are stored. If the behavior continues, it will reach the error level, indicating that a concept drift has occurred. At this moment, the base classifier is destroyed and a new base classifier is created and initially trained on the stored instances. On the other hand, if the classifier starts to correctly evaluate instances, this situation is considered a false alarm and stored instances are flushed. Algorithms that follow this approach can work with any type of classifier as they only analyze how the classifier evaluates instances.

One example of this approach is Drift Detection Method (DDM), proposed by Gama et al. (2004a). It works by controlling the algorithm's error rate. For each point $i$ in the sequence of arriving instances, the error rate is computed as the probability of misclassifying ($p_i$), with standard deviation given by $s_i = \sqrt{p_i(1-p_i)/i}$. Statistical theory guarantees that, when the distribution changes, the error will increase. The values of $p_i$ and $s_i$ are stored when $p_i + s_i$ reaches its minimum value during the process (obtaining $p_{min}$ and $s_{min}$). The warning level is reached when $p_i + s_i \geq p_{min} + 2 \times s_{min}$ and the error at $p_i + s_i \geq p_{min} + 3 \times s_{min}$.

Another similar method is Early Drift Detection Method (EDDM), presented by Baena-

García et al. (2006). It works similarly to DDM, but instead of controlling solely the amount of error of the classifier, it uses the distance between two errors to identify concept drifts. It computes the average distance between two errors ($p_i$) and its standard deviation ($s_i$). These values are stored when $p_i + 2 \times s_i$ reaches its maximum value (obtaining $p_{max}$ and $s_{max}$). Thus, the value of $p_{max} + 2 \times s_{max}$ corresponds to the point where the distribution of distances between errors is maximum. EDDM was shown to be more adequate to detect gradual concept drifts while DDM was better suited for abrupt concept drifts (Baena-García et al., 2006).

The Page-Hinkley Test (PHT) (Page, 1954) is a concept drift detector that identifies a change in the average of a Gaussian signal. It computes the cumulative difference between the observed values and their mean up to the current moment ($U_T$). It also stores the minimum difference between these two values ($m_T$), and, when the difference between $U_T$ and $m_T$ is above a specified threshold ($\lambda$), a change in the distribution is detected. Higher $\lambda$ values results in fewer false alarms, but might miss or delay some changes.

Exponentially Weighted Moving Average (EWMA) charts (Roberts, 1959) were originally proposed for detecting an increase in the mean of a sequence of random variables, considering that the mean and standard deviation of the stream are known. Yeh et al. (2008) proposed an EWMA change detector for a sequence of random variables that form a Bernoulli distribution. A recent proposal of concept drift detector is named EWMA for Concept Drift Detection (ECDD) (Ross et al., 2012), which extends the work of Yeh et al. (2008) to monitor the misclassification rate of a streaming classifier, allowing the rate of false positive detection to be controlled and kept constant over time.

Adaptive Windowing (ADWIN) (Bifet and Gavaldà, 2007) is another proposal to detect concept drifts. It keeps a variable size window which may contain bits or real numbers. ADWIN automatically grows the window when no change is apparent, and shrinks it when data changes. The window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window". To save space, instead of storing explicitly the window, it compresses it by using a variant of the exponential histogram technique. Thus, for a window of length $W$, only $O(logW)$ memory and processing time per item are required.

Paired Learners (PL) (Bach and Maloof, 2008) is both a classifier and drift detection method proposal that uses two classifiers to identify when a concept drift has occurred. One is a stable classifier, trained on all previously seen instances, and the other is a reactive classifier, trained on a window of most recent data. The idea here is to compare the performance of both classifiers. If the reactive classifier has a better accuracy when

compared to the stable learner over a window of recent data of parameterized size, the method infers that a context change has occurred. The stable classifier is substituted by the reactive one, and the learning process continues.

Nishida and Yamauchi (2007) presented Statistical Test of Equal Proportions (STEPD), a concept drift detector that also uses two classifiers to detect drifts. The difference to PL is that the accuracies of both classifiers are compared using a statistical test of equal proportions, where the proportions are the division between the number of correct classifications and the number of examples each classifier was trained on. A drift can only be detected when the number of instances is at least twice the size of the window of most recent examples. Besides the proportions, a statistic is calculated by using the Pearson's chi-square test with Yates' correction for continuity (Yates, 1934). If the proportion of the stable classifier is higher than the reactive classifier and statistically significant difference is detected with a certain on-tailed significance level, it is detected a significant decrease in the recent accuracy.

### 2.2.3 Ensemble Classifiers

Several proposals try to deal with concept drifts by the use of ensemble classifiers. This approach maintains a collection of learners and combine their decisions to make an overall decision. To deal with concept drifts, ensemble classifiers must take into account the temporal nature of the data stream.

One of the first proposals to use ensemble classifiers to deal with concept drifts was the Streaming Ensemble Algorithm (SEA), proposed by Street and Kim (2001). It builds separate classifiers, each one trained on a different sequential chunk of data of configurable size. These classifiers are then combined into an ensemble of fixed-size using a heuristic replacement strategy. If there is free space in the ensemble, SEA adds the newly created classifier to the ensemble. If the ensemble is full, the new classifier is only added if it outperforms a stored classifier, substituting it. The performance is measured on the current batch of examples.

The Accuracy Weighted Ensemble (AWE) classifier, introduced by Wang et al. (2003) is another proposal of ensemble classifier. It uses batch classifiers and each one is built in different chunks of data.

Wang et al. (2003) showed that, if classifiers are weighted by their expected accuracy in test data, the ensemble possesses a better accuracy than using a single classifier. Thus, they proposed to infer the weights by estimating the error rate in the most recent data blocks $x_i$, as presented in equations 2.2 below:

$$MSE_i = \frac{1}{|x_i|} \sum_{(x,c) \in x_i} (1 - f_c^i(x))^2 \tag{2.2a}$$

$$MSE_r = \sum_c p(c)(1 - p(c))^2 \tag{2.2b}$$

$$w_i = MSE_r - MSE_i \tag{2.2c}$$

where $f_c^j(x)$ denotes the probability given by classifier $C_i$ that $x$ be of class $c$. The value of $MSE_r$ is the average squared error of a classifier and is used as a threshold to zero the weights of classifiers that are not precise enough.

This weight function can create a problem in environments that suffer from abrupt concept drifts. In this situation, it is common that the classifier's accuracy reduces making it possible that no classifier surpasses the stipulated threshold, leaving no class to be predicted.

Accuracy Updated Ensemble (AUE) was proposed by Brzeziński and Stefanowski (2011) and is an enhancement of the AWE classifier. Both use classifier ensembles and associate to them weights that are updated as data arrives. To avoid the problem of no class being predicted, AUE uses a simpler weighting function:

$$w_i = \frac{1}{(MSE_r + \epsilon)} \tag{2.3}$$

where $MSE_r$ is computed the same as in equation 2.2b and $\epsilon$ is a very small constant value, allowing the computation of the weight event in the situation where $MSE_i = 0$.

In summary, the differences between AUE and AWE are the following:

1. Instead of using batch classifiers, AUE uses on-line classifiers, which adapt to data while they arrive;

2. AUE updates both classifiers and their weights, while AWE only updates the weights;

3. AUE changed the weight computation to avoid situations where no classifier should have precision higher than the stipulated threshold, as in abrupt concept drifting environments, zeroing the weights of all classifiers and no class being predicted.

The Weighted Majority Algorithm (WMA) implements a weighted ensemble classifier as proposed by Blum (1997), which extends the previous work of Littlestone and Warmuth (1994) to specifically handle concept drifts. In WMA, all the arriving instances are passed

to all the classifiers in the ensemble. The initial weight of the classifiers is 1, and, if a classifier makes an error, its weight is reduced by a factor of $\beta$ provided it is higher than a specified minimum threshold. Then, the classifier is trained. After all the ensemble classifiers have been trained, their weights are normalized.

Dynamic Weighted Majority (DWM), proposed by Kolter and Maloof (2007), is another example of ensemble classifier. It extends WMA and implements a weighted ensemble classifier designed specifically to identify concept drifts. This method adds and removes classifiers according to the algorithms global performance. If the ensemble commits an error, then a classifier is added. If one classifier commits an error, its weight is reduced. If after many examples a classifier continues with a low accuracy, indicated by a low weight, it is removed from the ensemble. This method is general and, in principle, can be used with any classifier.

Another proposal for ensemble classifier is Dynamic Weight Assignment and Adjustment (DWAA). It was proposed by Wu et al. (2008) and creates classifiers based on data chunks, using the next chunk to evaluate the classifier previously built. If the ensemble is not full, the classifier is added; otherwise, the worst classifier in the last data chunk is replaced. To set the weight, it uses a formula that considers how many of the ensemble classifiers have actually made correct predictions. If more than half of the classifiers predictions are correct, each one receives a normal reward. Otherwise, each one receives a higher reward, making those influence more the global decision of the ensemble, as they are better suited to represent the concept.

Aljaafreh and Dong (2010) proposes "a heuristic to enhance cooperative detection of moving targets within a region that is monitored by a wireless sensor network. This heuristic is based on fuzzy dynamic weighted majority voting for decision fusion." Based on the decision of each sensor it determines the number of moving targets and their types. The weights of each local decision is determined by fuzzy logic "based on the signal to noise ratio of the acoustic signal for target detection and the signal to noise ratio of the radio signal for sensor communication." (Aljaafreh and Dong, 2010)

Diversity for Dealing with Drifts (DDD) (Minku and Yao, 2012) is a recent proposal that uses four ensemble classifiers: with high and low diversity, before and after a concept drift is detected. A previous study (Minku et al., 2010) analyzed how these ensembles behaved in data sets suffering from abrupt and gradual concept drifts with several speeds of change, right after the drift and longer after. With the results obtained, DDD was proposed, trying to select the best ensemble (or weighted majority of ensembles) before and after drifts, detected by the use of a drift detection method.

### 2.2.4 Recurring Contexts

One approach commonly used to treat recurring concept drifts, implemented in FLORA3 (Widmer and Kubat, 1996), Prediction Error Context Switching (PECS) (Salganicoff, 1997), and SPLICE-2 (Harries et al., 1998) is storing information about the concepts, and, if necessary, reusing them.

FLORA3 deals with categorical attributes and represents data using a simple representation language based on attribute-value logic without negation. It also uses the notion of description items, which is a conjunction of attribute-value pairs. It represents a *concept description* in the form of three description sets: the set ADES (*Accepted Descriptors*) contains description items matching only positive examples. The set NDES (*Negative Descriptors*) summarizes the negative examples. PDES (*Potential Descriptors*) contains description items that are too general, matching positive examples, but also some negative ones. ADES is used to classify new incoming examples, NDES summarizes the negative examples and is used to prevent over-generalization of ADES, while PDES acts as a repository of hypotheses that are currently too general but might become relevant.

PECS uses KNN to locally compare close instances and check whether they agree on their outcomes. If an instance differs considerably from its neighbors it is moved from an active learning set to an inactive set for re-evaluation over future instances. Statistics are kept for each of these instances on its most recent agreement and disagreement over a sliding window to detect trends in their reliability. Accuracy of an instance in the current task context is computed. As described by Salganicoff (1997), "Higher probabilities indicate that the given observation is appropriate for the current mapping context and should be in the active learning set. Lower probabilities indicate that the instance should be switched to the dormant set of instances".

SPLICE-2 also deals with categorical attributes but the classifier training is made in batch mode. SPLICE-2 assumes that a concept will be stable over some interval of time. Sequences of examples in the data set are combined into intervals if they appear to belong to the same context. SPLICE-2 then attempts to cluster similar intervals by applying the notion that similarity of context is reflected by the degree to which intervals are well classified by the same concept. This is called *contextual clustering*. To perform training, each example must have an attribute that uniquely identifies its position in the sequence of training data. SPLICE-2 begins by guessing an initial partitioning of the data and subsequent stages refine the initial guess. Initial partitioning may be performed through random partitioning, partitioning by C4.5 (Quinlan, 1993), using the tests on *time*, or prior domain knowledge. The next step is to refine the contextual clusters. With each

iteration, a new set of contextual clusters is created in an attempt to better identify stable concepts in the data set. This process may also reduce the number of contextual clusters and, consequently, the number of stable concepts eventually learned.

Ramamurthy and Bhatnagar (2007) presents an ensemble classifier, here named Ensemble Building (EB), similar to the approaches taken by AWE and AUE, where classifiers are created from sequential data chunks and a subset of them is used to be part of the ensemble. Classifier weight is based on its accuracy on the last data chunk. If none of the stored classifiers performs above a specified threshold in the current chunk, a new classifier is built and stored in the ensemble, indicating that the stored classifiers do not correctly represent current data. Thus, differently than AWE and AUE which create a new classifier for each data chunk, this proposal only adds a new classifier to the ensemble if no classifier is well suited to current data.

Gama and Kosina (2009) presents the two-layers learning system, consisting of two classifiers: the first one (level 0) is the primary model that serves as normal classifier while the second one (level 1) is denoted as the referee. Each instance is used to train the current classifier and the same instance to the referee, only changing the class attribute to zero if the prediction of the current classifier is wrong, or one otherwise. "Doing so, the meta-learner learns the regions of the instance space where the base classifier performs well" (Gama and Kosina, 2009).

Katakis et al. (2010) presented Conceptual Clustering and Prediction (CCP), an ensemble classifier specially built to handle recurring concept drifts in an application to email filtering. CCP uses a clustering algorithm to group similar instances (based on a distance measure) and applies a classifier to learn the concept represented by the instances of each cluster. Instances that do not belong to any stored cluster are included as a new cluster and a classifier is trained on that instances. When a batch of instances are similar to the ones presented in a stored cluster, the stored classifier is reused to train/test that instances.

Gomes et al. (2011) is a proposal that uses ensemble classifiers to deal with recurring concept drifts. It uses a similar schema as defined by AWE together with a measure called *Conceptual equivalence*: two classifiers are compared against a sample of recent data by testing each instance in the sample and returning 1 if they agree in their classifications and -1 otherwise. These values are summed and divided by the sample size. Values above a specified threshold indicate conceptual equivalence between the classifiers. DDM is used as a drift detection method, and, when a drift occurs, it stores the current classifier in the ensemble, if the concept is new (no stored classifier has a conceptual equivalence

to current classifier); or stored classifiers are reused, if they are conceptually equivalent to the current classifier. Weights of the classifiers are computed based on Ramamurthy and Bhatnagar (2007) and the context distance, a measure that computes the distance between two data samples.

Another recent proposal of an ensemble classifier built to handle recurring concept drifts is Learn++.NSE (LNSE). This method implements a classifier ensemble for incremental learning in non-stationary environments. The original algorithm, as proposed by Muhlbaier and Polikar (2007), works as follows: a single classifier is created to each data chunk that becomes available. The algorithm first evaluates the classification accuracy of the current ensemble on the newly available data, obtained by the weighted majority voting of all classifiers in the ensemble. Its error is computed as a simple ratio of the correctly identified instances of the new data set and normalized in the interval [0,1]. Then, the weight of the instances are updated: the weights of the instances misclassified by the ensemble are reduced by a factor of the normalized error. The weights are normalized. A new classifier is created and all the classifiers generated so far are evaluated on the current data chunk, by computing their weighted error. If the error of the most recent classifier is greater than 0.5, it is discarded and a new one is created. For each of the other classifiers, if its error is greater than 0.5, its voting power is removed during the weighted majority voting. No classifier is ever removed from the ensemble.

A comparison of this algorithm's performance using several classifier models, and on different environments as a function of time for several values of rate-of-change is presented by Karnick et al. (2008a). Another comparison of this algorithm concerning its performance using different base learners in different environments with varying types of drift is presented by Karnick et al. (2008b). Elwell and Polikar (2009a) included various ensemble pruning methods to prevent potential outvoting from irrelevant classifiers or simply to save memory. Elwell and Polikar (2009b) presented preliminary results on the ability of the algorithm to accommodate addition and subtraction of classes over time. Elwell and Polikar (2011) presented a comparison with several other approaches.

## 2.3 DATA SETS

The following sections present the data sets used in the comparison of the framework from Chapters 4 to 6. The data sets used are well known in the field of data streams and concept drifts and have been used in previous experiments. They were divided between artificial and real-world data sets.

### 2.3.1 Artificial data sets

We first describe the artificial data sets. All these data sets are available through the Massive Online Analysis (MOA) framework (Bifet et al., 2010a). Some represent abrupt concept drifts while others represent gradual concept drifts.

**2.3.1.1 Hyperplane** Hyperplane is an artificial data set that simulates gradual concept drifts through a moving hyperplane. A hyperplane in a $d$ dimensional space is the set of points $x$ that satisfies

$$\sum_{i=1}^{d} w_i x_i = w_0 \tag{2.4}$$

where $x_i$ is the $i^{th}$ coordinate of $x$. Examples where $\sum_{i=1}^{d} w_i x_i \geq w_0$ are classified as positive, and examples where $\sum_{i=1}^{d} w_i x_i < w_0$ are classified as negative. Hyperplanes are used to simulate gradual concept drifts where it is possible to smoothly change the orientation and position of the hyperplane by modifying its weights. It is possible to introduce changes in the data set changing the weight of each attribute $w_i = w_i + d\sigma$, where $\sigma$ is the chance the direction of change be inverted and $d$ is the amount of change applied to each example.

The Hyperplane data set has already been used in many publications (Hulten et al., 2001; Wang et al., 2003; Fan et al., 2004; Yang et al., 2005; Narasimhamurthy and Kuncheva, 2007; Sun et al., 2007; Tsymbal et al., 2008; Wu et al., 2008; Bifet et al., 2009b, 2010b; Brzeziński and Stefanowski, 2011).

**2.3.1.2 LED** The LED data set is composed of 24 categorical attributes, 17 of which are irrelevant, and one categorical class with ten possible values. It represents the problem of predicting the digit shown by a seven-segment LED display, where each attribute have 10% probability of being inverted (noise). We used a version of LED available at MOA that includes concept drifts to the data sets by simply changing the attributes positions. The number of drifting attributes chosen were 1, 3, 5, and 7. Like Hyperplane, this data set was previously tested by several authors (Breiman et al., 1984; Gama et al., 2003, 2004b, 2005, 2006; Gama and Kosina, 2009; Bifet et al., 2009b, 2010b; Brzeziński and Stefanowski, 2011).

**2.3.1.3 Mixed** The Mixed data set (Gama et al., 2004a; Baena-García et al., 2006) has two boolean ($v$ and $w$) and two numerical ($x$ and $y$) attributes. Three conditions are verified for these attributes: $v$, $w$, and $y < 0.5 + 0.3\sin(3\pi x)$. If at least two of these conditions are satisfied, the example is considered positive. When a concept drift occurs, the classification is reversed. Concept drift is made by gradually choosing examples from the old and new concepts, reducing the probability of selecting examples from the old context, while increasing the probability of the new context. In this data set, we have an initial stable context that gradually changes to a new stable context, differently than Hyperplane, which is always changing. This is a noise free data set.

**2.3.1.4 Random RBF** RBF (Radial Basis Function) creates complex concept drifts that are not straightforward to approximate with a decision tree model. It works as follows: a fixed number of random centroids are generated. Each center has a random position, a single standard deviation, a class label, and a weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Only numeric attributes are generated. Drift is introduced by moving the centroids with constant speed. This speed is initialized by a drift parameter. The Random RBF set was already used in the following papers: Bifet et al. (2009b, 2010b).

**2.3.1.5 SEA** The SEA concepts were first introduced by Street and Kim (2001) and are commonly used to test abrupt concept drifts. The values of each of their three attributes are in the interval [0,10), but the third one is irrelevant. In each concept, a data point belongs to class 1 if $f_1 + f_2 \leq \theta$, where $f_l$ and $f_2$ represent the first two features and $\theta$ is a threshold value between the two classes. Several authors have already used this data set in experiments (Kolter and Maloof, 2007; Sun et al., 2007; Bach and Maloof, 2008; Karnick et al., 2008b; Tsymbal et al., 2008; Wu et al., 2008; Elwell and Polikar, 2009b; Gama and Kosina, 2009; Sebastião and Gama, 2009; Bifet et al., 2010b; Elwell and Polikar, 2011; Brzeziński and Stefanowski, 2011; Gomes et al., 2011).

The expression below describes the common utilization of this data set. The subscript

value on $SEA$ indicates the value of $\theta$, the $\oplus$ value represents the concatenation of two instances set, with the subscript value indicating the point of change, and the superscript, the length of change.

$$(((SEA_9 \oplus_{t_0}^{W} SEA_8) \oplus_{2t_0}^{W} SEA_7) \oplus_{3t_0}^{W} SEA_{9.5})$$

**2.3.1.6 Sine** This data set presents the problem of identifying the position of coordinates, represented by two attributes, in relation to the curve $y = \sin(x)$. In the first context, points below the curve are classified as positive. After each concept drift, the classification is reversed. Each coordinate has values uniformly distributed in the [0,1] interval. It is possible to include two other attributes, filled with random data in the same interval, with no influence on the classification function (irrelevant data). Gama et al. (2004a) named these data sets as Sine1 and Sinirrel1, respectively. It also described Sine2, similar to Sine1 but using a different curve, $y < 0.5 + 0.3\sin(3\pi x)$. Positive and negative examples are interchanged to ensure a stable learning environment. Sine has already been used in several papers, specially when describing new drift detection methods (Gama et al., 2004a; Baena-García et al., 2006; Ross et al., 2012).

**2.3.1.7 STAGGER** Bifet et al. (2009b) described that "the STAGGER concepts are boolean functions of three attributes encoding objects". Each example consists of the following attributes: $color \in \{green, blue, red\}$, $shape \in \{triangle, circle, rectangle\}$, and $size \in \{small, medium, large\}$.

There are three kinds of different concepts, according to STAGGER original paper (Schlimmer and Granger, 1986):

- Concept A: $color = red \wedge size = small$;
- Concept B: $color = green \vee shape = circle$
- Concept C: $size = medium \vee size = large$

This data set is usually used to simulate abrupt concept drifts and has already been used in many papers (Schlimmer and Granger, 1986; Gama et al., 2004a; Kolter and Maloof, 2005; Yang et al., 2005, 2006; Kolter and Maloof, 2007; Narasimhamurthy and Kuncheva, 2007).

$$((STAGGER_A \oplus_{t_0}^{W} STAGGER_B) \oplus_{2t_0}^{W} STAGGER_C)$$

**2.3.1.8 Waveform** Waveform consists of a data stream with three decision classes where the examples are described by 21 numeric attributes. The goal of the task is to differentiate between three different classes of waveform, each of which is generated from a combination of two or three base waves. This data set was previously used in experiments in the following papers: Breiman et al. (1984); Gama et al. (2003, 2004b, 2005, 2009); Dries and Rückert (2009); Brzeziński and Stefanowski (2011).

### 2.3.2 Real data sets

The next sections present four real data sets used in the experiments. It is not easy to find large real-world datasets for public benchmarking, especially with substantial concept change. Another problem is that we do not know when drift occurs or if there is any drift at all. The first three described data sets were obtained from the MOA web site, in the following address: `http://moa.cs.waikato.ac.nz/`.

**2.3.2.1 Forest Covertype** This data set contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The goal is to predict the forest cover type from cartographic variables. It contains 581,012 instances and 54 attributes, including numeric and categoric ones. It has been used in several papers on data stream classification as in Blackard and Dean (1999); Bazan and Szczuka (2001); Oza and Russell (2001a); Gama et al. (2003); Oza (2005); Bifet et al. (2009b, 2010b).

**2.3.2.2 Electricity** This data set, composed of 45,312 instances and eight attributes, presents data collected from the Australian New South Wales Electricity Market. In that market the prices are not fixed, varying based on market demand and supply. The prices are set every five minutes and the class label identifies the change of the price related to a moving average of the last 24 hours. The goal of the problem is to predict if the price will increase or decrease. This data set was previously used in the following papers: Gama et al. (2004a, 2005); Baena-García et al. (2006); Kolter and Maloof (2007); Bifet et al. (2009b, 2010b); Brzeziński and Stefanowski (2011); Minku and Yao (2012).

**2.3.2.3 Poker Hand** The Poker Hand data set represents the problem of identifying the value of five cards in the Poker game. It is constituted of five categoric and five numeric attributes and one categoric class with 10 possible values informing the value

of the hand, for example, one pair, two pairs, a sequence, a street flush, etc. "In the Poker hand data set, the cards are not ordered, i.e., a hand can be represented by any permutation, which makes it very hard for propositional learners, especially for linear ones" (Bifet et al., 2010b). So, in the experiments we used a modified version, where the cards are sorted by rank and suit, and duplicates were removed. This data set was recently used in experiments in papers by Bifet et al. (2009b, 2010b) and is composed of 829,201 instances.

**2.3.2.4 Weather** This real-world data set was obtained in the Nebraska Weather Prediction, part of the U.S. National Oceanic and Atmospheric Administration. It provides eight numeric daily measurements of several meteorological features like temperature, sea level pressure, average wind speed, among others. Based on these values, it is demanded to indicate if rained or not in that specific day. This data set provides 50 years of measurements (1949–1999), offering diverse weather patterns, consisting of 18,159 daily readings, 5,698 (31%) of which are positive ("rain") while the remaining 12,461 (69%) are negative ("no rain"). It was already used in previous experiments by Elwell and Polikar (2011) and can be obtained at `http://users.rowan.edu/~polikar/research/nse/`.

CHAPTER 3

# THE RCD FRAMEWORK

As we could observe in the previous chapter, independently of using the single classifiers approach, the approach using a drift detection method, or ensemble classifiers, they always try to adapt to the current concept. The previous seen concepts are gradually forgotten, which is a waste of processed information that could be used in the future. Even the classifiers built to handle specifically recurring concept drifts have problems. For example, in EB, classifiers weights are computed based on their accuracy on the last data chunk in the training phase. Thus, in the testing phase, if EB receives instances that are similar to those observed in the beginning of the training phase, before drifts have occurred, the classifier trained on those data will have a small weight, as had been adjusted to data different from that of the last concept. A similar behavior occurs concerning the LNSE classifier.

The approach proposed here described changes the functioning of the drift detection methods as described in the previous chapter. Instead of dumping the old classifier when a drift is detected, it is stored for future use, when data similar to those used to train the classifier appear again. But how can we compare instances to verify if they come from the same distribution, indicating that they are similar, and that the stored classifier could be reused? A statistical test is commonly used to solve this kind of problem. The test must be non-parametric, as we do not know before hand the distributions of the data to be compared. Besides that, it must be multivariate, as instances to be compared are composed of various attributes.

This chapter describes Recurring Concept Drifts (RCD), a framework to compare the data distribution of samples to identify if a new context or an old context has occurred using a multivariate non-parametric statistical test.

Algorithm 1 presents the RCD pseudo-code used in the training phase. The proposed algorithm functions as follows: a classifier is built together with a concept drift detector; the scheme proposed by DDM and EDDM is used, but others could also be applied, for example, Kifer et al. (2004); Nishida and Yamauchi (2007); Dries and Rückert (2009); Ross et al. (2012). A new classifier and an empty buffer called current classifier ($c_a$) and current buffer ($b_a$), respectively, are created and stored in their respective lists (lines 2

and 3). The drift detector uses $c_a$ to identify if a concept drift is beginning to occur (line 5). While no drift is detected, $b_a$ is filled with the examples used in the training of $c_a$ (lines 28 and 31).

If the error rate of $c_a$ increases, the *warning level* of the drift detector is reached, indicating that a concept drift might be beginning to occur (line 7). Then, a new classifier ($c_n$) is created alongside with a new buffer ($b_n$). Examples are used to train $c_n$ and $c_a$ and are stored in $b_n$ (lines 12 and 13), but $b_a$ is not destroyed. If the error rate of $c_a$ decreases, it will return to the normal level considering that the concept drift was a false alarm. In this situation, $c_n$ and $b_n$ are disposed (line 26), and $b_a$ is used again to train and store examples.

On the other hand, if $c_a$ continues to increase its error rate, it will reach the *error level* (line 14). At this point, the drift detection method considers that a drift has really occurred. A statistical test is performed comparing $b_n$ with all stored buffers, trying to identify if this new context has already occurred in the past (line 15). If the test is positive, it means this is an old context that is occurring again. Then, the stored classifier and buffer are considered the new $c_a$ and $b_a$, respectively (line 16), and $c_n$ and $b_n$ are disposed (line 23). If the test is negative, it means that this is a new context. So, $c_n$ and $b_n$ are stored in the list and are considered the new $c_a$ and $b_a$ (lines 19 to 22).

The general idea is similar to the one presented by Widmer and Kubat (1996), but their contribution only deals with categorical data and uses decision rules to identify concept drift. RCD can deal with both categorical and numerical data and uses statistical tests to compare distributions (which offers significance levels of the similarity between distributions) and tests for concept drift using sample data.

Thus, RCD does not build a single classifier or a classifier ensemble but creates a collection of classifiers and chooses which one to use based on the data distribution. The classifier trained on data more similar to current data according to a statistical test is used. RCD is not limited to single classifiers as base learners; ensemble classifiers can also be used.

Algorithm 2 presents how RCD behaves in the testing phase. Initially, an empty test buffer of parameterized size is created (line 2) and each arriving instance is stored (line 4). When the buffer is full (line 5), it is time for RCD to verify if some other stored classifier is better suited to current data by the use of a statistical test (line 6). If the test is positive, the stored classifier is considered the current one (line 7). After the test, the test buffer is cleaned (line 9) and the current classifier is used to classify the current instance (line 11).

**Algorithm 1:** RCD algorithm in the training phase.

**Data**: ($c_a$) Current classifier, ($b_a$) Current buffer, ($c_n$) New classifier, ($S$) Data stream, ($b_n$) New buffer
**Result**: $C$: Classifiers list, $B$: Buffers list

```
1  begin
2      C ← {Create(c_a)};
3      B ← {Create(b_a)};
4      foreach s ∈ S do
5          level ← DDM(c_a, s);
6          switch level do
7              case WARNING
8                  if c_n = null then
9                      Create (c_n);
10                     Create (b_n);
11                 end
12                 b_n ← b_n ∪ {s};
13                 Train (c_n,s);
14             case DRIFT
15                 if StatTest (C, B, b_n) then
16                     (c_a, b_a) ← Stored (C, B, b_n);
17                 end
18                 else
19                     C ← C ∪ {c_n};
20                     B ← B ∪ {b_n};
21                     c_a ← c_n;
22                     b_a ← b_n;
23                 c_n = b_n = ∅;
24             otherwise
25                 if c_n ≠ null then
26                     c_n = b_n = ∅;
27                 end
28                 b_a ← b_a ∪ {s};
29             end
30         end
31         Train (c_a,s);
32     end
33 end
```

This is a big difference between RCD and the other approaches. While RCD can choose the best stored classifier in the testing phase, the other approaches use the classifier(s) as finished in the training phase. These classifiers better represent the last concepts seen in the training phase. Concepts seen in the beginning of the training phase are not well represented. RCD, on its turn, have classifiers adapted to several different concepts which can be selected in the testing phase based on current data.

There are many parameters that can be used to fine-tune RCD. Below the list of available parameters of RCD is presented.

**Algorithm 2:** RCD algorithm in the testing phase.

```
Data: C: Classifiers list, B: Buffers list, (c_c) Current classifier, (b_t) Test buffer, (S) Data stream
1  begin
2      b_t ← ∅;
3      foreach s ∈ S do
4          b_t ← b_t ∪ {s};
5          if b_t is full then
6              if StatTest (C, B, b_t) then
7                  c_c ← Stored (C, B, b_t);
8              end
9              b_t ← ∅;
10         end
11         Classify (c_c, s);
12     end
13 end
```

- $a$: Non-parametric multivariate statistical test to use. Up to the moment, two statistical tests are available: KNN and Cramer;

- $s$: the significance value (p-value). It informs the amount of similarity between distributions. Common values are 0.01 and 0.05. Lower values indicate that the similarity between the buffers must be higher to the statistical test indicate that they are similar;

- $b$: the maximum amount of instances to represent each distribution. With more instances, the algorithm tends to provide better accuracy but takes longer to compare the distributions;

- $t$: the rate the tests will be made in the testing phase. Smaller values mean tests are made more frequently, positively influencing the accuracy but reducing performance;

- $c$: the maximum amount of classifiers to store. Small sizes make RCD faster, but reduce its ability to represent different data distributions, possibly reducing its predictive accuracy;

- $m$: the thread pool size, indicating how many simultaneous tests are allowed to occur. If the computer has more than one core, RCD can perform several statistical tests in parallel, reducing the evaluation time (more details are given later);

- $d$: the drift detection method used by RCD to inform the state of concept drift. Possible values are DDM, EDDM, ECDD, and PHT.

The KNN statistical test available at RCD was based on the MTSKNN (Chen et al., 2010) package of The R Project for Statistical Computing tool. This package was implemented based on Schilling (1986) and Henze (1988). On its turn, Cramer was based on the implementation of the Cramer package (Franz, 2006) and was implemented based on Baringhaus and Franz (2004). The two following paragraphs briefly describe how these statistical tests work.

The multivariate KNN works by joining together the instances from both samples that need to be compared. The instances from the first sample are assigned a new attribute with value 1 while the instances from the second sample are assigned value 2. Each instance is compared to all the other instances from both samples. The metric used was the Euclidean distance. It is computed summing the distance from each corresponding attribute (the number of attributes must be the same), with the $k$ nearest ones being stored. The p-value is computed by analyzing how the $k$ nearest neighbors of all instances are normally distributed between the two samples.

The statistic computed by the Cramer test is "the difference of the sum of all the Euclidean interpoint distances between the random variables from the two different samples and one-half of the two corresponding sums of distances of the variables within the same sample. The asymptotic null distribution of the test statistic is derived using the projection method and shown to be the limit of the bootstrap distribution" (Baringhaus and Franz, 2004).

The proposal here presented was implemented using the MOA framework, developed at Waikato University, New Zealand, by the same group that developed and maintain WEKA (Hall et al., 2009). MOA is a framework to mine data streams. It offers a collection of machine learning algorithms, evaluation tools, as well as data set generators commonly used in data streams research. Instructions on how to download and use RCD can be found at `https://sites.google.com/site/moaextensions/`.

## 3.1 PARALLEL RCD

Originally, as presented at Figure 3.1, RCD performed the statistical tests sequentially. Thus, a statistical test would be performed comparing current data to data stored in the buffer for classifier 1 to verify if both represented the same data distribution. If positive, this classifier was considered the new current classifier. If negative, a statistical test would be performed on classifier 2 buffer data, and so on.

One improvement made was to perform several tests *simultaneously* by the use of a thread pool of configurable fixed size to allow the user to fine tune its value based on the

hardware being used. Figure 3.2 presents an example illustrating how the thread pool works. It considers a thread pool of size two and a classifiers set of size six.

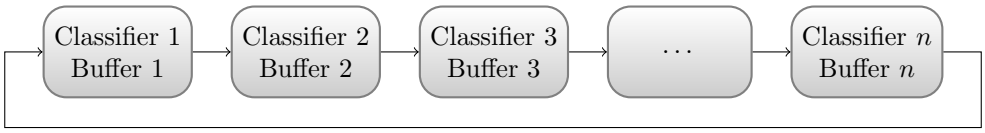



**Figure 3.1** RCD classifiers set.

When a concept drift occurs, it means the current classifier does not correctly represent the current context. So, it is necessary to check whether any stored classifier better represents the current context. The remaining classifiers stored in the set must be tested, comparing a sample of current data to the data stored in the buffer associated with each classifier which represents the data the classifier was trained on.

In the example of Figure 3.2, five threads are built to perform the statistical tests – the data associated to the current classifier is not sent as it is not correctly classifying recent instances. These threads are sent to the thread pool using a FIFO scheme to associate each test to a position in the thread pool, but only the first two are active, i.e., only these two are actually performing a statistical test. At Figure 3.2 they are represented by dashed lines, and inactive threads by solid lines. At this point ($t = 0$), two statistical tests are active and the remaining three are waiting for their turn to be executed.

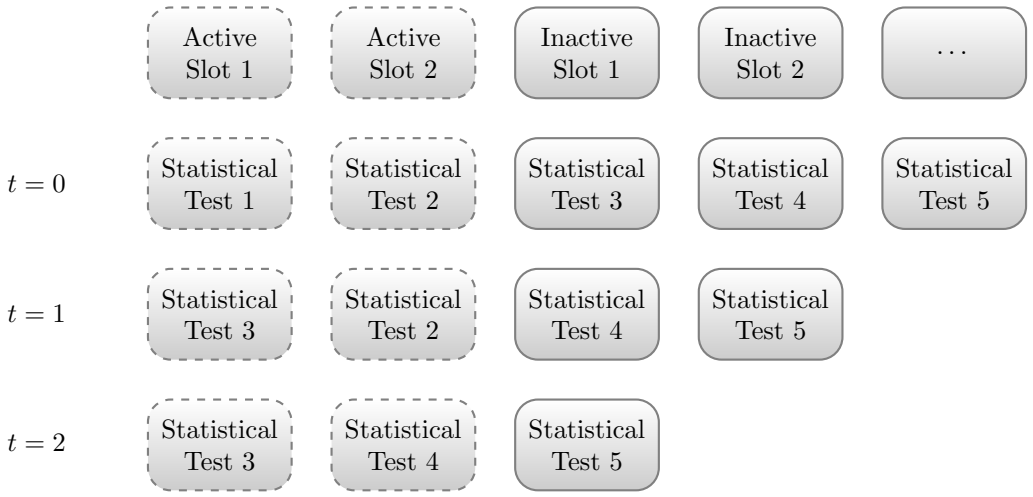



**Figure 3.2** Example of a thread pool execution.

When the first statistical test finishes execution (let's consider it is statistical test 1), if the result indicates that current data and sample data from classifier 1 do not represent the same data distribution, the next inactive statistical test (in this case, statistical test

3) executes in the place of statistical test 1 ($t = 1$). At $t = 2$, the same occurs. Classifier 2, represented by statistical test 2, also does not better represent current data and the next statistical test (number 4) occupies its place.

Now, let's consider that statistical test 3 has finished and it identified that current data and the data stored in the buffer of classifier 3 represent the same distribution. In this situation, classifier 3 substitutes the current classifier, all other active statistical tests are stopped from executing, and the inactive ones are canceled.

This scheme is interesting because, if a given test is negative, the next test is already being executed, allowing a faster performance of the algorithm. If a test is positive, all other executing tests are stopped and tests yet to be executed do not enter the active thread pool. This scheme is general and allows the execution of any statistical tests in parallel.

Several tests were performed to identify what was the best thread pool size to use based on the number of active cores in the processor. Results can be obtained at Chapter 6.

Other possibility in using parallel statistical tests would be to compare all the stored buffers with the current data and select the classifier associated with the buffer more similar to the current data, instead of selecting the first classifier whose buffer is similar.

It is also possible to compare the data samples using more than one statistical test (for example, both KNN and Cramer), and only identify a similarity if any or all the tests indicates so.

CHAPTER 4

# EMPIRICAL ANALYSIS

In this chapter, the RCD framework is compared with seven algorithms (among single and ensemble approaches) in terms of accuracy and concept drift handling. The tests were performed in commonly used data sets in concept drift research, including abrupt and gradual concept drifts, in both artificial and real data sets. In this chapter, the KNN statistical test is used. At Chapter 5, a comparison between KNN and Cramer is performed.

In section 4.1, it is informed the algorithms used in the experiments. Section 4.2 describes the data sets used in the experiments, how they were created, training and testing set sizes, and information about them. Section 4.3 informs tests performed to identify the best parameters to use in the RCD framework. Section 4.4 presents the results of the comparison between RCD and the algorithms described at Section 4.1. Finally, Section 4.5 presents the conclusions.

## 4.1 ALGORITHMS

In this section it is informed the algorithms used in the experiments, as well as its parameters. Their descriptions can be found at Chapter 2.

The base classifier used in the experiments was a Hoeffding tree that is a mixture of VFDTc and CVFDT, more specifically, CVFDT supporting numerical attributes and using naive Bayes at the leaves. From now on, we will call it Hoeffding Tree with Naive Bayes (HTNB). The same classifier was also used as base learner in the other algorithms, so we can exclude its influence in the comparison. The statistical tests were performed sequentially.

Concerning the tested algorithms, the following were used: HTNB, EDDM, AUE, DWM, EB, LNSE, and WMA. All have freely available implementations in the MOA framework, except for DWM, EB, and LNSE, which were implemented and are available at `https://sites.google.com/site/moaextensions/` as MOA extensions.

The parameters used by HTNB in the experiments are the default values defined in the MOA framework: number of instances a leaf should observe between split attempts

(200), split criterion to use (information gain), allowable error in split decision ($10^{-7}$), and threshold below which a split will be forced to break ties (0.05). Being the base classifier of RCD, it was important to analyze how RCD compared to it.

The parameters of EDDM are also the default values in MOA. This algorithm was chosen because RCD uses it as its drift detection method, so it is important to compare them to verify if RCD improves EDDM and, if so, in which conditions.

AUE also used the default values in the MOA framework: the maximum number of classifiers in the ensemble was set to 15, the maximum number of classifiers to store and choose from when creating an ensemble was set to 30, and the chunk size used for classifier creation and evaluation was set to 500. The ensemble size of AUE is equal to the classifiers set in RCD. Other values for the chunk size were tested (250 and 1,000), but 500 obtained the best results.

The parameters used in the EB classifier were: acceptable error threshold for creating new classifiers ($\tau$) was set to 0.05; acceptance factor ($a$), used to select classifiers to form the ensemble, was set to 0.4; and chunk size ($c$) was set to 400 (Ramamurthy and Bhatnagar, 2007). As this method is similar to AUE, we added a similar pruning strategy storing at most 30 classifiers and selecting up to 15 to form the ensemble.

To implement the DWM approach, we used the values presented by Kolter and Maloof (2007): the factor for decreasing classifier's weights ($\beta$) was set to 0.5; the threshold for deleting experts ($\theta$) was set to 0.01; the ensemble size is unlimited; and the period between expert removal, creation, and weight update ($p$) was set to 50. The $p$ value was also tested with the other values presented at the original article ($p = 1$ and $p = 10$), but $p = 50$ presented the best results. The other values were not modified because, according to the authors, "varying $\beta$ affected performance little; the selected value for $\theta$ did not affect accuracy, but did reduce considerably the number of experts".

The parameters of LNSE were based on Elwell and Polikar (2011): size of the environments ($p$) was set to 250; the slope ($a$) and the halfway crossing point ($b$) of the sigmoid function controlling the number of previous periods taken into account during weighting were set to 0.5 and 10, respectively; the ensemble size ($e$) was set to 15; and the pruning strategy ($s$) to classifiers with higher error on recent data chunk.

In the WMA approach, we also used the default configuration of the MOA framework: factor to punish mistakes of classifiers (0.9), minimum fraction of weight per model (0.01), and an ensemble with four classifiers of the following types: naive Bayes, CVFDT (Hulten et al., 2001), HTNB, and a decision trees with Hoeffding bounds and adaptive naive Bayes and majority class at the leaves (Bifet et al., 2009a). This ensemble uses the diversity of

classifiers to improve accuracy over new instances.

In summary, we compare RCD to one classifier that tries to adapt to concept drifts (HTNB); one classifier that creates a new HTNB classifier every time it identifies a concept drift (EDDM); one ensemble where each classifier is responsible for representing part of the data (AUE); one ensemble where all classifiers are trained with all data, that uses more than one base classifier but does not prune them (WMA); one ensemble classifier that removes models when the global performance is below a specified threshold (DWM); and, finally, two ensemble classifiers built to handle recurring concept drifts (EB and LNSE).

## 4.2 DATA SETS

We selected the following artificial data sets to perform the experiments: Hyperplane, LED, SEA, STAGGER, and Waveform, described at Chapter 2. They are commonly used in the concept drift research area and are also freely available at MOA framework. We used them to test for both abrupt and gradual concept drifts.

To create the abrupt concept drifts data sets, we used the following scheme: 25,000 examples of each concept were generated. The first 12,500 examples of each concept were appended together and used to form the training set. Thus, an abrupt concept drift is simulated at each 12,500 examples. The remaining 12,500 examples were kept separate and used as testing set to analyze how the algorithms represented the specific concepts. We repeated this procedure 30 times, thus creating 30 training sets and 30 testing sets for each concept. The first training set is tested on the first testing set of each concept, and so on.

To create the data sets with gradual concept drifts, we used a similar approach. However, a gradual concept drift was simulated by increasing the probability that instances of the new concept be selected during a period of 1,000 instances around each 12,500 instances. Therefore, when the stream reaches 12,000 instances, the probability that instances of the new concept be selected increases from 0 to 1, for 1,000 instances.

In these data sets, we first executed the training phase, constituted of different contexts and concept drifts, and then the testing phase. This methodology better verifies how the classifiers adapted to the concepts seen in the training phase and is more representative of real-world usage.

To evaluate the real data sets, a different approach must be taken, because the data sets already exist. An evaluation methodology that has already been used in real-world data sets is Interleaved Test-Then-Train (Bifet et al., 2009b, 2010b), also named Prequential. Every example is used for testing the model; then, it is used to train it. Here, all the

instances are used for both training and testing, making maximum use of the data, yet the classifier is always tested on examples it has not seen. The accuracy was measured as the final percentage of instances correctly classified over the interleaved evaluation, as described by Bifet et al. (2010b).

To compare the classifiers, we followed the procedure described by Demšar (2006), using the Iman and Davenport (1980) statistic ($F_F$), which is derived from the Friedman non-parametric statistical test (Friedman, 1937, 1940). The Friedman test ranks the algorithms for each data set separately, giving rank 1 for the best performing algorithm, rank 2 to the second best, and so on. The average rank of each algorithm ($\overline{R}$) and the Friedman statistic ($\chi^2_F$), presented at Equation 4.1, are computed (considering $k$ algorithms and $N$ data sets).

$$\chi^2_F = \frac{12N}{k(k+1)} \left[ \sum \overline{R}^2 - \frac{k(k+1)^2}{4} \right] \tag{4.1}$$

Iman and Davenport (1980) computed a better statistic ($F_F$), presented at Equation 4.2, after proving that the Friedman statistic was undesirably conservative. $F_F$ is distributed according to the F-distribution. If $F_F$ is lower than the critical value found in the F-distribution, the null-hypothesis is satisfied, indicating all algorithms have similar performances. If the null-hypothesis is rejected, a post-hoc test is used.

$$F_F = \frac{(N-1)\chi^2_F}{N(k-1) - \chi^2_F} \tag{4.2}$$

One post-hoc test used was the Nemenyi test (Nemenyi, 1963). A Critical Difference (CD) is computed and compared with the average ranks to obtain classifiers with similar performance. If the difference between two classifiers is above CD, we can conclude that the classifier with lower average rank is significantly better.

The Bonferroni-Dunn post-hoc test (Dunn, 1961), on the other hand, was used to compare one classifier to the others. It is similar to the Nemenyi test, also computing a CD, but controls the family-wise error rate.

Table 4.1 presents a summary of the main information of the artificial and real-world data sets used in the experiments like number of attributes, classes, numerical and categorical attributes, number of data sets created, if data has noise and irrelevant attributes.

**Table 4.1** Information about data sets.

| | Data sets | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Hyper-plane** | **LED** | **SEA** | **Stagger** | **Wave-form** | **Cover-type** | **Electri-city** | **Poker Hand** | **Weather** |
| **Attributes** | 10 | 24 | 3 | 3 | 21 | 54 | 8 | 10 | 8 |
| **Classes** | 2 | 10 | 2 | 2 | 3 | 7 | 2 | 10 | 2 |
| **Categoric** | 0 | 24 | 0 | 3 | 0 | 44 | 1 | 5 | 0 |
| **Numeric** | 9 | 0 | 3 | 0 | 21 | 10 | 7 | 5 | 8 |
| **Data sets** | 9 | 4 | 4 | 3 | 4 | 1 | 1 | 1 | 1 |
| **Noise** | 5% | 10% | 10% | No | No | - | - | - | - |
| **Irrelevant** | No | Yes | Yes | No | No | - | - | - | - |
| **Size** | 112,500 | 50,000 | 50,000 | 37,500 | 50,000 | 581,012 | 45,312 | 829,201 | 18,159 |

## 4.3 RCD PARAMETRIZATION

The maximum allowed number of stored classifiers in the RCD framework was set to 15, the same number used in the AUE and EB ensembles. If the set is full, older classifiers are replaced in a FIFO scheme. A comparison with different levels of parallelization is presented at Chapter 6.

The values used in the tests are presented in Table 4.2. To identify the best parameter set, 240 different configurations were tested for the five parameters described in the first column. The tests were performed in the data sets with abrupt and gradual concept drifts. Each RCD configuration was compared to the algorithms described at Section 4.1, in the data sets described at Section 4.2, summing 8,400 comparisons (remembering that each comparison was repeated 30 times).

The configuration used in the experiment is informed in the last column of Table 4.2, based on the overall test error across the tested data sets.

**Table 4.2** Values of the parameters used to find the best configuration to use in RCD.

| **Parameter** | **Start value** | **End value** | **Step** | **Configuration** |
|---|---|---|---|---|
| $s$ | 0.01 | 0.05 | - | 0.01 |
| $b$ | 100 | 500 | 100 | 400 |
| $t$ | $b$ | 500 | 100 | 400 |
| $k$ | 1 | 7 | 2 | 5 |
| $d$ | DDM | EDDM | - | EDDM |

## 4.4 RESULTS

In this section, results are presented for each data set, considering abrupt and gradual concept drifts, predictive accuracy of the algorithm in each context, concept drift handling, and train and test times.

### 4.4.1 LED

Concerning the LED problem, four data sets were generated, each one with a different number of drifting attributes (1, 3, 5, and 7). This configuration is similar to the one used by Bifet et al. (2010b).

At Table 4.3, columns 1 to 7 present the average accuracy results of the algorithms in each testing set. Columns $\mu$ and $\sigma$ present the average accuracy and standard deviation over all data sets, in all repetitions. The last column presents the average rank and is used to compute the Friedman statistic. The bold values inform the best algorithm for a specific configuration. It is interesting to note that the best accuracy of every algorithm resides in the last concept. As most of the algorithms tend to adapt to the last instances presented to them in the training phase or give to classifiers that correctly represent them higher weights, it is no surprise that the best accuracy concentrates in the last concept.

**Table 4.3** Statistics in LED data set with abrupt concept drifts.

| Classifier | 1 | 3 | 5 | 7 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|
| AUE | 13.97 | 27.88 | 58.56 | **74.04** | 43.61 | 23.98 | 4.00 |
| DWM | 10.57 | 12.92 | 18.68 | 22.50 | 16.17 | **12.15** | 8.00 |
| HTNB | 18.70 | 35.53 | 69.33 | 72.04 | 48.90 | 22.87 | 3.25 |
| EB | 13.97 | 27.74 | 58.74 | 74.01 | 43.61 | 24.03 | 4.50 |
| EDDM | 15.12 | 31.14 | 67.94 | 73.29 | 46.87 | 24.76 | 3.50 |
| LNSE | 13.65 | 26.41 | 52.81 | 68.28 | 40.29 | 21.62 | 6.75 |
| WMA | 18.76 | 35.61 | **69.38** | 72.08 | 48.96 | 22.85 | **2.25** |
| RCD | **48.95** | **67.34** | 49.94 | 70.97 | **59.30** | 15.59 | 3.75 |

As we can see, RCD better represented concepts 1 and 3; WMA was better at concept 5; and AUE at concept 7. Considering all concepts, RCD had the higher average of all algorithms in this data set. Computing $F_F$ for this data set, we obtain 4.75, which is higher than the critical value 2.49 (valid when testing eight algorithms in four data sets with 95% confidence), obtained by the F-distribution, thus indicating that some classifiers have different performances. The CD computed by the Nemenyi test was equal to 5.25 (valid in the same conditions of the critical value), indicating two groups of algo-

rithms: DWM is significantly worse than that of WMA. Concerning the other algorithms, the experimental data is not sufficient to reach any conclusion about which group they belong. The Bonferroni-Dunn test returns a CD of 4.66, yielding similar results. Reducing the confidence level to 90%, the CD of the Bonferroni-Dunn test is equal to 4.24, now indicating that RCD is also significantly better than DWM.

Figure 4.1 presents how the algorithms deal with abrupt concept drifts in the testing phase. The values presented are the averages of 30 train/test executions, as used to create Table 4.3. It is possible to verify that RCD performed better during all concept drifts. Besides accuracy, having a low standard deviation is another important characteristic an algorithm that deals with concept drifts must have. It informs that, even when a drift occurs, the algorithm's accuracy does not change much. Analyzing the $\sigma$ column, we can observe that DWM had the smallest standard deviation (but also the lowest accuracy), followed by RCD.

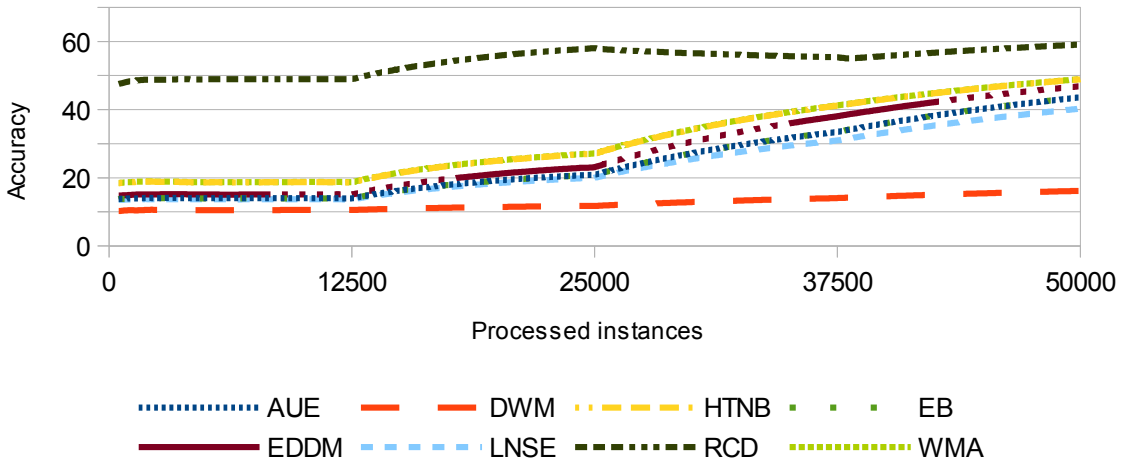
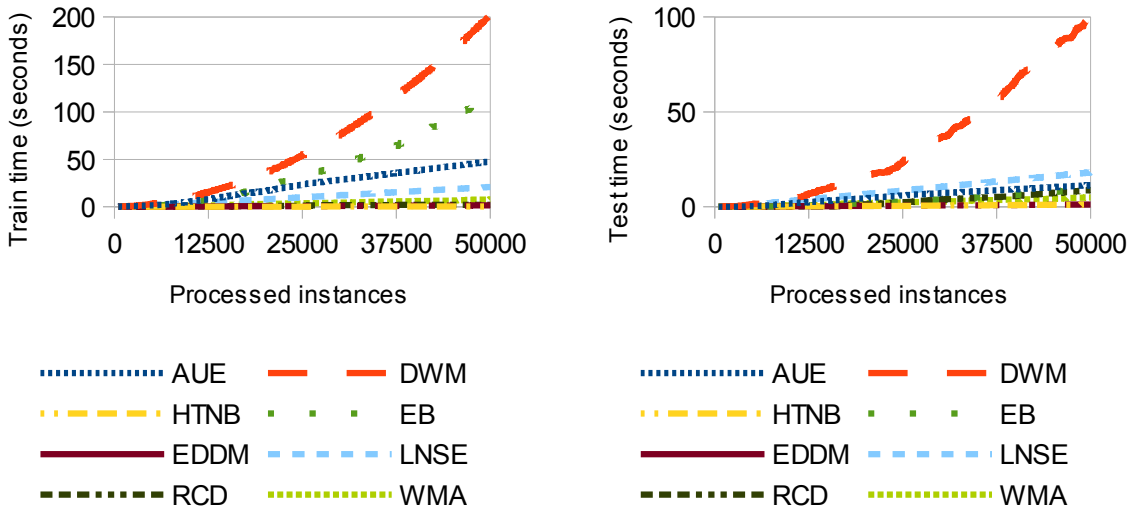



**Figure 4.1** Accuracy on the LED data set with abrupt concept drifts.

To obtain the train and test times it was used the interleaved chunks approach presented by Brzeziński (2010) and used by Brzeziński and Stefanowski (2011). It works evaluating the classifiers using data chunks. It stores a data chunk of size $s$, tests the current classifier in this data chunk, trains it, and then disposes the chunk. It is also actually available at the MOA framework.

Figures 4.2(a) and 4.2(b) present the train and test times taken by the algorithms to process one set with 50,000 instances and three abrupt concept drifts. In both the train and test times, the single classifier approach (represented by HTNB) was faster than the ensemble approach. HTNB took 0.312 seconds to train the classifier. On the other

hand, DWM took approximately 200 seconds. The training time of the DWM approach takes much more time than the other algorithms. It was followed by EB, which took approximately 121 seconds, and AUE, with 48 seconds.




(a) Train time.

(b) Test time.

**Figure 4.2** Train and test times on the LED data set with abrupt concept drifts.

DWM is the method that takes longer to train because it does not have a maximum number of classifiers to store. It only removes a classifier when its weight is less than a specified threshold. Thus, its tendency is to grow the ensemble size without limits as more instances arrive. It is possible to observe its effect by looking at its curve: it is not linear, reflecting that the ensemble takes longer to train because of the number of stored classifiers.

The reasons why EB is one of the methods that spends more time training are that, for each data chunk, (1) all stored classifiers must be tested against all instances in the current data chunk and, only those with classification error below the acceptance factor become part of the ensemble; (2) sort the ensemble based on the error and keep the 15 best ones; (3) if the error of the ensemble on the current chunk is above a specified threshold, create a new classifier, train it on all instances of the current data chunk and substitute the classifier with lower accuracy. The original EB did not prune the ensemble, taking even more time than the version here tested.

AUE also takes so much time because of its internal structure: to each new data chunk that arrives, AUE creates a candidate classifier, performs a 10-fold cross validation and computes the weight of the candidate classifier and of the stored classifiers. If there is

space left in the ensemble, it stores the candidate classifier; otherwise, it substitutes the poorest stored classifier with the candidate one (if it has a higher weight). Then, it uses the best stored classifiers to form the ensemble. In the default configuration of the MOA framework, AUE stores 30 classifiers and uses the best 15 to form the ensemble. So, the use of cross-validation and the number of stored classifiers explain why it takes more time than the other algorithms to perform the training phase.

In the testing phase, DWM also had the worse performance. It needed approximately 101 seconds to test the instances while LNSE, with the second worse performance, took 18 seconds. HTNB was the fastest algorithm, taking 1.2 seconds to complete the testing phase.

RCD was the only algorithm that spent more time in the testing phase than in the training phase. In the testing phase RCD periodically needs to verify if recent data is better represented by a different classifier. So, performing the statistical test is the reason for RCD taking more time when compared to the other approaches, which use the classifier without modification. This is the cost to dynamically select the best classifier based on current data. All the other classifiers in the testing phase use the model as it was built on the training phase, differently of RCD.

Table 4.4 is similar to Table 4.3 but refers to the LED data set with gradual concept drifts. The majority of the algorithms had very similar average performances, around 39%. AUE better represented concepts 3 to 7, and had highest average accuracy in all concepts. DWM was the algorithm with the lowest standard deviation.

The $F_F$ statistic returned the value 5.06, higher than the critical value, 2.49, indicating that there are classifiers with different performances. Performing the Nemenyi and the Bonferroni-Dunn tests, with 95% confidence, returned CD's equal to 5.25 and 4.66, respectively. Both tests inform that AUE have better performance than DWM. Concerning the other algorithms, it is not possible to inform to which group they belong.

Differently from the data set with abrupt concept drifts, here, the accuracy of the algorithms decrease in time. The best represented concept was always the first observed one. Dealing with gradual concept drifts is generally a more difficult task. The classifiers do not forget the concepts as rapidly as with abrupt concept drifts.

Figure 4.3 presents how the algorithms deal with gradual concept drifts. DWM had the worse performance in all contexts. The curve of LNSE is slightly lower than the majority of the other classifiers. The other classifiers performed similarly during all the instances, handling the concept drifts with similar behavior.

Figures 4.4(a) and 4.4(b) present the training and testing times for the LED data

**Table 4.4** Statistics in LED data set with gradual concept drifts.

| Classifier | 1 | 3 | 5 | 7 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|
| AUE | 69.87 | **45.34** | **25.11** | **17.24** | **39.39** | 23.07 | **2.00** |
| DWM | 20.82 | 17.57 | 12.26 | 11.13 | 15.44 | **9.99** | 8.00 |
| HTNB | 69.90 | 44.37 | 24.62 | 16.81 | 38.92 | 23.13 | 4.75 |
| EB | 69.89 | 44.88 | 24.73 | 17.07 | 39.15 | 23.06 | 3.00 |
| EDDM | 69.87 | 44.43 | 24.95 | 16.78 | 39.01 | 23.18 | 4.50 |
| LNSE | 64.01 | 42.44 | 24.27 | 16.85 | 36.89 | 20.74 | 6.00 |
| WMA | 69.94 | 44.50 | 24.78 | 17.04 | 39.07 | 23.06 | 2.75 |
| RCD | **71.95** | 44.42 | 23.63 | 16.25 | 39.06 | 23.20 | 5.00 |

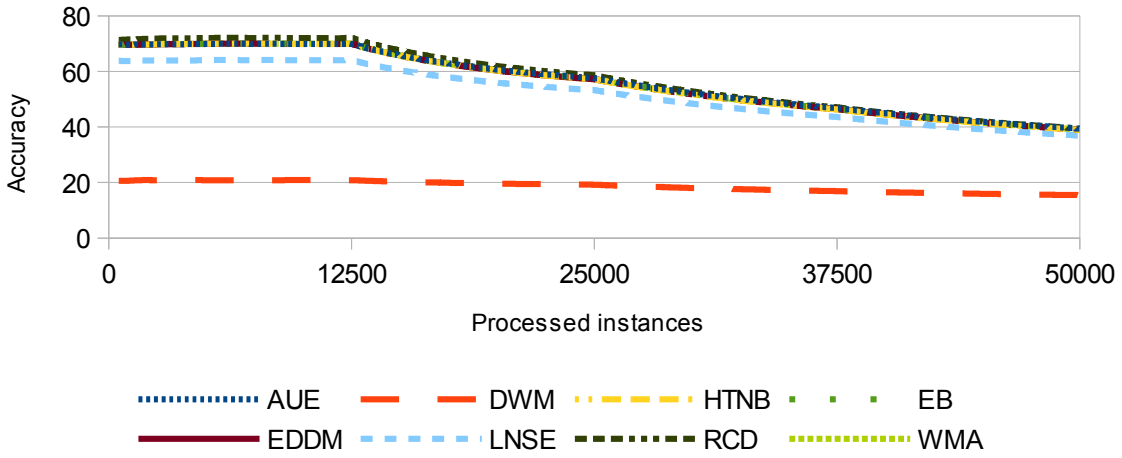
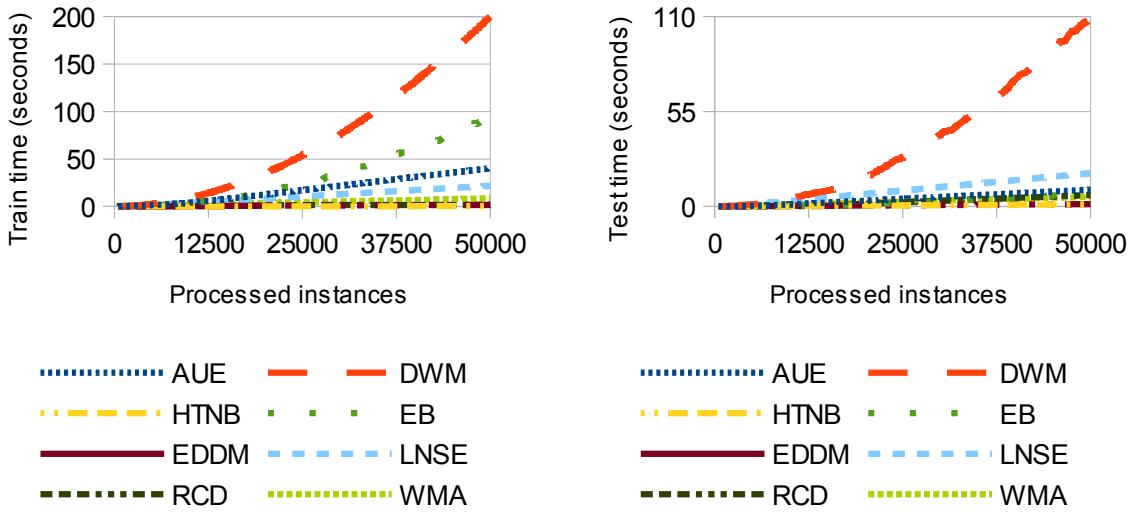



**Figure 4.3** Accuracy on the LED data set with gradual concept drifts.

set with gradual concept drifts. The results were quite similar to the ones presented at Figures 4.2(a) and 4.2(b), respectively. DWM was also the slowest algorithm in both training and testing phases, while the second slowest was EB, in the training phase, and LNSE, in the testing phase. On the other hand, HTNB was the fastest algorithm in both phases.

### 4.4.2 Hyperplane

The data set used in the experiments is constituted of nine numerical attributes and one binary categorical class. To perform the tests, we used the same settings as Fan (2004) and Tsymbal et al. (2008). There are nine distinct hyperplane data sets, varying the number of attributes that suffer concept drift ($k$) and the magnitude of change for each




(a) Train time.

(b) Test time.

**Figure 4.4** Train and test times on the LED data set with gradual concept drifts.

example ($t$). Table 4.5 describes the values of the parameters used in the construction of the data sets. The probability that the direction of change is reversed was set to 10%.

**Table 4.5** Values of the parameters used in the Hyperplane data set.

| Parameter | Data sets | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| $k$ | 2 | 2 | 2 | 5 | 5 | 5 | 8 | 8 | 8 |
| $t$ | 0.1 | 0.5 | 1.0 | 0.1 | 0.5 | 1.0 | 0.1 | 0.5 | 1.0 |

Table 4.6 presents the results for the Hyperplane data set with abrupt concept drifts. It shows that the number of drifting concepts ($k$) has a bigger influence in the results than the magnitude of change ($t$) for all data sets. In these algorithms, concepts 1 to 3, 4 to 6 and 7 to 9 have very similar accuracies, indicating that the concepts they represent are also similar. RCD best represented concepts 1 to 6, while AUE best represented concepts 7 to 9. The best average accuracy and rank were obtained by RCD. Like in the LED data set, DWM was again the stablest algorithm, followed by RCD. The algorithms also usually better represented the last observed concepts.

Figure 4.5 presents how the algorithms deal with abrupt concept drifts in the testing phase. Here we can visualize that DWM and RCD are really the stablest algorithms, better adapting their base learners when they identify a concept drift. The other algorithms

**Table 4.6** Statistics in Hyperplane data set with abrupt concept drifts.

| Classifier | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AUE | 37.14 | 37.10 | 37.10 | 36.66 | 36.63 | 36.63 | **87.84** | **87.82** | **87.82** | 53.86 | 25.74 | 5.33 |
| DWM | 49.47 | 49.47 | 49.46 | 49.16 | 49.15 | 49.16 | 51.03 | 51.04 | 51.04 | 49.89 | **2.56** | 4.67 |
| HTNB | 45.50 | 45.49 | 45.49 | 50.82 | 50.80 | 50.80 | 74.51 | 74.53 | 74.53 | 56.94 | 14.03 | 3.67 |
| EB | 37.74 | 37.69 | 37.69 | 37.20 | 37.18 | 37.17 | 85.91 | 85.90 | 85.90 | 53.60 | 24.67 | 5.00 |
| EDDM | 36.83 | 36.80 | 36.80 | 37.13 | 37.10 | 37.10 | 87.15 | 87.13 | 87.13 | 53.69 | 25.33 | 5.67 |
| LNSE | 38.05 | 38.01 | 38.01 | 37.33 | 37.30 | 37.30 | 84.76 | 84.75 | 84.75 | 53.36 | 24.01 | 4.67 |
| WMA | 45.02 | 45.01 | 45.01 | 50.48 | 50.45 | 50.45 | 74.90 | 74.92 | 74.92 | 56.80 | 14.47 | 4.00 |
| RCD | **57.71** | **57.71** | **57.71** | **58.48** | **58.46** | **58.45** | 61.41 | 61.43 | 61.44 | **59.20** | 6.83 | **3.00** |

have similar behavior, increasing their accuracies as they reach the last concept, clearly indicating that they better represent it.

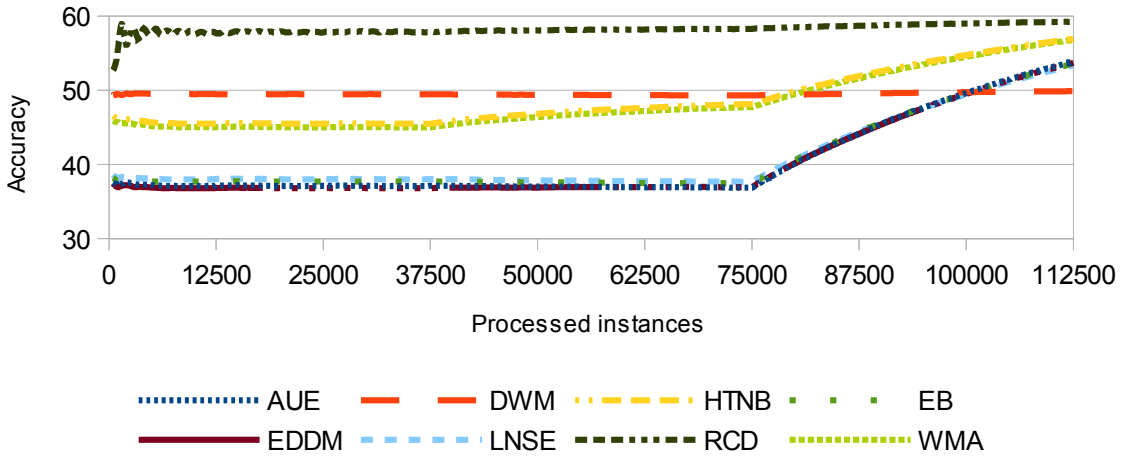
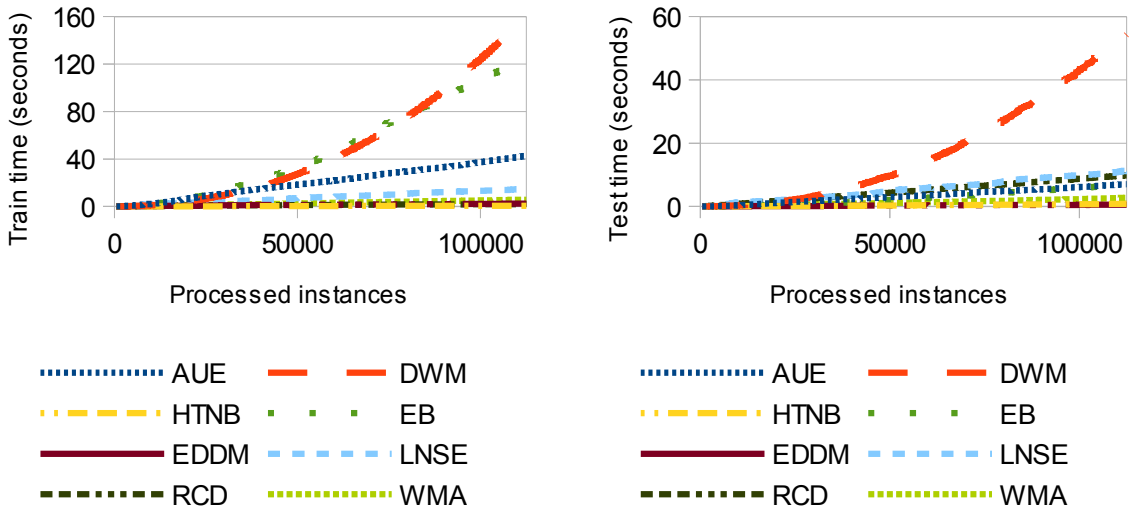



**Figure 4.5** Accuracy on the Hyperplane data set with abrupt concept drifts.

Figures 4.6(a) and 4.6(b) present the train and test times needed by the algorithms to process this data set. Both in the training and testing phases, DWM was the slowest algorithm, similarly to the results obtained in the LED data set.

Again, the single classifiers were the fastest algorithms in both train and test phases. HTNB and EDDM were the fastest, followed by RCD and WMA, in the training phase, and by WMA and EB in the testing phase.

Concerning gradual concept drifts, Table 4.7 presents the accuracy averages, the standard deviation, and the average ranks for the Hyperplane data set. Similarly to the data set with abrupt concept drifts, the majority of the algorithms had practically the same average accuracies, around 55%. LNSE better represented concepts 1 to 3, AUE, concepts




(a) Train time.

(b) Test time.

**Figure 4.6** Train and test times on the Hyperplane data set with abrupt concept drifts.

4 to 6, and DWM, concepts 7 to 9. AUE also had the highest average accuracy over all data sets, while DWM presented best standard deviation. Even not having best represented any concept, RCD presented the lowest average rank. For this data set, $F_F$ is 0.95, below the critical value (2.18), indicating that all classifiers have similar performances.

Concerning the algorithms stability, discarding DWM due to its lower statistical accuracy (in spite of its lower statistical results), all other algorithms had very similar results, with RCD presenting third best results.

**Table 4.7** Statistics in Hyperplane data set with gradual concept drifts.

| Classifier | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AUE | 61.59 | 61.52 | 61.52 | **62.80** | **62.71** | **62.70** | 42.81 | 42.77 | 42.77 | **55.69** | 12.80 | 3.67 |
| DWM | 54.28 | 54.25 | 54.25 | 55.07 | 55.05 | 55.04 | **46.89** | **46.87** | **46.86** | 52.06 | **7.03** | 5.67 |
| HTNB | 60.35 | 60.27 | 60.27 | 62.37 | 62.28 | 62.27 | 43.23 | 43.18 | 43.18 | 55.27 | 12.49 | 5.33 |
| EB | 61.32 | 61.25 | 61.24 | 62.62 | 62.54 | 62.53 | 42.94 | 42.89 | 42.89 | 55.58 | 12.52 | 4.00 |
| EDDM | 61.05 | 60.97 | 60.96 | 62.38 | 62.30 | 62.29 | 43.04 | 42.99 | 42.99 | 55.44 | 12.79 | 5.00 |
| LNSE | **61.67** | **61.61** | **61.60** | 62.55 | 62.47 | 62.46 | 42.87 | 42.83 | 42.83 | 55.66 | 12.17 | 4.00 |
| WMA | 60.60 | 60.53 | 60.52 | 62.49 | 62.41 | 62.40 | 43.11 | 43.06 | 43.06 | 55.35 | 12.47 | 4.78 |
| RCD | 60.83 | 60.77 | 60.77 | 62.68 | 62.61 | 62.60 | 43.09 | 43.06 | 43.06 | 55.49 | 12.31 | **3.56** |

Differently from the LED data set with gradual concept drifts where the accuracies decreased with time, here, all the algorithms initially increased their accuracies when

reaching the second context, decreasing again in the third one. One similarity with LED is that, again, the last context had the lowest accuracy, indicating that the algorithms did not represent it properly, better representing the second context.

Figure 4.7 presents how the algorithms deal with gradual concept drifts. Here we can see that all the algorithms had similar behavior when dealing with concept drifts. DWM, as expected, appears below the curves of the other algorithms, again indicating its lower accuracy in this data set configuration.

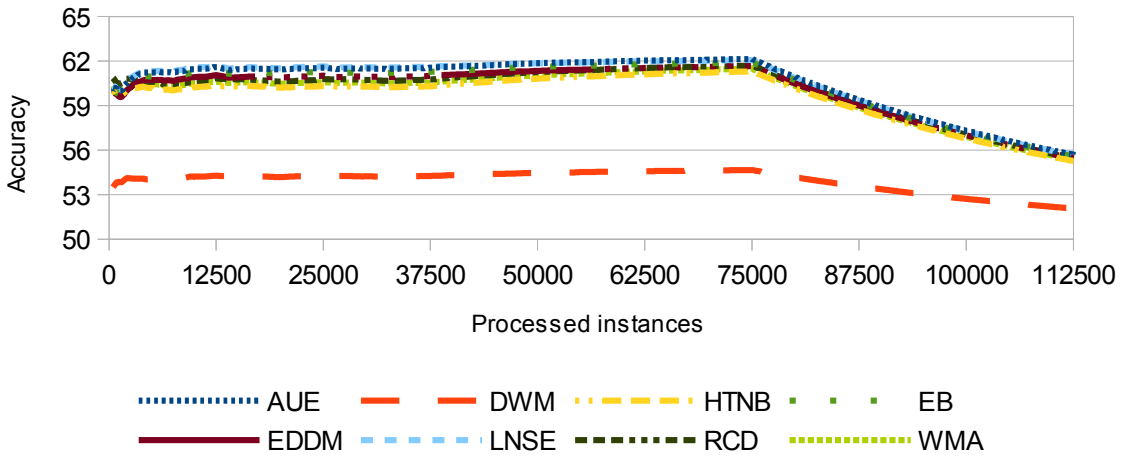


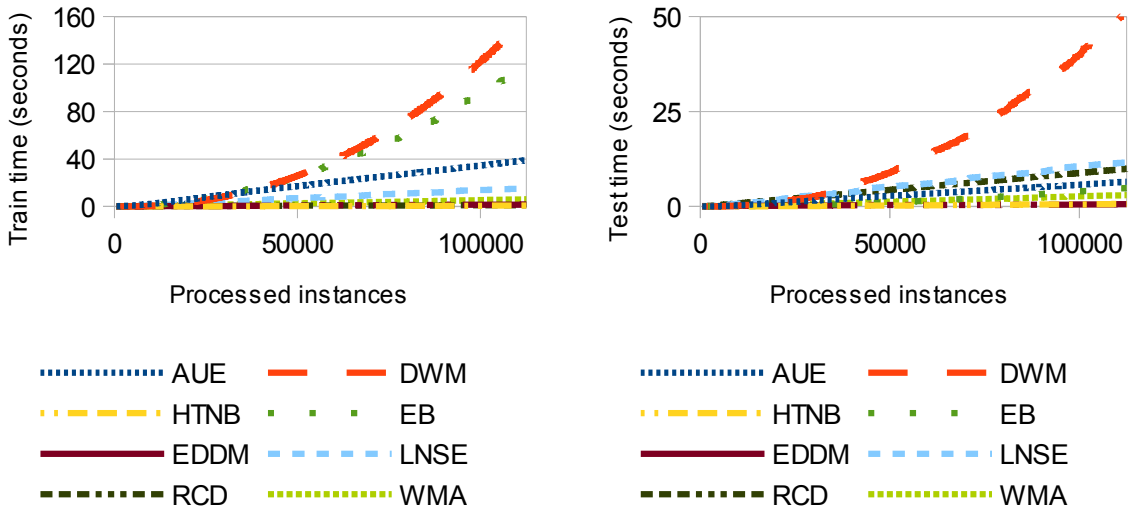

**Figure 4.7** Accuracy on the Hyperplane data set with gradual concept drifts.

Analyzing the time spent in the training and testing phases at Figures 4.8(a) and 4.8(b), similar results were obtained compared to the abrupt concept drifts version. In the training phase, beginning with the fastest, the order of the classifiers were HTNB, RCD, EDDM, WMA, LNSE, AUE, EB, and again, DWM was the slower algorithm. Concerning the testing phase, EDDM was the fastest, followed by HTNB, WMA, EB, AUE, RCD, LNSE, and DWM. Again, RCD was the only algorithm that spent more time in the testing phase than in the training phase.

### 4.4.3 Waveform

Like in the LED data set, we used a version of Waveform that creates drifts by changing the position of attributes. Again, we created four data sets, with one, three, five, and seven drifting attributes.

Table 4.8 shows how the algorithms represent each concept for the Waveform data set concerning abrupt concept drifts. In this data set, the algorithm that had the highest




(a) Train time.

(b) Test time.

**Figure 4.8** Train and test times on the Hyperplane data set with gradual concept drifts.

average accuracy and rank was WMA, with RCD being second best. The $F_F$ statistic returned 2.48, slightly below the critical value. Thus, with 95% confidence, we can not reject the null-hypothesis. Reducing the confidence to 90%, the critical value becomes 2.02, now rejecting the null-hypothesis. The Nemenyi and Bonferroni-Dunn tests are not able to identify differences among the classifiers with 95% confidence. Reducing to 90%, it is possible to affirm that WMA is significantly better than DWM.

**Table 4.8** Statistics in Waveform data set with abrupt concept drifts.

| **Classifier** | **1** | **2** | **3** | **4** | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|
| AUE | 35.23 | 36.32 | 53.30 | **81.29** | 51.54 | 18.74 | 5.75 |
| DWM | 36.14 | 38.02 | 38.99 | 46.20 | 39.84 | 8.49 | 6.50 |
| HTNB | 70.93 | 63.55 | 70.05 | 79.29 | 70.95 | **6.97** | 4.00 |
| EB | 35.22 | 36.40 | 54.02 | 80.57 | 51.55 | 18.43 | 6.00 |
| EDDM | 49.42 | 50.19 | 59.56 | 80.26 | 59.86 | 17.61 | 4.25 |
| LNSE | 35.36 | 37.22 | 53.52 | 80.69 | 51.70 | 18.44 | 5.00 |
| WMA | 73.37 | **69.13** | **73.41** | 80.67 | **74.14** | 7.12 | **1.75** |
| RCD | **76.85** | 64.45 | 70.73 | 79.57 | 72.90 | 11.53 | 2.75 |

Figure 4.9 presents how the algorithms deal with abrupt concept drifts. Here, we can confirm that AUE, DWM, LNSE, and EB had worse performances in this data set, with much lower curves when compared to those of the other algorithms.

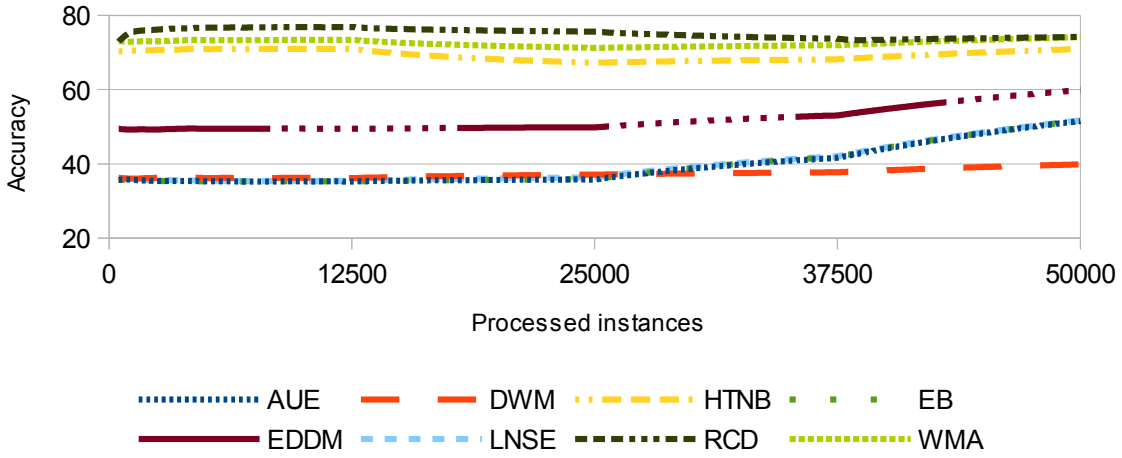
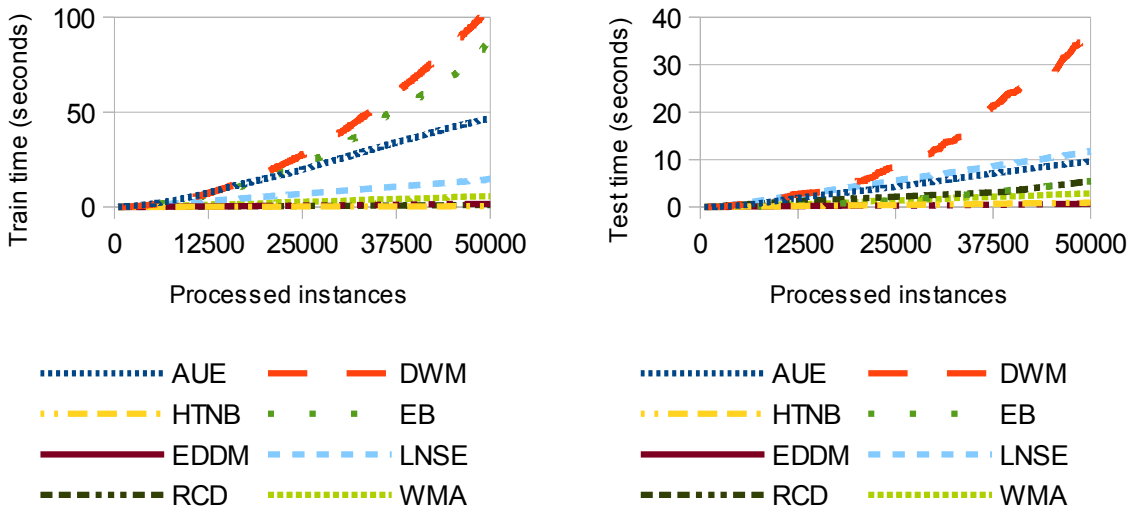



**Figure 4.9** Accuracy on the Waveform data set with abrupt concept drifts.

The training time, as presented at Figure 4.10(a), and the testing time, as presented at Figure 4.10(b), have similar results to the ones presented in the LED data set. Again, HTNB was the fastest and DWM the slowest in both phases.




(a) Train time.

(b) Test time.

**Figure 4.10** Train and test times on the Waveform data set with abrupt concept drifts.

Table 4.9 presents the results of the tests performed considering the version of the Waveform data set with gradual concept drifts. WMA presented the best average accuracy, closely followed by RCD. DWM was again the stablest algorithm, and RCD second best.

Even though RCD did not better represent any individual context, it had the lowest average rank, together with AUE. $F_F$ was equal to 1.27, below the critical value and confirming the null-hypothesis.

**Table 4.9** Statistics in Waveform with gradual concept drifts.

| Classifier | 1 | 2 | 3 | 4 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|
| AUE | 79.44 | **76.13** | 60.19 | 37.20 | 63.24 | 18.37 | **3.25** |
| DWM | 44.79 | 41.91 | 40.62 | 37.09 | 41.10 | **10.16** | 7.00 |
| HTNB | 80.28 | 73.45 | 63.21 | 36.13 | 63.27 | 17.73 | 4.25 |
| EB | 78.68 | 75.67 | 59.02 | 37.06 | 62.61 | 17.99 | 5.00 |
| EDDM | 78.60 | 72.97 | 62.34 | 36.26 | 62.54 | 17.90 | 5.75 |
| LNSE | 78.76 | 74.57 | 60.70 | **37.57** | 62.90 | 17.74 | 3.50 |
| WMA | **81.87** | 73.31 | **63.38** | 35.91 | **63.62** | 18.14 | 4.00 |
| RCD | 80.63 | 74.25 | 62.02 | 37.15 | 63.51 | 17.53 | **3.25** |

Figure 4.11 presents how the algorithms deal with the gradual concept drift version of this data set in the testing phase. They have similar patterns in dealing with concept drifts in this phase. The standard deviation of the algorithms were very similar, varying from 17.53 (RCD) to 18.37 (AUE).

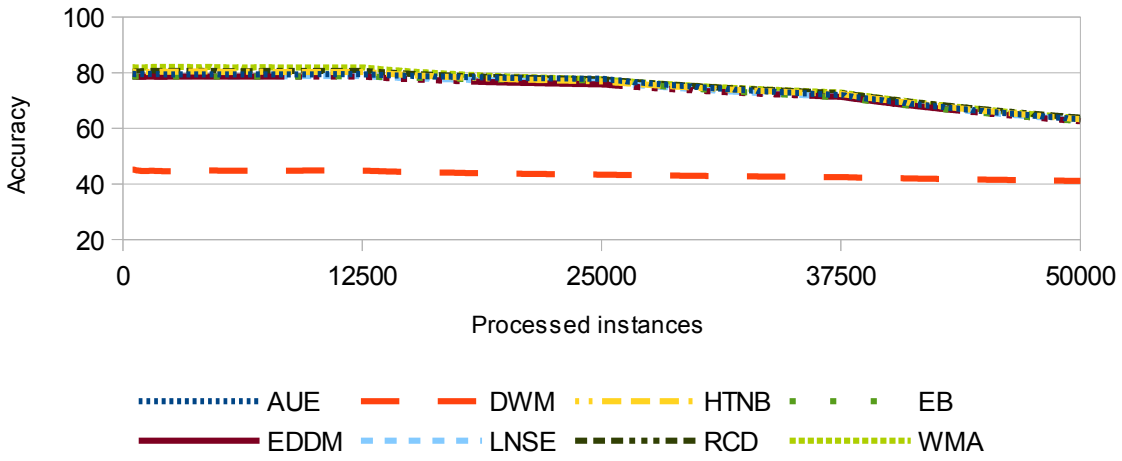
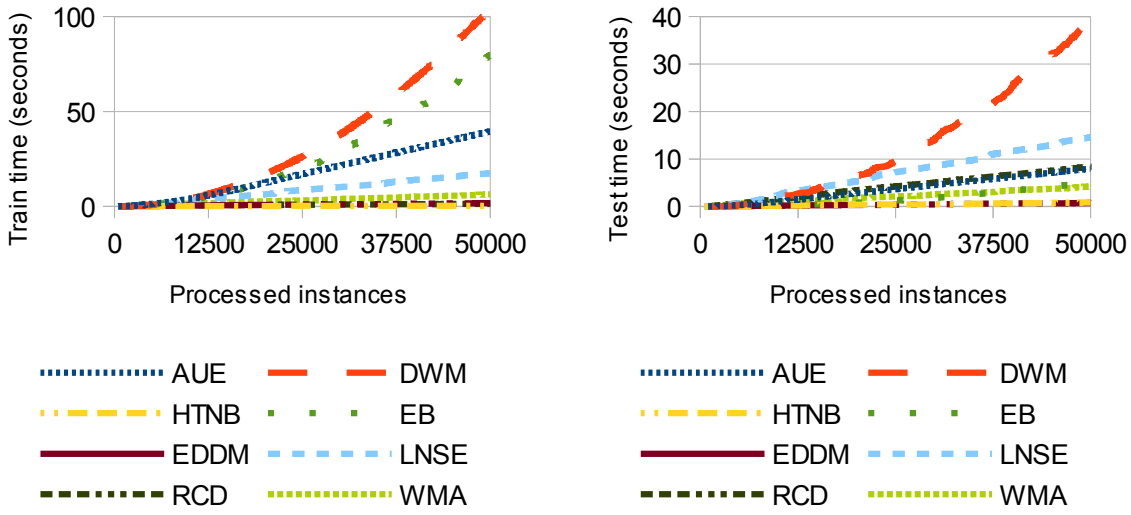



**Figure 4.11** Accuracy on the Waveform data set with gradual concept drifts.

Figures 4.12(a) and 4.12(b) show the time taken by the algorithms in the training and testing phases in the Waveform data set with gradual concept drifts. DWM was again the slowest algorithm in both phases followed by EB and AUE in the training phase, and by LNSE and RCD in the testing phase.




(a) Train time.

(b) Test time.

**Figure 4.12** Train and test times on the Waveform data set with gradual concept drifts.

### 4.4.4 SEA

Table 4.10 presents the average accuracies, ranks, and standard deviations for the SEA data set with abrupt concept drifts. HTNB best represented concepts 1 and 2, and had the best average accuracy and rank. Computed $F_F$ returned 3.07, rejecting the null-hypothesis. The Nemenyi test informs two groups of performances: HTNB is significantly better than DWM. The Bonferroni-Dunn confirms the results.

**Table 4.10** Statistics in SEA with abrupt concept drifts.

| Classifier | 1 | 2 | 3 | 4 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|
| AUE | 79.87 | 86.48 | 73.83 | **88.36** | 82.13 | 5.80 | 3.50 |
| DWM | 56.87 | 61.48 | 51.91 | 63.44 | 58.43 | 14.28 | 8.00 |
| HTNB | **84.69** | **87.30** | 78.65 | 84.52 | **83.79** | 3.38 | **2.25** |
| EB | 81.03 | 85.71 | 75.33 | 85.98 | 82.01 | 4.73 | 3.75 |
| EDDM | 79.20 | 85.34 | 73.24 | 87.08 | 81.21 | 5.71 | 5.25 |
| LNSE | 79.77 | 84.29 | 74.08 | 84.67 | 80.70 | 5.05 | 5.50 |
| WMA | 83.98 | 85.93 | 78.54 | 84.20 | 83.16 | 2.98 | 3.75 |
| RCD | 84.47 | 85.03 | **79.96** | 83.05 | 83.13 | **2.28** | 4.00 |

Most of the algorithms improve their accuracies after the first concept drift (RCD keeps it practically constant). This is an expected behavior because there is not much difference between the first two concepts: instead of the sum of the first two attributes

being below eight to belong to class 1, the sum becomes below nine. In the second concept drift, after 25,000 instances, the algorithms decrease their accuracies around 10%, while RCD decreases just 5%. Again, the behavior is expected. The decrease is due to the difference between the second and third concept: from below 9 to below 7. Entering the last context, the algorithms again increase their accuracies as this was the last concept the algorithms were trained on.

Another form to analyze the results is the following: as the algorithms tend to better represent the last concept, and in this case it was the sum of the first two attributes below 9.5, concepts closer to 9.5 might have better results. After the first concept drift, the algorithms tend to improve their accuracies because the threshold of being class 1 was increased from eight to nine, approaching 9.5. In the second concept drift, when the threshold is decreased to 7, the accuracy of the algorithms decrease as well. Finally, when the algorithms reach the last concept, they start increasing again their accuracies as this was the last concept dealt in the training phase. This can also be analyzed at Table 4.10. All the algorithms improved their performance from the first and second concepts, decreased in the third concept, and improved again in the fourth concept.

In this data set, RCD was the algorithm with the lowest standard deviation, followed by WMA, HTNB, EB, LNSE, EDDM, AUE, and finally, DWM. This can be noticed observing Figure 4.13.

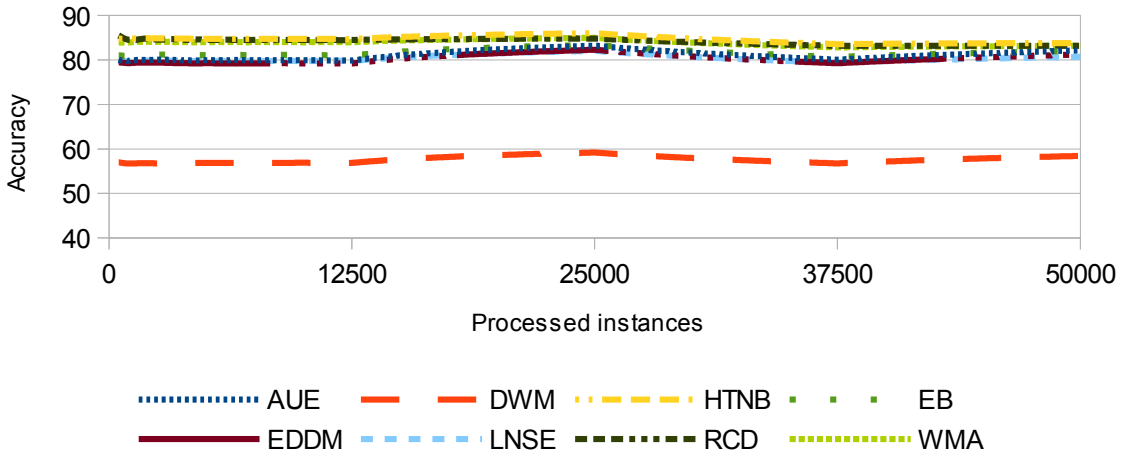
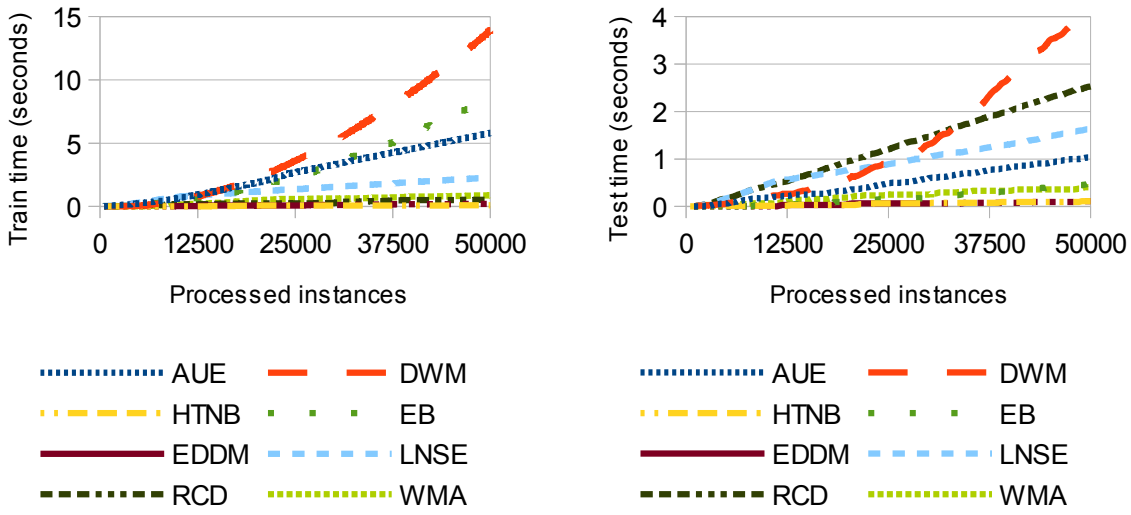



**Figure 4.13** Accuracy on the SEA data set with abrupt concept drifts.

Figures 4.14(a) and 4.14(b), concerning the time spent training in this data set, DWM was again the slowest one, followed by EB and AUE. In the testing phase, DWM was followed by RCD, LNSE, and AUE as the slowest ones.




(a) Train time.

(b) Test time.

**Figure 4.14** Train and test times on the SEA data set with abrupt concept drifts.

Concerning the gradual concept drifts, HTNB best represented concepts 1 and 2, had the highest average accuracy and lowest average rank. The average accuracy of the majority of the algorithms ranged between 82.14% and 83.92%, showing that they had very similar performances, as presented at Table 4.11. $F_F$ was equal to 8.20, indicating classifiers with different performances. The Nemenyi and Bonferroni-Dunn test informed that HTNB was significantly better than DWM. Again, the performance of the algorithms decreased when encountering concept drifts, like results have already shown in the previously tested data sets.

**Table 4.11** Statistics in SEA with gradual concept drifts.

| Classifier | 1 | 2 | 3 | 4 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|---|
| AUE | 88.20 | 83.26 | **83.71** | 79.76 | 83.73 | 3.62 | 3.00 |
| DWM | 53.16 | 56.89 | 49.02 | 58.59 | 54.42 | 14.96 | 8.00 |
| HTNB | **88.58** | **83.61** | 83.52 | 79.97 | **83.92** | 3.35 | **2.25** |
| EB | 85.29 | 81.42 | 83.55 | 78.32 | 82.14 | 3.69 | 6.25 |
| EDDM | 87.77 | 83.48 | 83.20 | **80.15** | 83.65 | 3.41 | 3.50 |
| LNSE | 85.69 | 82.64 | 83.16 | 79.38 | 82.72 | 3.38 | 6.25 |
| WMA | 88.57 | 83.57 | 83.57 | 79.93 | 83.91 | 3.39 | 2.50 |
| RCD | 87.39 | 82.92 | 83.67 | 79.49 | 83.37 | **3.13** | 4.25 |

Similar to the version with abrupt concept drifts, RCD was again the stablest algorithm

with the lowest standard deviation as can be seen in the $\sigma$ column at Table 4.11. In Figure 4.15 it is possible to see that the algorithms had similar behavior dealing with concept drifts.

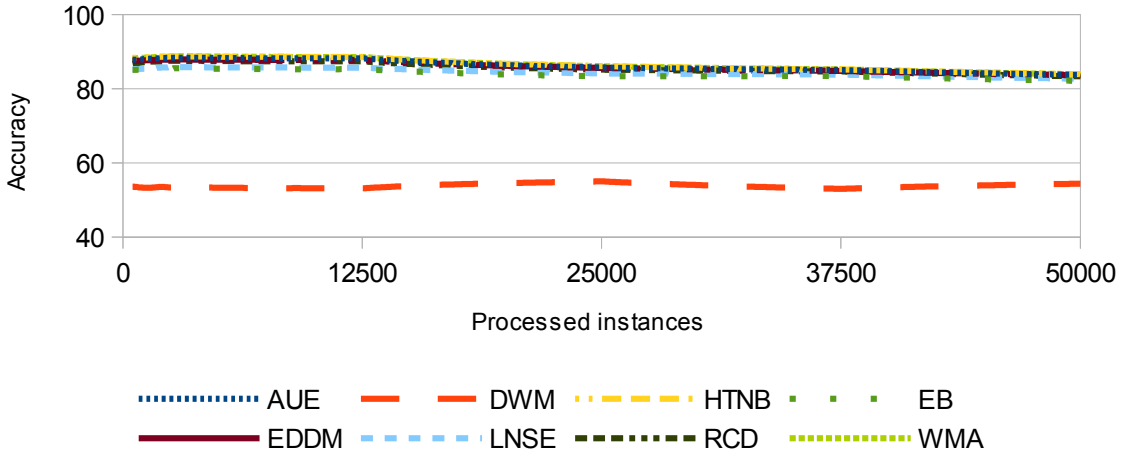


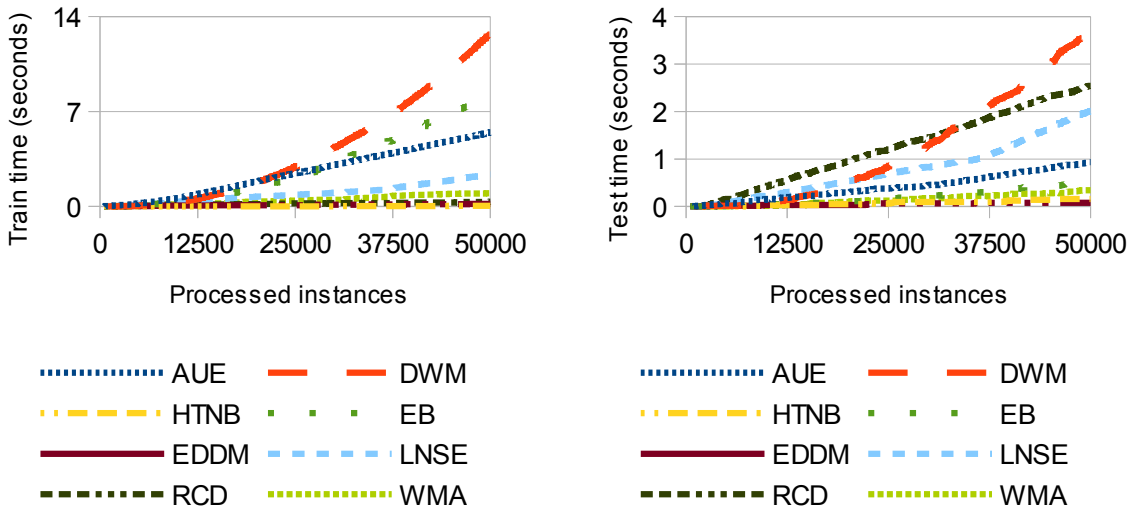

**Figure 4.15** Accuracy on the SEA data set with gradual concept drifts.

The training time, as presented at Figure 4.16(a), shows that HTNB was the fastest algorithm, followed by EDDM, RCD, WMA, LNSE, AUE, EB, and DWM. Concerning the testing time, as presented at Figure 4.16(b), HTNB is still the fastest, followed by EDDM, RCD, WMA, EB, AUE, LNSE, and DWM. These results are similar to the ones obtained in the abrupt concept drift version of this data set.

### 4.4.5 STAGGER

In this data set, concept A (represented by set 1) is almost a reversal of concept B (set 2). Concepts B and C (set 3), on the other hand, share a great number of positive examples. Table 4.12 presents the accuracy averages obtained in this data set with abrupt concept drifts. After being trained in a data set with three different concepts, the algorithms did not have good accuracy in the first concept, for two different reasons. First, the models do not represent the concept well because it was the first concept seen and they had to adapt to two more concepts. The second reason is that the first concept is very different from the other two.

In this data set configuration, RCD was again the stablest algorithm. It also had the lowest average rank, besides better representing the first two concepts. Nevertheless, statistically, all the algorithms can be considered having the same performance. It




(a) Train time.

(b) Test time.

**Figure 4.16** Train and test times on the SEA data set with gradual concept drifts.

is interesting to notice that AUE, DWM, EB, and EDDM obtained 100% in the last context. This is usually an indication that the algorithm is suffering from over-fitting. In this situation, the algorithm describes random error or noise instead of the underlying distribution. This is probably the case because in the first and second contexts these algorithms have the lowest accuracies (except from LNSE) and, besides that, they have exactly the same performance in all contexts. Even making 100% in the last context, the average accuracy of these algorithms was lower compared to the other ones that did not over-fit.

**Table 4.12** Statistics in STAGGER with abrupt concept drifts.

| Classifier | 1 | 2 | 3 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|
| AUE | 22.20 | 51.72 | **100.00** | 57.97 | 32.25 | 4.83 |
| DWM | 22.20 | 51.72 | **100.00** | 57.97 | 32.25 | 4.83 |
| HTNB | 32.86 | 62.37 | 89.35 | 61.53 | 24.35 | 3.67 |
| EB | 22.20 | 51.72 | **100.00** | 57.97 | 32.25 | 4.83 |
| EDDM | 22.20 | 51.72 | **100.00** | 57.97 | 32.25 | 4.83 |
| LNSE | 11.14 | 55.51 | 66.65 | 44.43 | 24.11 | 6.67 |
| WMA | 33.83 | 63.61 | 88.11 | **61.85** | 23.63 | 3.33 |
| RCD | **39.82** | **67.03** | 72.27 | 59.71 | **18.22** | **3.00** |

Figure 4.17 presents how the algorithms deal with concept drifts on the STAGGER

data set with abrupt concept drifts. During the first 12,500 instances, related to the first concept, the algorithms keep their accuracies almost constant, with RCD being the best one. In the first concept drift, they all start to increase their performance, with RCD ending the second context with the higher accuracy. In the third and last concept drift, the algorithms keep increasing their accuracies.

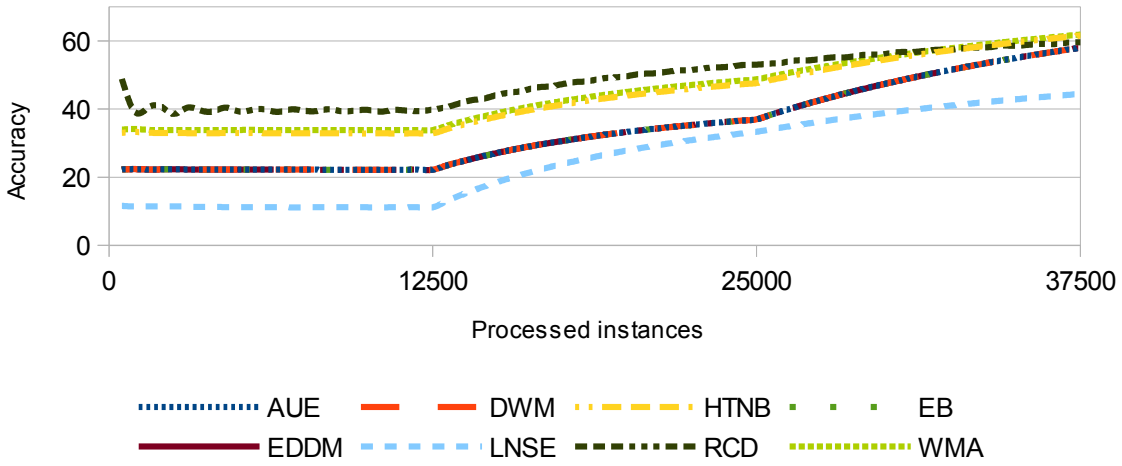



**Figure 4.17** Accuracy on the STAGGER data set with abrupt concept drifts.

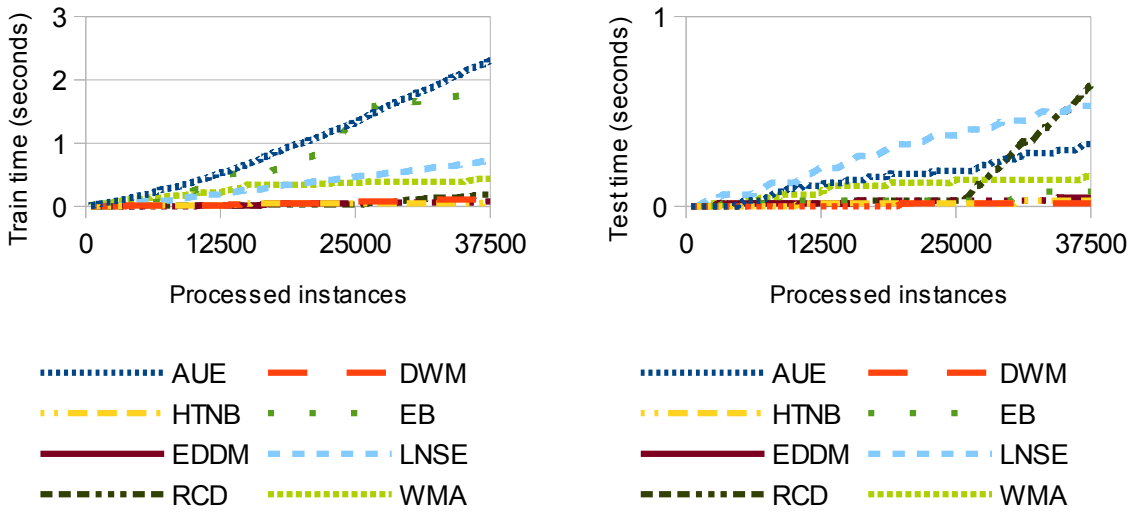
Figures 4.18(a) and 4.18(b) present the time spent by the algorithms in the training and testing phases. The majority of the algorithms needed less than 1 second to train and test its internal models. The fastest algorithms in both phases are similar to the ones presented in the SEA data set. RCD was the slowest algorithm in the testing phase, but the difference to the other algorithms is low, less than one second.

Concerning the gradual concept drifts, all the algorithms reduce their accuracies when encountering new contexts, except for LNSE which increase its performance. It better represented the last two concepts, presented lowest standard deviation and average rank. Statistically, like in the data set with abrupt concept drifts, all the algorithms can be considered to have similar accuracies.

Figure 4.19 shows that the accuracies of all algorithms are practically the same. Only LNSE starts lower and increases its accuracy as it reaches the end of the data set.

Figures 4.20(a) and 4.20(b) present similar results as the ones obtained in the version with abrupt concept drifts.

RCD was the method that best represented the first concept in four out of five configurations of the abrupt data sets. As the last concepts arise, other methods present better results, but differences to RCD are small. In other words, RCD stored classifiers specific to




(a) Train time.

(b) Test time.

**Figure 4.18** Train and test times on the STAGGER data set with abrupt concept drifts.

**Table 4.13** Statistics in STAGGER with gradual concept drifts.

| Classifier | 1 | 2 | 3 | $\mu$ | $\sigma$ | $\overline{R}$ |
|---|---|---|---|---|---|---|
| AUE | 95.33 | 43.25 | 25.84 | 54.81 | 33.06 | 5.00 |
| DWM | 95.45 | 43.13 | 25.71 | 54.76 | 33.17 | 5.33 |
| HTNB | **97.19** | 43.61 | 24.98 | **55.26** | 32.56 | 3.67 |
| EB | 95.08 | 43.50 | 26.08 | 54.89 | 32.84 | 4.67 |
| EDDM | 94.46 | 44.12 | 26.19 | 54.92 | 32.74 | 3.67 |
| LNSE | 22.31 | **54.24** | **61.16** | 45.90 | **25.09** | **3.33** |
| WMA | 96.95 | 43.60 | 24.99 | 55.18 | 32.57 | 4.00 |
| RCD | 96.78 | 42.94 | 24.40 | 54.71 | 32.64 | 6.33 |

each concept, yielding higher average accuracies in the initial concepts and close to best in the last concepts, resulting in stabler results across several concept drifts.

In the abrupt data sets, WMA presented the lowest average rank (3.21), closely followed by RCD (3.25). Performing the comparison of the classifiers on all abrupt data sets, the $F_F$ statistic returned 5.64, indicating classifiers with different levels of accuracy. The Nemenyi test returned CD equal to 2.14, indicating that DWM and LNSE are significantly inferior to WMA and RCD. The Bonferroni-Dunn test returned 1.90, confirming the results of the previous test.

In the gradual data sets, AUE presented the lowest average rank (3.38), followed by WMA (3.83) and RCD (4.21). The $F_F$ statistic computed for all gradual data sets was

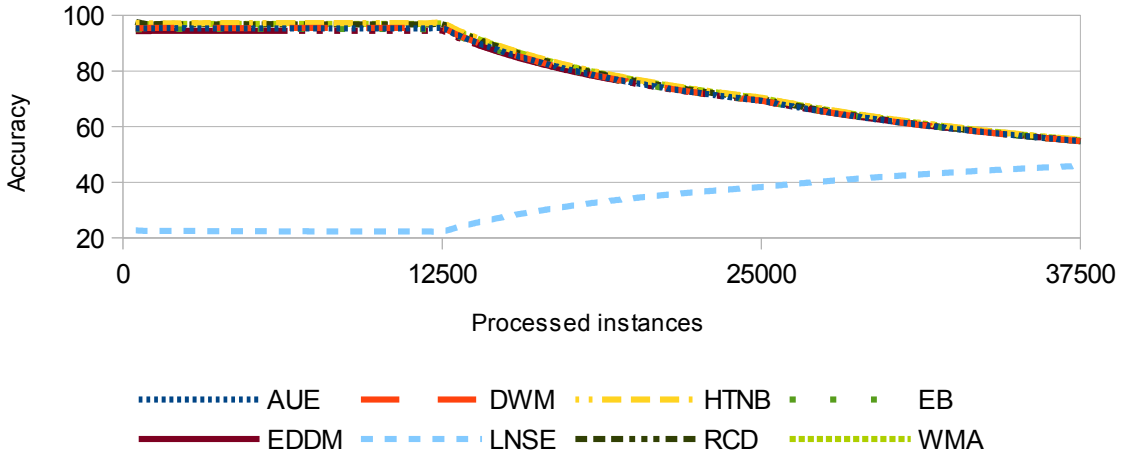
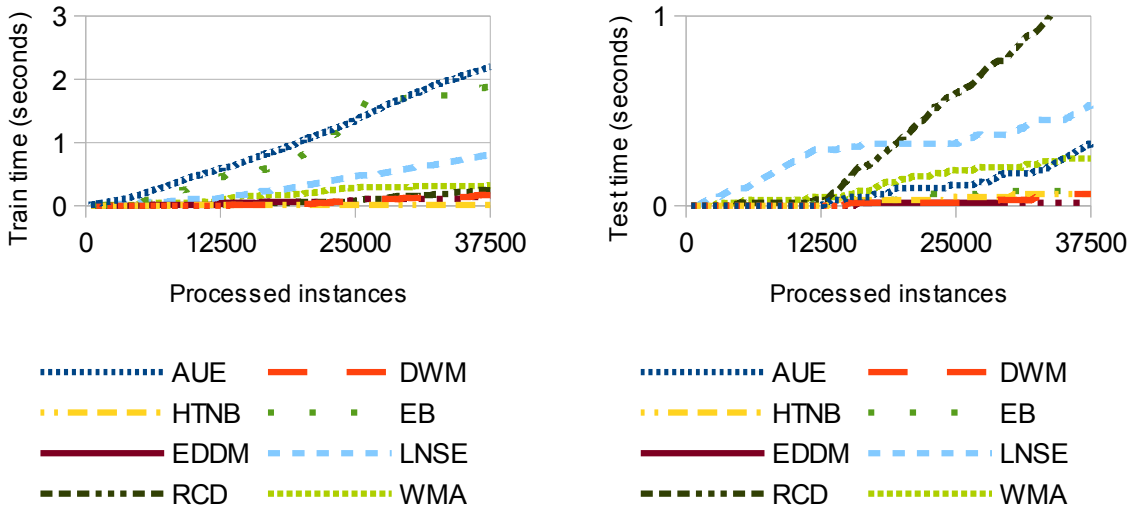



**Figure 4.19** Accuracy on the STAGGER data set with gradual concept drifts.




(a) Train time.

(b) Test time.

**Figure 4.20** Train and test times on the STAGGER data set with gradual concept drifts.

equal to 4.10, higher than the critical value (2.07), also indicating classifiers with different performances. The Nemenyi test with 95% confidence indicated that AUE, WMA, RCD, HTNB, and EB are significantly better than DWM. Reducing the confidence to 90%, LNSE and EDDM are also better than DWM. The Bonferroni-Dunn yields similar results.

Comparing all data sets, $F_F$ returned 9.54, higher than the critical value (2.04), indicating classifiers with different levels of performance. WMA presented the lowest average

rank (3.41), closely followed by RCD (3.57). The Nemenyi test considering all data sets has a CD equal to 1.52. Thus, it informs that WMA and RCD are significantly better than LNSE and DWM. Reducing the confidence to 90%, WMA and RCD are also better than EDDM. Again, the Bonferroni-Dunn confirms the results of the Nemenyi test.

### 4.4.6 Real data sets

In our experiments, we also considered four real-world data sets: Forest Covertype, Poker Hand, Electricity, and Weather. The results are presented at Table 4.14.

**Table 4.14** Accuracy in real-world data sets.

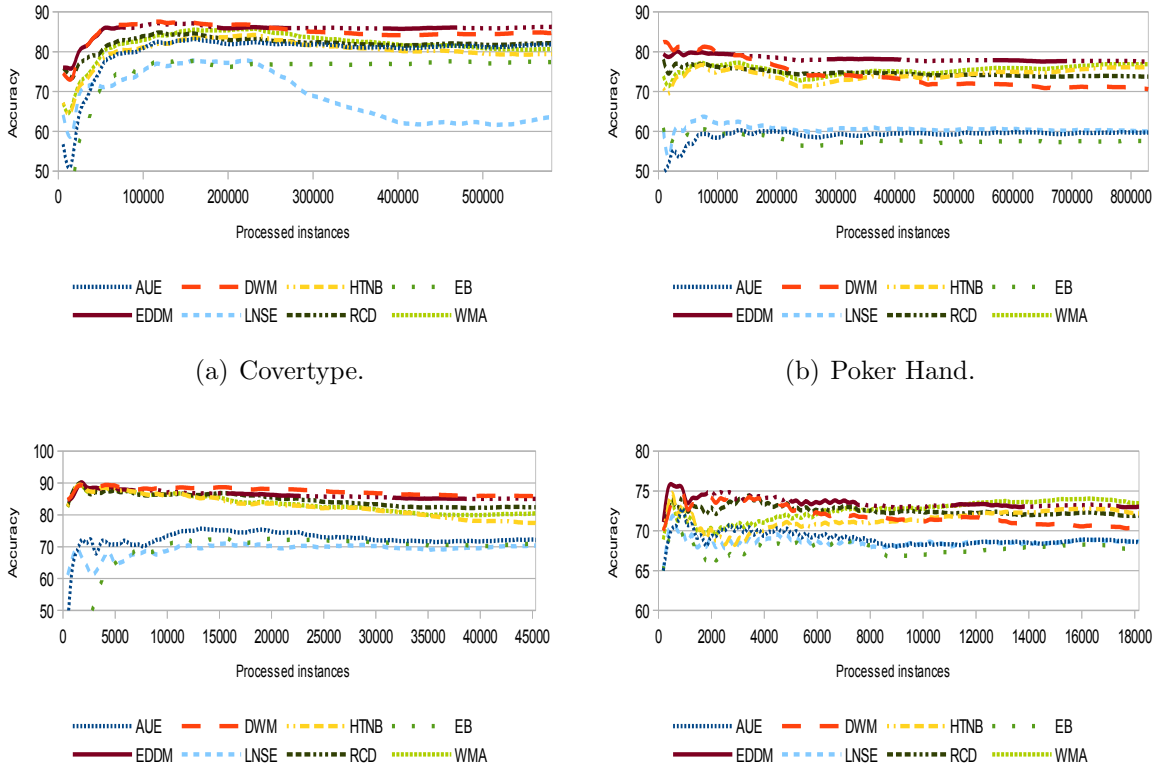
| Classifier | Covertype | Electricity | Poker Hand | Weather | $\overline{R}$ |
|---|---|---|---|---|---|
| AUE | 81.85 | 72.13 | 59.70 | 68.52 | 6.00 |
| DWM | 84.71 | **85.84** | 70.57 | 70.36 | 3.25 |
| HTNB | 79.41 | 77.43 | 76.07 | 72.18 | 4.25 |
| EB | 77.39 | 70.99 | 57.52 | 67.54 | 7.50 |
| EDDM | **86.24** | 85.03 | **77.50** | 73.03 | **1.50** |
| LNSE | 63.57 | 70.31 | 59.95 | 68.64 | 7.00 |
| WMA | 80.64 | 80.46 | 76.82 | **73.45** | 3.00 |
| RCD | 82.18 | 82.39 | 73.73 | 71.89 | 3.50 |

DWM, which did not present good accuracy results in the artificial data sets, had the best performance in the *Electricity* data set and the second best performance in the *Covertype* data set. The algorithm with lowest average rank in these data sets was EDDM. Computing the $F_F$ statistic for the real-world data sets, it was obtained the value 8.86, higher than the critical value (2.49), rejecting the null-hypothesis. The Nemenyi test informs that EDDM is significantly better than LNSE and EB. The Bonferroni-Dunn test informs that RCD has similar performance compared to the other classifiers. Figure 4.21 presents the predictive accuracies obtained by the classifiers in the real-world data sets.

The first three data sets were previously tested by Bifet et al. (2010b) with nine different algorithms: perceptron, naive Bayes, Hoeffding tree, Hoeffding Naive Bayes Tree (HNBT), Hoeffding Perceptron Tree (HPT), Hoeffding Naive Bayes Perceptron Tree (HNBPT), and ADWIN bagging using HNBT, HPT, and HNBPT.

Concerning the perceptron classifier, it is used an on-line version of the perceptron that employs the sigmoid activation function, instead of the threshold activation function, and optimizes the squared error, with one perceptron per class value.

Traditional Hoeffding trees perform prediction by choosing the majority class at each leaf. Their predictive accuracy can be increased by adding naive Bayes models at the




(a) Covertype.

(b) Poker Hand.

(c) Electricity.

(d) Weather.

**Figure 4.21** Accuracies in the real-world data sets.

leaves of the trees. The HNBT classifier proposes a hybrid adaptive method that generally outperforms the two original prediction methods for both simple and complex concepts. Bifet et al. (2010b) describe that "this method works by performing a naive Bayes prediction per training instance, and comparing its prediction with the majority class. Counts are stored to measure how many times the naive Bayes prediction gets the true class correct as compared to the majority class. When performing a prediction on a test instance, the leaf will only return a naive Bayes prediction if it has been more accurate overall than the majority class, otherwise it resorts to a majority class prediction." HPT works the same, only substituting naive Bayes by a perceptron. HNBPT is also similar, using three classifiers at each leaf: a majority class, naive Bayes, and a perceptron.

ADWIN (Bifet and Gavaldà, 2007) is a change detector and estimator that offers a solution to the problem of tracking the average of a stream of bits or real-valued numbers. It keeps a window of recently seen items, changing its size to avoid changes in the average value inside the window. ADWIN Bagging is the on-line method of Oza and Russell (2001b) using ADWIN as its change detector. When a change is detected, the ensemble's worst

classifier is removed and a new one is added. ADWIN Bagging was tested using HNBT, HPT, and HNBPT as base classifiers.

Table 4.15 presents the results according to Bifet et al. (2010b). Comparing to RCD, it is possible to notice that it had comparable results to the algorithms presented. In the Covertype, Electricity and Poker Hand data sets, RCD had better accuracy than four algorithms in each set.

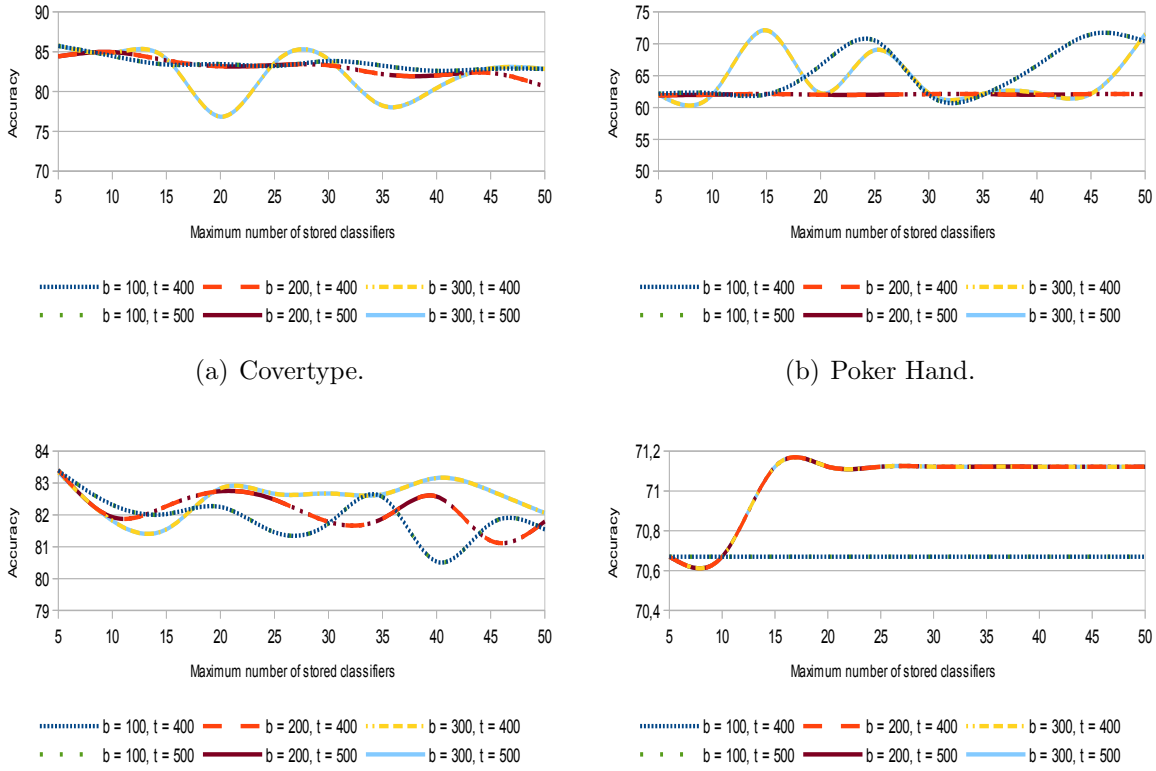
**Table 4.15** Accuracy in real-world data sets according to Bifet et al. (2010b).

| Classifier | Covertype | Electricity | Poker Hand |
|---|---|---|---|
| Perceptron | 81.68 | 79.07 | 3.34 |
| Naive Bayes | 60.52 | 73.36 | 59.55 |
| Hoeffding Tree | 68.30 | 75.35 | 73.62 |
| HNBT | 81.06 | 80.69 | 83.05 |
| HPT | 83.59 | 84.24 | 74.02 |
| HNBPT | 85.77 | 84.34 | 82.93 |
| ADWIN Bagging HNBT | 85.73 | 84.36 | 74.56 |
| ADWIN Bagging HPT | 86.33 | 85.22 | 65.76 |
| ADWIN Bagging HNBPT | 87.88 | 86.44 | 74.36 |

### 4.4.7 Classifiers set size comparison

It was also analyzed the impact of the classifiers set size in the accuracy of RCD. Six different parametrization for the buffer size and test frequency were tested varying the classifiers set size from 5 to 50 with step 5. Results are presented at Figure 4.22. To perform the experiments, it is needed data sets that suffer from several concept drifts. The artificial data sets used in this chapter do not meet this characteristic. The data set with most concept drifts is Hyperplane with nine changes. Thus, only the real-world data sets were used.

It is possible to observe that results are the same in buffers of equal size in all four real-world data sets. Also, the average accuracies in all configurations are very similar, with low standard deviation. The highest difference was less than 6%, in the Poker Hand data set, comparing the buffers with 100 and 200 instances while in the Electricity and Weather data sets, the differences were less than 1%. Thus, RCD is stable with different values for the buffer size, test frequency, and classifiers set size.




(a) Covertype.

(b) Poker Hand.

(c) Electricity.

(d) Weather.

**Figure 4.22** Accuracies in the real-world data sets in several classifiers set sizes.

## 4.5 CONCLUSION

In this chapter RCD was compared with seven different algorithms including single approaches, like Hoeffding trees with naive Bayes and EDDM, and also with ensemble classifiers, like AUE, WMA, EB, LNSE, and DWM. To perform the comparison, we selected five artificial and four real-world data sets among the most used in the concept drift research area.

Concerning abrupt concept drifts, tests indicated that RCD had better performance when the buffer size and test frequency had the same values. In the data sets with gradual concept drifts, the opposite applies.

Considering the data sets suffering from abrupt concept drifts, RCD was significantly better than LNSE and DWM. In the gradual data sets, RCD was significantly better than DWM. Comparing the classifiers in all artificial data sets, RCD was significantly better than LNSE and DWM (with 95% confidence) and also better than EDDM (with 90% confidence).

In the real-world data sets, EDDM was the best algorithm at Covertype and Poker Hand, DWM in Electricity, and WMA was the best in Weather. RCD had results very close to the ones presented by the best algorithms in each data set. The biggest difference between RCD and these algorithms was 4.06% in the Covertype data set.

Analyzing the train and test times, DWM was the slowest tested algorithm in all data sets, independently of the type of concept drift. Considering the train times, DWM was followed by EB and AUE as the slowest ones. In the testing phase, DWM was followed by LNSE and AUE. RCD was the only classifier that took more time in the testing phase than the training phase, due to the statistical tests that are periodically performed to identify the best classifier to the current data. In both training and testing phases, single learners were faster than ensemble classifiers, as expected.

It is interesting to notice that, in data sets with abrupt concept drifts, the best accuracy of every algorithm tends to reside in the most recent concept. As most of the algorithms tend to adapt to the last instances presented to them in the training phase, or to give higher weights to classifiers that correctly represent them, the occurrence of this behavior is no surprise. On the other hand, in data sets with gradual concept drifts, the performance decreases with time; the best represented concept was almost always the first one observed.

CHAPTER 5

# STATISTICAL TESTS COMPARISON

In this chapter, a version of RCD using KNN and a version using Cramer are compared using five different base learners in terms of accuracy and evaluation time. The evaluated base learners were naive Bayes, Multilayer Perceptron (MLP), J48, and two decision trees built to handle streaming data: a pure Hoeffding tree and a Hoeffding tree with naive Bayes at the leaves. The tests were performed in four artificial data sets: Random RBF, Hyperplane (presenting gradual concept drifts), SEA, and LED (presenting abrupt concept drifts); and in three real-world data sets: Forest Covertype, Poker Hand, and Electricity.

The evaluation methodology used in the tests was the Interleaved Test-Then-Train approach. Every example is used for testing the model, then it is used to train it. The accuracy was measured as the final percentage of instances correctly classified over the interleaved evaluation. For the Hyperplane and Random RBF data sets, ten million instances were generated; for LED and SEA, one million instances. The experiments were repeated ten times. The parameters of these streams are the following:

- RBF$(x, v)$: Random RBF data stream with $x$ centroids moving at speed $v$.
- HYP$(x, v)$: Hyperplane data stream with $x$ attributes changing at speed $v$.
- SEA$(v)$: SEA data set, with length of change $v$.
- LED$(v)$: LED data set, with length of change $v$.

The chosen evaluation methodology, data sets, and configurations are exactly the same as used by Bifet et al. (2009b, 2010b). For the tests, it was used an Intel Core i3 330M processor (with two cores and emulating two more cores), with 4GB of main memory.

Initially, tests were performed to identify how much time both statistical tests need to compare samples of varying sizes. Table 5.1 presents this information for samples sizes varying from 100 to 300 instances. The samples were obtained from the Electricity data set. We can see that the time spent by the Cramer test almost doubles each time the

sample size is increased by 50 instances while KNN is less impacted. Comparing both tests, it is clear that KNN is considerably faster than Cramer performing the statistical tests. Thus, considering the data set sizes used in the experiments (1 to 10 million, with 10 repetitions), the buffer size used in RCD was set to 100 to increase its speed. The other parameters used were: testing frequency each 500 instances, up to 15 stored classifiers, significance value set to 0.05, and DDM as drift detection method.

**Table 5.1** Evaluation time using statistical tests (in seconds).

| | **100** | **150** | **200** | **250** | **300** |
|---|---|---|---|---|---|
| **Cramer** | 0.413 | 0.947 | 1.835 | 3.263 | 6.375 |
| **KNN** | 0.003 | 0.007 | 0.012 | 0.017 | 0.024 |

Tables 5.2 to 5.6 present the average evaluation times in seconds and the accuracy of the HTNB, naive Bayes, MLP, Hoeffding tree (Hoeffding Tree (HT)), and J48 (Quinlan, 1993) base classifiers, respectively. For the artificial data sets, the accuracy also contains the 95% confidence interval. The second and third columns present the results for the base classifier; columns 4 to 6, RCD using the Cramer test; and columns 7 to 9, RCD using the KNN test. The last column of Cramer and KNN present the average number of drifts detected in all repetitions in the artificial data sets, and the total number of drifts in the real-world data sets, respectively. The best results in performance are highlighted in boldface and the values marked by an asterisk (*) represent statistically significant differences between RCD and the base learner.

## 5.1 HOEFFDING TREE WITH NAIVE BAYES AT THE LEAVES

Analyzing the results presented at Table 5.2, it is possible to observe that RCD outperformed the HTNB base classifier in the majority of the artificial data sets, independently of the statistical test used. Statistically, RCD had better performance in HYP(10,0.0001), HYP(10,0.001) (when using the KNN statistical test), and in the Random RBF configurations with 50 centroids. In all the other configurations, both algorithms had statistically similar performances.

In the Random RBF configuration without concept drifts, RBF(0,0), all algorithms had exactly the same results. This is an expected result of RCD because, when no concept drift is detected, no statistical test is performed and RCD performs the same as the base learner. In the other RBF data sets with concept drift, the configurations that RCD performed better than the base learner were the ones with 50 centroids. For this data

**Table 5.2** Results using the HTNB base classifier.

| | HTNB | | RCD HTNB CVM | | | RCD HTNB KNN | | |
|---|---|---|---|---|---|---|---|---|
| | **Time** | **Accuracy** | **Time** | **Accuracy** | **# Drifts** | **Time** | **Accuracy** | **# Drifts** |
| **RBF(0,0)** | 162.45 | 93.04±0.07 | 224.93 | 93.04±0.07 | 0.0 | 221.38 | 93.04±0.07 | 0.0 |
| **RBF(50,0.001)** | 247.17 | 55.27±0.04 | 392.35 | 55.37±0.11* | 17.2 | 451.63 | **55.47±0.15*** | 15.4 |
| **RBF(10,0.001)** | 178.06 | 88.29±0.08 | 242.27 | **88.30±0.09** | 0.4 | 238.70 | 88.29±0.09 | 0.3 |
| **RBF(50,0.0001)** | 258.82 | 63.40±0.08 | 380.44 | 64.39±0.27* | 87.9 | 292.93 | **69.90±2.69*** | 125.0 |
| **RBF(10,0.0001)** | 180.57 | **89.37±0.07** | 247.55 | 89.35±0.09 | 0.8 | 241.56 | 89.36±0.08 | 0.9 |
| **HYP(10,0.001)** | 182.55 | 88.71±1.83 | 261.13 | 89.06±1.47 | 8.0 | 246.00 | **89.01±1.56*** | 8.9 |
| **HYP(10,0.0001)** | 181.44 | 89.01±0.61 | 251.45 | 89.24±0.50* | 10.4 | 243.27 | **89.18±0.58*** | 10.6 |
| **SEA(50)** | 6.87 | 88.67±0.85 | 9.93 | **89.20±0.19** | 0.7 | 9.78 | **89.20±0.19** | 0.7 |
| **SEA(50000)** | 6.78 | 88.68±0.85 | 9.84 | **89.19±0.22** | 0.9 | 9.66 | **89.19±0.22** | 0.9 |
| **LED(50000)** | 26.14 | 73.24±0.93 | 43.37 | 73.50±0.37 | 3.2 | 41.47 | **73.51±0.37** | 3.3 |
| **Covertype** | 23.60 | 79.57 | 326.84 | 75.95 | 2329 | 97.83 | **84.50** | 2779 |
| **Poker Hand** | 11.48 | **77.11** | 164.94 | 76.59 | 798 | 32.26 | 74.46 | 892 |
| **Electricity** | 0.51 | 77.43 | 17.30 | 78.68 | 119 | 2.73 | **84.90** | 211 |

set, the higher the number of moving centroids, the bigger the number of concept drifts. Thus, RCD had better performance exactly in the configurations with higher number of concept drifts while in the configurations with 10 centroids, the ones with less concept drifts, the performances were similar.

It is possible to check the influence of the number of moving centroids comparing it to the average number of concept drifts that occurred in each configuration. In the RBF(50,0.001) configuration, using the Cramer test, 17.2 concept drifts were detected, while using KNN there were 15.4 in the ten repetitions. In the RBF(10,0.001) configuration, only 0.4 and 0.3 concept drifts were detected using the Cramer and KNN statistical tests, respectively, in the ten repetitions.

Comparing the usage of the two statistical tests, KNN had a better performance in two data sets: in RBF(50,0.0001) and in the LED data set. In the rest, both statistical tests had comparable performances. Analyzing the number of detected concept drifts, KNN is more sensible than Cramer. Using KNN, 5,542 concept drifts were detected while Cramer detected 4,541 in all data sets, considering all repetitions.

Considering the time spent in the evaluation procedure, as expected, the base learner was faster than the RCD framework. In average, HTNB was approximately 75.41% faster than RCD using the Cramer test and 45.19% using KNN. In 12 out of 13 data sets, RCD was faster using KNN than with the Cramer test.

## 5.2 NAIVE BAYES

The results presented for the naive Bayes base learner in Table 5.3 show that, again, the RCD framework had better performance in the majority of the data sets and configurations. Statistically, the results were similar to the ones obtained using the HTNB base learner: RCD also had higher accuracy values in both configurations of Random RBF with 50 centroids and, in this case, in both configurations of the Hyperplane data set. In the other data set configurations, RCD and naive Bayes obtained statistically similar results.

**Table 5.3** Results using the naive Bayes base classifier.

| | NB | | RCD NB CVM | | | RCD NB KNN | | |
|---|---|---|---|---|---|---|---|---|
| | **Time** | **Accuracy** | **Time** | **Accuracy** | **# Drifts** | **Time** | **Accuracy** | **# Drifts** |
| **RBF(0,0)** | 53.90 | 72.02±0.02 | 93.94 | 72.02±0.02 | 0.2 | 92.68 | 72.02±0.02 | 0.2 |
| **RBF(50,0.001)** | 117.17 | 53.18±0.01 | 161.20 | 53.19±0.01* | 10.0 | 162.71 | **53.22±0.02*** | 16.7 |
| **RBF(10,0.001)** | 66.40 | 75.78±0.03 | 105.95 | **75.79±0.03** | 5.7 | 103.61 | 75.78±0.03 | 5.5 |
| **RBF(50,0.0001)** | 116.78 | 53.25±0.05 | 160.61 | 53.33±0.03* | 13.6 | 160.86 | **53.76±0.31*** | 26.2 |
| **RBF(10,0.0001)** | 66.43 | **75.77±0.04** | 106.29 | 75.76±0.04 | 6.8 | 105.09 | **75.77±0.04** | 8.1 |
| **HYP(10,0.001)** | 54.91 | 87.10±3.14 | 96.88 | 87.64±2.78* | 7.6 | 96.14 | **87.91±2.47*** | 8.6 |
| **HYP(10,0.0001)** | 54.90 | 88.34±1.23 | 97.08 | 88.71±1.24* | 9.3 | 95.03 | **88.84±1.25*** | 8.4 |
| **SEA(50)** | 2.60 | 87.71±1.33 | 4.35 | **87.86±1.22** | 0.3 | 4.29 | **87.86±1.22** | 0.3 |
| **SEA(50000)** | 2.51 | 87.71±1.32 | 4.36 | **88.42±0.35** | 0.8 | 4.20 | **88.42±0.35** | 0.8 |
| **LED(50000)** | 17.77 | 72.58±2.36 | 31.89 | **73.18±1.25** | 0.2 | 32.59 | **73.18±1.25** | 0.2 |
| **Covertype** | 29.13 | 60.52 | 391.97 | 61.68 | 2513 | 128.54 | **85.47** | 3961 |
| **Poker Hand** | 15.09 | 59.55 | 231.88 | 60.25 | 1151 | 45.97 | **70.88** | 1790 |
| **Electricity** | 0.44 | 73.36 | 26.01 | 73.58 | 180 | 2.56 | **82.21** | 198 |

Considering RBF(0,0), two concept drifts occurred in the ten repetitions. The results of the algorithms are practically the same, only differing in the fifth decimal. This is a similar result compared to the same configuration using HTNB: when no concept drift was found, the same results occurred; when only two concept drifts were detected, the accuracy difference was negligible. Again, it is possible to notice that a higher number of centroids leads to a higher number of concept drifts. Using the Cramer test, there were an average of 10.0 concept drifts in RBF(50,0.001) and 5.7 with 10 centroids. The same behavior occurred when using the KNN test: 16.7 and 5.5 concept drifts, respectively.

Comparing the statistical tests, KNN was statistically more accurate than Cramer in HYP(10,0.0001) and in both configurations of Random RBF with 50 centroids. In all other situations, both statistical tests had comparable performances. Comparing the

number of detected concept drifts, similar results to the ones using the HTNB base learner were obtained. Using KNN, 6,699 concept drifts were detected while Cramer detected 4,389 in all data sets.

Again, the base learner was faster in evaluating the data sets compared to RCD. The base learner was 2.5 times faster compared to RCD using the Cramer test, and 72% faster than RCD using KNN. The KNN statistical test was faster than the Cramer test in 10 out of 13 data set configurations. The differences between the statistical tests is low in the artificial data sets, due to its reduced number of concept drifts. In the real-world data sets, where more concept drifts are identified, the differences are higher, with KNN being much faster than Cramer.

## 5.3 MULTILAYER PERCEPTRON

Table 5.4 presents the results for the MLP base learner. The evaluation results show that, statistically, RCD had higher accuracy values compared to MLP only in RBF(50,0.0001). In all other situations, RCD and MLP had comparable performances, independently of the statistical test used.

**Table 5.4** Results using the MLP base classifier.

| | MLP | | RCD MLP CVM | | | RCD MLP KNN | | |
|---|---|---|---|---|---|---|---|---|
| | **Time** | **Accuracy** | **Time** | **Accuracy** | **# Drifts** | **Time** | **Accuracy** | **# Drifts** |
| **RBF(0,0)** | 56.46 | **87.70±0.57** | 101.86 | 87.44±0.53 | 0.2 | 101.90 | 87.44±0.53 | 0.2 |
| **RBF(50,0.001)** | 120.26 | 50.28±0.24 | 174.82 | 50.27±0.19 | 14.0 | 188.73 | **50.30±0.33** | 17.5 |
| **RBF(10,0.001)** | 69.09 | 83.95±0.47 | 128.76 | **84.20±0.43** | 6.8 | 133.79 | 83.93±0.45 | 10.5 |
| **RBF(50,0.0001)** | 120.23 | 49.92±0.26 | 176.99 | 50.10±0.26* | 10.9 | 321.69 | **53.23±1.94*** | 248.5 |
| **RBF(10,0.0001)** | 69.12 | 84.41±0.67 | 122.14 | 84.40±0.48 | 8.0 | 121.69 | **84.78±0.40** | 7.6 |
| **HYP(10,0.001)** | 56.45 | **82.55±6.70** | 111.98 | 82.24±7.33 | 8.6 | 110.47 | 82.27±7.36 | 9.2 |
| **HYP(10,0.0001)** | 56.44 | 84.31±2.14 | 113.46 | 84.34±2.22 | 8.9 | 112.71 | **84.43±2.22** | 9.0 |
| **SEA(50)** | 4.13 | 87.99±1.68 | 6.97 | **88.75±0.98** | 1.0 | 7.03 | 88.74±0.98 | 0.9 |
| **SEA(50000)** | 4.08 | 87.99±1.78 | 7.37 | **88.04±1.70** | 0.9 | 7.14 | **88.04±1.70** | 0.9 |
| **LED(50000)** | 27.48 | 63.67±4.73 | 50.84 | 63.68±4.72 | 0.5 | 51.05 | **65.17±2.98** | 0.5 |
| **Covertype** | 49.00 | **60.69** | 410.19 | 49.02 | 1661 | 388.15 | 49.05 | 1779 |
| **Poker Hand** | 23.35 | 43.03 | 265.26 | 59.80 | 1421 | 918.81 | **63.47** | 1284 |
| **Electricity** | 1.53 | **57.51** | 20.76 | 49.30 | 98 | 7.68 | 55.49 | 67 |

In the RBF data set without concept drifts, both versions of RCD performed the same, like when using the other base learners. In the 10 repetitions, only two concept drifts were found. These were false positives raised by the concept drift detector, giving the

slightly different results from RCD and MLP. Similarly from the other base learners, RCD was statistically better in RBF(50,0.0001).

Again, it is possible to verify that the number of centroids has a much higher influence in the number of concept drifts than the speed of change. For example, using the Cramer test, an average of 6.8 concept drifts were identified in the RBF(10,0.001) configuration. Increasing the number of centroids to 50, the number of concept drifts raised to 14.0, while changing the speed of change to 0.0001 augmented the number of concept drifts to 8.0.

Comparing Cramer and KNN, only in the RBF(50,0.0001) configuration the tests had statistical different performances: KNN performed better than Cramer. In all other data sets, KNN and Cramer performed similarly. Analyzing the evaluation time, the base learner was again faster. Here, Cramer was faster than KNN in average. While Cramer spent more than two times compared to MLP in the evaluation procedure, KNN was more than three times slower than the base learner. Again, KNN identified more concept drifts than Cramer: 6,178 versus 3,778. The difference between the drift detection of the two statistical tests was similar to the obtained using the naive Bayes base learner.

## 5.4 HOEFFDING TREE

Table 5.5 presents the results for the Hoeffding tree base learner. Statistically, the results were the same as using the MLP classifier: RCD was superior in the RBF(50,0.0001) data set. In all other situations, RCD and HT had comparable performances, independently of the statistical test used. In the thirteen data set configurations, HT had better average performance in four situations, RCD using Cramer was better in nine, and RCD using KNN, in seven.

Like in the previous results, the base learner did not have a better statistical result (considering the 95% confidence interval) than RCD in any data set configuration, independently of the statistical test used. The influence of the number of centroids can also be verified: the Cramer and KNN tests identified two concept drift in the ten repetitions of the RBF(10,0.001) configuration. Increasing the number of centroids to 50, Cramer and KNN identified, respectively, 70 and 50 concept drifts. Changing the speed of change to 0.0001 the results did not change.

Comparing the statistical tests, KNN and Cramer had exactly the same performance in six configurations. The number of concept drifts was also the same. This was the base learner where the results using the different statistical tests were most similar. Like when using the MLP base learner, KNN was statistically superior to Cramer in the

**Table 5.5** Results using the HT base classifier.

| | HT | | RCD HT CVM | | | RCD HT KNN | | |
|---|---|---|---|---|---|---|---|---|
| | **Time** | **Accuracy** | **Time** | **Accuracy** | **# Drifts** | **Time** | **Accuracy** | **# Drifts** |
| **RBF(0,0)** | 133.89 | **91.71±0.09** | 162.06 | 91.68±0.07 | 0.2 | 161.68 | 91.68±0.07 | 0.2 |
| **RBF(50,0.001)** | 211.62 | **52.19±0.03** | 270.10 | **52.19±0.03** | 7.0 | 270.81 | 52.17±0.02 | 5.0 |
| **RBF(10,0.001)** | 147.95 | 87.43±0.10 | 176.51 | **87.45±0.10** | 0.2 | 176.00 | **87.45±0.10** | 0.2 |
| **RBF(50,0.0001)** | 223.31 | 53.62±0.05 | 306.65 | 53.75±0.07* | 20.9 | 306.80 | **55.29±0.96*** | 75.2 |
| **RBF(10,0.0001)** | 150.48 | 87.45±0.09 | 180.63 | **87.47±0.08** | 0.2 | 179.81 | **87.47±0.08** | 0.2 |
| **HYP(10,0.001)** | 146.02 | 82.25±2.10 | 171.32 | **82.50±1.68** | 2.4 | 165.50 | 82.39±1.84 | 1.5 |
| **HYP(10,0.0001)** | 145.16 | 82.20±0.65 | 167.84 | **82.22±0.66** | 1.7 | 167.11 | **82.22±0.66** | 1.7 |
| **SEA(50)** | 5.83 | 87.51±0.79 | 7.70 | **87.81±0.47** | 0.6 | 7.60 | **87.81±0.47** | 0.6 |
| **SEA(50000)** | 5.74 | 87.51±0.78 | 7.65 | **87.94±0.23** | 0.6 | 7.49 | **87.94±0.23** | 0.6 |
| **LED(50000)** | 12.55 | 63.86±2.33 | 14.62 | **63.99±2.09** | 0.1 | 14.55 | **63.99±2.09** | 0.1 |
| **Covertype** | 13.21 | **68.03** | 357.24 | 64.47 | 2831 | 124.63 | 67.93 | 4725 |
| **Poker Hand** | 5.77 | 67.55 | 221.05 | **67.87** | 1274 | 27.88 | 64.62 | 1472 |
| **Electricity** | 0.39 | **75.72** | 19.36 | 74.99 | 135 | 2.28 | 67.12 | 162 |

RBF(50,0.0001) configuration. In the other configurations, both had comparable performances.

Considering the evaluation time, Cramer was, on average, 71.62% slower than HT. KNN was 34.13% slower than HT, this being the best relative result compared to the other base learners. Once more, KNN identified more concept drifts than the Cramer test: 7,212 versus 4,579.

## 5.5 J48

Table 5.6 presents the results for the J48 base learner. Statistically comparing J48 to RCD using KNN, the latter had statistical better results in the RBF(50,0.0001) data set. In the other configurations, the results were similar.

Comparing the evaluation time, J48 was 27.74% faster than RCD using the Cramer test and 53.14% when using KNN. This was the experiment where Cramer was closest to the base learner. The KNN statistical test also identified more concept drifts than the Cramer test: 4,699 versus 3,641. KNN outperformed the Cramer test in the RBF(50,0.0001) configuration and Cramer had statistical better results in the RBF(10,0.0001). In the other data sets, the results were statistically similar.

**Table 5.6** Results using the J48 base classifier.

| | J48 | | RCD J48 CVM | | | RCD J48 KNN | | |
|---|---|---|---|---|---|---|---|---|
| | **Time** | **Accuracy** | **Time** | **Accuracy** | **# Drifts** | **Time** | **Accuracy** | **# Drifts** |
| **RBF(0,0)** | 23.74 | 84.38±0.32 | 33.35 | **84.46±0.34** | 0.2 | 33.33 | **84.46±0.34** | 0.2 |
| **RBF(50,0.001)** | 86.91 | 50.13±0.15 | 99.12 | 50.06±0.08 | 6.5 | 112.06 | **50.18±0.09** | 14.2 |
| **RBF(10,0.001)** | 35.57 | **81.24±0.62** | 48.33 | 81.20±0.60 | 3.5 | 49.45 | 80.97±0.58 | 3.7 |
| **RBF(50,0.0001)** | 86.68 | 50.20±0.09 | 99.36 | 50.19±0.13 | 7.1 | 125.29 | **51.20±0.56*** | 120.1 |
| **RBF(10,0.0001)** | 35.42 | 81.62±0.75 | 46.64 | **81.63±0.38** | 5.4 | 48.34 | 81.34±0.36 | 4.3 |
| **HYP(10,0.001)** | 22.32 | **72.84±4.95** | 32.92 | 72.42±5.64 | 3.2 | 33.67 | 72.42±5.64 | 4.4 |
| **HYP(10,0.0001)** | 22.44 | **73.42±0.81** | 33.34 | 72.73±1.49 | 3.0 | 33.44 | 72.73±1.49 | 3.0 |
| **SEA(50)** | 2.82 | 85.12±1.16 | 4.22 | **85.73±0.49** | 0.4 | 4.22 | **85.73±0.49** | 0.4 |
| **SEA(50000)** | 2.73 | 84.86±1.11 | 4.30 | **84.88±1.10** | 0.8 | 4.29 | **84.88±1.10** | 0.8 |
| **LED(50000)** | 7.37 | 67.49±5.74 | 11.02 | 67.84±5.09 | 0.7 | 11.08 | **67.88±5.02** | 0.7 |
| **Covertype** | 10.62 | **59.22** | 15.80 | 55.28 | 2737 | 36.46 | 47.75 | 1902 |
| **Poker Hand** | 5.34 | 17.59 | 8.31 | 22.40 | 439 | 31.62 | **63.72** | 1247 |
| **Electricity** | 0.34 | 63.10 | 0.53 | **69.24** | 157 | 0.95 | 64.97 | 32 |

## 5.6 CONCLUSION

This chapter compared RCD using two multivariate non-parametric statistical tests (KNN and Cramer) with five base learners: Hoeffding trees with naive Bayes at the leaves, naive Bayes, Multilayer Perceptron, a pure Hoeffding tree, and J48. The tests were performed in seven data sets: four artificial data sets, with several rates-of-change, and three real world data sets.

The RCD configurations tested presented better performance than the Hoeffding tree base learner with naive Bayes at the leaves in 8 out of 13 possible situations using the Cramer test, in four situations the Hoeffding tree performed better, and in one situation their performances were similar. Using KNN, RCD has beaten the base learner by 9 to 2. In two situations, their performances were similar. Statistically, in three data set configurations RCD had a better performance than the base learners considering both statistical tests. In all other situations, the performances were similar.

Using the naive Bayes base learner, the results of RCD are even better considering the Cramer test: of the thirteen data set configurations, RCD performed better than naive Bayes in eleven. In one situation they performed similarly and in one naive Bayes had higher performance. Using KNN, RCD was better in ten data set configurations and in three the results were similar. Statistically, RCD performed better in four situations: both versions of Hyperplane and Random RBF with 50 centroids.

These results indicate that RCD tends to perform better than the base learners in environments with many concept drifts. This is an expected behavior because RCD stores classifiers and reuses them if current data is similar to the ones used to build it. In environments without concept drifts, RCD performs similarly as the base classifier, as the results from the tests show in the Random RBF(0,0) configuration using the Hoeffding tree with naive Bayes at the leaves as base learner. In the others, the drift detection method identifies two concept drifts, in data with no concept drift, leading to slightly different results (false positives).

Comparing the two statistical tests in the artificial data sets, KNN statistically outperformed Cramer in 7 situations, and the opposite occurred only once. In the remaining situations, both tests performed similarly. Considering the real data sets, KNN has beaten Cramer by 10 to 5. Results from the experiments clearly show that KNN is better suited for concept drift detection than the Cramer test.

Considering the evaluation times, as expected, the base learners were faster than using RCD. Comparing the two statistical tests, using both Hoeffding trees and naive Bayes, KNN was faster, while Cramer was faster when using Multilayer Perceptron and J48. In all the tests, KNN was faster than Cramer in 41 situations, while the opposite occurred 23 times.

These results are consistent with previous findings presented at Chapter 4 confirming that the usage of the RCD approach to handle concept drifts is promising and improves single classifiers results when using them as base learners of the framework, independently of the statistical test used.

Finally, it is important to emphasize that, in the experiments, there was no situation where the base learner had a statistically better performance compared to using RCD.

CHAPTER 6

# PARALLEL STATISTICAL TESTS

As described in the previous chapter, the statistical test to be used has a big influence in performance. Increasing the buffer size from 100 to 300 instances made RCD using the KNN statistical test to slow down eight times, while using the Cramer test slowed down the execution more than 15 times. Using a buffer size of 100 instances with KNN takes in average 0.003 seconds to complete while with Cramer takes 0.413 seconds, more than 130 times slower. Increasing the buffer size only makes the difference in performance bigger, indicating that it is monotonic.

It is necessary an approach to improve performance of slower statistical tests so RCD can make use of them. The general solution here presented, which can be used with any statistical test, is to perform them in parallel by the use of a thread pool of configurable size.

This chapter describes the improvements made in RCD to perform statistical tests in parallel and presents how concurrency impacts performance.

## 6.1 EXPERIMENTS CONFIGURATION

To perform the experiments, we selected artificial data sets suffering from abrupt and gradual concept drifts, as well as real-world data sets. Thus, we used the same data sets described at Chapter 5, except for Random RBF.

The RCD configuration used in the experiments was the following: naive Bayes as base learner, classifiers collection size set to 15, KNN as the statistical test used (with $k = 3$), and the minimum amount of similarity between data samples set to 0.05. Two buffer sizes and test frequencies (in the testing phase), and three thread pool sizes have been used.

The evaluation methodology used was Interleaved Chunks, also known as data block evaluation method (Brzeziński and Stefanowski, 2011). It initially reads a block of $d$ instances. When the block is formed, it uses the instances for testing the existing classifier and then the classifier is trained on the instances. This methodology was used because it is better suited to compute training and testing times. In the following experiments,

$d$ was set to 100,000 instances in the Hyperplane, Covertype, and Poker Hand data sets, and to 10,000 instances in the LED, SEA, and Electricity data sets, 1% of the size of the artificial data sets, thus the evaluation is performed 100 times for each artificial data set. Tests were repeated ten times and a 95% confidence interval was computed.

All the experiments were performed using the MOA framework in three different computer systems, all executing the Ubuntu 12.04 operating system with OpenJDK 1.7, as described at Table 6.1. By the use of these systems, it is also possible to analyze the influence of the number of available cores in the processor in the performance of the experiments.

**Table 6.1** Systems used for evaluation.

| Processor | Clock | Memory | Cores |
|---|---|---|---|
| **Pentium 4** | 3.06GHz | 1.2GB | 1 |
| **Celeron E3400** | 2.60GHz | 4.0GB | 2 |
| **Core i3 330M** | 2.13GHz | 4.0GB | 4 |

In the following sections it is presented how parallelism in RCD behaves in the three systems presented.

## 6.2 RESULTS FOR A SINGLE CORE PROCESSOR

Initially, we present how the parallelism of the statistical tests in RCD perform in a single core processor, here represented by a Pentium 4. It is important to notice that this processor does not have the Hyper-Threading functionality active, meaning that it really executes sequentially.

Table 6.2 presents the average evaluation, train, and test times, in seconds, to execute RCD with a buffer size of 100 instances and a test frequency of 500 instances, with one, two, and four parallel statistical tests. This configuration is used as a basis for comparison. Two other configurations are compared with this one, one of them increasing the buffer size and the other increasing the test frequency, to analyze their impact in performance and the influence of the parallel statistical tests.

Analyzing the average evaluation times, we can see that using one core was faster than using two or four cores, and using two cores was faster than using four (except for LED). The same results apply when considering the 95% confidence interval: one core is statistically faster than two and four cores in all tested data sets. Two and four cores have similar statistical results in two data sets: HYP(10,0.001) and LED. In all others, using two cores is statistically faster. In average, using one core is 3.12% and 3.08%

**Table 6.2** Results for a buffer with 100 instances and test frequency of 500 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | **eval** | **train** | **test** | **eval** | **train** | **test** | **eval** | **train** | **test** |
| **HYP(10,0.001)** | 178.26 | 78.92 | 71.00 | 183.79 | 82.31 | 71.66 | 183.98 | 82.39 | 74.08 |
| **HYP(10,0.0001)** | 178.86 | 79.13 | 72.56 | 183.73 | 82.53 | 74.23 | 184.25 | 82.55 | 74.39 |
| **SEA(50)** | 7.59 | 2.84 | 2.43 | 7.67 | 2.91 | 2.46 | 7.71 | 2.93 | 2.47 |
| **SEA(50000)** | 7.69 | 2.85 | 2.42 | 7.79 | 2.91 | 2.47 | 7.81 | 2.92 | 2.47 |
| **LED(50000)** | 50.62 | 20.96 | 19.15 | 53.26 | 22.38 | 20.45 | 53.24 | 22.36 | 20.43 |
| **Covertype** | 153.92 | 129.41 | 11.77 | 158.35 | 133.84 | 11.75 | 165.01 | 140.43 | 11.76 |
| **Poker Hand** | 70.63 | 59.14 | 5.34 | 72.97 | 61.50 | 5.35 | 74.73 | 63.28 | 5.55 |
| **Electricity** | 4.63 | 4.05 | 0.13 | 4.92 | 4.33 | 0.14 | 5.10 | 4.52 | 0.13 |

faster than two cores in the artificial and real-world data sets, respectively. Comparing one core to four cores, the former is 3.30% faster in the artificial data sets and 6.83% in the real-world data sets.

It is also possible to notice that the testing phase is faster than the training phase. This is a consequence of the configuration used. For example, KNN takes three milliseconds, in average, in the Electricity data set using a buffer size of 100 instances, every time a statistical test is performed. The testing phase occurs every 500 instances. Thus, the time taken to perform a test is small and the testing phase is performed more sparsely.

Also, the differences in training and testing times are much higher in the real-world data sets, compared to the artificial ones. This is probably due to the number of detected concept drifts. Table 6.3 presents the average number of detected concept drifts (# CD), classifiers set size (CS), and number of reused classifiers (# RC) for the three tested RCD configurations, where $b$ is the buffer size and $t$ is the test frequency. In the artificial data sets, the average number of detected concept drifts are reasonably low, as can be seen in the first column of each configuration. A small number of concept drifts demands few statistical tests to be performed. A large number of concept drifts demands more statistical tests, which take more time to complete.

For its turn, the classifiers set size is proportional to the number of detected concept drifts. In the artificial data sets, the average classifiers set size is below three, while in the real-world data sets, it is always full (15 classifiers). Finally, the # RC column shows that, in the artificial data sets, the number of reused classifiers is almost the same of the detected concept drifts, indicating that, besides identifying few concept drifts, most of the time a classifier is reused. This tends to reduce the benefits of using parallelism,

**Table 6.3** Detected concept drifts, classifiers set size and reused classifiers quantities.

| Data set configuration | b = 100, t = 500 | | | b = 100, t = 100 | | | b = 500, t = 500 | | |
|---|---|---|---|---|---|---|---|---|---|
| | # CD | CS | # RC | # CD | CS | # RC | # CD | CS | # RC |
| **HYP(10,0.001)** | 7.7 | 2.6 | 6.0 | 8.4 | 2.8 | 6.5 | 11.9 | 2.6 | 9.9 |
| **HYP(10,0.0001)** | 8.2 | 2.4 | 6.7 | 10.2 | 2.3 | 8.2 | 9.5 | 2.4 | 7.5 |
| **SEA(50)** | 0.1 | 1.1 | 0.0 | 0.1 | 1.1 | 0.0 | 1.1 | 1.1 | 1.0 |
| **SEA(50000)** | 0.4 | 1.1 | 0.3 | 0.2 | 1.1 | 0.1 | 0.2 | 1.1 | 0.1 |
| **LED(50000)** | 0.3 | 1.2 | 0.1 | 2.3 | 1.2 | 2.1 | 0.2 | 1.2 | 0.0 |
| **Covertype** | 2980.0 | 15.0 | 844.0 | 3376.0 | 15.0 | 944.0 | 3063.0 | 15.0 | 798.0 |
| **Poker Hand** | 1871.0 | 15.0 | 109.0 | 1871.0 | 15.0 | 109.0 | 410.0 | 15.0 | 18.0 |
| **Electricity** | 212.0 | 15.0 | 30.0 | 212.0 | 15.0 | 30.0 | 183.0 | 15.0 | 20.0 |

specially if the reused classifiers are the first ones to be tested. In the real-world data sets, Forest Covertype had the highest percentage of reused classifiers, around 27%, while Poker Hand had the lowest percentage, around 5%.

To better analyze how the thread pool influences performance, we computed the average amount of time (in milliseconds) needed to create the thread pool and to assign the statistical tests to their respective slots, to execute the thread pool, and to finalize it. In the assignment stage, if a statistical test is assigned to an active slot, it starts immediately executing, while other tests are still being assigned, so it is not necessary to wait for all tests to be assigned a specific slot to start execution, saving time. This information is presented at Table 6.4. The thread pool is used every time a statistical test is performed: in the training phase, when the concept drift detector identifies a change, and, in the testing phase, periodically based on the test frequency parameter.

**Table 6.4** Thread pool management for a buffer with 100 instances and test frequency of 500 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 3.62 | 3.11 | 2.89 | 4.21 | 2.54 | 3.11 | 0.08 | 0.00 | 0.03 |
| **HYP(10,0.0001)** | 3.04 | 2.72 | 2.75 | 3.53 | 2.88 | 2.47 | 0.01 | 0.04 | 0.04 |
| **SEA(50)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 4.00 | 1.25 | 1.25 | 1.75 | 1.75 | 3.00 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 3.67 | 4.33 | 4.33 | 7.00 | 5.67 | 5.67 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 3.28 | 6.84 | 14.24 | 42.28 | 40.47 | 35.65 | 0.03 | 0.02 | 0.03 |
| **Poker Hand** | 3.16 | 5.15 | 9.97 | 23.93 | 23.22 | 19.31 | 0.01 | 0.02 | 0.03 |
| **Electricity** | 3.11 | 4.66 | 8.61 | 15.14 | 14.95 | 11.85 | 0.02 | 0.02 | 0.04 |

In the artificial data sets, using one single active core usually takes more time in the creation of the pool compared to using parallelism (only in the LED data set it was faster). In the real-world data sets, the opposite occurs. The execution time is faster when using more than one active core in the real-world data sets and in the artificial data sets. Just in the SEA(50) data set, the execution times were similar. The destruction times are usually very close one to another.

The data may seem irrelevant due to the magnitude of the results. But it must be considered that these values accumulate each time a statistical test is performed. For example, considering a thread pool with one core and the Poker Hand data set, a test is performed in average in 23.93 milliseconds. Multiplying this value by the number of identified concept drifts, presented at Table 6.3 (1871.0), we obtain 44.78 seconds, which is more than 60% of the evaluation time. Thus, we can conclude that, in the majority of the time RCD is performing the statistical tests.

Table 6.5 presents similar results as the ones shown at Table 6.2, but the test frequency was increased five times. The results present similar trends: using one core offered lower average times in evaluation, training, and testing phases. Only in SEA(50000) the three core sizes had similar statistical results. In all the others, one core was statistically faster.

**Table 6.5** Results for a buffer and test frequency of 100 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 671.78 | 80.94 | 564.40 | 818.31 | 85.67 | 707.76 | 872.98 | 85.65 | 761.80 |
| **HYP(10,0.0001)** | 659.52 | 80.48 | 553.31 | 724.30 | 85.21 | 614.20 | 733.26 | 85.91 | 621.51 |
| **SEA(50)** | 10.10 | 2.83 | 4.94 | 10.33 | 2.94 | 5.07 | 10.34 | 2.93 | 5.08 |
| **SEA(50000)** | 10.49 | 2.85 | 5.08 | 10.45 | 2.93 | 5.10 | 10.46 | 2.93 | 5.11 |
| **LED(50000)** | 57.18 | 20.93 | 25.90 | 62.11 | 22.37 | 29.43 | 62.10 | 22.38 | 29.41 |
| **Covertype** | 853.29 | 139.03 | 701.14 | 866.30 | 144.20 | 709.28 | 885.16 | 151.84 | 720.64 |
| **Poker Hand** | 469.79 | 59.06 | 404.61 | 485.06 | 61.13 | 418.14 | 507.70 | 64.10 | 437.51 |
| **Electricity** | 19.36 | 4.01 | 14.90 | 20.27 | 4.26 | 15.57 | 21.46 | 4.56 | 16.45 |

Comparing the results of Tables 6.2 and 6.5, the evaluation time increased considerably, specially in the data sets with more detected concept drifts. The training time stayed practically the same. The difference in evaluation time was mostly due to the testing time: increasing the test frequency means more statistical tests being performed in the testing phase, making the testing phase take longer to complete.

Table 6.6 presents similar information as Table 6.4. Using one single active core usually takes less time in the creation of the pool compared to using parallelism. In the

execution phase, the opposite occurs, but the differences are higher in the creation phase than in the execution phase. This explains why using one core is faster than using two or four cores. The destruction times are usually very close one to another.

**Table 6.6** Thread pool management for a buffer and test frequency of 100 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.67 | 2.76 | 3.51 | 4.27 | 3.59 | 3.38 | 0.01 | 0.01 | 0.02 |
| **HYP(10,0.0001)** | 0.58 | 1.56 | 1.57 | 4.25 | 3.86 | 3.91 | 0.01 | 0.01 | 0.01 |
| **SEA(50)** | 0.04 | 0.05 | 0.05 | 0.19 | 0.19 | 0.19 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.05 | 0.05 | 0.05 | 0.20 | 0.19 | 0.19 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.05 | 0.19 | 0.19 | 0.60 | 0.70 | 0.70 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 3.58 | 7.77 | 16.19 | 117.83 | 115.58 | 109.91 | 0.04 | 0.03 | 0.05 |
| **Poker Hand** | 3.24 | 6.64 | 14.54 | 47.69 | 46.06 | 40.70 | 0.04 | 0.04 | 0.05 |
| **Electricity** | 3.21 | 6.09 | 12.50 | 34.25 | 33.22 | 29.14 | 0.04 | 0.03 | 0.05 |

Table 6.7 also presents results similar to those of Table 6.2, but increasing the buffer size by five times. Again, using one core was faster than using two or four cores. Using one core was statistically faster than two cores in all data sets, except for HYP(10,0.0001) where the results are statistically similar regardless of the number of used cores. Compared to four cores, one core was also statistically faster in HYP(10,0.001) and LED; in the others, both had similar results. The training time practically does not change, but the testing time is considerably higher.

This last configuration is the one that takes longer to complete. In the Electricity data set, a KNN test performs in average in 62 milliseconds using buffer sizes with 500 instances. Even making tests more sparsely compared to the configuration used at Table 6.5, the influence of a bigger buffer size is much higher than that of performing the tests more frequently. In the real-world data sets, in all situations, using one core was statistically faster than using two cores and using two cores was statistically faster than using four cores.

Table 6.8 presents the thread pool management times to this RCD configuration. The creation times are lower when using sequential execution, quite similar to the ones presented at Table 6.6. The execution times, on the other hand, are also usually lower when using one single active core, which is a different behavior than the obtained in the previous two configurations. The destruction times are similar, being usually slightly faster when using sequential execution.

**Table 6.7** Results for a buffer and test frequency of 500 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 1962.65 | 83.13 | 1850.83 | 2720.77 | 85.80 | 2606.70 | 2816.04 | 85.77 | 2703.51 |
| **HYP(10,0.0001)** | 1987.29 | 82.65 | 1876.67 | 2355.89 | 85.45 | 2240.76 | 2357.33 | 85.08 | 2245.69 |
| **SEA(50)** | 16.56 | 3.08 | 10.10 | 16.67 | 3.12 | 10.17 | 16.66 | 3.10 | 10.23 |
| **SEA(50000)** | 17.09 | 3.05 | 10.49 | 17.21 | 3.09 | 10.57 | 17.06 | 3.10 | 10.41 |
| **LED(50000)** | 86.36 | 21.21 | 53.77 | 90.35 | 22.67 | 56.28 | 90.43 | 22.66 | 56.37 |
| **Covertype** | 4504.01 | 568.44 | 3922.63 | 4610.14 | 601.88 | 3995.27 | 4753.21 | 641.39 | 4098.89 |
| **Poker Hand** | 863.27 | 71.68 | 785.31 | 902.43 | 74.33 | 821.85 | 932.71 | 76.21 | 850.20 |
| **Electricity** | 49.74 | 13.39 | 35.83 | 53.87 | 14.41 | 38.93 | 55.40 | 14.99 | 39.88 |

**Table 6.8** Thread pool management for a buffer and test frequency of 500 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.76 | 2.48 | 2.79 | 89.05 | 125.32 | 129.90 | 0.03 | 0.04 | 0.05 |
| **HYP(10,0.0001)** | 0.70 | 1.57 | 1.60 | 90.48 | 107.82 | 108.02 | 0.03 | 0.04 | 0.06 |
| **SEA(50)** | 0.06 | 0.05 | 0.05 | 3.71 | 3.71 | 3.75 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.06 | 0.06 | 0.05 | 3.86 | 3.87 | 3.80 | 0.00 | 0.01 | 0.00 |
| **LED(50000)** | 0.05 | 0.15 | 0.15 | 17.27 | 17.79 | 17.81 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 3.18 | 7.70 | 16.33 | 1433.14 | 1462.05 | 1499.35 | 0.02 | 0.02 | 0.03 |
| **Poker Hand** | 3.08 | 8.31 | 18.88 | 458.16 | 474.85 | 481.83 | 0.02 | 0.02 | 0.03 |
| **Electricity** | 3.27 | 7.09 | 14.54 | 209.86 | 224.42 | 223.83 | 0.01 | 0.03 | 0.03 |

Analyzing the results here presented, we can conclude that the best option in a single core system is to use only one active thread to perform the statistical tests, i.e., sequentially. We could also observe that increasing the buffer size had a higher impact in performance than increasing the test frequency. Multiplying the test frequency by five increased the evaluation time between 4.12 and 4.55 times; the same increase in the buffer size impacted performance between 14.54 and 16.18 times.

Next, in sections 6.3 and 6.4, the same tests are performed in a two-core and in a four-core processor system, respectively, to check if similar results apply or not, and what the differences are. Tables regarding these computer systems were put in Appendix A to facilitate reading.

## 6.3 RESULTS FOR A TWO CORE PROCESSOR

In this section, we present the results for a two core processor. Table A.1 presents the evaluation, train, and test times for a configuration of RCD using a buffer size of 100 instances and a test frequency of 500 instances.

Evaluation times are quite similar in these experiments in the artificial data sets. Analyzing the confidence interval, statistics informs that using one core is similar to using two or four cores. Comparing two and four cores, the former is statistically faster in both SEA data sets. In the others, it is not possible to distinguish which one is best. In the real-world data sets, using two cores offered the best configuration. This is in line with the findings of the experiments using the single core processor: again, the best option was to use the number of available cores in the processor as the number of parallel statistical tests. The testing time was always lower than the training time because, in this configuration, the test frequency is not high and the buffer size is small, reflecting in faster execution of the statistical tests.

Next, Table A.2 presents the times taken to create the thread pool, execute the statistical tests, and destroy the thread pool. The creation time is usually faster with one active core in both artificial and real-world data sets. The execution time, however, is faster when using two active cores compared to a single core. Comparing two and four cores, each one is faster in two data sets and in another, SEA(50), both had similar performances. In the real-world data sets, using four cores was the fastest option. Finally, the destruction times presented very similar results using all three core sizes.

Table A.3 shows results similar to those of Table A.1, but increasing the test frequency by five times. In the artificial data sets, using one core was statistically better only in SEA(50). Using two cores was better than using one core for HYP(10,0.001). Four cores offered better performance in both configurations of the Hyperplane data set compared to using one core. Comparing the parallel configurations, using four cores was better at HYP(10,0.001) and two cores was better in the LED data set. In average, using two cores is faster than using one core by 6.41% and is slower by 0.59% than using four cores in the artificial data sets. Using four cores is faster than one core by 6.96%.

In the real-world data sets, using two active cores offered the best statistical results. Four cores offered second best results, and the sequential execution had the worst performance. The time spent in the testing phase is the bottleneck in this configuration, as was also the case when using a single core processor. In these data sets, using two cores was faster than using one core by 44.18% and by 3.90% compared to using four cores. One core is also slower than using four cores by 42.00%.

Table A.4 presents the thread pool management times of this configuration. In the artificial data sets, the creation time was usually faster using one active core, similarly to results obtained when using a single core processor. In the real-world data sets, using two cores was faster at Poker Hand and Electricity; at Forest Covertype, using one core was better. Considering the execution time, in the data sets with more detected concept drifts (Hyperplane and the real-world ones), using one active core was slower than using two active cores, which was slower than using four active cores. In the remaining data sets (SEA and LED), the results were statistically similar. The destruction times were quite similar, taking at most 0.09 milliseconds to complete in all three core size configurations.

In the last tested configuration for the two core processor, presented at Table A.5, the buffer size was increased to 500 instances. Here, only in HYP(10,0.001) using two cores was faster than using one core. In all other artificial data sets, independently of the number of active cores used, the results were statistically the same. In the real-world data sets, using two cores is statistically superior than using one or four cores: two cores is in average 36.62% superior than sequential execution, and 4.19% when compared to using four cores. Using four cores is also faster than using one core; in this case, by 33.97%.

Table A.6 presents similar information as described at Tables A.2 and A.4. The creation time is lower when using one core in five configurations; using two cores is lower in two configurations; in the remaining one, results were similar in the three core sizes. We can also see that, in the data sets with more detected concept drifts, the time taken to create the thread pool is also higher. In the other data sets (SEA and LED), the differences in time between the three core sizes were negligible. The execution times were faster when using four cores; except from HYP(10,0.001) and Forest Covertype, where using two cores was faster. The destruction times, similarly to results presented in previous configurations, were very similar in the three executed core sizes.

In this section, we observed how a two core processor deals with sequential and parallel execution of statistical tests. Similarly to results obtained when using a single core processor, the best number of statistical tests to perform in parallel was the number of cores available in the processor. In all the real-world data sets, in the three RCD configurations tested, using two cores had better statistical results than using one and four cores.

Also, considering the base configuration using 100 instances in the buffer and 500 instances as test frequency, an increase in the test frequency by five times, increased the evaluation time between 5.37 and 6.15 times. An increase in the buffer size by five times increased the evaluation time between 16.23 and 17.15 times, clearly indicating that the

buffer size has a much bigger influence than the test frequency in the evaluation time, as was also the case in the single core processor.

We can also see the influence of the buffer size and test frequency in the time taken to complete the testing phase. The increase in the test frequency resulted in the testing time taking more than 18 times to complete, while the increase in the buffer size resulted in more than 57 times.

## 6.4 RESULTS FOR A FOUR CORE PROCESSOR

Similarly, in this section it is analyzed how a four core processor deals with concurrent statistical tests. Table A.7 presents the evaluation, train and test times (in seconds) for RCD considering the ten runs, using a buffer size with 100 instances and a test frequency of 500, considering thread pools with one, two, and four active cores.

It is worth pointing out that the results were quite similar in the artificial data sets, regardless of the thread pool size, presenting behavior compatible to the other computer systems. This pattern did not occur in the real-world data sets and is probably due to the number of detected concept drifts. The number of statistical tests is directly related to concept drifts, in the training phase, and to the test frequency, in the testing phase.

In this RCD configuration, using one core did not have better statistical results than using two or four cores in any of the artificial data sets. On the other hand, one core was statistically inferior than two and four cores in the SEA(50000) data set. Using four cores was statistically better than using two cores in the SEA(50) data set. In all other artificial data sets, results were statistically similar independently of the thread pool size.

In the real-world data sets, however, more cores returned lower evaluation times: two cores were faster than one core and using four cores was faster than the other two thread pool sizes. In the artificial data sets, using one or two cores had, on average, practically identical performances, but using one core was 29.42% slower in the real-world data sets. Comparing one and four cores, similar results apply. One core and four cores had similar results in the artificial data sets and one core was 35.52% slower in the real data sets. Using four cores had practically the same performance than using two cores in artificial data sets, but it was 8.63% faster in the real-world data sets. The real-world data sets presented a huge amount of concept drifts, the classifiers set became full and the number of reused classifiers was also much bigger when compared to the artificial data sets.

Table A.8 presents the thread pool management times to the RCD configuration with a buffer size of 100 instances and test frequency of 500 instances. Analyzing the real-world data sets, it is possible to observe that, in general, the greater the number of cores,

the longer the time spent in the creation of the thread pool was, but the differences are usually very small. In the execution time, using more cores meant faster execution, with no exceptions. The destruction times were usually negligible, taking less than 0.14 milliseconds. Comparing these results to the ones performed in a single core processor, we can see that using four cores was faster in the three phases of the thread pool management.

The results of Table A.9 are similar to those presented at Table A.7 but the test frequency was increased to 100 instances. Again, parallelism outperforms the sequential solution in the real-world data sets, the ones with more detected concept drifts. Here, we can also notice that the evaluation time is mostly spent in the testing phase, differently from the results of Table A.7. As the test frequency is higher, the evaluation time and the time spent in the testing phase increased considerably. Making the tests more frequently also increased the number of detected concept drifts in most data sets: the exceptions were SEA(50000), which reduced from 0.4 to 0.2, and Poker Hand, Electricity, and SEA(50), where the number of detected concept drifts stayed the same. All thread pool sizes had practically the same performance in the artificial data sets, but using more cores returned slightly better results. In the real-world data sets, using one core was 40.29% slower than using two cores and 30.67% than using four cores. Using two cores offered better average results compared to using four cores by 16.12% in the real-world ones.

Table A.10 presents similar information to those presented at Table A.8. Times in the artificial data sets are usually very similar using the three thread pool sizes. In the real-world data sets, the creation time is usually slightly faster using less cores, the execution time was smaller when using two cores, and destruction times are 0.15 milliseconds or less.

Instead of increasing the test frequency, Table A.11 presents similar information as Tables A.7 and A.9 but increasing the buffer size to 500 instances. Here, the tests took longer to complete; in average, 62 milliseconds compared to 3 milliseconds when using a buffer with 100 instances. Nevertheless, the results were similar to the ones presented at Table A.9. Using parallelism was much faster in the real-world data sets and slightly slower in the artificial ones. Using one core was 2.08% faster than using two cores in the artificial data sets but 40.62% slower in the real-world ones.

Comparing one and four cores similar results apply: one core was 2.92% faster in the artificial data sets and 35.56% slower in the real-world ones. Using two cores was slightly better than using four: 0.82% in the artificial and 8.54% in the real-world data sets. This probably occurs because there are much more statistical tests to perform and they take longer to complete than the tests in the other configurations, putting a higher load on

the whole system and negatively affecting the performance.

Comparing Tables A.7, A.9 and A.11, it is possible to notice that once again the increase in the buffer size had a higher influence in the evaluation time than the increase in the test frequency. Increasing the test frequency by five times increased the evaluation time between 7.90 and 8.53 times. Increasing the buffer size by five times increased the evaluation time between 17.80 and 19.13 times. The training time practically did not change in the three configurations performed; the increase in the evaluation time was due to the testing time.

Table A.12 presents the times taken for the thread pool management, as previously described at Tables A.8 and A.10. In the artificial data sets, the creation times are very close in the three sizes of active cores used. In the real-world data sets, one and two cores take very similar times, and using four cores takes more time than the other two. This probably occurs because, in the creation time, the statistical tests associated with active cores begin executing while other tests are still being assigned. Thus, the creation time tends to be bigger when there are more active cores and they take longer to complete. We can see this by comparing the three tables concerning thread pool management. At Tables A.8 and A.10, the differences in the creation time between one, two and four cores are almost negligible. In these cases, the average time taken to perform a statistical test is three milliseconds. At Table A.12, using four cores takes longer than using one and two cores. Here, the average time taken to perform the statistical test is 62 milliseconds. The execution times are very similar in the artificial data sets and considerably lower in the parallel configurations.

## 6.5 PROCESSOR COMPARISON

In this section we compare the results presented in the three tested processors. Comparing the performances obtained using a processor with a single core versus using two cores, we can notice that in all three tested RCD configurations, using two cores offered the best performance. In some configurations, using two cores was more than four times faster than using a single core. Biggest differences occur in the real-world data sets, the ones with more detected concept drifts, and with more than one active core. This indicates that the single core processor does not handle concurrency as well as the two core processor, despite having a faster processor.

Comparing the four core processor with the two core processor, the latter presented average better results. Using a thread pool with one active core, the processor with four cores took in average 15% more time than the needed by the two core processor in the

artificial data sets. This can be explained by the processor clock of each core, which is faster in the Celeron. Using one single thread, the factor that mostly contributes to performance is the processor frequency. Even as more statistical tests are allowed to perform in parallel by incrementing the thread pool size, the differences in performance remain stable.

Comparing the single core processor with the four core processor, similar results as the ones obtained in the one versus two core processor apply. Better results occur in the data sets with more detected concept drifts and when more active cores are used. In these situations, using four cores was in average more than two times faster.

In the experiments here performed, neither the processor with the higher clock frequency nor the processor with more available cores offered the best results, but the processor that balanced the average number of available cores and clock frequency.

Comparing the best configuration for each processor, i.e., the configuration where the number of cores in the processor is equal to the number of parallel statistical tests, the same conclusion can be drawn: the processor with two cores had better results than the four core processor, which had better results than the single core processor.

## 6.6 CONCLUSION

Depending on the statistical test to be performed, the influence in the evaluation time may be considerable, as indicated in the previous chapter. A proposed approach to increase the speed of RCD was the execution of the statistical tests in parallel, by the use of a thread pool of configurable size. This solution is general and can be used with any statistical test to be implemented in RCD.

Thus, this chapter studied the influence of executing parallel statistical tests in the RCD framework in six data sets (summing eight configurations), with varying number of detected concept drifts, with abrupt and gradual concept drifts, and considering artificial and real-world data sets. Tests were performed with both sequential and parallel execution of two and four statistical tests.

It was possible to observe from this chapter that the execution of the statistical tests are the bottleneck of the RCD framework. The major part of the time of the RCD execution is spent executing the statistical tests.

Analysis of the experiment results led to the conclusion that the execution of parallel statistical tests was most beneficial when there was a high number of detected concept drifts leading to more statistical tests being performed. Also, parallelism is highly influenced by the number of available cores in the processor and processor clock frequency.

Results suggest that, in average, it is better to set the number of parallel statistical tests to the number of available cores in the processor.

Besides that, tests were carried out to analyze the performance results in the following conditions:

1. the buffer size was increased, making the statistical tests take longer to complete; and

2. increase the test frequency, making more statistical tests to be performed in the testing phase.

Multiplying the test frequency five times increased the evaluation time more than eight times, while multiplying the size of the buffer five times increased the evaluation time more than 17 times, indicating that the buffer size has a higher impact in performance than the test frequency.

In data sets with a small number of detected concept drifts (the artificial data sets), the performances were quite similar. On the other hand, in the data sets with a high number of detected concept drifts (the real-world ones), using parallelism increased performance in values ranging from 29.42% to 40.62%.

However, not only the number of detected concept drifts influences performance. Reusing classifiers also matters. If the first tests identify similarity between distributions, several other tests will not be executed, reducing the benefits of using a thread pool. On the other hand, if the last tests identify similarity (or none at all), more tests need to be executed.

For example, in the artificial data sets, the average classifiers collection size is below three, not being even close to fill the set maximum size (15 classifiers). Having few stored classifiers indicates that few statistical tests needed to be performed. The difference between the number of detected concept drifts and the number of stored classifiers is due to the reuse of classifiers. The number of reused classifiers were very close to the number of detected drifts, informing that, in the majority of the detected concept drifts, a classifier was reused, reducing the impact of parallelism.

Analysis of the thread pool creation, execution, and destruction times were also performed, showing that the differences in performance occur in the creation and execution phases. The destruction times were commonly very similar, usually taking less than 0.1 millisecond to complete.

In the experiments here performed, the best performance was obtained by the processor which balanced the average number of cores (2) and clock frequency (2.60GHz).

CHAPTER 7

# DRIFT DETECTION METHODS COMPARISON

RCD uses internally a drift detection method to identify when a drift is occurring. Thus, it is important to analyze several available methods and verify in which have better performance in data sets with different characteristics, similarly to the work of Kiang (2003), that compared five classification methods with different metrics.

Proposals of drift detection methods usually perform comparisons with few other alternatives, using also few metrics. For example, in the original paper of EDDM, it was only compared to DDM in terms of prequential error; DDM and ADWIN were only compared with themselves, in different configurations; PL was tested with other classifiers, but not with drift detection methods.

In this chapter several experiments were performed to analyze the predictive accuracies, evaluation times, false alarms rates, miss detection rates, and distance to the drift point of six different drift detection methods: PL, ADWIN, DDM, ECDD, PHT, and EDDM. To the best of our knowledge, this is the broadest (in numbers of methods) and deepest (in numbers of metrics) comparison of drift detection methods up to date. This experiment is important to identify what is the best drift method in several conditions, and so use it as the drift detection method of the RCD framework. Following there are described the parametrization of the drift detectors, and the evaluation methodology used in the experiments.

Two data sets with abrupt concept drifts (Sine and Stagger), two data sets suffering from gradual concept drifts (Hyperplane and Mixed) and also four real-world data sets (Forest Covertype, Electricity, Poker Hand, and Weather) were used in the experiments.

To evaluate the presented data sets, it were used methodologies similar to the ones described at Elwell and Polikar (2011). For the artificial data sets, the methodology is the following: For each time step $T$, a specified amount of instances of the training set ($t$) are read and used to train the classifier. Then, another quantity of examples of the testing set ($d$) are used to test the classifier. This procedure is repeated 50 times, and a confidence interval is computed. The values of $t$ and $d$ are different for each data set. For each artificial data set, two configurations were tested to analyze the influence of (a) the number of training examples, in the data sets with abrupt concept drifts, and of (b)

the speed of change, in the data sets with gradual drifts.

For the real-world data sets, the methodology is the following: At each time step $T$, 30 instances are used for training and the next 30 instances are used for testing. In step $T + 1$, the values previously used for testing are now used for training and the following 30 instances are used for testing.

We used naive Bayes (NB) as base learner in all tested drift detection methods, because of its speed, simplicity, freely available implementations, and widespread use in experiments in the data stream research area. The experiments here performed used the Massive Online Analysis (MOA) framework. `SingleClassifierDrift` is a classifier implemented in MOA which uses a parameterized drift detector to classify incoming instances. It initially tests an incoming instance using the base learner and passes the result to the drift detection method, which will flag the example for no drift, warning level, or drift level. If no drift is identified, the base learner is trained on the instance just arrived. If the warning level is reached, a new learner is built and both are trained. If the drift level is reached, the learner built on instances obtained in the warning level is used as the new base learner. MOA already had implementations of DDM and EDDM as drift detection methods of `SingleClassifierDrift`, and these were used in the experiments.

The ADWIN method was also implemented in MOA, but it is mainly used in other classifiers. So, it was implemented to be used as a drift method of `SingleClassifierDrift`. As ADWIN does not have a warning level, the drift level was used as a warning level, signaling the drift level after one instance, allowing the new base learner to be trained on that instance before indicating drift level.

The ECDD method was completely implemented as a drift detection method to be used together with `SingleClassifierDrift`. It has three parameters: the average run length that informs the rate of false positive alarms per data point ($a$), how much weight is given to more recent data compared to older data ($\lambda$), and the warning threshold ($w$). The values used were the same presented by Ross et al. (2012): $a = 100$, $\lambda = 0.2$, and $w = 0.5$. To allow ECDD to correctly recompute its statistics when a drift is raised, it was implemented an approach taken by DDM and EDDM: after one drift, during the next 30 instances processed by ECDD, a new drift is not allowed to happen. This considerably reduced the amount of false alarms raised by ECDD, also increasing its predictive accuracy results.

PHT was also implemented in the MOA framework as a drift detection method of `SingleClassifierDrift`. The parameter values used are the same presented by Gama et al. (2009): allowed magnitude of change ($10^{-3}$) and $\lambda = 2.5$. The warning level was

obtained by experimentation and set to 2. The 30 instances approach described in the previous paragraph was also used.

Differently from the other algorithms that were built to identify concept drifts based on the results of a base learner's classification (true or false), PL identifies a drift when the difference in performance between its two learners (stable and reactive) are above a parameterized threshold on a window with the last tested instances. Thus, PL is not implemented using `SingleClassifierDrift`, but as a pure classifier. Several window length sizes ($w$) and thresholds ($\theta$) were tested at Bach and Maloof (2008), each one reflecting the best values for a specific data set. There chosen values are $w = 12$ and $\theta = 0.2$, one of the tested configurations of the aforementioned paper.

Notice that the last three methods, ECDD, PHT, and PL, were not available in MOA. So, complete implementations were provided and made available as extensions of the MOA framework. These, together with ADWIN as well as the Sine and Mixed data sets are freely available at `http://sites.google.com/site/moaextensions/`, including the instructions on how to use them.

## 7.1 EXPERIMENTAL RESULTS

This section presents the results of the experiments with the drift detectors in the used data sets concerning accuracy, evaluation time, false alarm and miss detection rates, as well as distance to the drift point.

### 7.1.1 Predictive Accuracy

The predictive accuracy of the drift detection methods were computed by the average normalized Area Under the Curve (AUC), with 95% confidence intervals using the trapezoid rule on adjacent pairs of accuracies, and are presented at Table 7.1. Best results are indicated in boldface.

Sine(1,000) is the Sine data set where each context contains 1,000 instances and, at each time step, one instance is used for training and 100 instances for testing. It is similar to the data set named SINIRREL1 described at Gama et al. (2004a) and its results are presented at Figure 7.1(a). None of the tested methods suffered a great decrease in accuracy because of the concept drift.

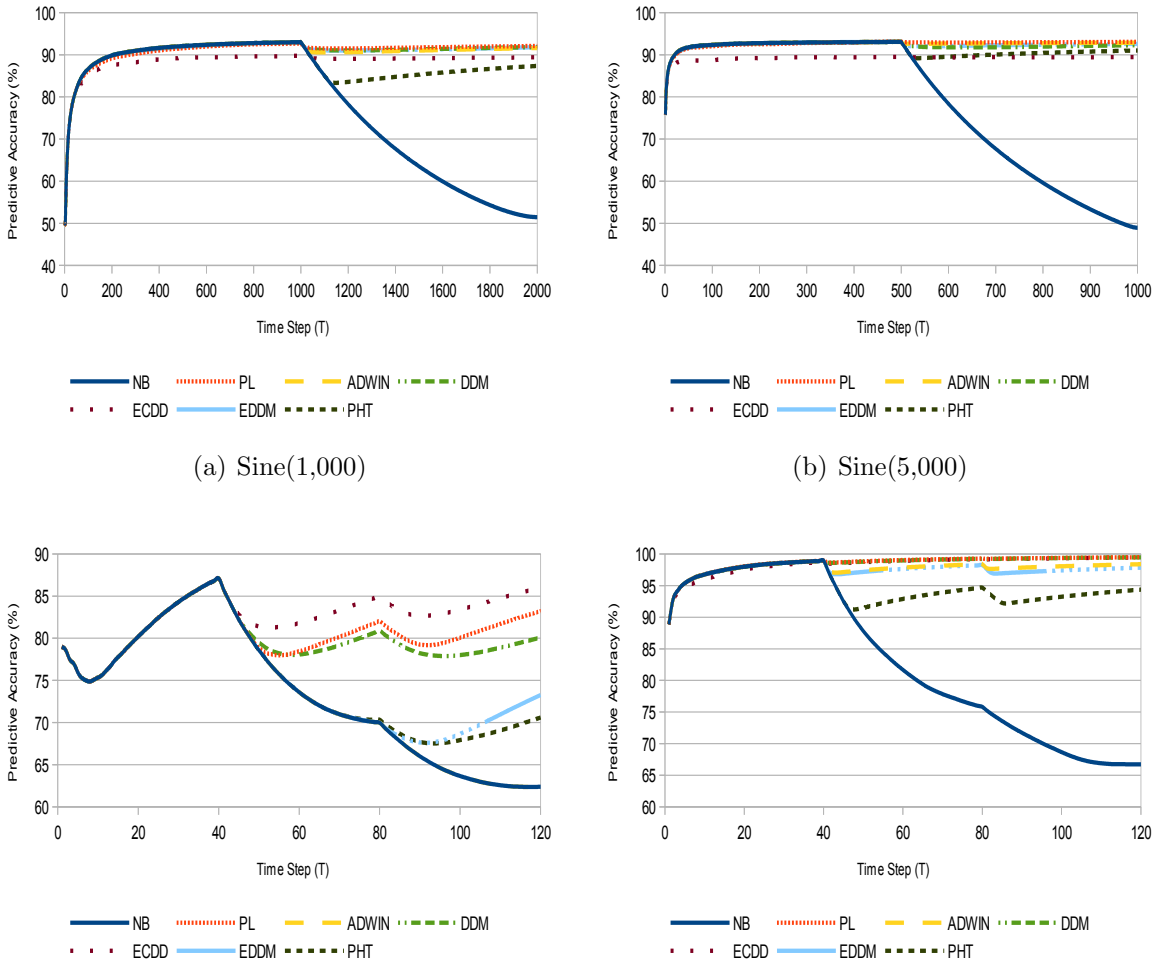
Sine(5,000) is similar to Sine(1,000), but each context is represented by 5,000 instances and at each time step 10 instances are used for training. It is based on the Sine200 data set presented at Ross et al. (2012), and the graphical curves are almost the same

**Table 7.1** Accuracy average normalized AUC with 95% confidence intervals.

| **Dataset** | NB | PL | ADWIN | DDM | ECDD | EDDM | PHT |
|---|---|---|---|---|---|---|---|
| Sine(1,000) | 78.32±0.67 | 90.87±0.16 | 90.74±0.15 | **90.90±0.15** | 88.49±0.14 | 90.82±0.15 | 88.11±0.19 |
| Sine(5,000) | 79.09±1.00 | **92.62±0.07** | 92.60±0.06 | 92.21±0.06 | 89.23±0.04 | 92.36±0.06 | 91.36±0.09 |
| Stagger(1) | 73.39±1.39 | 80.57±0.48 | 73.39±1.39 | 79.77±0.49 | **82.50±0.51** | 75.10±1.06 | 74.80±1.10 |
| Stagger(20) | 83.34±2.18 | **98.62±0.23** | 97.77±0.19 | 98.59±0.23 | 98.41±0.26 | 97.49±0.18 | 94.70±0.42 |
| Mixed(200) | 77.75±2.95 | 87.37±0.23 | **89.48±0.25** | 89.29±0.27 | 81.17±0.23 | 89.16±0.25 | 87.82±0.50 |
| Mixed(1,000) | 76.99±2.78 | 82.51±0.89 | 84.29±0.97 | 84.86±0.86 | 76.20±0.89 | 84.67±0.84 | **85.00±0.79** |
| Hyp(0.1) | 75.85±0.30 | 76.82±0.41 | 76.88±0.42 | 77.35±0.44 | 68.67±0.24 | **79.67±0.58** | 77.14±0.41 |
| Hyp(0.001) | 89.03±0.26 | 80.15±0.09 | 89.03±0.26 | **89.06±0.26** | 68.12±0.05 | 88.68±0.29 | 88.98±0.29 |
| Covertype | 66.92±1.12 | 70.87±0.61 | **74.01±0.46** | 71.02±0.72 | 69.05±0.63 | 69.81±0.69 | 73.91±0.45 |
| Electricity | 75.40±0.51 | 74.87±0.28 | **77.94±0.28** | 76.56±0.34 | 66.62±0.36 | 74.73±0.25 | 77.13±0.34 |
| Poker | 58.83±0.18 | **72.81±0.17** | 71.31±0.19 | 63.80±0.80 | 68.28±0.14 | 70.89±0.12 | 69.90±0.16 |
| Weather | 69.53±0.14 | 71.27±0.13 | 70.40±0.20 | 70.90±0.15 | 69.01±0.13 | **71.47±0.07** | 69.91±0.20 |

as the ones presented by Sine(1,000). Results are presented at Figure 7.1(b). All the methods slightly increased their performances as more data were used for training. For example, ADWIN and PL were the methods that most enhanced their performances with the increase of the training set, by 2.04% and 1.93%, respectively. In both Sine data set configurations, ECDD presented the stablest accuracy results, as can be observed by the confidence interval.

The first tested Stagger configuration, here named Stagger(1), was already used in previous experiments by Schlimmer and Granger (1986); Kolter and Maloof (2007); Narasimhamurthy and Kuncheva (2007) and Bach and Maloof (2008). The three contexts are represented by 40 instances. At each time step, one instance is used for training and 100 are used for testing. Figure 7.1(c) presents the accuracy results for this data set. It is worth pointing out that, around time steps 40 and 80, when the concept drifts occur, the performances of all the algorithms were reduced, until their base learners were changed to adapt to the new context. Also notice that ECDD had the best statistical results, specially after the first concept drift. PL was second best and had the same results of ECDD in the first context, starting to diverge right after the first drift. It was also the stablest method in the data set. Only in the third context EDDM differed from the base learner performance, while ADWIN performed exactly the same as naive Bayes, not identifying any of the two concept drifts. This occurred because, in the tested configuration, ADWIN only computes its two internal windows every 32 instances. Thus, ADWIN tries to identify a drift slightly before the drift occurs, when it has stored instances of a stable concept,




(a) Sine(1,000)

(b) Sine(5,000)

(c) Stagger(1)

(d) Stagger(20)

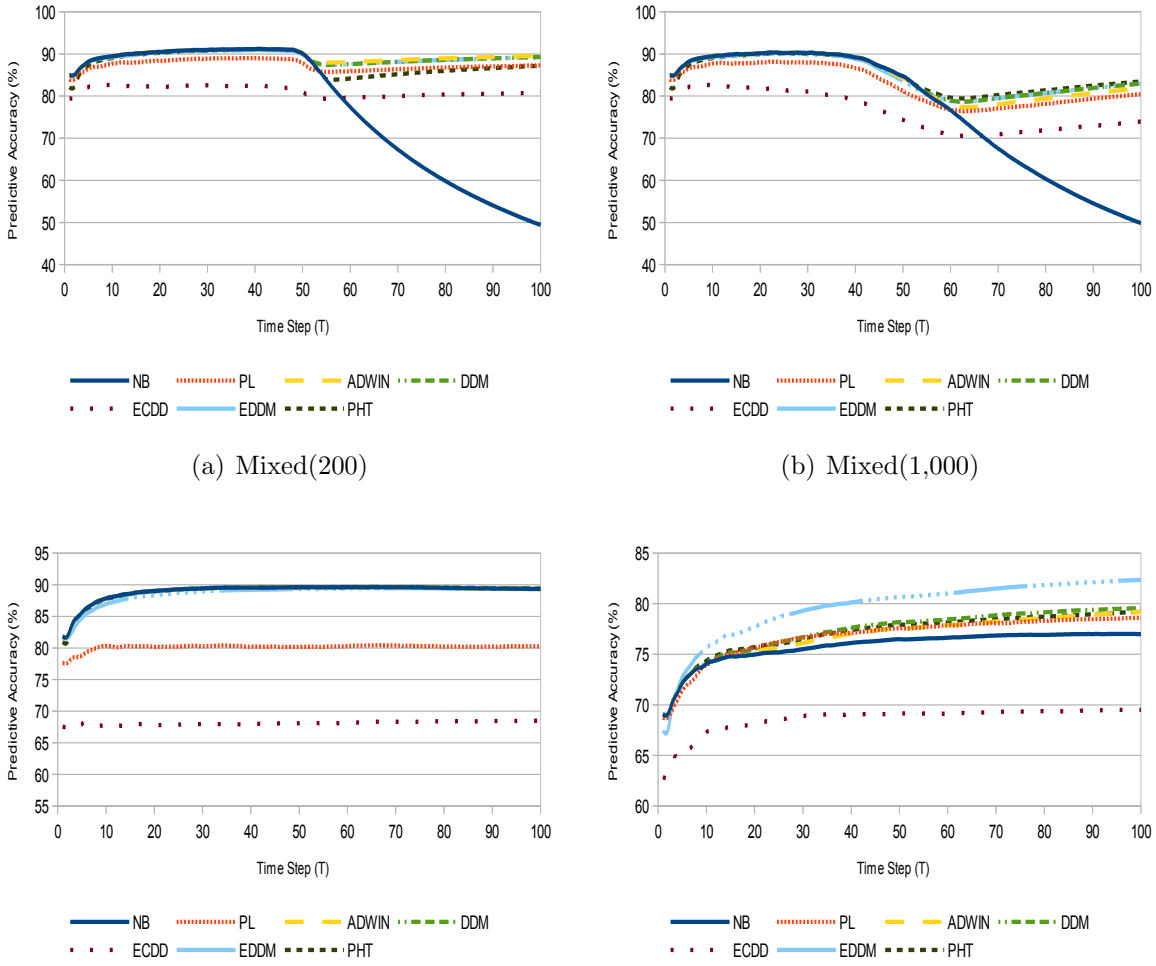
**Figure 7.1** Accuracies in the abrupt data sets.

making the detection difficult.

Figure 7.1(d) presents a modified configuration of Stagger, increasing the number of instances to be used for training at each time step from 1 to 20. Now, all the algorithms diverge from the base learner much earlier. Increasing the amount of training instances had a positive influence in the evaluation of the testing instances, similarly to what happened in the Sine data set configurations. ADWIN (by 33.22%) and EDDM (by 29.81%) were the methods that most benefited from the increase in the size of the training set. They were also the stablest methods in this data set.

To analyze how the algorithms perform in the presence of gradual drifts, two configurations of the Mixed data set were evaluated with different widths of change: 200, a

faster gradual configuration, and 1,000, a slower one. For this data set, each context is composed of 2,000 instances with 40 instances being used for training and other 40 used for testing. In the results of the Mixed(200) data set, presented at Figure 7.2(a), the drift takes 200 instances to completely change the context, comprised between time steps 47 and 53.




(a) Mixed(200)

(b) Mixed(1,000)

(c) Hyperplane(0.001)

(d) Hyperplane(0.1)

**Figure 7.2** Accuracies in the gradual data sets.

The Mixed(1,000) data set is similar to the Sine1g configuration used at Baena-García et al. (2006). The results of Figure 7.2(b) show that the performance of the detectors begin to drop around $T = 37$, as the new context starts to appear. After $T = 62$, they increase their performances, except for NB, which continues dropping its performance. Thus, all algorithms decrease their performances when the concept starts to drift but

recover slightly before the new context is completely present. When compared to the previous configuration, all the algorithms had lower accuracy results when the width of change increased. ADWIN was the method most affected by the slow changing gradual data set: it was 5.82% less precise. In this configuration, DDM presented the best accuracy statistical results, followed by EDDM.

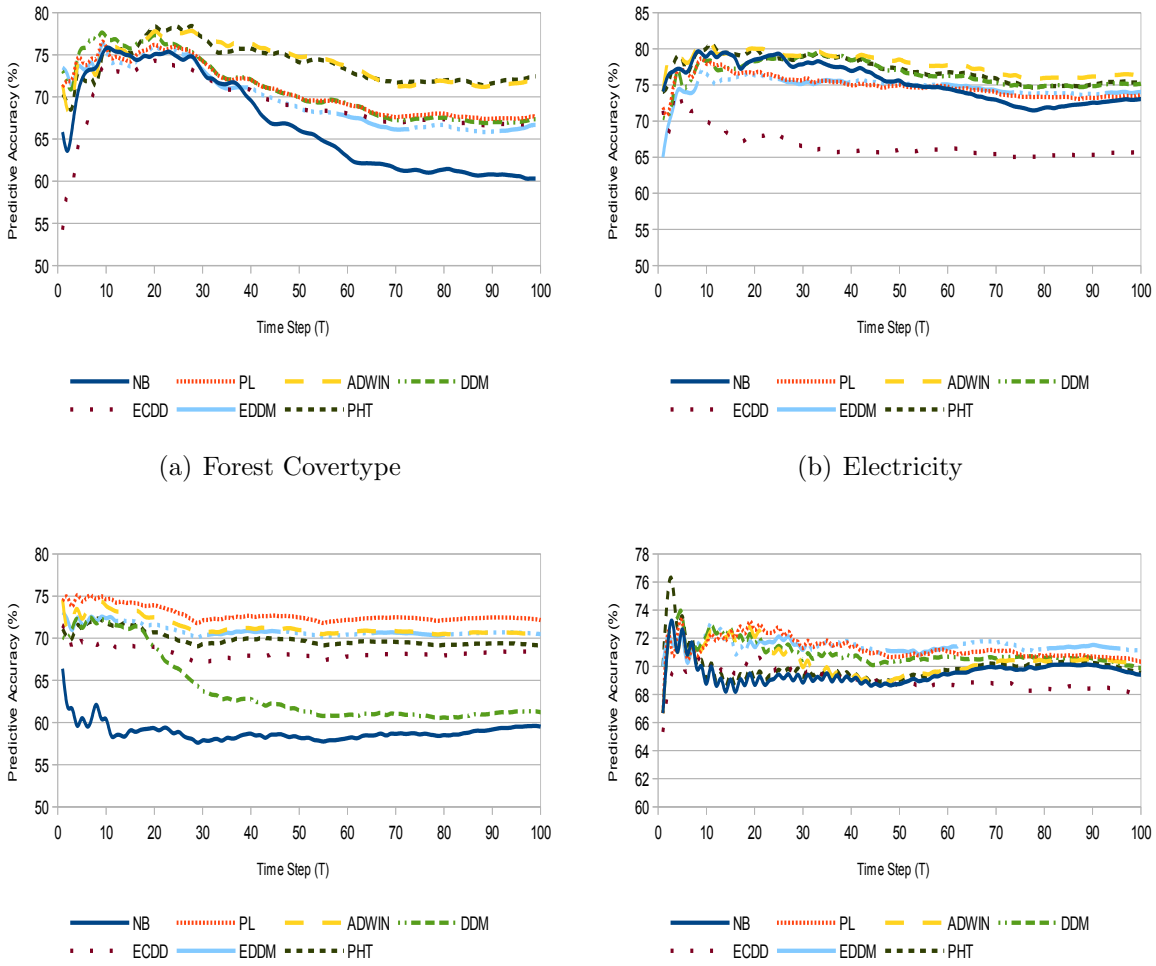
Differently from the Mixed data set, where there are two stable periods, i.e. before and after the gradual drift, the Hyperplane data set is in a continuous state of drift. Also, two versions of Hyperplane were tested with different speeds of change, based on the magnitude of the change for every example: 0.001 (slower change) and 0.1 (faster change), during 100,000 instances, and using 100 instances for both training and testing. Figures 7.2(c) and 7.2(d) present their results, respectively.

The Hyp(0.001) is similar to that used at Bifet et al. (2010b). All the methods performed better in the slower drift version of this data set, except for ECDD, which had a slightly worse accuracy. In this slow changing configuration, DDM was the best method, followed by NB, ADWIN, and EDDM; but the differences in performance between these four algorithms were very subtle, of less than one percentage point. The drift being so slow makes it almost resemble a single context data set, explaining why the base learner had a comparatively good performance.

ECDD was the method with the lowest accuracies in all datasets suffering from gradual concept drifts. This is in line with Ross et al. (2012), when it says that ECDD "is primarily intended to be used to detect abrupt change". Reducing the value of the warning level should improve ECDD in these datasets. Similarly, PL was the second worse method in the gradual drifts datasets, due to the window length used. Increasing it would make it better suited to these datasets.

Analyzing the results of the real-world data sets, presented at Figure 7.3, ADWIN was the best algorithm in Forest Covertype and in the Electricity data sets, and also second best in Poker Hand. PL was the best algorithm in Poker Hand and second best in Weather. EDDM was the algorithm with the highest accuracy in the Weather data set, and the stablest one at Electricity, Poker Hand, and Weather data sets.

To statistically compare the methods in all datasets, we computed the $F_F$ statistic (Iman and Davenport, 1980), derived from the Friedman non-parametric statistical test (Friedman, 1940) as described by Demšar (2006). The $F_F$ statistic indicated that some methods are significantly better than others. To identify which ones are better, we performed the Nemenyi test Nemenyi (1963) with 95% confidence. Results indicated that DDM, ADWIN, and PL are significantly better than NB, and DDM is also better than




(a) Forest Covertype

(b) Electricity

(c) Poker Hand

(d) Weather

**Figure 7.3** Accuracies in the real-world data sets.

ECDD. The Holm procedure (Holm, 1979) is a much more confident test that controls the family-wise error when comparing all methods with NB. Its results indicate that all methods, except for ECDD, are significantly better than NB.

### 7.1.2 Evaluation Time

The average evaluation times are presented at Table 7.2, together with the 95% confidence interval for the artificial data sets. The values presented are in CPU seconds, meaning that they are the times taken by the methods inside the CPU to process the data sets. The best results in performance are highlighted in boldface.

PL had the highest average evaluation times in practically all tested datasets, mainly

**Table 7.2** Evaluation time with 95% confidence intervals.

| **Dataset** | NB | PL | ADWIN | DDM | ECDD | EDDM | PHT |
|---|---|---|---|---|---|---|---|
| Sine(1,000) | 4.09±0.55 | 6.29±0.58 | 5.79±0.59 | 5.74±0.66 | 6.55±0.50 | **5.43±0.57** | 5.81±0.56 |
| Sine(5,000) | 1.21±0.09 | 2.37±0.11 | 2.02±0.10 | 1.97±0.10 | 2.29±0.10 | **1.95±0.10** | 2.18±0.17 |
| Stagger(1) | 0.09±0.01 | 0.22±0.01 | 0.16±0.00 | 0.15±0.00 | 0.19±0.01 | **0.15±0.00** | 0.21±0.01 |
| Stagger(20) | 0.09±0.00 | 0.21±0.00 | 0.17±0.00 | **0.15±0.00** | 0.18±0.00 | 0.15±0.00 | 0.21±0.01 |
| Mixed(200) | 0.08±0.00 | 0.22±0.00 | 0.15±0.00 | **0.14±0.00** | 0.18±0.00 | 0.14±0.00 | 0.18±0.01 |
| Mixed(1,000) | 0.08±0.00 | 0.22±0.00 | 0.15±0.00 | **0.14±0.00** | 0.18±0.00 | 0.14±0.00 | 0.18±0.01 |
| Hyp(0.1) | 0.18±0.23 | 0.66±0.83 | **0.32±0.41** | 0.33±0.43 | 0.37±0.47 | 0.33±0.43 | 0.37±0.57 |
| Hyp(0.001) | 0.18±0.23 | 0.65±0.83 | 0.32±0.41 | **0.31±0.43** | 0.37±0.46 | 0.34±0.43 | 0.37±0.46 |
| Covertype | 29.86 | 69.95 | 33.88 | 30.31 | **27.03** | 29.53 | 33.85 |
| Electricity | 0.51 | 1.58 | 0.80 | 0.76 | 0.89 | **0.75** | 0.76 |
| Poker Hand | 15.58 | 30.17 | 14.38 | 24.82 | 13.45 | **11.73** | 13.42 |
| Weather | 0.30 | 1.12 | 0.45 | 0.45 | 0.51 | **0.42** | **0.42** |

because, for each arriving instance, it must create a new learner trained on the last $w$ instances. If the retractable version were used, the evaluation times might be reduced, but at the cost of not being general, as not all online methods have a retractable version. The $F_F$ statistic computed also indicated differences in performance. The Nemenyi test indicated that NB was significantly better than PL, ECDD, and PHT; EDDM was better than PL and ECDD; and DDM and ADWIN were better than PL. The Holm procedure informed that NB was significantly faster than ADWIN, PL, ECDD, and PHT.

### 7.1.3 Drift identification

Table 7.3 presents the mean distance until drift detection ($\mu$ DT), its standard deviation ($\sigma$ DT), the false alarm rates (FA), and the miss detection rates (MD) for the abrupt drifts data sets. In these data sets it is previously known exactly where the concept changes occur and, thus, it is possible to use this information to check which method more closely identified the correct drift position as well as had the best false alarm and miss detection rates. The drift points are $T = 1001$ and $T = 501$ for Sine(1,000) and Sine(5,000), respectively, and $T = 41$ and $T = 81$ for both Stagger configurations. For the ADWIN method, the results present drift points in the position raised by the original method, correcting the one instance shift implemented to allow ADWIN to work with `SingleClassifierDrift`.

In the Sine(1,000) and Stagger(1) data sets, the methods train on a single instance before performing the testing phase. Thus, it is quite difficult for any method to detect

**Table 7.3** Mean distance after drifts, false alarm rates and miss detection rates in the abrupt datasets.

| Dataset | PL | | | | ADWIN | | | | DDM | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\mu$ DT | $\sigma$ DT | FA | MD | $\mu$ DT | $\sigma$ DT | FA | MD | $\mu$ DT | $\sigma$ DT | FA | MD |
| **Sine(1,000)** | 15.62 | 52.66 | 0.00 | 0.28 | 23.00 | 0.00 | 0.00 | 1.00 | 17.00 | 5.10 | 0.00 | 0.98 |
| **Sine(5,000)** | 0.18 | 0.39 | 0.02 | 0.18 | 2.00 | 0.00 | 0.00 | 1.00 | 2.90 | 1.33 | 0.00 | 0.92 |
| **Stagger(1)** | 11.64 | 5.67 | 0.01 | 0.65 | 40.00 | 0.00 | 0.00 | 1.00 | 18.73 | 14.20 | 0.01 | 0.66 |
| **Stagger(20)** | 0.02 | 0.14 | 0.00 | 0.01 | 1.04 | 0.28 | 0.02 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Dataset** | ECDD | | | | EDDM | | | | PHT | | | |
| | $\mu$ DT | $\sigma$ DT | FA | MD | $\mu$ DT | $\sigma$ DT | FA | MD | $\mu$ DT | $\sigma$ DT | FA | MD |
| **Sine(1,000)** | 19.68 | 74.44 | 0.01 | 0.16 | 15.28 | 8.51 | 0.00 | 0.74 | 123.34 | 11.23 | 0.00 | 1.00 |
| **Sine(5,000)** | 1.40 | 5.95 | 0.06 | 0.12 | 4.02 | 4.05 | 0.01 | 0.80 | 23.52 | 2.92 | 0.00 | 1.00 |
| **Stagger(1)** | 4.97 | 5.01 | 0.00 | 0.09 | 26.99 | 15.38 | 0.00 | 0.78 | 32.53 | 12.14 | 0.01 | 0.92 |
| **Stagger(20)** | 0.00 | 0.00 | 0.00 | 0.00 | 1.62 | 0.72 | 0.02 | 0.92 | 5.73 | 1.16 | 0.02 | 1.00 |

a drift based on only one instance because they usually need more information after the drift before detecting the context change.

It is possible to confirm this statement by analyzing the mean distance taken to identify a concept drift. In both Sine(1,000) and Stagger(1), the mean distances are much higher than in the configurations where the methods are trained on more instances before testing. Lower distances mean that the methods identified the drift closer to the real change point. PL presented good results, with the lowest mean distance in Sine(5,000), being the second best in Sine(1,000) and Stagger(1), and third best in Stagger(20). ECDD was the best method in both Stagger configurations while EDDM was the best in Sine(1,000).

The standard deviation allows us to analyze the stability of the methods regarding the detected drift position. Lower values mean that, in the 50 repetitions, the method identified the drift in close positions. Here, ADWIN clearly outperformed the other methods. In both versions of Sine and in Stagger(1), all the repetitions identified the drift point in exactly the same time step, resulting in $\sigma = 0$. DDM was the second best method in this criterion. On the other hand, ECDD had, on average, much higher values.

Training on single instances raises a question: how should the miss detection rates be computed? To expect that the methods could correctly identify drifts based on one instance training of a new context is implausible: the miss detection rate can easily reach 100%. So, to avoid this problem, in the Sine(1,000) and Stagger(1) data sets, if the

algorithm identifies a drift in the first 10 instances following the correct drift point, *no* miss detection is computed. In the other abrupt data sets, as they train on at least 10 instances, the miss detection rate *is* computed if the drift identification does not happen in the exact time step where it occurred.

Figure 7.4 present the average number of detected concept drifts per time step for the data sets suffering from abrupt concept drifts. For the Sine data set, the methods identified the drift point slightly after half the data set, indicating that they signaled in the correct position. Analyzing the results, the PL and ECDD methods signaled for a drift at least once in the first context.

Observing the figure relative to Stagger(1), we can confirm that ECDD had average lower distances to the drift points, followed by PL. Before the first drift point, only DDM and ECDD presented false alarms, but only in one of the 50 repetitions.

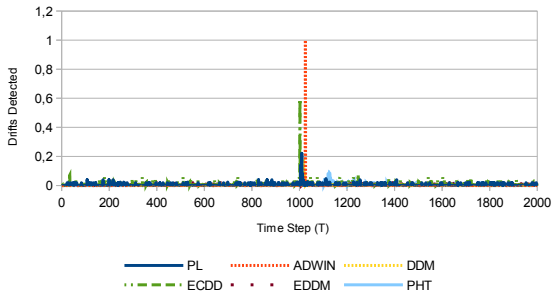
Observing Figure 7.4(d), it is possible to notice that, right after time steps 40 and 80, there is an increase in the number of detected concept drifts, indicating that the methods generally identify the drift at the right position. In this data set, only PL and ECDD signaled for drift in the first stable context. Training on 20 instances instead of one, the methods have much more information about the contexts and, as a consequence, they identify the drift points much closer, as can be observed comparing Figures 7.4(c) and 7.4(d).

In the Mixed data set configurations, presented at Figures 7.4(e) and 7.4(f), the methods usually identified the drifts in the correct positions, between $T = 47$ and $T = 52$ (for Mixed(200)) and $T = 37$ and $T = 62$ (for Mixed(1,000)), but it is observed that both PL and ECDD raised many more drifts in the stable contexts compared to ADWIN, DDM, and EDDM. Similar behavior occurs in the Hyperplane data sets. This behavior may indicate that PL and ECDD do not handle gradual drift data sets well. In fact, examining their predictive accuracies, these methods did not present the best results in any data set suffering from gradual concept drifts.

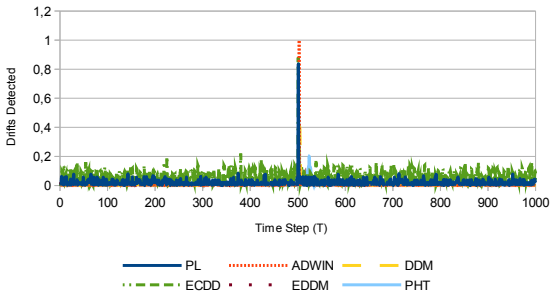
## 7.2 CONCLUSION

In this chapter several concept drift detectors were compared and their behavior were analyzed in the presence of abrupt and gradual concept drifts, with different speeds of change, in artificial and real-world data sets, with noise and irrelevant attributes.

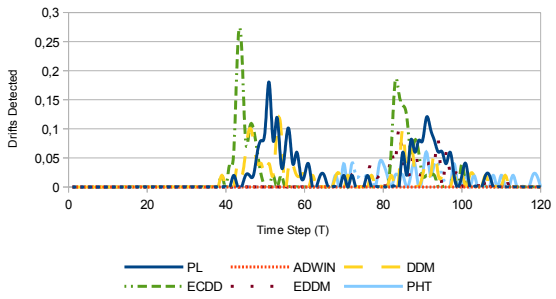
Three different drift detection methods, PHT, ECDD and Paired Learners, and two artificial data sets, Sine and Mixed, were implemented in the MOA framework and made freely available to the public, allowing further comparison by other researchers.



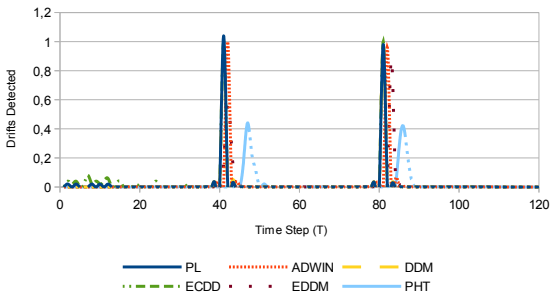

(a) Sine(1,000)



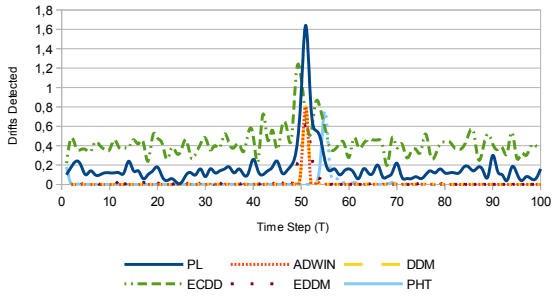

(b) Sine(5,000)



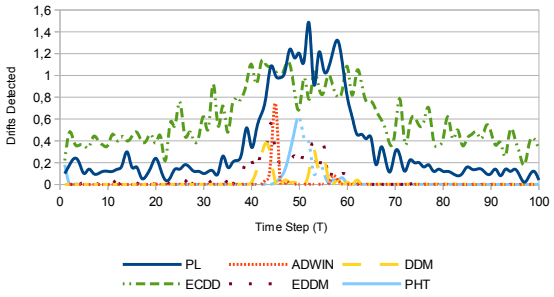

(c) Stagger(1)



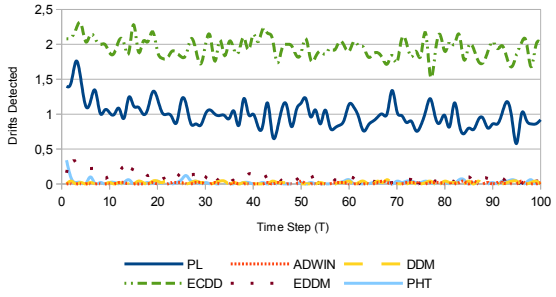

(d) Stagger(20)



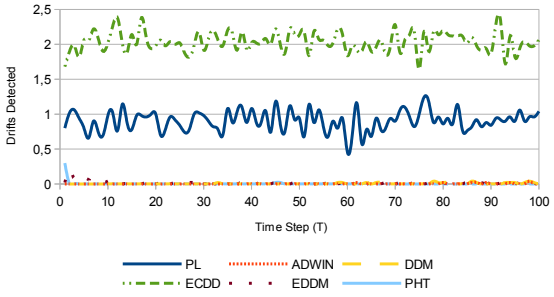

(e) Mixed(200)




(f) Mixed(1,000)




(g) Hyperplane(0.1)




(h) Hyperplane(0.001)

**Figure 7.4** Average number of detected concept drifts in the artificial data sets.

Experiments indicated that PL was the method that presented the best average accuracy results in data sets with abrupt concept drifts, followed by DDM, ECDD, and EDDM. DDM was the best in data sets suffering from gradual concept drifts, followed by EDDM, ADWIN and PHT. In the real-world data sets, ADWIN was the best in Covertype and Electricity, and second best in Poker Hand.

For its turn, ECDD was the stablest method in seven out of eight artificial data sets, while EDDM was the stablest in three out of four real-world data sets.

Considering the evaluation time, EDDM was the fastest drift detection method in 6 out of 12 tested data sets, being even faster than the base learner in some real-world ones. On the other hand, PL was the slowest in 10 out of 12 data sets and ECDD was the second slowest.

In the abrupt drifts data sets, the algorithm that returned the lowest number of false alarms was DDM, closely followed by ADWIN. ECDD presented a highly unstable behavior, practically indicating a context change at each time step, while ADWIN had the stablest behavior. PL was the best method at identifying the drift position close to the real drift points.

The experiments reported in this chapter are a first step towards identifying which drift detection method is best suited to solve different kinds of problems. Other kinds of tests are possible, like using other drift detection methods, like the Fixed Cumulative Windows Model (FCWM) (Sebastião et al., 2010), using other data set configurations and methods parametrization, or testing the drift detection methods with other base learners.

CHAPTER 8

# CONCLUSION

This chapter summarizes the contributions of the thesis and gives directions for future work.

Dealing with concept drifts is a challenging topic. Several approaches have been proposed for modifying traditional classifiers to adapt to concept drifts, to identify when a drift has occurred, and to use ensemble classifiers with and without diversity to better manage it. The difficulty in managing drifts increases especially when algorithms have to deal with several concept drifts and they recur in time. Up to date, algorithms that deal with concept drifts give higher importance to currently seen data, gradually forgetting older examples. This can be viewed as a waste of information because classifiers are trained in data with information that is not represented by the models. Models fit one context instead of managing possible recurring contexts.

In this thesis we presented RCD, a framework to deal with recurring concept drifts. It uses a multivariate non-parametric statistical test to compare data samples and identify if new or old concepts are occurring and reusing previously generated classifiers if a similarly seen concept has occurred.

The proposed framework is highly configurable, allowing the user to select the statistical test, the confidence level, the amount of instances to represent each distribution, the rate of tests to be performed, the drift detection method to be used, the number of classifiers to store, and the number of concurrent tests. The work also gives experimental support for controlling all parameter settings.

In the first experiment, presented at Chapter 4, RCD was compared not only to seven different algorithms among single approaches like Hoeffding trees with naive Bayes and Early Drift Detection Method, but also to ensemble classifiers like Accuracy Updated Ensemble, Weighted Majority Algorithm, Learn++.NSE, Ensemble Building, and Dynamic Weighted Majority. To perform the comparison we selected five artificial (each one with abrupt and gradual concept drift versions) and four real-world data sets among the most used in the concept drift research area. In this experiment, the statistical test used was KNN.

The evaluation methodology used was similar to what is usually performed in the

data mining research area. Instead of using Interleaved Test-Then-Train (Prequential), which uses all instances for both testing and training, a holdout method composed of 30 training and testing sets with different contexts were created. After training the classifier, testing sets representing each context were used to analyze how the classifiers represented each of the contexts that they were trained on.

The metrics used were the predictive accuracy, its standard deviation, and the training and testing times. It was observed that in the data sets suffering from abrupt concept drifts, the classifiers accuracies were higher in the last trained contexts. This means that, as contexts change, the classifiers really forget the first contexts and try to adapt to the new ones. RCD, on its turn, maintains a stabler behavior, usually being the classifier that best represents the first seen contexts.

In the data sets suffering from gradual concept drifts the accuracies decreased in time, presenting lower values in the last observed contexts. This indicated that the classifiers had difficulty in adapting to the new contexts, presenting a harder problem to solve.

Considering the train and test times, DWM was the slowest one, followed by EB, AUE, and RCD. RCD is the only classifier that spends more time in the testing phase than in the training phase, due to statistical tests performed in the testing phase to select the best trained classifier in current data.

To statistically compare the eight tested classifiers over multiple data sets, a modified version of the Friedman non-parametric test was used. In none of the tested data sets RCD presented a statistical lower performance compared to all the other classifiers. RCD presented performance statistically similar to the best classifiers in the abrupt, gradual, and real-world data sets. RCD also presented the lowest standard deviation in three data set configurations, second best in other three data sets. This result associated with low average rank, indicate that RCD was able to select stored classifier suited to actual distributions after drift detection.

Another experiment, presented at Chapter 5, was performed to analyze the influence of the statistical test in accuracy and evaluation time. Two tests were compared: KNN and Cramer. RCD was tested using five different base learners: two versions of a Hoeffding tree (on-line classifiers), naive Bayes, and two traditional batch classifiers (Multilayer Perceptron and J48). Tests were performed in four artificial and three real-world data sets. KNN presented a much faster evaluation time as the sample sizes used to perform the tests increased. Thus, an RCD version using a buffer with only 100 instances was used. To reduce the influence of the testing phase, the test frequency was set to be performed each 500 instances. Results have shown that RCD improved the base learners accuracy

in several data set configurations considering a 95% confidence interval, specially in data sets with several concept drifts. KNN was more sensitive, identifying more concept drifts than the Cramer test. The most interesting conclusion was that there was no situation where the base learner was statistically superior to the use of RCD, independently of the statistical test used. This occurred even using a very small buffer size, with little impact on RCD accuracy. In the tests performed in Chapter 4, a buffer size and test frequency of 400 instances were used. Thus, results in this chapter indicate that even with fewer instances to compare distributions and making tests more sparsely in the testing phase, RCD still had a good performance compared to the other tested classifiers.

Considering the slow speed of the Cramer test, improvements in how RCD performs the statistical tests were considered to increase performance. A general solution, independent of the test performed, is to execute the statistical tests in parallel by the use of a thread pool of configurable size. An experiment performing the statistical tests in parallel was executed and was presented at Chapter 6. Tests were performed using sequential execution and parallelism with a thread pool of two and four cores in three computer systems with processors with one, two, and four cores. Results indicated that best results were obtained when the thread pool size was equal to the number of cores in the processor. Thus, overloading the processor reduced performance. In environments with many detected concept drifts, when the buffer size is big, and with a high frequency of tests, parallelism gives the best results, outperforming sequential execution. Besides that, bigger buffers (i.e., more instances in the statistical tests) are better handled in processors with higher processor clock frequency. Also, the influence of the buffer size in performance is bigger than test frequency. Increasing the buffer size by five times, increased the evaluation time in more than 17 times, while multiplying the test frequency by five times increased the evaluation time by more than eight times. The best result was obtained by the processor which balanced core size and processor frequency.

Finally, at Chapter 7, six different drift detection methods were compared to analyze their performances concerning accuracy, evaluation time, false alarm rates, miss detection rates, and distance from drift point. As expected, each method is better suited to a different metric in data sets suffering from concept drifts with different characteristics. Paired Learners had higher predictive accuracies in abrupt concept drifting data sets, similarly to the findings of the proposed paper (Bach and Maloof, 2008). On the other hand, DDM was the best in the gradual data sets. This result was different than the published in the EDDM paper (Baena-García et al., 2006), which indicated it to have better results than DDM in slow gradual concept drifting data sets. In the real-world

data sets, ADWIN presented the best results.

Concerning accuracy stability, ECDD was the stablest in the artificial data sets, while EDDM was the stablest in the real-world ones. EDDM was the fastest method, concerning the evaluation time. DDM had the lowest rate of false alarms while ECDD had the lowest rate of miss detection. Paired Learners identified changes closer to the correct drift point. ADWIN, on its turn, was the stablest. In three data sets, ADWIN identified the drift point in exactly the same position, in all 30 repetitions.

## 8.1 CONTRIBUTIONS

The main contributions of this thesis are:

- The use of non-parametric multivariate statistical tests to identify previously used classifiers that were trained on data similar to the most recent distribution.
- The use of a general approach to identify similarity between data samples based on the comparison of data distributions. It is important to notice that the user can parameterize the confidence level, allowing variable identification of data samples.
- RCD allows, in the testing phase, to dynamically select the best classifier on current data.

Other minor contributions are:

- RCD was made freely available on-line as an extension of the MOA framework. It can be obtained at the following address: `http://sites.google.com/site/moaextensions/`.
- Ensemble classifiers (DWM and EB), drift detection methods (Paired Learners, PHT, and ECDD), and artificial data sets (Sine and Mixed) were implemented and made freely available to the public to be used integrated with the MOA framework, contributing to the analysis and comparison of new techniques in the field of data streams with concept drifts.

## 8.2 FUTURE WORK

As previously described, RCD is highly configurable. Even so, as future work, much research might be made to allow the user to choose the best configuration for a specific problem.

One possible improvement would be to create an ensemble classifier based on the individual classifiers associated with each distribution. As more than one stored classifier can match the current context, one approach would be to set higher weights to classifiers where their samples return higher confidence levels.

Another direction could be to test other possibilities in the selection of the best stored classifier, like performing the statistical tests for all stored buffers and select the one with the highest significance value.

Other possibilities could be:

- To implement a pruning strategy different than FIFO and analyze the influence of the number of stored classifiers in terms of accuracy and performance.
- To use the two available statistical tests in conjunction and analyze if they provide better results than their individual use. If the statistical tests are complementary, each one would better represent different distributions, making the combined output achieve better results. This is a technique commonly used in ensemble classifiers.
- To use other representations for the categorical attributes. Different representations can yield better results for different data sets.
- To create an ensemble of drift detection methods and test forms to combine their results to better indicate when a concept drift has really occurred.

APPENDIX A

# RESULT TABLES FOR THE PARALLEL RCD

Here, it is presented the results for the two and four-core processors, described at Chapter 6.

## A.1 RESULTS FOR THE TWO CORE COMPUTER SYSTEM

**Table A.1** Results for a buffer with 100 instances and test frequency of 500 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 56.85 | 25.24 | 20.44 | 56.91 | 25.25 | 20.55 | 56.86 | 25.21 | 20.49 |
| **HYP(10,0.0001)** | 56.87 | 25.26 | 20.46 | 56.79 | 25.20 | 20.43 | 56.83 | 25.18 | 20.47 |
| **SEA(50)** | 2.35 | 0.89 | 0.75 | 2.35 | 0.90 | 0.74 | 2.36 | 0.90 | 0.74 |
| **SEA(50000)** | 2.53 | 0.90 | 0.75 | 2.53 | 0.90 | 0.75 | 2.54 | 0.90 | 0.75 |
| **LED(50000)** | 13.61 | 5.21 | 4.44 | 13.62 | 5.20 | 4.44 | 13.57 | 5.18 | 4.42 |
| **Covertype** | 72.76 | 64.23 | 2.75 | 47.26 | 38.60 | 2.75 | 50.89 | 42.19 | 2.75 |
| **Poker Hand** | 28.41 | 25.13 | 1.26 | 19.88 | 16.58 | 1.25 | 21.08 | 17.75 | 1.26 |
| **Electricity** | 1.82 | 1.62 | 0.04 | 1.36 | 1.15 | 0.04 | 1.50 | 1.30 | 0.04 |

**Table A.2** Thread pool management for a buffer with 100 instances and test frequency of 500 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.34 | 0.53 | 0.82 | 1.92 | 1.08 | 1.49 | 0.03 | 0.01 | 0.01 |
| **HYP(10,0.0001)** | 0.30 | 0.41 | 0.77 | 1.91 | 1.85 | 1.30 | 0.04 | 0.00 | 0.02 |
| **SEA(50)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.00 | 0.00 | 0.25 | 0.75 | 0.75 | 2.75 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.67 | 0.00 | 0.67 | 3.33 | 2.67 | 2.00 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 1.49 | 1.60 | 4.31 | 22.61 | 12.03 | 10.79 | 0.05 | 0.06 | 0.07 |
| **Poker Hand** | 1.47 | 1.51 | 3.08 | 10.76 | 5.99 | 5.08 | 0.06 | 0.07 | 0.08 |
| **Electricity** | 1.30 | 1.43 | 2.49 | 5.86 | 3.43 | 3.08 | 0.06 | 0.06 | 0.07 |

**Table A.3** Results for a buffer and test frequency of 100 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 361.47 | 25.53 | 325.09 | 328.96 | 26.21 | 291.23 | 327.72 | 26.19 | 290.43 |
| **HYP(10,0.0001)** | 370.81 | 25.51 | 334.51 | 354.94 | 26.25 | 317.17 | 351.93 | 26.18 | 314.71 |
| **SEA(50)** | 3.92 | 0.93 | 2.30 | 3.97 | 0.95 | 2.32 | 4.02 | 0.94 | 2.35 |
| **SEA(50000)** | 4.18 | 0.93 | 2.38 | 4.21 | 0.94 | 2.38 | 4.12 | 0.94 | 2.29 |
| **LED(50000)** | 17.46 | 5.27 | 8.23 | 17.18 | 5.20 | 8.04 | 17.28 | 5.22 | 8.08 |
| **Covertype** | 464.05 | 69.97 | 387.58 | 253.06 | 41.70 | 205.43 | 262.79 | 45.75 | 211.08 |
| **Poker Hand** | 217.25 | 25.09 | 189.93 | 126.87 | 16.49 | 108.24 | 131.74 | 17.77 | 111.85 |
| **Electricity** | 8.37 | 1.63 | 6.58 | 5.07 | 1.16 | 3.76 | 5.47 | 1.31 | 4.00 |

**Table A.4** Thread pool management for a buffer and test frequency of 100 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.42 | 0.51 | 0.83 | 2.47 | 2.05 | 1.73 | 0.02 | 0.02 | 0.02 |
| **HYP(10,0.0001)** | 0.37 | 0.41 | 0.42 | 2.60 | 2.39 | 2.36 | 0.02 | 0.02 | 0.02 |
| **SEA(50)** | 0.03 | 0.03 | 0.03 | 0.11 | 0.11 | 0.12 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.03 | 0.03 | 0.03 | 0.12 | 0.12 | 0.11 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.03 | 0.04 | 0.04 | 0.33 | 0.30 | 0.30 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 1.48 | 1.60 | 7.70 | 65.89 | 34.17 | 29.53 | 0.07 | 0.06 | 0.08 |
| **Poker Hand** | 1.61 | 1.50 | 4.15 | 22.20 | 12.10 | 9.96 | 0.08 | 0.04 | 0.09 |
| **Electricity** | 1.52 | 1.49 | 3.42 | 14.70 | 8.03 | 6.79 | 0.07 | 0.07 | 0.09 |

**Table A.5** Results for a buffer and test frequency of 500 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 941.47 | 27.50 | 902.35 | 935.57 | 27.46 | 896.59 | 965.18 | 27.45 | 926.22 |
| **HYP(10,0.0001)** | 955.18 | 27.38 | 916.31 | 952.85 | 27.30 | 914.05 | 955.28 | 27.33 | 916.42 |
| **SEA(50)** | 6.33 | 1.05 | 4.34 | 6.32 | 1.05 | 4.32 | 6.27 | 1.05 | 4.28 |
| **SEA(50000)** | 6.53 | 1.03 | 4.39 | 6.47 | 1.02 | 4.34 | 6.47 | 1.04 | 4.33 |
| **LED(50000)** | 34.26 | 5.30 | 24.74 | 30.65 | 5.34 | 21.11 | 30.02 | 5.33 | 20.47 |
| **Covertype** | 1635.72 | 309.94 | 1319.49 | 1077.54 | 193.30 | 877.97 | 1127.29 | 219.49 | 901.44 |
| **Poker Hand** | 432.44 | 31.01 | 399.21 | 235.16 | 19.20 | 213.80 | 240.62 | 19.76 | 218.57 |
| **Electricity** | 23.45 | 5.96 | 17.31 | 12.98 | 3.56 | 9.24 | 13.27 | 3.78 | 9.30 |

**Table A.6** Thread pool management for a buffer and test frequency of 500 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.36 | 0.51 | 1.05 | 43.87 | 43.46 | 44.40 | 0.02 | 0.02 | 0.02 |
| **HYP(10,0.0001)** | 0.32 | 0.39 | 0.53 | 44.65 | 44.47 | 44.44 | 0.02 | 0.02 | 0.02 |
| **SEA(50)** | 0.03 | 0.02 | 0.03 | 1.71 | 1.70 | 1.68 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.03 | 0.03 | 0.03 | 1.73 | 1.71 | 1.70 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.03 | 0.04 | 0.04 | 10.12 | 8.30 | 7.98 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 1.56 | 1.73 | 11.46 | 520.68 | 339.68 | 347.38 | 0.05 | 0.06 | 0.08 |
| **Poker Hand** | 1.43 | 1.54 | 15.57 | 234.85 | 124.29 | 113.19 | 0.06 | 0.08 | 0.10 |
| **Electricity** | 1.46 | 1.43 | 6.74 | 99.44 | 53.63 | 49.56 | 0.07 | 0.08 | 0.10 |

## A.2 RESULTS FOR THE FOUR CORE COMPUTER SYSTEM

**Table A.7** Results for a buffer with 100 instances and test frequency of 500 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 54.00 | 26.16 | 21.27 | 53.87 | 26.10 | 21.22 | 53.92 | 26.13 | 21.23 |
| **HYP(10,0.0001)** | 53.90 | 26.10 | 21.23 | 53.97 | 26.11 | 21.28 | 53.92 | 26.09 | 21.24 |
| **SEA(50)** | 2.26 | 0.94 | 0.78 | 2.25 | 0.94 | 0.78 | 2.24 | 0.94 | 0.78 |
| **SEA(50000)** | 2.46 | 0.94 | 0.77 | 2.45 | 0.94 | 0.78 | 2.45 | 0.94 | 0.78 |
| **LED(50000)** | 12.53 | 5.74 | 4.82 | 12.52 | 5.74 | 4.81 | 12.51 | 5.73 | 4.81 |
| **Covertype** | 82.71 | 73.27 | 2.88 | 58.16 | 49.03 | 2.80 | 53.77 | 42.68 | 3.01 |
| **Poker Hand** | 35.80 | 32.38 | 1.25 | 25.51 | 22.11 | 1.25 | 22.47 | 18.42 | 1.51 |
| **Electricity** | 2.60 | 2.38 | 0.04 | 1.80 | 1.59 | 0.04 | 1.86 | 1.57 | 0.06 |

**Table A.8** Thread pool management for a buffer with 100 instances and test frequency of 500 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.70 | 0.42 | 0.45 | 3.64 | 3.46 | 3.68 | 0.00 | 0.00 | 0.00 |
| **HYP(10,0.0001)** | 0.54 | 0.42 | 0.41 | 3.42 | 3.57 | 3.52 | 0.01 | 0.00 | 0.01 |
| **SEA(50)** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.50 | 0.00 | 0.25 | 2.25 | 2.25 | 2.25 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.67 | 0.33 | 0.33 | 9.00 | 4.00 | 6.67 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 1.65 | 2.24 | 2.65 | 24.68 | 14.24 | 12.13 | 0.05 | 0.09 | 0.11 |
| **Poker Hand** | 1.66 | 2.38 | 2.66 | 13.64 | 7.26 | 5.39 | 0.06 | 0.10 | 0.14 |
| **Electricity** | 1.73 | 2.54 | 2.93 | 8.67 | 4.08 | 3.55 | 0.06 | 0.08 | 0.13 |

**Table A.9** Results for a buffer and test frequency of 100 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 616.78 | 27.05 | 583.28 | 592.94 | 27.64 | 558.87 | 593.74 | 27.61 | 559.62 |
| **HYP(10,0.0001)** | 619.56 | 26.84 | 586.40 | 606.20 | 27.55 | 572.26 | 604.14 | 27.51 | 570.20 |
| **SEA(50)** | 4.93 | 1.00 | 3.40 | 4.89 | 1.00 | 3.35 | 4.93 | 1.00 | 3.39 |
| **SEA(50000)** | 5.18 | 0.99 | 3.46 | 5.15 | 1.00 | 3.41 | 5.15 | 1.00 | 3.39 |
| **LED(50000)** | 20.26 | 5.63 | 12.64 | 20.37 | 5.79 | 12.60 | 20.17 | 5.79 | 12.39 |
| **Covertype** | 480.40 | 79.48 | 394.26 | 283.09 | 55.48 | 218.73 | 350.08 | 51.46 | 289.80 |
| **Poker Hand** | 239.60 | 32.72 | 204.58 | 146.56 | 24.12 | 119.60 | 149.58 | 20.24 | 126.37 |
| **Electricity** | 9.80 | 2.42 | 7.19 | 6.09 | 1.70 | 4.17 | 6.31 | 1.59 | 4.52 |

**Table A.10** Thread pool management for a buffer and test frequency of 100 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 1.09 | 0.78 | 0.76 | 4.76 | 4.56 | 4.35 | 0.05 | 0.04 | 0.02 |
| **HYP(10,0.0001)** | 0.20 | 0.17 | 0.18 | 2.59 | 2.54 | 2.58 | 0.01 | 0.01 | 0.01 |
| **SEA(50)** | 0.03 | 0.03 | 0.03 | 0.22 | 0.22 | 0.22 | 0.00 | 0.00 | 0.00 |
| **SEA(50000)** | 0.03 | 0.03 | 0.03 | 0.22 | 0.22 | 0.22 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.04 | 0.04 | 0.04 | 0.69 | 0.69 | 0.67 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 1.69 | 2.37 | 2.74 | 67.08 | 36.24 | 46.69 | 0.06 | 0.13 | 0.14 |
| **Poker Hand** | 1.62 | 2.33 | 2.60 | 24.47 | 12.98 | 13.17 | 0.07 | 0.12 | 0.15 |
| **Electricity** | 1.65 | 2.42 | 2.82 | 17.08 | 8.69 | 8.78 | 0.07 | 0.10 | 0.15 |

**Table A.11** Results for a buffer and test frequency of 500 instances (in seconds).

| Data set configuration | 1 core | | | 2 cores | | | 4 cores | | |
|---|---|---|---|---|---|---|---|---|---|
| | eval | train | test | eval | train | test | eval | train | test |
| **HYP(10,0.001)** | 1118.03 | 27.98 | 1083.10 | 1157.49 | 30.18 | 1119.82 | 1185.39 | 29.30 | 1148.60 |
| **HYP(10,0.0001)** | 1137.97 | 27.75 | 1103.34 | 1150.56 | 28.67 | 1114.77 | 1136.83 | 28.07 | 1101.63 |
| **SEA(50)** | 6.90 | 1.05 | 5.08 | 6.87 | 1.05 | 5.06 | 6.95 | 1.04 | 5.15 |
| **SEA(50000)** | 7.14 | 1.02 | 5.15 | 7.10 | 1.02 | 5.11 | 7.06 | 1.01 | 5.11 |
| **LED(50000)** | 34.87 | 5.77 | 26.87 | 30.83 | 5.78 | 22.84 | 35.91 | 8.35 | 25.32 |
| **Covertype** | 1831.74 | 353.76 | 1470.86 | 1078.44 | 230.53 | 840.03 | 1202.38 | 259.40 | 933.58 |
| **Poker Hand** | 493.03 | 35.40 | 455.12 | 301.23 | 24.60 | 273.56 | 296.03 | 25.72 | 266.97 |
| **Electricity** | 26.72 | 7.04 | 19.47 | 16.54 | 4.76 | 11.54 | 17.00 | 4.83 | 11.86 |

**Table A.12** Thread pool management for a buffer and test frequency of 500 instances (in milliseconds).

| Data set configuration | Creation time | | | Execution time | | | Destruction time | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores | 1 core | 2 cores | 4 cores |
| **HYP(10,0.001)** | 0.61 | 0.64 | 0.91 | 51.68 | 53.48 | 54.58 | 0.02 | 0.02 | 0.10 |
| **HYP(10,0.0001)** | 0.57 | 0.57 | 0.83 | 52.70 | 53.28 | 52.28 | 0.02 | 0.02 | 0.09 |
| **SEA(50)** | 0.04 | 0.04 | 0.05 | 2.08 | 2.07 | 2.08 | 0.00 | 0.00 | 0.01 |
| **SEA(50000)** | 0.04 | 0.04 | 0.05 | 2.11 | 2.08 | 2.07 | 0.00 | 0.00 | 0.00 |
| **LED(50000)** | 0.05 | 0.05 | 0.07 | 10.88 | 8.83 | 8.62 | 0.00 | 0.00 | 0.00 |
| **Covertype** | 2.24 | 2.81 | 4.02 | 580.93 | 335.76 | 374.69 | 0.07 | 0.14 | 0.08 |
| **Poker Hand** | 2.14 | 2.67 | 3.89 | 265.42 | 156.56 | 151.53 | 0.06 | 0.14 | 0.08 |
| **Electricity** | 2.02 | 2.48 | 2.99 | 112.50 | 67.32 | 68.75 | 0.06 | 0.14 | 0.07 |

# BIBLIOGRAPHY

D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, August 2003. ISSN 1066-8888. URL http://dx.doi.org/10.1007/s00778-003-0095-z.

D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *2nd Biennial Conference on Innovative Data Systems Research*, CIDR '05, pages 277–289, Asilomar, CA, January 2005.

A. Aljaafreh and L. Dong. Cooperative detection of moving targets in wireless sensor network based on fuzzy dynamic weighted majority voting decision fusion. In *International Conference on Networking, Sensing and Control*, ICNSC '10, pages 544–548, April 2010. URL http://dx.doi.org/10.1109/ICNSC.2010.5461603.

A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: The stanford data stream management system. Technical report, Stanford InfoLab, 2004.

A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, June 2006. ISSN 1066-8888. URL http://dx.doi.org/10.1007/s00778-004-0147-z.

B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM. ISBN 1-58113-507-6. URL http://dx.doi.org/10.1145/543613.543615.

S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30 (3):109–120, September 2001. ISSN 0163-5808. URL http://dx.doi.org/10.1145/603867.603884.

S. Babu and J. Widom. Streamon: an adaptive engine for stream query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 931–932, New York, NY, USA, 2004. ACM. ISBN 1-58113-859-8. URL http://dx.doi.org/10.1145/1007568.1007702.

S. H. Bach and M. A. Maloof. Paired learners for concept drift. In *IEEE International Conference on Data Mining*, ICDM '08, pages 23–32, Los Alamitos, CA, USA, December 2008. IEEE Computer Society. URL http://dx.doi.org/10.1109/ICDM.2008.119.

M. Baena-García, J. Del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno. Early drift detection method. In *International Workshop on Knowledge Discovery from Data Streams*, IWKDDS '06, pages 77–86, 2006. URL http://eprints.pascal-network.org/archive/00002509/.

H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik. Retrospective on aurora. *The VLDB Journal*, 13(4):370–383, December 2004. ISSN 1066-8888. URL http://dx.doi.org/10.1007/s00778-004-0133-5.

L. Baringhaus and C. Franz. On a new multivariate two-sample test. *Journal of Multivariate Analysis*, 88(1):190–206, 2004. ISSN 0047-259X. URL http://dx.doi.org/10.1016/S0047-259X(03)00079-4.

J. Bazan and M. Szczuka. Rses and rseslib - a collection of tools for rough set computations. In W. Ziarko and Y. Yao, editors, *Rough Sets and Current Trends in Computing*, volume 2005 of *Lecture Notes in Computer Science*, pages 106–113. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-43074-2. URL http://dx.doi.org/10.1007/3-540-45554-X_12.

A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, SDM '07, pages 443–448, Lake Buena Vista, Florida, USA, April 2007. SIAM.

A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà. Improving adaptive bagging methods for evolving data streams. In Z.-H. Zhou and T. Washio, editors, *Advances in Machine Learning*, volume 5828 of *Lecture Notes in Computer Science*, pages 23–37. Springer Berlin / Heidelberg, 2009a. ISBN 978-3-642-05223-1. URL http://dx.doi.org/10.1007/978-3-642-05224-8_4.

A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 139–148, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-495-9. URL http://dx.doi.org/10.1145/1557019.1557041.

A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, August 2010a. ISSN 1532-4435. URL http://portal.acm.org/citation.cfm?id=1859890.1859903.

A. Bifet, G. Holmes, B. Pfahringer, and E. Frank. Fast perceptron decision tree learning from evolving data streams. In M. Zaki, J. Yu, B. Ravindran, and V. Pudi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 6119 of *Lecture Notes in Computer Science*, pages 299–310. Springer Berlin / Heidelberg, 2010b. ISBN 978-3-642-13671-9. URL http://dx.doi.org/10.1007/978-3-642-13672-6_30.

J. A. Blackard and D. 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, December 1999. ISSN 0168-1699. URL http://dx.doi.org/10.1016/S0168-1699(99)00046-0.

A. Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26(1):5–23, 1997. ISSN 0885-6125. URL http://dx.doi.org/10.1023/A:1007335615132.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth Statistics / Probability series. Wadsworth International Group, Belmont, California, 1984.

D. Brzeziński. Mining data streams with concept drift. Master's thesis, Poznan University of Technology, Poznań, Poland, 2010.

D. Brzeziński and J. Stefanowski. Accuracy updated ensemble for data streams with concept drift. In E. Corchado, M. Kurzynski, and M. Wozniak, editors, *Hybrid Artificial Intelligent Systems*, volume 6679 of *Lecture Notes in Computer Science*, pages 155–163. Springer Berlin / Heidelberg, 2011. URL http://dx.doi.org/10.1007/978-3-642-21222-2_19.

Y. D. Cai, D. Clutter, G. Pape, J. Han, M. Welge, and L. Auvil. Maids: Mining alarming incidents from data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 919–920, New York, NY, USA, 2004. ACM. ISBN 1-58113-859-8. URL http://dx.doi.org/10.1145/1007568.1007695.

L. Chen, P. Dai, and W. Dou. Mtsknn: Multivariate two-sample tests based on k-nearest-neighbors, 2010. URL http://cran.r-project.org/web/packages/MTSKNN/index.html.

M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Conference on Innovative Data Systems Research*, CIDR '03, January 2003.

Y. Chi, H. Wang, and P. S. Yu. Loadstar: load shedding in data stream mining. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1302–1305. VLDB Endowment, 2005. ISBN 1-59593-154-6. URL http://dl.acm.org/citation.cfm?id=1083592.1083757.

C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 647–651, New York, NY, USA, June 2003. ACM. ISBN 1-58113-634-X. URL http://dx.doi.org/10.1145/872757.872838.

S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4-5):187–195, 2005. ISSN 0950-7051. URL http://dx.doi.org/10.1016/j.knosys.2004.10.002. AI-2004, Cambridge, England, 13th-15th December 2004.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1248547.1248548.

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM. ISBN 1-58113-233-6. URL http://dx.doi.org/10.1145/347090.347107.

A. Dries and U. Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining,*, 2(5-6):311–327, December 2009. ISSN 1932-1872. URL http://dx.doi.org/10.1002/sam.10054.

O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961. URL http://dx.doi.org/10.1080/01621459.1961.10482090.

R. Elwell and R. Polikar. Incremental learning of variable rate concept drift. In J. Benediktsson, J. Kittler, and F. Roli, editors, *Multiple Classifier Systems*, volume 5519 of *Lecture Notes in Computer Science*, pages 142–151. Springer Berlin / Heidelberg, June 2009a. URL http://dx.doi.org/10.1007/978-3-642-02326-2_15.

R. Elwell and R. Polikar. Incremental learning in nonstationary environments with controlled forgetting. In *IEEE International Joint Conference on Neural Networks*, IJCNN '09, pages 771–778, Los Alamitos, CA, USA, June 2009b. IEEE Computer Society. URL http://dx.doi.org/10.1109/IJCNN.2009.5178779.

R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, October 2011. ISSN 1045-9227. URL http://dx.doi.org/10.1109/TNN.2011.2160459.

W. Fan. Systematic data selection to mine concept-drifting data streams. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 128–137, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. URL http://dx.doi.org/10.1145/1014052.1014069.

W. Fan, Y. an Huang, H. Wang, and P. S. Yu. Active mining of data streams. In M. W. Berry, U. Dayal, C. Kamath, and D. Skillicorn, editors, *Proceedings of the Fourth SIAM International Conference on Data Mining*, SDM '04, pages 457–461, Lake Buena Vista, Florida, USA, April 2004. SIAM.

F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme. Discovering decision rules from numerical data streams. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 649–653, New York, NY, USA, 2004. ACM. ISBN 1-58113-812-1. URL http://dx.doi.org/10.1145/967900.968036.

F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme. Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005*

*ACM Symposium on Applied Computing*, SAC '05, pages 568–572, New York, NY, USA, 2005a. ACM. ISBN 1-58113-964-0. URL http://dx.doi.org/10.1145/1066677.1066808.

F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme. Incremental rule learning and border examples selection from numerical data streams. *Journal of Universal Computer Science*, 11(8):1426–1439, August 2005b. URL http://dx.doi.org/10.3217/jucs-011-08-1426.

C. Franz. cramer: Multivariate nonparametric cramer-test for the two-sample-problem, June 2006. URL http://cran.r-project.org/web/packages/cramer/index.html.

M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. URL http://dx.doi.org/10.1080/01621459.1937.10503522.

M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, March 1940. URL http://www.jstor.org/stable/2235971.

M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: A review. *SIGMOD Record*, 34(2):18–26, June 2005. ISSN 0163-5808. URL http://dx.doi.org/10.1145/1083784.1083789.

J. Gama and P. Kosina. Tracking recurring concepts with meta-learners. In L. S. Lopes, N. Lau, P. Mariano, and L. M. Rocha, editors, *Progress in Artificial Intelligence*, volume 5816 of *Lecture Notes in Computer Science*, pages 423–434. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04685-8. URL http://dx.doi.org/10.1007/978-3-642-04686-5_35.

J. Gama and P. Rodrigues. Stream-based electricity load forecast. In J. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenic, and A. Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, volume 4702 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin / Heidelberg, 2007. URL http://dx.doi.org/10.1007/978-3-540-74976-9_45.

J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 523–528, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. URL http://dx.doi.org/10.1145/956750.956813.

J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In A. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 66–112. Springer Berlin / Heidelberg, 2004a. URL http://dx.doi.org/10.1007/978-3-540-28645-5_29.

J. Gama, P. Medas, and R. Rocha. Forest trees for on-line data. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 632–636, New York, NY, USA, 2004b. ACM. ISBN 1-58113-812-1. URL http://dx.doi.org/10.1145/967900.968033.

J. Gama, P. Medas, and P. Rodrigues. Learning decision trees from dynamic data streams. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 573–577, New York, NY, USA, 2005. ACM. ISBN 1-58113-964-0. URL http://dx.doi.org/10.1145/1066677.1066809.

J. Gama, R. Fernandes, and R. Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1):23–54, March 2006. ISSN 1088-467X. URL http://iospress.metapress.com/content/858v8q1w2284pyrn/.

J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 329–338, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. URL http://dx.doi.org/10.1145/1557019.1557060.

L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, June 2003. ISSN 0163-5808. URL http://dx.doi.org/10.1145/776985.776986.

J. B. Gomes, E. Menasalvas, and P. A. C. Sousa. Learning recurring concepts from data streams with a context-aware ensemble. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 994–999, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0113-8. URL http://dx.doi.org/10.1145/1982185.1982403.

P. M. Gonçalves, Jr. and R. S. M. Barros. A comparison on how statistical tests deal with concept drifts. In H. R. Arabnia, D. de la Fuente, E. B. Kozerenko, P. M. LaMonica, R. A. Liuzzi, J. A. Olivas, A. M. G. Solo, and T. Waskiewicz, editors, *Proceedings of the 2012 International Conference on Artificial Intelligence*, volume 2 of *ICAI '12*, pages 832–838, Las Vegas, Nevada, USA, July 2012. CSREA Press. ISBN 1-60132-218-6. URL http://world-comp.org/proc2012/icai.html.

P. M. Gonçalves, Jr. and R. S. M. Barros. RCD: A recurring concept drift framework. *Pattern Recognition Letters*, 34(9):1018–1025, July 2013a. ISSN 0167-8655. URL http://dx.doi.org/10.1016/j.patrec.2013.02.005.

P. M. Gonçalves, Jr. and R. S. M. Barros. Speeding up statistical tests to detect recurring concept drifts. In R. Lee, editor, *Computer and Information Science*, volume 493 of *Studies in Computational Intelligence*, pages 129–142. Springer International Publishing, 2013b. ISBN 978-3-319-00803-5. URL http://dx.doi.org/10.1007/978-3-319-00804-2_10.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, November 2009. ISSN 1931-0145. URL http://dx.doi.org/10.1145/1656274.1656278.

M. B. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Machine Learning*, 32(2):101–126, 1998. ISSN 0885-6125. URL http://dx.doi.org/10.1023/A:1007420529897.

N. Henze. A multivariate two-sample test based on the number of nearest neighbor type coincidences. *The Annals of Statistics*, 16(2):772–783, June 1988. URL http://www.jstor.org/stable/2241756.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. URL http://www.jstor.org/stable/4615733.

X.-G. Hu, P.-P. Li, X.-D. Wu, and G.-Q. Wu. A semi-random multiple decision-tree algorithm for mining data streams. *Journal of Computer Science and Technology*, 22(5):711–724, 2007. ISSN 1000-9000. URL http://dx.doi.org/10.1007/s11390-007-9084-9.

G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. URL http://dx.doi.org/10.1145/502512.502529.

R. L. Iman and J. M. Davenport. Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595, 1980. URL http://dx.doi.org/10.1080/03610928008827904.

R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 571–576, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. URL http://dx.doi.org/10.1145/956750.956821.

H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. VEDAS: A mobile and distributed data stream mining system for real-time vehicle monitoring. In M. W. Berry, U. Dayal, C. Kamath, and D. Skillicorn, editors, *Proceedings of the Fourth SIAM International Conference on Data Mining*, SDM '04, pages 300–311, Lake Buena Vista, Florida, USA, April 2004. SIAM.

M. Karnick, M. Ahiskali, M. D. Muhlbaier, and R. Polikar. Learning concept drift in nonstationary environments using an ensemble of classifiers based approach. In *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, IJCNN '08, pages 3455–3462, Los Alamitos, CA, USA, June 2008a. IEEE Computer Society. URL http://dx.doi.org/10.1109/IJCNN.2008.4634290.

M. Karnick, M. D. Muhlbaier, and R. Polikar. Incremental learning in non-stationary environments with concept drift using a multiple classifier based approach. In *19th International Conference on Pattern Recognition*, ICPR '08, pages 1–4, December 2008b. URL http://dx.doi.org/10.1109/ICPR.2008.4761062.

I. Katakis, G. Tsoumakas, and I. Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22:371–391, 2010. ISSN 0219-1377. URL http://dx.doi.org/10.1007/s10115-009-0206-2.

M. Y. Kiang. A comparative assessment of classification methods. *Decision Support Systems*, 35(4):441–454, July 2003. ISSN 0167-9236. URL http://dx.doi.org/10.1016/S0167-9236(02)00110-0.

D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proceedings of the 30th International Conference on Very Large Data Bases*, volume 30 of *VLDB '04*,

pages 180–191. VLDB Endowment, 2004. ISBN 0-12-088469-0. URL http://portal.acm.org/citation.cfm?id=1316689.1316707.

J. Z. Kolter and M. A. Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 449–456, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. URL http://dx.doi.org/10.1145/1102351.1102408.

J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, December 2007. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1314498.1390333.

T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In R. Agrawal and P. Stolorz, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD '98, pages 259–263, Menlo Park, CA, USA, August 1998. AAAI Press. URL http://www.aaai.org/Papers/KDD/1998/KDD98-045.pdf.

Y. K. Lee, L. Wang, and K. H. Ryu. A system architecture for monitoring sensor data stream. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, CIT '07, pages 1026 –1031, Los Alamitos, CA, USA, October 2007. IEEE Computer Society. URL http://dx.doi.org/10.1109/CIT.2007.178.

C. Li, Y. Zhang, and X. Li. Ocvfdt: one-class very fast decision tree for one-class classification of data streams. In *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '09, pages 79–86, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-668-7. URL http://dx.doi.org/10.1145/1601966.1601981.

N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, February 1994. ISSN 0890-5401. URL http://dx.doi.org/10.1006/inco.1994.1009.

M. A. Maloof. Concept drift. In J. Wang, editor, *Encyclopedia of Data Warehousing and Mining*, pages 202–206. IGI Global, 2005. ISBN 9781591405573. URL http://dx.doi.org/10.4018/978-1-59140-557-3.ch039.

G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02,

pages 346–357. VLDB Endowment, 2002. URL http://dl.acm.org/citation.cfm?id=1287369.1287400.

O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, San Mateo, California, 1994. Morgan Kaufmann.

L. L. Minku and X. Yao. DDD: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):619–633, April 2012. ISSN 1041-4347. URL http://dx.doi.org/10.1109/TKDE.2011.58.

L. L. Minku, A. P. White, and X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, May 2010. ISSN 1041-4347. URL http://dx.doi.org/10.1109/TKDE.2009.156.

M. Muhlbaier and R. Polikar. An ensemble approach for incremental learning in nonstationary environments. In M. Haindl, J. Kittler, and F. Roli, editors, *Multiple Classifier Systems*, volume 4472 of *Lecture Notes in Computer Science*, pages 490–500. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-72481-0. URL http://dx.doi.org/10.1007/978-3-540-72523-7_49.

A. Narasimhamurthy and L. I. Kuncheva. A framework for generating data to simulate changing environments. In *Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications*, AIAP '07, pages 384–389, Anaheim, CA, USA, 2007. ACTA Press. URL http://dl.acm.org/citation.cfm?id=1295303.1295369.

P. B. Nemenyi. *Distribution-free multiple comparisons*. PhD thesis, Princeton University, 1963.

K. Nishida and K. Yamauchi. Detecting concept drift using statistical testing. In V. Corruble, M. Takeda, and E. Suzuki, editors, *Proceedings of the 10th International Conference on Discovery Science*, volume 4755 of *DS '07*, pages 264–269, Berlin, Heidelberg, October 2007. Springer-Verlag. ISBN 978-3-540-75487-9. URL http://portal.acm.org/citation.cfm?id=1778942.1778972.

N. C. Oza. Online bagging and boosting. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3 of *SMC '05*, pages 2340–2345, Los Alamitos, CA, USA,

October 2005. IEEE Computer Society. URL http://dx.doi.org/10.1109/ICSMC.2005.1571498.

N. C. Oza and S. Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 359–364, New York, NY, USA, 2001a. ACM. ISBN 1-58113-391-X. URL http://dx.doi.org/10.1145/502512.502565.

N. C. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics*, pages 105–112. Morgan Kaufmann, 2001b.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, June 1954. URL http://www.jstor.org/stable/2333009.

R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

S. Ramamurthy and R. Bhatnagar. Tracking recurrent concept drift in streaming data using ensemble classifiers. In *Proceedings of the 6th International Conference on Machine Learning and Applications*, ICMLA '07, pages 404–409, Los Alamitos, CA, USA, 2007. IEEE Computer Society. ISBN 0-7695-3069-9. URL http://dx.doi.org/10.1109/ICMLA.2007.80.

S. W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, August 1959. ISSN 00401706. URL http://www.jstor.org/stable/1266443.

G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198, January 2012. ISSN 0167-8655. URL http://dx.doi.org/10.1016/j.patrec.2011.08.019.

M. Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, February 1997. ISSN 0269-2821. URL http://dx.doi.org/10.1023/A:1006515405170.

M. F. Schilling. Multivariate two-sample tests based on nearest neighbors. *Journal of the American Statistical Association*, 81(395):799–806, September 1986. URL http://www.jstor.org/stable/2289012.

J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986. ISSN 0885-6125. URL http://dx.doi.org/10.1023/A:1022810614389.

R. Sebastião and J. Gama. A study on change detection methods. In L. S. Lopes, N. Lau, P. Mariano, and L. M. Rocha, editors, *Proceedings of the 14th Portuguese Conference on Artificial Intelligence*, EPIA '09, pages 353–364, Aveiro, Portugal, October 2009. Designeed - Design e Publicidade, UNIP. LDA. ISBN 978-972-96895-4-3. URL http://epia2009.web.ua.pt/onlineEdition/353.pdf.

R. Sebastião, J. Gama, P. Rodrigues, and J. Bernardes. Monitoring incremental histogram distribution for change detection in data streams. In M. Gaber, R. Vatsavai, O. Omitaomu, J. Gama, N. Chawla, and A. Ganguly, editors, *Knowledge Discovery from Sensor Data*, volume 5840 of *Lecture Notes in Computer Science*, pages 25–42. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-12518-8. URL http://dx.doi.org/10.1007/978-3-642-12519-5_2.

K. O. Stanley. Learning concept drift with a committee of decision trees. Technical report, Department of Computer Sciences, University of Texas at Austin, USA, 2003.

W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. URL http://dx.doi.org/10.1145/502512.502568.

Y. Sun, G. Mao, X. Liu, and C. Liu. Mining concept drifts from data streams based on multi-classifiers. *International Conference on Advanced Information Networking and Applications Workshops*, 2:257–263, 2007. URL http://dx.doi.org/10.1109/AINAW.2007.250.

N. Tatbul, U. Çetintemel, and S. Zdonik. Staying fit: efficient load shedding techniques for distributed stream processing. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 159–170. VLDB Endowment, 2007. ISBN 978-1-59593-649-3. URL http://dl.acm.org/citation.cfm?id=1325851.1325873.

H. Thakkar, B. Mozafari, and C. Zaniolo. A data stream mining system. In *IEEE International Conference on Data Mining Workshops*, ICDMW '08, pages 987–990, Los Alamitos, CA, USA, December 2008. IEEE Computer Society.

A. Tsymbal. The problem of concept drift: Definitions and related work. Technical report, Department of Computer Science, Trinity College, Dublin, Ireland, April 2004. URL http://www.cs.tcd.ie/publications/tech-reports/reports.04/TCD-CS-2004-15.pdf.

A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Dynamic integration of classifiers for handling concept drift. *Information Fusion*, 9(1):56–68, January 2008. ISSN 1566-2535. URL http://dx.doi.org/10.1016/j.inffus.2006.11.002. Special Issue on Applications of Ensemble Methods.

H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. URL http://dx.doi.org/10.1145/956750.956778.

S. Wang, S. Schlobach, and M. Klein. Concept drift and how to identify it. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(3):247–265, September 2011. ISSN 1570-8268. URL http://dx.doi.org/10.1016/j.websem.2011.05.003.

G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996. ISSN 0885-6125. URL http://dx.doi.org/10.1023/A:1018046501280.

D. Wu, K. Wang, T. He, and J. Ren. A dynamic weighted ensemble to cope with concept drifting classification. In *The 9th International Conference for Young Computer Scientists*, ICYCS '08, pages 1854–1859, November 2008. URL http://dx.doi.org/10.1109/ICYCS.2008.491.

Y. Yang, X. Wu, and X. Zhu. Combining proactive and reactive predictions for data streams. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 710–715, New York, NY, USA, 2005. ACM. ISBN 1-59593-135-X. URL http://dx.doi.org/10.1145/1081870.1081961.

Y. Yang, X. Wu, and X. Zhu. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data Mining and Knowledge Discovery*, 13(3):261–289, 2006. ISSN 1384-5810. URL http://dx.doi.org/10.1007/s10618-006-0050-x.

F. Yates. Contingency table involving small numbers and the $\chi^2$ test. *Journal of the Royal Statistical Society Supplement*, 1:217–235, 1934.

A. B. Yeh, R. N. Mcgrath, M. A. Sembower, and Q. Shen. Ewma control charts for monitoring high-yield processes based on non-transformed observations. *International Journal of Production Research*, 46(20):5679–5699, September 2008. ISSN 0020-7543. URL http://dx.doi.org/10.1080/00207540601182252.

Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 336–345, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. URL http://dx.doi.org/10.1145/956750.956789.