LUCCA PORTES CAVALHEIRO

DYNAMIC SELECTION OF CLASSIFIERS IN DATA STREAMS

Thesis project presented to the Graduate Program in Informatics of Pontifícia Universidade Católica do Paraná (PUCPR) as partial requirement for the degree of Master in Informatics.

Curitiba
2021

LUCCA PORTES CAVALHEIRO

DYNAMIC SELECTION OF CLASSIFIERS IN DATA STREAMS

Dissertation project presented to the Graduate Program in Informatics of Pontifícia Universidade Católica do Paraná (PUCPR) as partial requirement for the degree of Master in Informatics.

Major Field: Computer Science

Adviser: Jean Paul Barddal
Co-adviser: Alceu Britto

Curitiba
2021

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Edilene de Oliveira dos Santos CRB-9/1636

C376d
2021

Cavalheiro, Lucca Portes
Dynamic selection of classifiers in data streams / Lucca Portes Cavalheiro; Adviser, Jean Paul Barddal; co-adviser, Alceu Brito. 2021
104 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2021
Bibliografia: f.93-101

1. Informática. 2. Classificação. 3. Fluxo de dados (Computadores). 4. Algoritmos computacionais. 5. Mineração de dados (Computação). 6. Aprendizado do computador. I. Barddal, Jean Paul. II. Brito, Alceu. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título

.   CDD. 20.ed. – 004

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

16-2021

DECLARAÇÃO

Declaro para os devidos fins que o aluno **LUCCA PORTES CAVALHEIRO**, defendeu sua dissertação de Mestrado intitulada **"Dynamic Selection of Classifiers in Data Streams"**, na área de concentração Ciência da Computação, no dia 03 de março de 2021, no qual foi aprovado.

Declaro ainda que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

Curitiba, 17 de março de 2021.

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná

Contents

Contents i

List of Algorithms v

List of Figures vi

List of Tables x

Abstract xii

Resumo xiii

I Introduction and Preliminaries

Chapter 1

Introduction 2

1.1 Objectives . . . 3

1.2 Hypotheses . . . 3

1.3 Contributions . . . 4

1.4 Financial Support . . . 4

1.5 Overview . . . 5

Chapter 2

Classification in Data Streams 6

2.1 Evaluation . . . 7

2.2 Validation . . . 9

2.3 Dynamic Environments . . . 9

2.3.1 Concept Drift . . . 10

2.3.2 Detecting Changes . . . 13

2.3.3 Forgetting . . . 13

2.4 Validation in Data Stream Mining . . . 14

2.4.1 Periodic Holdout . . . 14

2.4.2 Prequential . . . 15
2.5 Data Stream Generators . . . 15
2.5.1 Streaming Ensemble Algorithm - SEA . . . 16
2.5.2 AGRAWAL . . . 16
2.5.3 Random Tree Generator - RTG . . . 17
2.5.4 Asset Negotiation Generator . . . 17
2.6 Real-World Datasets . . . 18
2.6.1 NOMAO . . . 18
2.6.2 Spam Corpus . . . 18
2.6.3 Electricity . . . 18
2.7 Base Classifiers for Data Streams . . . 19
2.7.1 K-Nearest Neighbors . . . 19
2.7.2 Naive Bayes . . . 20
2.7.3 Hoeffding Tree . . . 20
2.8 Dynamic Ensembles . . . 22
2.8.1 OzaBag . . . 23
2.8.2 OzaBoost . . . 25
2.8.3 OzaBagAdwin . . . 25
2.8.4 Adaptive Random Forest . . . 26
2.8.5 Kappa Update Ensemble . . . 27
2.9 Frameworks for Data Stream Mining . . . 28
2.9.1 Massive Online Analysis - MOA . . . 28
2.9.2 Scikit-Multiflow . . . 28
2.10 Final Considerations . . . 29

Chapter 3

Dynamic Classifier Selection 30
3.1 Methods for Offline Machine Learning . . . 32
3.1.1 A Priori and A Posteriori . . . 32
3.1.2 DCS-LA . . . 32
3.1.3 DCS-RANK . . . 33
3.1.4 KNORA . . . 33
3.1.5 K-Nearest Output Profiles . . . 34
3.1.6 Multiple Classifier Behavior . . . 35
3.1.7 META-DES . . . 35
3.2 Methods for Online Machine Learning . . . 37

3.2.1 Minority Driven Ensemble 37
3.2.2 Dynamic Ensemble Selection for Drift Detection 38
3.2.3 Semi-supervised Drift Detection Method 39
3.2.4 Dynamic Selection Based Drift Handler 39
3.2.5 Preprocessed DCS I 40
3.2.6 Preprocessed DCS II 41
3.3 Applying Dynamic Selection of Classifiers 41
3.4 Final Considerations 43

II Proposal

Chapter 4
Framework for Applying DCS in Data Stream Mining 45
4.1 Scikit-Dyn2Sel 46
4.1.1 Example of DYNSE Implementation 48

Chapter 5
Behavior of Dynamic Selection on Data Stream Mining 51
5.1 Dynamic Selection in Data Stream Mining 53
5.1.1 Weak Classifiers 53
5.1.2 Complex Datasets 54
5.2 Experimental Protocol 57
5.2.1 Statistical Testing 57
5.3 Results 59
5.3.1 Adaptive Random Forest and Kappa Updated Ensemble using Hoeffding Trees 59
5.3.2 OzaBag using Hoeffding Trees 62
5.3.3 OzaBag using Perceptrons 67
5.3.4 Single Base Classifiers 70
5.4 Conclusion 70

Chapter 6
DCS Method Focused on Data Stream Mining 73
6.1 Double Dynamic Classifier Selection 74
6.2 Experimental protocol 75
6.2.1 Experimental protocol 75

6.3 Results 77

6.3.1 No Drift 77

6.3.2 Gradual Drift 80

6.3.3 Abrupt Drift 84

6.3.4 Real World 87

6.4 Conclusion 89

III Conclusions

Chapter 7

Conclusions 91

Bibliography 93

Appendix A

Results DCS Techniques in Data Stream Mining 102

List of Algorithms

1 Training Step of DDCS 74
2 Prediction Step of DDCS 75

List of Figures

2.1 Binary Confusion Matrix Example (MARKHAM, 2019). 9

2.2 Evolution of Data During Gradual Drift 12

2.3 Types of Concept Drift, adapted from (FOUCHÉ, 2017) 12

2.4 Periodic Holdout 14

2.5 Prequential 15

2.6 Data Distribution Following a Function 16

2.7 Example of Decision Tree, adapted from (GUANGA, 2019) 21

2.8 Reasons for the Adoption of Ensembles, adapted from (DIETTERICH, 2000) 24

2.9 Code Adapted from Scikit-Multiflow repository (Left) and Possible Optimized Version (Right) 29

3.1 Dynamic Selection of Classifiers 31

4.1 Class Diagram of Scikit-Dyn2Sel 47

4.2 DYNSE Training Phase 49

4.3 DYNSE Testing Phase 49

4.4 Implementation of the `partial fit` on DYNSEMethod 50

4.5 Implementation of the `predict` on DYNSEMethod 50

5.1 Decision Space of P2 Generator 52

5.2 Evolution of the Accuracy Overtime 54

5.3 Decision Space of SEA Generator - Function 0 55

5.4 Decision Space of Agrawal's Second Function 56

5.5 Critical Distance Plot Example 58

5.6 Critical Distance Plot for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - Without Drift and Real-World Datasets . 60

5.7 Bayesian Analysis Matrix for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - Without Drift and Real-World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 60

5.8 Accuracy Evolution Over Time of ARF and KUE with SEA Generator - Without Drift and Real-World datasets 62

5.9 Accuracy Evolution Over Time of ARF and KUE with Asset Generator - With Drift 63

5.10 Critical Distance Plot for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - With Drift 63

5.11 Bayesian Analysis Matrix for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - With Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 64

5.12 Accuracy Evolution Over Time of Ozabag with Hoeffding Tree on the P2 Generator - Without Drift 64

5.13 Critical Distance Plot for OzaBag with Hoeffding Tree - Without Drift and Real-World Datasets 64

5.14 Bayesian Analysis Matrix for OzaBag with Hoeffding Tree - Without Drift and Real-World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 65

5.15 Accuracy Evolution Over Time of Ozabag with Hoeffding Tree on the SEA and RTG Generator - With Drift 66

5.16 Critical Distance Plot for OzaBag with Hoeffding Tree - With Drift 66

5.17 Bayesian Analysis Matrix for OzaBag with Hoeffding Tree - With Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 67

5.18 Accuracy Evolution Over Time of OzaBag with perceptron on the P2 Generator 68

5.19 Critical Distance Plot for OzaBag with Perceptron - Without Drift and Real World Dataset 68

5.20 Bayesian Analysis Matrix for OzaBag with Perceptron - Without Drift and Real-World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 69

5.21 Critical Distance Plot for OzaBag with Perceptron - With Drift 69

5.22 Bayesian Analysis Matrix for OzaBag with Perceptron - With Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 70

6.1 Critical Distance Comparing DDCS with Streams Without Drift 78

6.2 Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Streams Without Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 79

6.3 Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Streams Without Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 79

6.4 Critical Distance Comparing DDCS with Streams with Gradual Drift . . . 81

6.5 Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Streams with Gradual Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 82

6.6 Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Streams with Gradual Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 82

6.7 Critical Distance Comparing DDCS with Streams with Abrupt Drift 84

6.8 Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Streams with Abrupt Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 85

6.9 Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Streams with Abrupt Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 85

6.10 Critical Distance Comparing DDCS with Real World Datasets 87

6.11 Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Real World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 88

6.12 Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Real World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively. 88

List of Tables

2.1 Agrawal Generator Features, adapted from (AGRAWAL; IMIELINSKI; SWAMI, 1993) 17

2.2 Asset Negotiation Generator Features, adapted from (ENEMBRECK et al., 2007) 18

2.3 Probability of Numbers 0 to 5 on Poisson Distribution with $\lambda = 1$. 25

2.4 Probability of Numbers 0 to 11 on Poisson Distribution with $\lambda = 6$. 26

2.5 Time Processing of Original Code versus With Possible Optimization . . . 29

3.1 Methods Implemented in DESLIB 42

4.1 Methods Contemplated in Scikit-Dyn2Sel 48

5.1 Results of DESLIB Experiment 53

5.2 Results of Generators with Batch Random Forest 56

5.3 Ensemble Algorithms, DCS Methods and Generator used in the Experiments 58

5.4 Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - Without Drift and Real-World datasets 59

5.5 Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - With Drift 61

5.6 OzaBag with Hoeffding Tree - Without Drift and Real-World Datasets . . 63

5.7 OzaBag with Hoeffding Tree - With Drift 66

5.8 OzaBag with Percepton - Without Drift and Real-World Datasets 67

5.9 OzaBag with Percepton - With Drift 69

5.10 Accuracy of the Base Classifiers in the Streams 71

6.1 Stream Generators and Datasets Used in the Experiments 76

6.2 Results Comparing DDCS with Streams Without Drift 77

6.3 Results Comparing Processing Time and Memory Consumption with Streams Without Drift Against ARF_ACC 80
6.4 Results Comparing DDCS with Streams with Gradual Drift 80
6.5 Results Comparing Processing Time and Memory Consumption with Streams with Gradual Drift Against ARF_ACC 83
6.6 Results Comparing DDCS with Streams with Abrupt Drift 84
6.7 Results Comparing Processing Time and Memory Consumption with Streams with Abrupt Drift Against ARF_ACC 86
6.8 Results Comparing DDCS with Real World Datasets 87
6.9 Results Comparing Processing Time and Memory Consumption with Streams for Real World Datasets Against OzaBag 87

A.1 Results for all experiments reported in Section 5.3 102

Abstract

Machine learning has been an essential tool these days. It allows the extraction of information that would be impossible to extract manually. One of the areas widely studied in the traditional machine learning environment, which aims to improve the predictive power of algorithms, is the dynamic classifier selection (DCS). However, with the increasing volume of stored data, traditional machine learning techniques are beginning to prove unfeasible in certain cases. The field of data stream mining grows as an alternative to applying similar algorithms in such situations. DCS, however, has been scarcely explored in the data stream mining environment. Nonetheless, it is intuitive that dynamic selection might be useful in an environment with dynamic learners, since learners are constantly evolving with data arrival. This highlights a gap to be filled. This project is dedicated to propose a new DCS method for mining data streams, as well as understanding the behavior of traditional DCS algorithms in common data stream mining ensembles. A python framework for applying DCS methods for data streams, Scikit-Dyn2Sel, was also proposed. Regarding the behavior, results showed that DCS works similarly in the data stream mining context as it does in the batch environment, however, there is little to no gain in applying such algorithms to the online state-of-art ensemble as is. Tests with the proposed method showed very competitive results against both general and DCS state-of-art of data stream mining. Our method achieved better accuracy in most cases, while using only a small fraction of the time and memory.

Keywords: Data Stream Classification; Dynamic Selection of Classifiers.

Resumo

O aprendizado de máquina tem sido uma ferramenta essencial atualmente. Ele permite a extração de informações que seriam impossíveis de serem extraidas manualmente. Uma das áreas amplamente estudadas no ambiente tradicional de aprendizado de máquina, que visa melhorar o poder preditivo dos algoritmos, é a seleção dinâmica de classificadores (DCS, em inglês). No entanto, com o aumento do volume de dados armazenados, as técnicas tradicionais de aprendizado de máquina começam a se mostrar inviáveis em alguns casos. O campo da mineração de fluxo de dados cresce como uma alternativa à aplicação de algoritmos semelhantes em tais situações. DCS, no entanto, tem sido pouco explorado no ambiente de mineração de fluxo de dados. Mas é intuitivo que seleção dinâmica pode ser útil em um ambiente com algoritmos também dinâmicos, uma vez que os classificadores são continuamente atualizados com a chegada de novos dados. Isso destaca uma lacuna a ser preenchida. Este projeto é dedicado a propor um novo método DCS para mineração de fluxos de dados, bem como compreender o comportamento de algoritmos DCS tradicionais em ambientes de mineração de fluxo de dados comuns. Um framework python para aplicação de métodos DCS para fluxos de dados, Scikit-Dyn2Sel, também foi proposto. Com relação ao comportamento, os resultados mostraram que DCS funciona de forma semelhante no contexto de mineração de fluxo de dados como no ambiente tradicional, no entanto, há pouco ou nenhum ganho em aplicar tais algoritmos aos algoritmos do estado da arte do ambiente online. Os testes com o método proposto mostraram resultados muito competitivos em relação ao estado da arte geral e DCS de mineração de fluxo de dados. Nosso método alcançou melhor precisão na maioria dos casos, usando apenas uma pequena fração do tempo e da memória.

Palavras-chave: Classificação de Fluxos Contínuos de Dados; Seleção Dinâmica de Classificadores.

PART I

INTRODUCTION AND PRELIMINARIES

Chapter 1

Introduction

In the modern world, enterprises and governments well understood the importance of data in decision making. That is why the habit of storing data is constantly growing. However, extracting knowledge from data is not a trivial task whatsoever. Nowadays, with the great amount of data available, it is humanly impossible to do so without specialized tools.

One of the most helpful tools used with this purpose is machine learning. This area studies methods from recognizing patterns from data without human intervention in the learning process, only from examples contained in the data. The discovery of these patterns can be very helpful in the decision-making process. For example, patterns discovered from purchases historic from a store may help the store-owner decide which products should be offered for which clients.

The area of machine learning is not new. Since the decade of 1950, statistical methods were already used for this purpose. To this day, the academic community and enterprises are constantly creating new methods, for increasing the efficiency and usefulness of the area. One of the sub-areas studied for increasing the quality of machine learning methods is the dynamic classifier selection (DCS). Classifiers are one of the most basic concepts of the supervised machine learning area, they will be further explained in Chapter 2, but a rough explanation is that they learn from data to make predictions. DCS works with ensembles of classifiers, which again will be thoroughly explained in Section 2.8, but in summary, an ensemble is a group of classifiers working together to predict the same data. The idea of DCS is that not all classifiers of the ensemble are able to make a good prediction on every instance of data, so the classifiers are selected before making the prediction. The intrinsic details of this area will be explained in Chapter 3. DCS was already widely studied in the traditional machine learning area, with many techniques and satisfactory results for many applications.

However, with the increasing amount of data and the dynamicity of the relations represented by it, in some cases, the application of traditional machine learning techniques started to become difficult. To fill this gap, the area of data stream mining was proposed. It consists of similar methods for recognizing patterns in data, but it is capable of update the patterns learned in an efficient way. That way, smaller pieces of data can be constantly inputted, without causing the algorithms to deal with a giant amount of data at the same time.

For data stream mining, however, the dynamic selection is yet to be deeply studied. That means that there are not many studies applying traditional dynamic selection methods in data stream mining, or creating new methods focused on it. But in the dynamic context, where we have continuously changing learners, it is intuitive that dynamic selection might present satisfactory results. DCS might be able to not only select the classifiers that are more competent for a given instance, but for the current state of the stream. That is why we consider this to be an important gap in the current data stream mining envinroment.

This project proposes to explore and understand the behavior of traditional dynamic selection methods in data stream mining. To determine its potential contributions to the areas and in which cases they are beneficial. It also proposes new dynamic selection methods that take advantage of data stream mining characteristics, and respect its restrictions.

1.1 Objectives

This project's main objective is to propose a dynamic selection method focused on the data stream mining context. As specific objectives, we can cite:

- Framework for applying the current state-of-art DCS techniques for data streams.
- Robust experimental protocol to ensure the feasibility of the proposed DCS method.
- Analysis of how traditional DCS methods behave on traditional ensembles in the data stream environment.

1.2 Hypotheses

This project has two hypotheses. The first is to verify if the impact of dynamic selection in data stream mining ensembles accuracy is positive. The second hypothesis is to check the viability of proposing a dynamic selection method for mining data streams, with a positive impact on the accuracy, without adding significant overhead to its execution time and memory consumption.

The hypotheses are:

- The application of dynamic selection methods positively impacts the classification accuracy of dynamic ensemble methods.
- Dynamic selection method tailored for data streams positively impact the classification accuracy in the data stream mining while keeping a low use of memory and processing time.

1.3 Contributions

The proposed contributions of this project are:

- A framework for applying dynamic selection methods on data stream mining. A paper explaining its structure and operation is available at (CAVALHEIRO et al., 2020) and will soon be submitted to a peer-reviewed journal.
- Adaptation of the traditional structure of application of dynamic selection methods for data stream mining.
- Evaluation and analysis of the impact of dynamic selection methods in the data stream mining context.
- Proposal of a dynamic selection method designed towards data stream mining. A paper explaining the method and experiments was submitted and is currently being reviewed in the International Joint Conference on Neural Networks (IJCNN-2021).

1.4 Financial Support

This project was financed partially by the Pontifical Catholic University of Paraná, with a monthly tuition waiver in 2019. From the beginning of 2020 to August 2020, financial support was done by the “Coordenação de Aperfeiçoamento de Pessoal de Nível Superior” (CAPES) via the “Programa de Suporte à Pós-Graduação de Instituições Comunitárias de Ensino Particulares” (PROSUC) program. From March 2020 to November 2020, an internship on the University of Rouen, France, started with support from the same university.

1.5 Overview

This project is divided as follows. Chapter 2 introduces machine learning and data stream mining. Chapter 3 introduces the dynamic selection of classifiers, as well as its state-of-art methods. Chapter 4 presents a framework structure for applying dynamic selection in data stream mining environments. Chapter 5 presents an analysis of the behavior of traditional dynamic selection methods in data stream mining environments. Chapter 6 proposes dynamic selection methods designed for data stream mining. Chapter 7 presents the conclusion of the current state of the project, as well as its future steps.

Chapter 2

Classification in Data Streams

As time passed, the value of data began to be more and more appreciated. The same type of data that enterprises would normally discard because there was no point in storing is now seen as a potential source of information for decision making. For example, to help them find patterns in their clients' habits and consequently explore such patterns in order to bring benefits to the company, e.g., increase the profit, increase efficiency, reduce costs, etc.

This led to a major adoption of a data-driven culture globally. With the advances in computational power, storage capability, and network speed, companies and institutions began to generate and collect more data each day. Laney (2001) stated that the way corporations handled data at that time was already at its limit in terms of computational and storage capacity.

It is physically impossible to extract valuable information from enormous masses of data manually. Therefore, automatic tools are needed. One of the most important tools for this objective is Inductive Learning. It is the base of the area of Supervised Machine Learning (BISHOP, 2006). The algorithms in this area aim to learn patterns from a set of labeled examples.

These examples are called instances $\{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\}$ in which x is a set of descriptive features and y is at least one target feature[1] (BURKOV, 2019). The objective of this pattern discovery is to generate a predictive model, also called a *classifier* or *regressor*, which predicts the target feature y in unlabeled instances.

The descriptive features x of the instances are commonly inexpensive to be extracted, and this process can often be automated, e.g., the height of an animal, blood tests, word count in a document, etc. Each feature should contribute to identifying a characteristic of the instance. The target feature y frequently requires a specialist to be

[1] target features are also referred to as label or class

determined, which makes its collecting process more expensive, such as the species of an animal, if a patient has a certain disease, the type of a document, etc. That is the main motivation of supervised machine learning, predict labels that would require an expensive analysis to be determined. If the possible values for the target feature are a finite or discrete set, the machine learning problem is called a classification problem. Otherwise, if the values are continuous, it is then called a regression problem. The focus of this project is classification. A usual protocol used to test how well the model learned is to split the dataset into two parts: train and test. The train part is inputted into the algorithm to build the model, and then, the test part is utilized to compute how successful was the model generalization.

2.1 Evaluation

The evaluation of a model is done using a set of measures called performance metrics. To compute these metrics, the test set is inputted into the trained model as if it was real-world data that needed to be predicted, but since the labels or values of the test instances are available, it is possible to assess how good the predictions were.

The most known and simple metric is the accuracy of the model, represented by Equation 2.1, which is essentially the number of correct predictions p_c over the total number of predictions p_t.

$$\text{Accuracy} = \frac{p_c}{p_t} \tag{2.1}$$

This metric alone might represent the quality of a model that attempts to solve a simple prediction problem. However, when the data starts to be more complex, using only this metric might lead to a wrong conclusion.

A more complex problem, for example, could be when the distribution of classes in the dataset is imbalanced, i.e., significantly more instances from one of the classes than the others. In this case, the accuracy by itself does not provide useful information. For instance, if a binary dataset has a distribution of classes of 99% for class A and 1% for class B, and the model incorrectly predicts all of the instances as belonging to class B, the model will achieve 99% of accuracy.

For these types of problems, usually, a per class analysis is preferable. In the imbalanced class example, the analyzed class is usually the one with fewer samples. The classes are divided into positive and negative instances, with positive being the class to be analyzed and negative as the other classes. With this separation, four counters are extracted:

- True Positive - The number of correctly predicted instances from the analyzed class.
- False Positive - The number of incorrectly predicted instances from the analyzed class.
- True Negative - The number of correctly predicted instances from the other classes.
- False Positive - The number of incorrectly predicted instances from the other classes.

From these counters, two metrics can be computed (POWERS, 2008): precision and recall. Precision, as stated in Equation 2.2, is the proportion of correctly predicted positive instances (TP) over all the ones that were predicted (correctly or not) as positive ($TP + FP$). In other words, of all instances returned as positive, how many really were.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.2}$$

On the other hand, recall, as shown in 2.3, is the proportion of correctly predicted positive instances (TP) over all the positive instances ($TP + FN$) in the dataset. This is to say, of all positive instances, how many were correctly predicted.

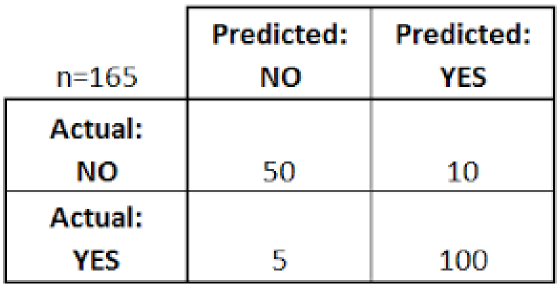
$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.3}$$

With precision and recall computed, the F1 score metric combines, with equal proportions, information from both into a single number. It can be seen as a weighted average of precision and recall. It ranges from zero to one with the values closer to one being better. Equation 2.4 displays how to compute the F1 score.

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Other vastly used metrics in the imbalanced context are the AUROC (area under the receiver operating characteristic), AUPRC (area under the precision-recall curve), and geometric mean. These are not detailed in this document since the imbalanced problem is not in the scope of this work. The interested reader is referred to (JAPKOWICZ, 2013) for a complete explanation on these metrics.

Another vastly used evaluation approach is the confusion matrix or contingency table. It allows errors between classes to be visually observed. For example, in Figure 2.1, there are two possible classes in the problem: YES or NO. The horizontal lines represent the real instances of the classes, and the vertical lines represent the predicted classes.



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Figure 2.1: Binary Confusion Matrix Example (MARKHAM, 2019).

Therefore, the main diagonal of the matrix is where the right predictions are, it should be the sequence with the highest concentration. In this example, the biggest confusion between classes was predicted YES, when the instances were actually NO (10 errors).

2.2 Validation

If the dataset is divided into train and test once, it is called the holdout protocol. Another commonly used option is the k-fold cross-validation protocol, which consists of repeatedly splitting the dataset into train and test k times and evaluating the model accordingly. This method ensures that every instance on the dataset will be used at least one time for training the model. Both methods ensure the model did not memorize the training instances of the dataset.

The problem of models memorizing training instances, instead of generalizing them, is fairly common in machine learning problems. This problem is referred to as overfitting (BURKOV, 2019). It can happen with either regression problems or classification problems. It is normally detected when comparing the performance of the prediction on the training set versus the test set. Usually, the metric used to detect it is accuracy. If it decreases significantly when evaluating using the test set, then it is an indicator that this might be happening.

2.3 Dynamic Environments

The traditional way of applying machine learning is called the offline approach. In this approach, the data that is used to build the model should be available all at once. The whole training set is loaded into the RAM, and the calculations needed to build the model are computed. This is the area that contains the most known algorithms of supervised machine learning, such as K-Nearest Neighbors (AHA; KIBLER; ALBERT, 1991), ID3 (QUINLAN, 1986), and Random Forest (HO, 1995).

However, with the ever-growing quantity of data produced by corporations and institutions, the traditional approach became unfeasible in certain scenarios. One of the reasons for this is due the limitations in computational processing availability, for example, the ability (or lack thereof) of loading all data into RAM. Another limitation is the processing time required to generate models with such a quantity of instances. When such circumstances are present, one possible approach is to resort to Data Stream Mining.

In the streaming approach, the definition of train and test set is not as precise as in the offline or static environment (BIFET et al., 2018). The quantity of available data, labeled or not, is potentially infinite since it arrives as streams of data. Possible examples of data streams are features generated by sensors, clicks on an e-commerce website, stock market prices, etc. Furthermore, the same data used for testing the model might later be used to train the model. The step of building the model is never complete, as the classifier should evolve and adapt its prediction structures as new labeled data arrives.

Other aspects of dynamic environments include: (BIFET et al., 2018):

- The learning algorithms should be able to accept new instances, either for training the model or to be classified, at any time.
- There is no definite order that the instances should be expected.
- Efficiency in the use of RAM is expected since the data arrival might never cease. This is one of the key points in data stream mining. The algorithms must be structured to deal with enormous (or even infinite) quantities of data, which makes it impossible to load all the instances into RAM.
- A limitation of the amount of time used to process each instance might exist. This aspect is highly correlated to the previous one. Since the instances might never stop arriving, the algorithm should not spend too much time with a single instance.
- Instances should be processed just once and should be then discarded. This is also correlated to the RAM item, as only the highly important information to the model should be kept in memory.

2.3.1 Concept Drift

Concepts are logic functions that machine learning algorithms learn based on patterns. These functions connect the descriptive features of a set of instances to their respective target. In other words, it is the representation of the available data behavior.

With the concept learned, the model should be ready to predict the target value of new instances. For example, in a problem with the descriptive features $\{x_1, x_2, x_3\}$, all with the same domain range $[0, 1]$, and the target feature y has a binary domain. One possible learned concept might be

$$x_1 < 0.5 \wedge (0.1 > x_2 > 0.2 \vee x_3 > 0.9) \rightarrow y = 1.$$

Machine learning is expected to solve or contribute to the solution of real-world problems. And in the real world, the behavior or the concept of certain phenomena is not always stable. However, if an approach accepts new data to be inputted in the model, which is the case of Data Stream Mining algorithms, this change of behavior must be dealt with. This change of behavior is called concept drift (SCHLIMMER; GRANGER JR., 1986).

One of the reasons for the concept drift is that sometimes the data available in the dataset does not contemplate all the descriptive features needed to fully understand the concept of a problem, an issue called Hidden Context (TSYMBAL, 2004). An example of a concept drift might occur, for instance, in a clothing store that wants to predict what type of clothes a given customer is more likely to buy. The dataset might contain variables that very well describe the customers. However, if it does not contemplate the country's inflation or the current season of the year, the patterns in the data may suddenly change with no apparent reason.

Considering the frequency, the changes in concepts can be divided into two groups: abrupt and gradual (NÚÑEZ; FIDALGO-MERINO; MORALES-BUENO, 2007). An abrupt drift occurs when the change in the concept usually occurs over one time-window to another. The gradual change occurs partially over several time windows, the percentage of one class in one or more areas of the data distribution slowly increases as the percentage of another class in the same area slowly decreases. During the occurrence of a gradual drift, it is common for a mixture between classes to happen around the decision boundary (the line or hyperplane that separates the distribution of the different classes in the data). This is illustrated in Figure 2.2, in which $f(x)_1$ and $f(x)_2$ are, respectively, the old and new decision boundaries. During the drift, instances belonging to both classes "empty circle" and "full circle" can be seen between the boundaries.

In order to define if a drift is gradual or abrupt, we must analyze the drift window W_{drift}. Considering that a drift started after an instance x_i, and the new concept was stabilized after instance x_j. If the W_{drift}, which is the number of instances passed between x_i and x_j, is equal to one, then the drift is abrupt. Otherwise, it is gradual. The difference between the two types of drift is illustrated in Figure 2.3.

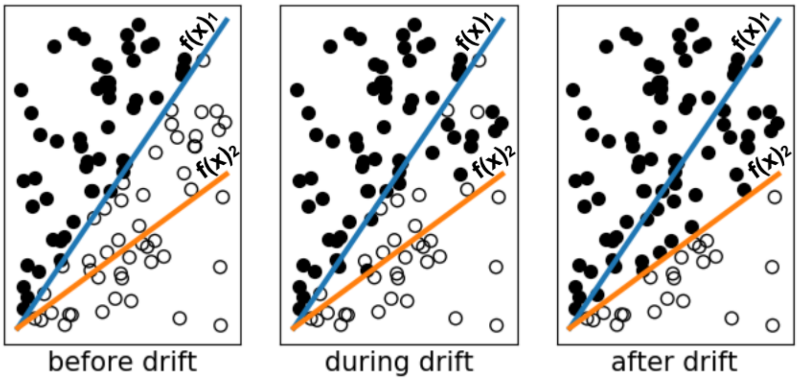


Figure 2.2: Evolution of Data During Gradual Drift

Finally, concept drifts may also be classified regarding the type of drift. There are many kinds in this category, some of the most notable are the real, virtual (PESARANGHADER; VIKTOR; PAQUET, 2018), local, and feature drift (TSYMBAL et al., 2008). In both real and virtual concept drifts, the data distribution changes, however in virtual drifts, the decision boundary remains unaffected. Local concept drifts happen only in specific parts of the data, without changing the whole concept for one class or a set of classes. An example of this phenomenon is when dealing with the resistance of bacteria to certain types of antibiotics (TSYMBAL et al., 2008). The resistance level can change to a particular colony of bacteria without affecting the global resistance. Finally, feature drift is when one or more features become or cease to be relevant for learning the concept.

An adaptive algorithm is not able to predict these changes in the concept. Yet, it should be able to detect and react to the changes in order to keep the algorithm performance metrics at an acceptable level.

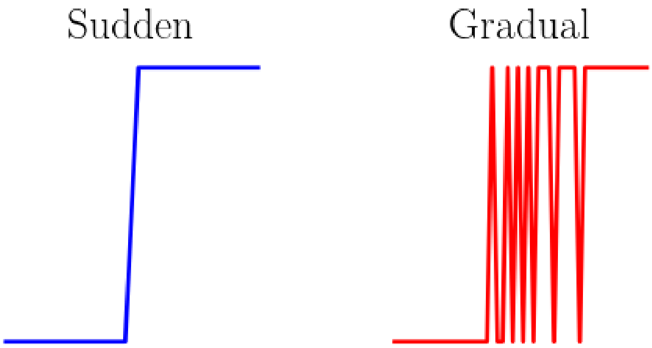


Figure 2.3: Types of Concept Drift, adapted from (FOUCHÉ, 2017)

2.3.2 Detecting Changes

One of the most common techniques to detect a change in the data distribution is to monitor the error rate of the learning algorithm. The Probably Approximately Correct Learning (PAC Learning) theory (VALIANT, 1984) explains that a concept is **approximately** correct, over a certain data distribution D, if the model error is below a certain ϵ and, it is **probably** correct if the probability of an algorithm generates a classifier that fits the previous condition is higher than $1 - \delta$. Therefore, if the data distribution is stationary, i.e., without concept drifts, an adaptive algorithm should be continuously increasing its predictive capacity as more data arrives, and consequently, its error rate should decrease or at least keep below ϵ. If a significant variation in the error rate occurs, the model might not be probably approximately correct anymore, which could possibly mean that drift in the concept occurred. This method is known as Drift Detection Method (DDM) (GAMA et al., 2004). There are other methods that are based on the same idea, such as Early Drift Detection Method (EDDM) (JOSE et al., 2006) and Reactive drift detection method (RDDM) (BARROS et al., 2017).

Another existing approach to detect changes is by continuously estimating the probability distribution values. A drift is detected when a significant change in the value of the estimated parameters happens. In this category, the most representative method is the Page-Hinkley (PH) test (PAGE, 1954). A variant of this approach is to monitor these distributions using more than one window of data, comparing reference data from old instances with newly arrived data, if there is a significant difference, a drift might have happened. Examples of methods following this approach are ADWIN (BIFET; GAVALDÀ, 2007) and Hoeffding Drift Detection Method (HDDM) (FRIAS-BLANCO et al., 2015).

The last approach used to detect changes in data is the contextual approach. Methods in this category combine incremental learning and forgetting techniques in the detection process, using an estimated window. A representative method in this category is the SPLICE method (HARRIES; SAMMUT; HORN, 1998).

A deeper explanation of each method and the main advantages of each approach is not under the scope of this project. For this analysis, the reader is referred to (DUONG; RAMAMPIARO; NØRVÅG, 2018).

2.3.3 Forgetting

The forgetting approach for dealing with concept drifts consists of disregarding old instances in the predictive model (KOYCHEV, 2004). This is an implicit way of dealing

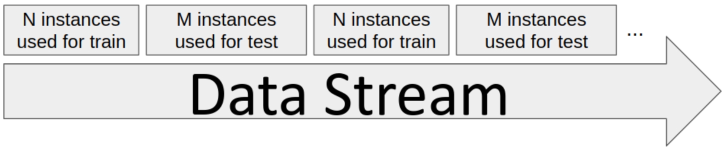

N instances used for train
M instances used for test
N instances used for train
M instances used for test
...
Data Stream

Figure 2.4: Periodic Holdout

with changes in the data distribution, as the most recent data is considered in detriment of older data.

One of the challenges with this type of approach is to determine for how long should the model buffer instances. If the threshold for the forgetting to start is big, the model should be more robust when no drift is present, however, when drift occurs, the instances belonging to the outdated distribution are going take longer to be disregarded. On the other hand, if the threshold is small, the model should quickly adapt to drifts, but in problems with no drift, the model might not be as robust. This is referred to as the stability-plasticity dilemma.

2.4 Validation in Data Stream Mining

In streaming scenarios, data is continuously arriving, and thus, the separation between training and test data may be unclear. Therefore, specific validation processes should be considered in streaming scenarios. Two of the most commons are the Periodic Holdout and Interleaved Test-Then-Train or Prequential (BIFET et al., 2010). It is noteworthy that both these methods are based on the test-then-train principle, which ensures that the instances were used for testing **before** they are used for training. This is important because it avoids a potential bias on the classifiers predicting instances already seen.

2.4.1 Periodic Holdout

Since the quantity of instances is uncertain in data stream mining problems, it is not possible to apply the normal offline holdout method. The periodic holdout protocol consists of a common holdout repeated over time until there are no more instances arriving. Since a percentage division for train and test is not possible, an absolute number of instances for each part is set. As presented in Figure 2.4, when the stream starts flowing, the first N instances are used only for training the model, without evaluation. After this step is completed, the next M instances are used solely for testing. Once this cycle is over, the process is repeated.

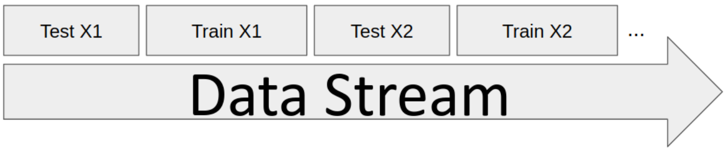


Figure 2.5: Prequential

2.4.2 Prequential

The idea of prequential (GAMA; SEBASTIÃO; RODRIGUES, 2013) is to assess instances individually in a test-then-train fashion. For each instance that arrives from the stream, it is used first for testing the model, then for training it, as shown in Figure 2.5. This order is important because it guarantees that the model will only be evaluated with instances that it has never been trained on. Therefore, it creates a solid representation of how the accuracy evolved over time, and how it behaved when certain known events occurred, e.g., concept drifts. As stated by the authors in the original paper, the performance of this protocol converges to a holdout estimate.

Prequential can also be executed with a ponderation applied upon the instances, so they have different weights when computing the evaluation metrics (e.g, accuracy). It was originally designed to work with one instance at a time, however it can also receive chunks of instances. The rationale remains the same, the chunk is first used to test, then to train.

2.5 Data Stream Generators

Data stream generators are common tools utilized when validating methods in data stream mining. Their purpose is to generate random instances following a certain data distribution. First, a random vector of features x is created and is labeled according to a function $f(x)$.

For example, in Figure 2.6 the descriptive features of the instances are x_1 and x_2, with both values ranging from 0 to 1. The target feature is “empty circle” or “full circle”. The decision boundary between the classes is defined by the function $f(x) = x$. The generation of an instance occurs as follows: a random pair x_1 and y_2 is generated, then, if the pair is above the decision boundary, it is considered of the class “full circle”, otherwise “empty circle”.

Generators are widely used by the scientific community because of the vast control and reproducibility they provide for experiments. Most generators have more than one function to be selected as a parameter, making it easy to simulate concept drifts at any desired time. In this project, four generators are described in detail.

2.5.1 Streaming Ensemble Algorithm - SEA

The SEA generator (STREET; KIM, 2001) creates a random feature vector with tree elements $\{x_1, x_2, x_3\}$, but only two of them ($\{x_1, x_2\}$) contribute to describing the instance, while the third one's purpose is to create noise. Each of the features is bounded in the [0;10] interval. The definition of the target feature once the vector of descriptive features is done follows a linear threshold. If $x_1 + x_2 > \theta$ then it belongs to class "A", else to class "B". The value of θ is variable and changing it in the middle of an algorithm execution creates a concept drift.

2.5.2 AGRAWAL

The AGRAWAL generator (AGRAWAL; IMIELINSKI; SWAMI, 1993) creates data that are separated in two groups. It creates instances with a total of nine descriptive features, with three of them being categoric and the other ones numeric, as seen in Table 2.1. The target feature can assume "Group 1" (defaulting customers) and "Group 2" (non-defaulting customers). The features are generated randomly, and there are ten different functions for determining the class of the instance once the features were generated. Changing the function creates a drift in the concept to be learned.

In order to introduce noise into the data generation process, the generator receives as parameter a perturbation factor p. After the features were generated and the class was

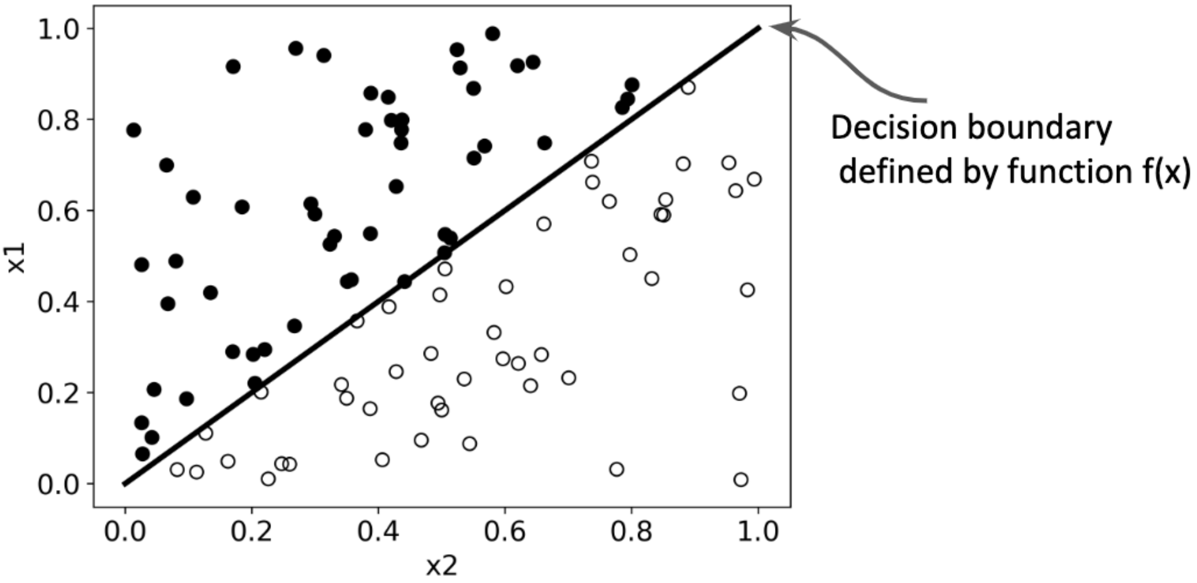


Figure 2.6: Data Distribution Following a Function

Table 2.1: Agrawal Generator Features, adapted from (AGRAWAL; IMIELINSKI; SWAMI, 1993)

Feature	Type	Value Range
salary	Continuous	Random in 20000 to 150000
commission	Continuous	0 if salary $>=$ 75000 else random in 10000 to 75000
age	Continuous	Random in 20 to 80
elevel	Categoric	{0, 1, 2, 3, 4}
car	Categoric	{1, 2, ..., 20}
zipcode	Categoric	$\{zipcode_0, zipcode_1, ..., zipcode_8\}$
hvalue	Continuous	Random in $\frac{k}{2} \times 100000$ to $\frac{3k}{2} \times 100000$ for k = zipcode index
hyears	Continuous	Random in 1 to 30
loan	Continuous	Random in 0 to 500000

assigned, the values of the continuous features are modified accordingly to this parameter. For each feature, the formula $v + r \times p \times a$, is applied, where v is the initial generated value for the feature, r is uniformly distributed between -0.5 and 0.5 and a is the range of values that the feature can assume.

2.5.3 Random Tree Generator - RTG

The Random Tree Generator (DOMINGOS; HULTEN, 2000) is a highly flexible generator as the number of continuous and categorical features, as well as the number of target features, are user-given parameters. The generator first spawns a decision tree based on random splits on attributes and random definitions of class for each leaf. After the tree is built, the instances are created by uniformly generating attributes for the instances and, defining the class by traversing the tree. To simulate concept drifts, a new tree must be created with a different random seed.

2.5.4 Asset Negotiation Generator

The Asset Negotiation Generator (ENEMBRECK et al., 2007) simulates instances with negotiation proposals. All the five descriptive features are categorical, as given in Table 2.2, and the target feature is a judgment if the offer is “interesting” or “not interesting”. It has five different functions to simulate changes in the concept.

2.6 Real-World Datasets

Stream generators are widely accepted by the scientific community. However, in order to fully validate that a method is suited for deployment in a real world environment, it must first be tested with real-world datasets. In this section we explain three datasets that are commonly used in streams.

2.6.1 NOMAO

The NOMAO dataset (CANDILLIER; LEMAIRE, 2013) was presented as a challenge. Each instance consists of data about two places, and the class is if they are the same place or not. The objective of the task is to train a model capable of deduplicating instances from a search engine. This dataset consists of 34,465 isntances with the dimension of 118.

2.6.2 Spam Corpus

The Spam Corpus dataset (KATAKIS; TSOUMAKAS; VLAHAVAS, 2006) consists of 9,324 instances with dimension 39,917. The features are all boolean values that represent if the instance, which is an email, contains a word or not. The class is if the email is a spam or not.

2.6.3 Electricity

The Electricity dataset (RODRIGUES; GAMA; PEDROSO, 2008) was extracted from raw data from sensors of multiple electrical stations. It contains 45,312 instaces with dimension 8. The label is if the electricity price will increase (up) or decrease (down).

Table 2.2: Asset Negotiation Generator Features, adapted from (ENEMBRECK et al., 2007)

Feature	Value Range
Color	{black, blue, cyan, brown, red, green, yellow, magenta}
Price	{very low, low, normal, high, very high, quite high, enormous, non salable}
Payment	{0, 30, 60, 90, 120, 150, 180, 210, 240}
Amount	{very low, low, normal, high, very high, quite high, enormous, non ensured}
Delivery Delay	{very low, low, normal, high, very high}

2.7 Base Classifiers for Data Streams

As previously mentioned, classifiers or models are normally built based on labeled data provided for training. Either in the offline or online machine learning, the data itself is not needed anymore once the classifier is built or updated. There are many different techniques to build classifiers, and in this section, three different methods for data stream classification (that allows the model to be updated) are described in detail.

2.7.1 K-Nearest Neighbors

K-Nearest Neighbors for data streams is a method that works almost the same as its offline version (FIX; HODGES, 1989). It does not have a building process for the model, and all training instances are stored. When an instance x for prediction arrives, its K most similar instances in the training set are selected. Next, the most common class in these K instances are selected as the predicted class for x.

The similarity is defined by a distance function, and this function is applied to x with every instance on the training set. Therefore, the instances with the lowest values are the most similar to x. The most common distance function used is Euclidean distance, which can be seen in a two or three-dimensional context as a straight line.

The search for the K-closest instances can also be optimized with a pre-processing step on top of the training set (BENTLEY, 1975). KD-Tree subdivides the training instances into subspaces, and thus, when predicting instances, distances are only computed on the group that most likely contains their similars. This subdivision is often done by discriminating the instances using as threshold the median of each feature values. For example, let a dataset D with instances with two dimensions x_1 and x_2. First, the instances would be separated accordingly to the median value of x_1. We would then have two groups of instances, then, these groups would be divided regarding the median value of x_2 for each group. This creates a tree structure, and when searching for the neighbors, the euclidean distance search would only have to be performed on the group where the instance lands.

While KD-Tree provides a great performance increase regarding time compared to traditional KNN, its results may not be as accurate as the latter. However, KD-Tree usually provides a satisfactory trade-off between statistical accuracy and processing time. There also other optimizations of KNN such as Ball-Tree (OMOHUNDRO, 2009).

In the online version of this algorithm, a sliding window size N is received as a user-given input. When new instances arrive, they begin filling the window. Once its maximum size is reached, every time a new instance arrives, the oldest one is forgotten. The definition of the size of the window recalls the plasticity-stability dilemma. With bigger windows, more processing is needed at every prediction as a larger number of distance computation occurs. On the other hand, with smaller windows, less robust the model is, but considering concept drifts, they can adapt faster.

2.7.2 Naive Bayes

The Naive Bayes algorithm for data streams is almost identical to the offline version (BISHOP, 2006). It is an algorithm based on the Bayes Theorem given in Equation (2.5).

$$P(y|\vec{x}) = \frac{P(\vec{x}|y) \times P(y)}{P(\vec{x})} \tag{2.5}$$

This theorem calculates the conditional probability of y happening, given that $\vec{x}$ happened. The training part of the algorithm consists of gathering a series of statistics of the training set, in order to compute the statistics required during the prediction step.

During the prediction step, the exact same idea of the theorem is followed. What is the probability of the instance i belonging to the class y, given that it has the following set of features $\vec{x}$? This probability is calculated for all the possible classes, the one with the biggest probability is set as the prediction.

One particularity of the Naive Bayes is that it assumes that all of the features are independent of one another. Although this is not always true, in many cases, this should not be a problem, because even though the probabilities values might be biased, it may not affect the final prediction (in a binary problem, a probability of 70% or 80% will output the same class).

2.7.3 Hoeffding Tree

A decision tree is a type of classifier that builds its model following the structure of a tree. Figure 2.7 illustrates an example of a decision tree. Each node of the tree is a test concerning an attribute. For example, the first node, also called root, when predicting will test if an instance has the attribute "age" is less than 30. If the condition is true, the instance will walk down the tree to one of the sides, in this case, the left, and then the same protocol is repeated until it reaches one of the tree leaves (node that does not have children). Each leaf of the tree will contain a class, which is the prediction for the instance that reaches it.

There are plenty of ways to build a decision tree, but the idea of selecting an

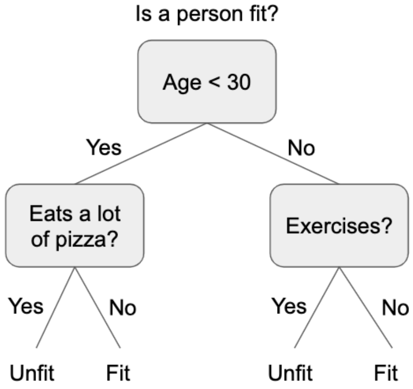


Figure 2.7: Example of Decision Tree, adapted from (GUANGA, 2019)

attribute to a node split is that it is, at a given moment, the more useful in terms of separating different classes. The Hoeffding Tree (DOMINGOS; HULTEN, 2000) algorithm was the first of this type of algorithm that allowed online learning, as previous batch algorithms needed to load the whole dataset in the memory or scan it sequentially multiple times, not fulfilling a stream mining requirement.

The Hoeffding Tree proved that with just a few instances and a confidence level, it was possible to determine the best attribute to assign to a node. The chosen attribute would, in most cases, be the same as if it was chosen based on infinite instances. This allowed the algorithm to process the instances just once, without the necessity of storing them. That way, the tree built by this algorithm should be almost identical to the ones built with offline decision trees algorithms. However, there are studies that challenge this claim, such as (MATUSZYK; KREMPL; SPILIOPOULOU, 2013), but the original algorithm is still the most used version, given its positive results.

Several adaptations of the original Hoefding Tree algorithm were made, such as the Hoeffding Option Tree (PFAHRINGER; HOLMES; KIRKBY, 2007), which generates new nodes as optional paths in the tree. There is also the Vertical Hoeffding Tree (KOURTELLIS et al., 2016), which is a distributed algorithm that allows to deal with enormous datasets. It also noteworthy citing the Extremely Fast Decision Tree (MANAPRAGADA; WEBB; SALEHI, 2018), which unlike the original algorithm, it revisits the node splits to make changes in the decision boundary, in order to make the tree more similar to its offline version. In this work, only the traditional Hoeffding Tree is considered, because it is the currently most used version in scientific experiments.

2.8 Dynamic Ensembles

An ensemble of classifiers is a diverse combination of multiple models trained to solve the same problem (OPITZ; MACLIN, 1999). The basic idea is that each element of the ensemble will be trained to be competent in a different area of the problem. Therefore, when appropriately combined, all elements should come up with a better solution than one individual element. Nonetheless, it is not guaranteed that an ensemble will perform better than an individual classifier. The more diverse an ensemble is, the higher are the chances that the members will complement each other, therefore increasing the performance metrics over single classifiers (SAMMUT; WEBB, 2010). This diversity is especially important when related to the classification errors of the elements, the classifiers should make different mistakes in a successful ensemble.

The type of classifier that composes an ensemble is called the base classifier. Technically, every type of classifier can constitute an ensemble, but an ensemble will usually present better results if the base classifiers are unstable (SAMMUT; WEBB, 2010). It is also important to highlight that some ensemble methods require specific base classifiers. Unstable base classifiers are those in which small changes in the training data cause the learning concept to vary widely. An example of an unstable classifier is a decision tree.

There are many different techniques to train an ensemble. One possible way is to randomly distribute instances to the base classifiers (BREIMAN, 1996). The ensemble can also be built by training each base classifier based on the previous errors of others already trained members (BREIMAN, 1996). Another method is to randomly select the features that each base classifier will take into consideration when training itself (HO, 1998). Among many other methods described in the literature, most of them are usually based on one or more of these three. When it comes to predicting, they all resort to a combination rule.

Every element of the ensemble E will have a vote about what the prediction should be, and the combination rule takes all of these votes and combines them into one final prediction. The most common and simple rule is the Majority Vote, given in Equation 2.6, where $\hat{y}$ is the predicted class, T is the number of classifiers in the ensemble and v is the vote of each classifier as an array with one in the predicted class index and zero in all other indexes.

$$\hat{y} = argmax \sum_{i=0}^{T} v_i \tag{2.6}$$

For example, in a binary problem, if a classifier predicts the instance as the class "zero", v would be equal to $\{1, 0\}$. The votes of every classifier are summed into a single array, and the `argmax` function selects the index of the class with the highest value. In other words, this method selects the class that was most voted by the classifiers. There is also the weighted vote, in which the vote of each classifier is weighted by a metric, e.g., the accuracy of the individual classifier.

There is an step that can happen before the vote combination, which is the dynamic classifier selection, which is the main focus of this work. A detailed discussion on this area is presented in Chapter 3.

There are three main reasons that justify the use of ensembles in machine learning (DIETTERICH, 2000). The first one is statistical as what a machine learning algorithm aims is to find a hypothesis h that explain the relations between the descriptive features x and the class y. In other words, it seeks to find the best approximation to the real function $f(x)$. Sometimes, especially when few instances are available to train the model, many different hypotheses $\{h_1, h_2, \ldots, h_n\}$ can be found in the whole space H with the same accuracy on the training set. This does not mean they will have the same accuracy when predicting new instances. Hence, instead of picking one hypothesis by chance, combining them brings the hypotheses to a mean, increasing the chance of a better generalization. This is represented in Figure 2.8, top-left.

The second reason is computational. When searching for the best hypothesis, an algorithm might find what is actually a local optimum. For some algorithms, it is very costly, or even impractical, to perform a global search. Consequently, combining multiple local optima increases the probability of a mean hypothesis performing better than a single one. This can be seen in Figure 2.8, top-right.

The last reason is representational. It is possible that the hypotheses found by classifiers do not well represent the function $f(x)$. One reason this might happen regards limitations in the algorithm, e.g., a linear classifier for a non-linear problem. Another reason might be incomplete or non-descriptive enough data. Therefore combining hypotheses might create a mean hypothesis that no individual classifier is capable of achieving. As demonstrated in Figure 2.8, bottom.

In the following sections, different ensemble learning techniques tailored for data stream learning are presented.

2.8.1 OzaBag

The Online Bagging or OzaBag (OZA, 2005) is an ensemble method for data streams based on the popular offline ensemble method Bootstrap Agreggating or Bagging

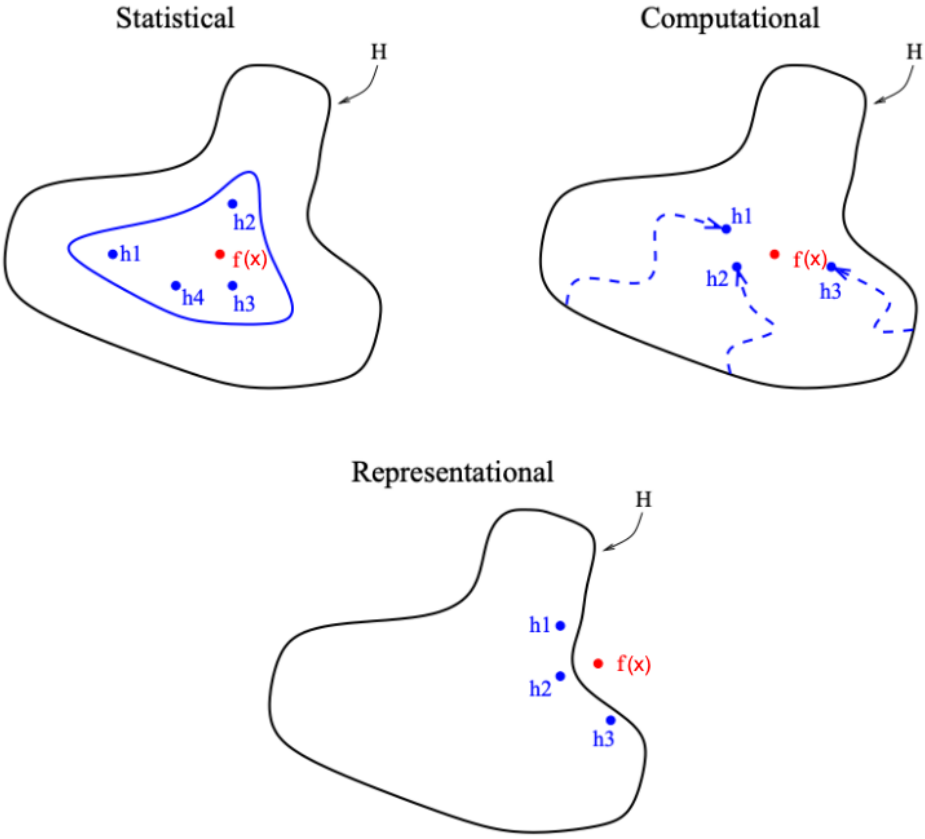


Figure 2.8: Reasons for the Adoption of Ensembles, adapted from (DIETTERICH, 2000)

Table 2.3: Probability of Numbers 0 to 5 on Poisson Distribution with $\lambda = 1$.

Number	Probability
0	0.36788
1	0.36788
2	0.18394
3	0.061313
4	0.015328
5	0.0030657

(BREIMAN, 1996). The traditional Bagging works by creating N "bags" of data, with replacement, with the same size K as the original dataset. Then, each of the bags is used to train a base classifier. On the prediction step, a simple majority vote is used.

One major difficulty when adapting the bagging algorithm to the online environment is that the whole dataset is not available at the same time. Therefore it is impossible to determine the size of the bags and create the bags. To overcome this issue, to each arriving instance, for each base classifier, a random number k following the distribution of Poisson with $\lambda = 1$ is generated. This k will be the number of times the instance will be used for training a base classifier. Table 2.3 illustrates the probability of each number from zero to five to be generated (above five, the probability is practically negligible).

2.8.2 OzaBoost

OzaBoost (OZA, 2005) is the first online adaptation of the AdaBoost method (SCHAPIRE, 1999). The rationale behind these methods is to create a sequential ensemble of learners that are trained based on the errors made by the last learner.

The training step of OzaBoost works as follows: For a training instance x_i, the first member c_0 of the ensemble trains the instance in the same way as OzaBag with $\lambda = 1$. If the instance is misclassified by c_0, the λ value is increased for that instance in the next member c_1. This process repeats for the whole ensemble. In that way, if the last member does not well classify the instance, the next member will give more attention to it, because the higher the λ, the higher the probability of the instance being sampled more times.

2.8.3 OzaBagAdwin

OzaBagAdwin (BIFET et al., 2009) combines the Online Bagging with the change detector ADWIN (BIFET; GAVALDÀ, 2007), described in Section 2.3.2.

The way it works is identical to OzaBag, but with one addition, each member of the ensemble has an ADWIN detector attached. When ADWIN detects a change, the

Table 2.4: Probability of Numbers 0 to 11 on Poisson Distribution with $\lambda = 6$.

Number	Probability	Number	Probability
0	0.0024788	6	0.16062
1	0.014873	7	0.13768
2	0.044618	8	0.10326
3	0.089235	9	0.068838
4	0.13385	10	0.041303
5	0.16062	11	0.022529

classifier is replaced by a new one.

2.8.4 Adaptive Random Forest

Adaptive Random Forest (ARF) (GOMES et al., 2017) is an adaptation of the well-known method for offline environments Random Forest (BREIMAN, 1996). The original Random Forest creates an ensemble of decision trees that are trained not only using the Bagging method but also randomizing which features each tree will take into consideration when splitting the nodes of the trees. This is expected to create a diverse ensemble because the trees will only be trained and construct a hypothesis between correlations of the features selected to it at that split.

Focusing on the Adaptive Random Forest, it initially creates bags for each tree in a similar way to OzaBag. The only difference is that instead of Poisson with $\lambda = 1$, it uses $\lambda = 6$. This drastically decreases the probability of an instance not being selected and increases the probability of it being selected with a higher resampling rate, as illustrated in Table 2.4. To adapt the selection of features in tree splits, ARF modifies the Hoeffding Tree (DOMINGOS; HULTEN, 2000), which was described in 2.7.3. In ARF, the Hoeffding Tree algorithm does not prune the trees, and when a tree split is done, only a subset of features are taken into consideration.

ARF also includes two detectors for each tree, but it does not simply reset a tree when a drift is detected, as OzaBagAdwin does. ARF sets one of the detectors with a low and one with a high level of confidence. When the one with low level is triggered, it is treated as a warning for a new tree to start to be constructed in the background. When the drift is actually detected (detector with a higher level of confidence is triggered), the tree is replaced for the new one that was already being trained on.

2.8.5 Kappa Update Ensemble

Kappa Updated Ensemble (KUE) (CANO; KRAWCZYK, 2019) is a method based on the Kappa metric. This metric indicates how well the classifier performs when comparing to pure statistical probability, given the current state of the stream. The formula for computing Kappa is illustrated in Equation 2.7, in which acc_o is the currently observed accuracy and acc_e is the currently expected accuracy, given the distribution of samples.

$$\text{Kappa} = \frac{acc_o - acc_e}{1 - acc_e} \tag{2.7}$$

This is especially useful when dealing with imbalanced datasets, for example, a distribution of classes of 99% to 1%. If the classifier simply guesses the predictions based on the probability of being of the first class, it might get an accuracy close to 99%. Hence, if a trained classifier gets the same accuracy, it does not necessarily mean it is a good result. Therefore, Kappa, in this case, is a more meaningful metric.

In constrast to the methods described above, KUE works with chunks of data. When the first data chunk arrives, the instances are distributed to each classifier to be trained using Online Bagging with $\lambda = 1$. Besides, what and how many features each classifier will take into consideration when training is also randomized for each ensemble element. This is similar to what happens in ARF, but in this case, the number of features is also randomized, and the subset of features is selected to the whole classifier, not only for the node splits. After the training step, the Kappa for each classifier is computed on the chunk. This is interpreted, in this algorithm, as the competence of the classifier.

When new data chunks arrive, the ensemble needs to be updated. The new chunk is sampled according to the Online Bagging process. Next, the Kappa (competence) is recalculated only on the last arrived chunk.

As a dynamic ensemble, however, the algorithm is expected to deal with possible drifts in the concept. In order to eliminate the need for a drift detector, after the update step, before accepting the next chunk, a new ensemble of q (which is passed as a parameter) is created using the initialization step with the last chunk. Then if the competence (Kappa) of the newest classifiers is better than competence of any member in the main ensemble, the weakest classifiers are replaced with the new ones.

In the prediction step, the vote of each classifier is weighted by its Kappa on the last chunk. The vote is only taken into consideration if the Kappa is equal or bigger than zero, which enforces that only the most well-adapted elements to the current data distribution contribute to the final decision.

2.9 Frameworks for Data Stream Mining

There are two major frameworks used by the scientific community for Data Stream Mining: Massive Online Analysis (MOA) and Scikit-Multiflow.

2.9.1 Massive Online Analysis - MOA

MOA (BIFET et al., 2010) was developed in the University of Waikato as a complement to the popular framework for offline machine learning, WEKA (HALL et al., 2009). It was developed as an environment for researchers to implement new methods and test them with the state-of-art, as well as for users to apply data stream mining algorithms in their own data.

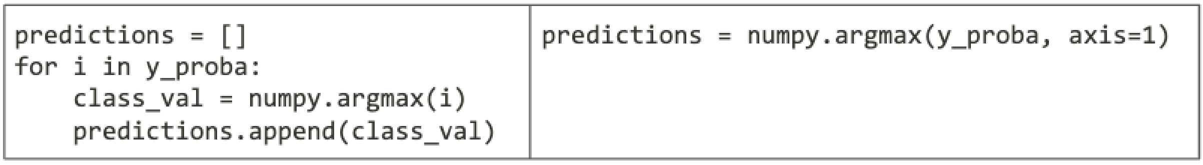
MOA gives access to a great number of machine learning algorithms, especially when considering the ones available in WEKA. It was developed in Java, and it is fully open-source and easily extendable. It also includes several data stream generators, and it allows input of ARFF (Attribute-Relation File Format) files as the data stream.

2.9.2 Scikit-Multiflow

Scikit-Multiflow (MONTIEL et al., 2018) is based on MOA, but it follows the base structure of the popular Python machine learning framework Scikit-Learn (PEDREGOSA et al., 2012). It focuses not only on applying data stream mining algorithms but also on methods that involve multi-output responses (when the target feature y is a vector and not a single value).

Scikit-Multiflow is also fully open-source and easily extendable. Since it is more recent and not widely embraced by the scientific community yet, it still lacks some methods and functionalities that are present in MOA, such as some stream generators and specific adaptations of traditional methods that are directly implemented in MOA by their authors. It is written in Python, which presents an advantage over MOA since Python is considered to be more beginner-friendly than Java (PELLET; DAME; PARRIAUX, 2019). This could help a faster adoption of the framework in the near future.

Python is also known by its various scientific tools that allow computations in high performance. However, currently, MOA presents better performance than Scikit-Multiflow. This is because the implementation of the data stream algorithms in Scikit-Multiflow was highly inspired in the MOA implementation. However, in an extremely high-level language such as Python to present good performance, non-conventional techniques must be applied. For example, in Figure 2.9 left a real code extracted from the



```
predictions = []
for i in y_proba:
    class_val = numpy.argmax(i)
    predictions.append(class_val)
```

```
predictions = numpy.argmax(y_proba, axis=1)
```

Figure 2.9: Code Adapted from Scikit-Multiflow repository (Left) and Possible Optimized Version (Right)

Table 2.5: Time Processing of Original Code versus With Possible Optimization

Original	Optimized
3.3678 sec.	0.0331 sec.

library[2] is displayed. In it, a loop through an array of arrays (matrix) is performed applying the function `argmax`[3], from the library Numpy (OLIPHANT, 2006), to each element. However, Numpy is a library written in low-level languages, such as C and Fortran, it provides an interface for Python to access highly-optimized array functions in much less time. The same loop can be rewritten as the code presented in Figure 2.9 right, with expressive differences in processing time, as presented in Table 2.5[4].

2.10 Final Considerations

This section presented the fundamental concepts for understanding the next sections of this document. The main ideas of machine learning and its extension to data stream mining were briefly described. In the next section, dynamic classifier selection (DCS) will be presented. Even though DCS is part of the machine learning toolkit, the next section is devoted specifically to it since it is at the core of this work.

[2] https://github.com/scikit-multiflow/scikit-multiflow/blob/master/src/skmultiflow/bayes/naive_bayes.py

[3] Function that returns the index of the element with the higher value in an array.

[4] The experiments were run with a matrix randomly generated, with one million rows and five columns. Each timing was executed ten times.

Chapter 3

Dynamic Classifier Selection

One of the most important aspects to set up when using an ensemble method is the combination method. Recapitulating, the combination method is how the predictions of all classifiers in an ensemble are combined to a single prediction. As explained in Section 2.8, two common methods for combining predictions are the majority vote and the weighted vote.

Besides the combination method, there are also methods that select which classifiers are going to be used to predict, these are called *Selection Methods*. The idea of this approach is to identify which members of the ensemble are more "competent" to classify a specific piece of data. There are two types of selection, static and dynamic. The static selection aims to select which classifiers were most successful during the training phase of the ensemble. This is not suitable for data streams, at first, for some reasons. The first one is that there is not a defined training phase in data stream algorithms, which prevents the functioning of traditional selection methods. Another reason is that classifiers are not static in this context, they are constantly updated and can be replaced, also, they are subject to concept drifts.

Dynamic Classifier Selection aims to select which members of the ensemble are more competent to predict each instance individually. As illustrated in Figure 3.1, for each instance that arrives to be predicted, a selection logic selects from the ensemble which elements shall be used to predict the instance. Then, from the selected ensemble, a prediction is outputted.

Recent literature divides selection methods into two types: Dynamic Classifier Selection (DCS) and Dynamic Ensemble Selection (DES). The difference between the groups is that the former selects a single classifier to perform the prediction, while the latter selects a subset of the ensemble (which in some cases may also contain just one classifier or even the entire ensemble). Since the second group might have more than one

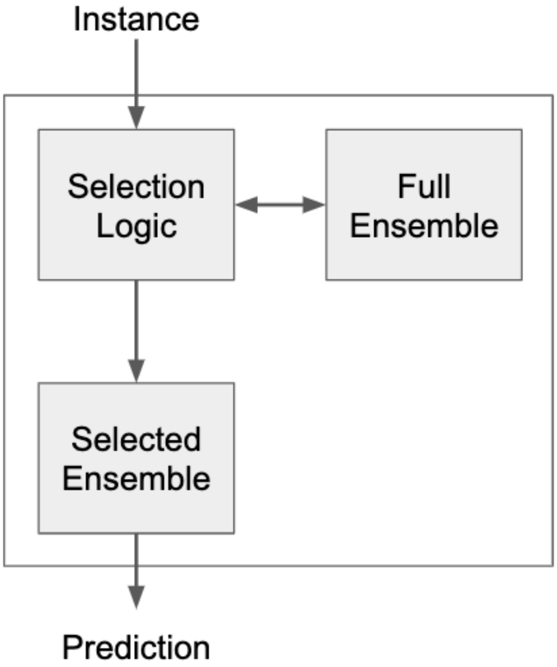


Figure 3.1: Dynamic Selection of Classifiers

classifier, it still needs to apply a combination rule as a final step to the prediction.

A great tool that helps clarify if it makes sense to apply Dynamic Selection in an ensemble is the Oracle (KUNCHEVA, 2000). The idea is to check if there is at least one member of the ensemble that correctly predicted the instance. If there is, the whole ensemble is set to as correctly predicted the instance. This is obviously not suitable for real problems since it depends on having the class of the instances to predict. The sole purpose of this method is to check if there is room available for increasing the performance of the ensemble. In other words, if the Oracle considerably outperforms a normal execution, it means that there are classifiers that are predicting correctly when the majority is predicting wrongly, thus it is expected that a good selection method will bring benefits.

Dynamic selection for the offline machine learning environment has a vast literature. Most of the methods described in this section have very similar structures. The protocol starts with separating part of the train set to be a validation set. Then, the ensemble is usually constructed using the new train set. On the prediction step, to every instance to be predicted, a search to its most similar instances on the validation set is performed using the K-Nearest Neighbors (FIX; HODGES, 1989). Since these instances are close to the instance to be predicted, their behavior on each classifier of the ensemble is analyzed to determine which elements will be selected. This analysis is where the difference between the methods resides.

For the understanding of the methods KNOP and META-DES, both explained in Section 3.1, it is necessary the understanding of what an output profile stands for. Notation-wise, the output profile of an instance is an array $\{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n}\}$ in which each element is the class predicted by each one of the n classifiers of the ensemble. For example, an output profile of $\{1, 0, 1\}$ means that a given instance was predicted by the first classifier of the ensemble as of class 1, predicted as class 0 by the second classifier and as class 1 by the third classifier.

There are few methods concerning the application of Dynamic Selection of Classifiers in online machine learning. In this section, it is presented the main methods for dynamic classifier selection, divided by offline and online machine learning methods, thus, highlighting this gap.

3.1 Methods for Offline Machine Learning

In this section, the DCS methods considered to be the most representative to the state-of-art for batch classifiers will be presented.

3.1.1 A Priori and A Posteriori

The A Priori method (GIACINTO; ROLI, 1999) provides a selection method based on probabilities. It gathers the most similar instances in the validation set to the instance to be classified using K-Nearest Neighbors. Then, for each classifier c, it sums the probability outputted by c of each neighbor being of the class they really are. In other words, this creates a metric that represents how accurate each classifier is the subset of neighbors. The classifier with the highest value for this metric is selected for prediction. The sum is weighted by the Euclidean distance of each neighbor with the query instance. The classifier is only selected if the difference in the metric is significantly better than the other classifiers. This is controlled by a user-given threshold.

The A Posteriori method (GIACINTO; ROLI, 1999) is similar to A Priori. The difference is that it only takes into consideration when computing the metric for each classifier c the neighbors that belong to the class predicted by c, for the instance to be classified.

3.1.2 DCS-LA

DCS-LA (Dynamic Classifier Selection - Local Accuracy) (WOODS; JR; BOWYER, 1996) is a family of algorithms for the Dynamic Classifier Selection approach. The algorithms are Local Class Accuracy (LCA) and Overall Local Accuracy (OLA). As the names suggest, both methods are based on the local accuracy concept, and they select a single classifier to make the prediction of each test instance.

The Overall Local Accuracy works as follows: the K neighbors of the query instance are gathered. Next, the algorithm computes the accuracy of each classifier regarding only the neighbors returned. It then proceeds to pick the most accurate classifier to predict the instance.

The Local Class Accuracy method (LCA) is very similar to the Overall Local Accuracy (LCA), but it focuses on the ability of the classifier in the class it predicted the query instance. The idea is to answer the question: what classifier is the most accurate with respect to the class they predicted the instance? Only the neighbors of the same predicted class are returned, thus, the neighbors might vary for each classifier of the ensemble. It then computes the percentage of neighbors that were correctly predicted by each classifier, and the prediction of the classifier with this higher measure is picked.

In both algorithms, if two or more classifiers share the higher accuracy values, the most recurring class amongst the tied classifiers is selected. If the tie persists, a third classifier is taken into consideration.

3.1.3 DCS-RANK

The DCS-RANK method (SABOURIN et al., 1993) might be considered one of the first works in dynamic selection. The idea is to rank the classifiers by competence and selecting the first position of the rank to predict the instance. The method starts with gathering the neighbors using K-Nearest Neighbors, and these are sorted according to the Euclidean distance to the instance to be predicted. Next, the rank of competence of the classifiers is built, based on the number of consecutive correct predictions each classifier made on the list of neighbors.

There is also a proposed modified version of this method (WOODS; JR; BOWYER, 1996). This version includes the concept of Local Accuracy, previously described in Section 3.1.2. When an instance arrives to be predicted, for each classifier, the nearest instances that belong to the class that was predicted by the classifier are returned. Finally, the neighbors of that class are sorted, and the rank is computed.

3.1.4 KNORA

KNORA or K-Nearest-Oracles (KO; SABOURIN; BRITTO JR., 2008) is a family of dynamic classifier selection algorithms. Unlike its predecessors, which focused on selecting the single best classifier of the ensemble, all of the algorithms of the KNORA family select a subset of classifiers from the ensemble per test instance. The algorithms in the family are KNORA-ELIMINATE, KNORA-UNION, KNORA-ELIMINATE-W, KNORA-UNION-W.

All algorithms share the first step as the common K-Nearest Neighbor search is performed on the validation set on the instance to be predicted. Once the neighbors are gathered, each algorithm behaves differently.

KNORA-ELIMINATE looks for classifiers in the ensemble that classified all of the K-nearest neighbors correctly. If there is no classifier with such level of accuracy in the neighbors, the classifiers that got $(K - 1)$ instances corrected are selected, and subsequently decreasing K until at least one of the classifiers is selected.

KNORA-UNION is similar to KNORA-ELIMINATE, but instead of expecting all of the neighbors to be correctly classified, it selects the classifiers that predicted at least one of the instances right. The classifiers that predicted more neighbors correctly have more votes on the final combination rule.

KNORA-ELIMINATE-W and KNORA-UNION-W use the same mechanism as KNORA-ELIMINATE and KNORA-UNION, but they take into consideration the Euclidean distance of each neighbor from the given test instance as a weight for the influence of the neighbor in the decision.

3.1.5 K-Nearest Output Profiles

K-Nearest Output Profiles (KNOP) (CAVALIN; SABOURIN; SUEN, 2013) is similar to the KNORA-U, but instead of searching for the most similar instances in the validation set using the features of the instances, it searches for the most similar output profiles in the validation set. The next step follows KNORA-U, i.e, if the classifier correctly predicts at least one of the neighbors, it is then picked to classify the instance and the more correctly predicted neighbors, the more votes the classifier will have in the final prediction.

For example, in a binary problem, with classes “a” and “b”. Given the instance x as the instance to be predicted, in an ensemble of five classifiers, the output profile, or in other words, the predictions of each member for the instance is {a, b, a, a, b}. The first step is to find the K instances in the validation set that were most similarly predicted by each member. With the instances selected, the classifiers that correctly predicted at least one of them are selected.

3.1.6 Multiple Classifier Behavior

The Multiple Classifier Behavior (MCB) (HUANG; SUEN, 1995) is a mixture of the KNOP and the OLA techniques. It first gathers all the K similar instances to the query instance using K-Nearest Neighbors. It then computes the similarity of the output profile of the neighbors with the query instance. The neighbors with a similarity below a threshold are removed from the neighbors set, thus the size of the set might vary from instance to instance. After this step, the local accuracy of each classifier in the set is computed, and the classifier with the best metric is chosen to predict the instance. Similar to A Priori and A Posteriori methods, a classifier is only selected if it is significantly better (also defined by a threshold) than the others. If it is not, all of the members are used, applying the majority vote rule.

3.1.7 META-DES

The META-DES (CRUZ et al., 2015) technique is a different approach to the classifier selection problem. The previous method focused on single features of the relationship between each classifier and the test instances, for example, Local Accuracy. META-DES, instead, takes into consideration multiple features. And instead of trying to design a selection method, it considers this task another classification problem, thus delegating it to a meta-classifier.

The meta-classifier relies on five meta-features, each representing a measure that aims to estimate how competent a classifier is to classify an instance. The target of the meta-classification is if the classifier correctly estimated the instance, i.e., if it is competent enough to classify it or not.

The method is divided into three parts. The first one is overproduction, which is the training phase of the ensemble, on the training set. The second part, the meta-training phase, is subdivided into three steps, which are listed below. The meta-training set is segregated from the training set of the ensemble, in order to avoid overfitting. And the third part is the generalization phase, which is when the training instances are inputted into the ensemble to be predicted.

The first step is the sample selection. Previous work (SANTOS; SABOURIN; MAUPIN, 2008; CAVALIN; SABOURIN; SUEN, 2013) showed that the dynamic selection of classifiers is not very efficient in cases where the consensus on the ensemble is low, i.e., when the higher number of votes from the classifier to a class is almost equal to the number of votes to a second class. Therefore, the authors focused on the training of the meta-classifier on instances where this behavior is present. In order to achieve such selection, the level of consensus of the classifiers on a given instance is calculated. Then, only the instances that are below a certain threshold are selected to be trained on the meta-classifier.

If the instance is selected, its region of competence is extracted. That is performed by using the K-Nearest-Neighbors, finding its K most similar instances, and the K most similar output profiles.

With the output profile and the region of competence, step two starts. Five meta-features are extracted for each instance for each classifier in the ensemble.

- Neighbors Hard Classification - an array of K size, for each instance in the region of competence, one is set in the i position if the classifier correctly predicted the class of the instance, otherwise 0 is set to the position.

- Posterior Probability - an array of K size, for each instance in the region of competence, the probability for the classifier is calculated and set in the i position. The posterior probability is $P(y_i, x_i)$, which means the probability of the class y_i happening with the features x_i.

- Overall Local Accuracy - the value of local accuracy of the classifier over the most similar instances, i.e., the region of competence.

- Output Profiles Classification - an array of K size, for each instance in the validation set with the most similar output profiles, one is set in the i position if the classifier correctly predicted the class of the instance, otherwise 0 is set to the position.

- Classifier's Confidence - it is the perpendicular distance between the instance to the decision border of the classifier.

Since one meta-instance is generated for every classifier in the pool, for every single instance, even if there are few samples of the class, this problem is decreased due to the great number of meta-instances generated.

The meta-classifier is then trained with the produced instances.

In the generalization phase, for each test instance, the region of competence and most similar output profiles are calculated. Then, the meta-features are extracted and inputted into the meta-classifier. Only the classifiers that are predicted as competent are selected to predict the test instance. The majority rule voting is defined to combine the selected classifiers. If there is a tie, the class with a higher a posteriori probability is selected.

3.2 Methods for Online Machine Learning

In this section, the DCS methods considered to be the most representative to the state-of-art for online classifiers are presented.

3.2.1 Minority Driven Ensemble

The Minority Driven Ensemble method (ZYBLEWSKI; KSIENIEWICZ; WOŹNIAK, 2019) is not only a selection method but a whole framework that covers training an ensemble with a data stream and predicting new instances. It focuses on data that are affected by the class imbalance problem. Although this method proposes dealing with streams of data, it does not use adaptive algorithms as it provides an adaptation for traditional offline machine learning algorithms. This is done by replacing poor performing classifiers with new ones, similar to the Kappa Update Ensemble (KUE) does, yet, KUE uses online algorithms, as explained in Section 2.8.5.

The method divides the stream into chunks of data with a fixed size. Each chunk is passed as a mini-batch for the ensemble to train on. For each data chunk, the instances belonging to the minority class are filtered in order to remove outliers. That is done using K-Nearest Neighbors on the current data chunk. If the K nearest neighbors of each instance belong to the majority class, the instance is then considered an outlier and then removed from the chunk. This technique is somewhat questionable since it removes instances from the class that is in an already reduced number, compared to the majority class.

With the outliers removed, the data chunk is used to produce one new classifier to compose the ensemble. To create a rotation of classifiers, thus building a self-adaptive ensemble, some techniques are applied to remove some experts from the pool of classifiers. The ensemble is adapted to be as expert as possible in detecting the minority class. If a classifier has low (threshold is $0.5 + \alpha$, where α is user-given) Balanced Class Accuracy (BAC), it is removed from the ensemble. If the maximum ensemble size (which is also user-given) is exceeded, the classifier with the worst BAC is also removed. BAC is an alternative way to measure the accuracy of a model. It takes into consideration the balance between classes, avoiding a false good accuracy when the model just predicts the majority class correctly. The formula for calculating BAC for a binary problem is displayed in Equation 3.1. Where TP is true positive, FP is false positive, TN is true negative, and FN is false negative.

$$BAC = \frac{TP}{TP + FP} + \frac{TN}{TN + FN} \tag{3.1}$$

In the prediction step is where the selection method is applied. Since the ensemble is constructed focusing on building expertise in classifying the minority class, if only one classifier predicts the instance as being from the minority class, it is considered as such class.

Although the paper that introduced this method presented good results, some aspects require a more in-depth analysis. It was compared to the static methods KNORA-E, KNORA-U (KO; SABOURIN; BRITTO JR., 2008), LCA (GIACINTO; ROLI, 1999), and DCS-RANK (SABOURIN et al., 1993). The parameters for the evaluation, such as α and the ensemble size, were chosen based on a hyperparameter optimization (search for the parameters which the ensemble presents the highest BAC). However, these parameters (and the other results as well) were tested on a single stream with gradual and sudden drifts, which might not be true for other streams and real-world problems. Another point is that the ensemble size selected for the evaluation was three, which might not be a fair comparison with the other algorithms such as KNORA-E and KNORA-U. That is because they are intended to select a subset of a usually bigger ensemble.

3.2.2 Dynamic Ensemble Selection for Drift Detection

Dynamic Ensemble Selection for Drift Detection (DESDD) (ALBUQUERQUE et al., 2019) method provides a different concept in Dynamic Selection. Traditional DCS methods such as KNORA-E and KNORA-U are also referred to as dynamic **ensemble** selection methods because they select a subset of the ensemble, which is, by definition, also an ensemble. DESDD, in contrast, uses the same term to designate the selection of an ensemble e_i in a group G with n ensembles. In other words, G is an ensemble of ensembles, e.g, an ensemble of n OzaBags.

DESDD utilizes online classifiers, and it is designed to work with the prequential evaluator. There is no algorithm predefined to be used to build each ensemble e_i as any algorithm can be used. However, in the paper experiments, Online Bagging was used to create each e_i, with a randomized λ varying from a minimum and maximum value (user-given), thus, each ensemble is expected to be different from each other.

In the selection step, DESDD selects the ensemble e_i with the highest current accuracy, which is the average of the accuracy of the ensemble prior to the moment of selection.

DESDD is also proposed to deal with concept drift. Therefore, a drift detector (also not specified, any method can be used) is attached to G. Once a drift is detected, the whole G is discarded, and the initialization and training for each ensemble e_i is started again.

3.2.3 Semi-supervised Drift Detection Method

The Semi-supervised Drift Detection Method method (PINAGE; SANTOS; GAMA, 2019) provides a framework to create and maintain a dynamic ensemble in environments where labeled data are scarce. As the name suggests, it applies a semi-supervised technique, which means that part of the data used to train the model has labels, and the other part gets an automatically assigned label. The method starts with the ensemble creation. In this phase, a modified version of Online Bagging (MINKU; WHITE; YAO, 2010) is used. The main difference of this version to the traditional Online Bagging is that the λ used by the Poisson random number generator is passed as a parameter and smaller λ values result in higher diversity. The first m instances to arrive are expected to be labeled to create a model capable of predicting new instances with a certain confidence.

After the classifier is partially built on the first instances, the next training instances are not necessarily expected to be labeled. Consequently, the method uses dynamic selection of classifiers to define which member of the ensemble is the most competent to predict the instance. Then, the classifier is updated assuming the prediction of the selected member as a true label for it. The methods applied for the selection are the ones defined in DCS-LA and MCB, previously explained in Sections 3.1.2 and 3.1.6. The validation set required by the methods is initially defined offline for the whole ensemble with part of the first labeled instances.

In order to detect drift, all of the members of the ensemble hold a detector. However, since most detectors are expected to deal with labeled data, they were also adapted to treat the label predicted by the most competent classifiers as true. The drifts work with both warning and detection levels. If the warning level is set in any classifier, the instances and their assumed labels are started to be stored, if the level changes to detection, the instances previously-stored are used as a new validation dataset, and all of the statistics are restarted.

3.2.4 Dynamic Selection Based Drift Handler

The Dynamic Selection Based Drift Handler (DYNSE) (ALMEIDA et al., 2016) is a method that applies traditional offline techniques of Dynamic Selection in online

Machine Learning environments to deal with concept drift. It contemplates the whole construction of the structure of the ensemble, as well as the prediction step.

It does not use a traditional ensemble algorithm, such as Online Bagging (OZA, 2005). Instead, it proposes building each classifier with batches of labeled data that arrive at once. Each batch is used to train a new classifier of the ensemble, and if the size n of the batch is not sufficiently large, multiple batches can be accumulated before training.

Any base classifier can be used to compose the ensemble, even the ones intended for offline machine learning, since they receive the data for training all at once. On the prediction step, when an instance x arrives to be predicted, a K-Nearest Neighbors (FIX; HODGES, 1989) search is executed to find the most similar instances to x in the validation set, which is defined by the M latest supervised batches that arrived to be trained on. Once the similar instances are gathered, any selection method that depends on it can be applied, in the paper, the method applied was KNORA-E (KO; SABOURIN; BRITTO JR., 2008).

There is not a defined number of maximum member an ensemble should have, and the authors propose that the most base classifiers trained on different data, the greater the chance that the selection for the most fitted members for a specific instance succeeds. However, if concepts change, the older classifiers might never be selected, because the selection is based on the neighbors of the instance that are on the most recent batches.

Although the paper presented some results superior to other state-of-art methods, in the experimental protocol, the number of instances used to train each classifier was somewhat small (250 instances for some datasets), and the classifier used was the Hoeffding Tree. Perhaps a greater amount of instances and the training with offline classifiers, because they usually require fewer instances in order to learn a stable concept, should be tested for a greater comprehension of the method behavior.

3.2.5 Preprocessed DCS I

Focusing on the imbalanced data problem, this method proposes a similar approach to DYNSE. Preprocessed DCS I (PDCS I, the method is not given a specific name by its authors, it is named for the sake of clarity) (ZYBLEWSKI; SABOURIN; WOŹNIAK, 2021) processes the data stream in chunks and new classifiers are trained and added to the ensemble upon each chunk arrival.

PDCS I works with an ensemble of ensembles so that for each chunk, a new ensemble is trained using offline bagging, however, the data samples are stratified. Similar to Minority Driven Ensemble, if the maximum size is reached, the ensemble with the lowest balanced accuracy (BAC) is removed from the pool. The validation set is also defined as the last arrived batch, however, before updating it, a pre-processing technique focused on data imbalance, either under or oversampling, is applied to the chunk. Covering these methods is not under the scope of this project, for an overview of this type of method, the reader is referred to (WANG; PINEAU, 2016).

The training part follows the same protocol of DYNSE, for each test instance, the most similar instances of the validation set are gathered so any traditional DCS method can be applied.

3.2.6 Preprocessed DCS II

Preprocessed DCS II (PDCS II, again for the sake of clarity) (ZYBLEWSKI; SABOURIN; WOŹNIAK, 2020) is similar to PDCS I (ZYBLEWSKI; SABOURIN; WOŹNIAK, 2021), with the difference that it works with any classifier, not only with ensembles, and the pre-processing step is applied on the chunk before the training phase. It also adds a step that removes any classifiers from the ensemble that have their BAC (Equation 3.1 in Section 3.2.1) below a user-given threshold, which is the same protocol applied in the Minority Driven Ensemble (ZYBLEWSKI; KSIENIEWICZ; WOŹNIAK, 2019).

Both PDCS I and II can be used with both offline and online classifiers as they handle the data stream as chunks, and classifiers are only trained once. Consequently, the ensemble may not take advantage of online classifiers as its members are not updated as new data become available.

3.3 Applying Dynamic Selection of Classifiers

When new dynamic selection methods are published, it is common for their authors to provide the source code used in their experiments. However, it is usually “biased” to the type of the experimental protocol used in that specific paper. Thus, it is not always allowing an easy adaptation for other authors to test it with other datasets and other base classifiers. Not to mention that the language of the code might not be the same as other authors are willing to implement.

However, a tool that allows the applications of many implements methods using the same programming interface is very important to the scientific community. It allows other creators to easily compare their new methods with the state-of-art, providing a fair evaluation of its real contribution and a code that is completely reproducible with other datasets. That is the purpose of the previously explained frameworks (Section 2.9) for

Table 3.1: Methods Implemented in DESLIB

Name	Type	Name	Type
A Priori (GIACINTO; ROLI, 1999)	DCS	DES-KNN (SOARES et al., 2006)	DES
A Posteriori (GIACINTO; ROLI, 1999)	DCS	KNOP (CAVALIN; SABOURIN; SUEN, 2013)	DES
LCA (WOODS; JR; BOWYER, 1996)	DCS	KNORA-E (KO; SABOURIN; BRITTO JR., 2008)	DES
OLA (WOODS; JR; BOWYER, 1996)	DCS	KNORA-U (KO; SABOURIN; BRITTO JR., 2008)	DES
MCB (HUANG; SUEN, 1995)	DCS	DES-MI (GARCÍA et al., 2018)	DES
MLA (WOODS; JR; BOWYER, 1996)	DCS	RRC (WOLOSZYNSKI; KURZYNSKI, 2011)	DES
Mod. Rank (WOODS; JR; BOWYER, 1996)	DCS	DES-Kullback Leibler. (WOLOSZYNSKI et al., 2012)	DES
META-DES (CRUZ et al., 2015)	DES	DES-Minimum Difference. (ANTOSIK; KURZYNSKI, 2011)	DES
DES Clust. (SOARES et al., 2006)	DES	DES-Exponential. (ANTOSIK; KURZYNSKI, 2011)	DES
DES-P (WOLOSZYNSKI et al., 2012)	DES	DES-Logarithmic. (ANTOSIK; KURZYNSKI, 2011)	DES

data stream mining and of Scikit-Learn (PEDREGOSA et al., 2012) for batch machine learning.

In order to fill in this gap in the dynamic classifier selection scope, Cruz et al. (2018) proposed DESLIB, which is a framework that implements the most popular and, well-succeeded methods of Dynamic Selection for offline Machine Learning environments. DESLIB is intended to work with ensembles implemented in Scikit-Learn, or that at least follows the same specified programming interface.

DESLIB delegates the training step of the ensembles to the framework that implemented it. It encapsulates the ensemble and provides a “second training” step, which is essentially receiving a validation set and preparing it in order to work with a dynamic selection algorithm.

The prediction step is handled by DESLIB, as it automatically selects the classifiers that are most competent to predict a given instance, respecting the protocol of each specific algorithm, and returns a prediction. It also offers multiple parameters for the methods, such as optimized ways to apply the K-Nearest Neighbors and the number of neighbors that will be gathered to perform the selection.

DESLIB is also easily extendable, fully open-source, and written in Python. This allows new creators of methods to implement it following the structure of the framework, making it easier to share and publish them.

The methods implemented in the library are divided into DCS and DES, and there are also Static Selection methods implemented. The Oracle selector is also implemented. The methods implemented in the library, as well as their category, are displayed in Table 3.1.

3.4 Final Considerations

This chapter presented the rationale behind dynamic classifier selection (DCS) methods, as well as the state-of-art methods for both batch and online environments. In addition, it was also presented a currently available framework for applying DCS in the batch envinronment. In the next section, a similar framework, but for online envinroments is proposed.

PART II

PROPOSAL

Chapter 4

Framework for Applying DCS in Data Stream Mining

As discussed in the previous chapter, methods concerning the application of dynamic selection of classifiers in data stream mining are still scarce. However, it is a well-explored area in the batch environments, with a great number of developed methods. This produces an implicit expectation that there are many benefits from DCS that are yet to be brought to the online machine learning configuration.

For batch environments, DESLIB (CRUZ et al., 2018) provides an easy way to implement and test new classifier selection methods. The framework only has to take care of the prediction step of the methods.

With the task of selecting which classifiers of the ensemble will be used to perform the prediction for a test instance, it provides a generic framework that works with any ensemble that follows the same interface as scikit-learn (PEDREGOSA et al., 2012).

The only code required by the authors of a new classifier selection method is the selection logic.

However, the whole rationale of dynamic selection was designed for offline environments. In opposition to what we observe in batch scenarios, the appliance of DCS methods in data stream mining is not a generic task.

These types of methods, such as DYNSE (ALMEIDA et al., 2016), do not propose a novel approach of selecting the classifiers to predict a given instance. Instead, they require an architecture that also contemplates how the selection methods will be applied. For instance, let us analyze the concept of the validation set, which in batch environments is traditionally a subset of the available instances to train the model, is not trivial in this context.

In streaming scenarios, the instances that will compose the validation set must be extracted from the stream. Moreover, they should be representative of the stream's current state, i.e., they should follow the same distribution of the training set, which is potentially untrue due to concept drifts (TSYMBAL, 2004).

Or in some cases like MDE (ZYBLEWSKI; KSIENIEWICZ; WOŹNIAK, 2019) and DESDD (ALBUQUERQUE et al., 2019), the validation set is not even required for classifier selection.

The type of ensemble that the method will work with is another point that must be contemplated.

Some methods might be compatible with any dynamic ensemble implemented in scikit-multiflow (MONTIEL et al., 2018) or MOA (BIFET et al., 2010), which is similar to what DESLIB (CRUZ et al., 2018) does with scikit-Learn (PEDREGOSA et al., 2012).

On the other hand, some methods, such as DYNSE (ALMEIDA et al., 2016), require a different way of constructing the ensemble with individual online classifiers, while others do not even require the classifiers to be incremental or dynamic, such as in MDE (ZYBLEWSKI; KSIENIEWICZ; WOŹNIAK, 2019).

To the best of the author's knowledge, there is currently a lack of tools and frameworks that contemplate the aforementioned characteristics and allow researchers to develop and compare new classifier selection methods with others previously developed. This creates a delay in the development of this area because even if the authors provide their implementation source code, it is usually biased toward their experimental protocol and their datasets.

4.1 Scikit-Dyn2Sel

To fill in this gap, we propose Scikit-Dyn2Sel, a novel framework for dynamic classifier selection for data streams. The Dyn2Sel acronym comes from that it deals with a **Dyn**amic environment **2** "times", because it deals with dynamic ensembles and dynamic classifier **Sel**ection. Scikit-Dyn2Sel is designed to be fully compatible with Scikit-Multiflow (MONTIEL et al., 2018). The architecture of Scikit-Dyn2Sel is illustrated in the class diagram in Figure 4.1.

In order to correctly address the challenges presented, the following key classes of the architecture are:

- `BaseEnsemble` - The `BaseEnsemble` class belongs to Scikit-Multiflow. It is the class that all the ensemble methods from the library, such as AdaptiveRandomForest (GOMES et al., 2017), extend. Thus, all of these classifiers are accepted in Scikit-Dyn2Sel. It is also possible to deal with the dynamic construction of an ensemble.

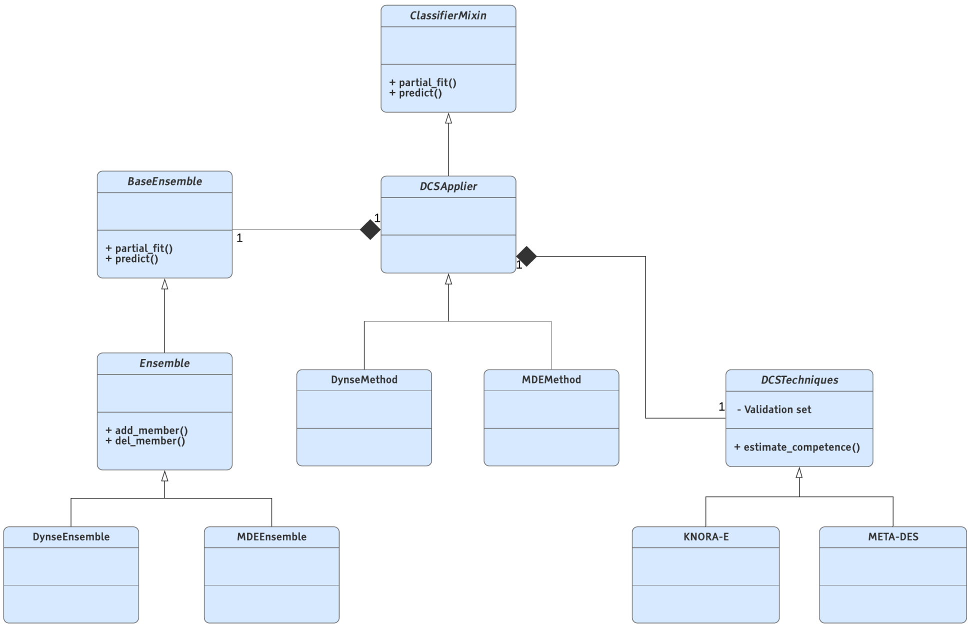


Figure 4.1: Class Diagram of Scikit-Dyn2Sel

This is required in methods like DYNSE. Therefore, a new class `Ensemble` is created, which extends the same `BaseEnsemble` class, forcing it to respect the same interface. Thus, it allows Scikit-Dyn2Sel to handle it the same way as a Scikit-Multiflow ensemble in which base classifiers can be either added or removed from the pool.

- `DCSTechnique` - The `DCSTechnique` class is responsible for defining the interface to be shared by the dynamic selection method. It is also stores the validation set of a method, when it is required.

The `BaseEnsemble`, `Ensemble`, and `DCSTechnique` classes are abstract classes, which means that they can provide abstract methods (without implementation), concrete methods (with implementation) and attributes that should be used by all of the concrete classes. The concrete classes are classes that extend these abstract classes and implement the abstract methods. For example, for the implementation of any method, it would be necessary to either implement an `Ensemble` concrete class and set it up in a way that mimics the growth of the ensemble defined in the method or to instantiate a Scikit-Multiflow ensemble. It would also be necessary to implement any DCS techniques that are desired to be applied in conjunction with the proposed method. The benefit of this separation is that each DCS technique only needs to be implemented once, independently of how many methods it is used with.

All of these classes are combined as attributes into the `DCSApplier` class, which is responsible for coordinating them. This class is also an abstract class, and thus, for any

Table 4.1: Methods Contemplated in Scikit-Dyn2Sel

DCSApplier	DCSTechnique
DYNSE	KNORA-E
DESDD	KNORA-U
MDE	A Priori and A Posteriori
Preprocessed DCS I	DCS-LA
Preprocessed DCS II	DCS-RANK
	KNOP
	MCB
	META-DES

method implemented, a new class inheriting from `DCSApplier` should be added.

For clarity, on the class diagram reported in Figure 4.1, the only concrete classes for `BaseEnsemble` that are displayed are `DYNSEEnsemble` and `MDEEnsemble`. The same happens for the concrete implementation of `DCSApplier`, where only `DYNSEMethod` and `MDEMethod` are shown. For `DCSTechnique`, only `KNORA-E` and `META-DES` are illustrated.

The `ClassifierMixin` class, which `DCSApplier` is inherited from, is also from Scikit-Multiflow. This is the class from which all classifiers implemented in Scikit-Multiflow extend. The fact that `DCSApplier` also has this inheritance is important for the architecture of the framework. It allows an implemented method to fully-compliant with Scikit-Multiflow's architecture, and thus, being able to be executed together with other functionalities of the library, such as generators and evaluators. The main methods that `DCSApplier` inherits for ensuring compatibility are `partial_fit` and `predict`, which are responsible for updating the model and predicting a set of instances, respectively.

Table 4.1 illustrates all the methods contemplated by the framework.

This structure is partially inspired by DESLIB, however, several adaptations were needed for it to work with dynamic ensembles. Scikit-Dyn2Sel was designed so most of the current and future DCS methods can be added to it.

4.1.1 Example of DYNSE Implementation

To illustrate how an implementation of a method could be done in Scikit-Dyn2Sel, a demonstration with two simple sequence diagrams was executed, following the explanation of the method DYNSE described in Section 3.2.4. For the sake of clarity, it is divided into two phases: training and testing.

For the training phase, as shown in Figure 4.2, inside a loop of n instances, for each instance, the first step is a call to the `partial_fit` method. This step is strictly the same for training any classifier from Scikit-Multiflow. Next, a conditional instruction is performed: if the number of instances sent to update the model `current_chunk_size`

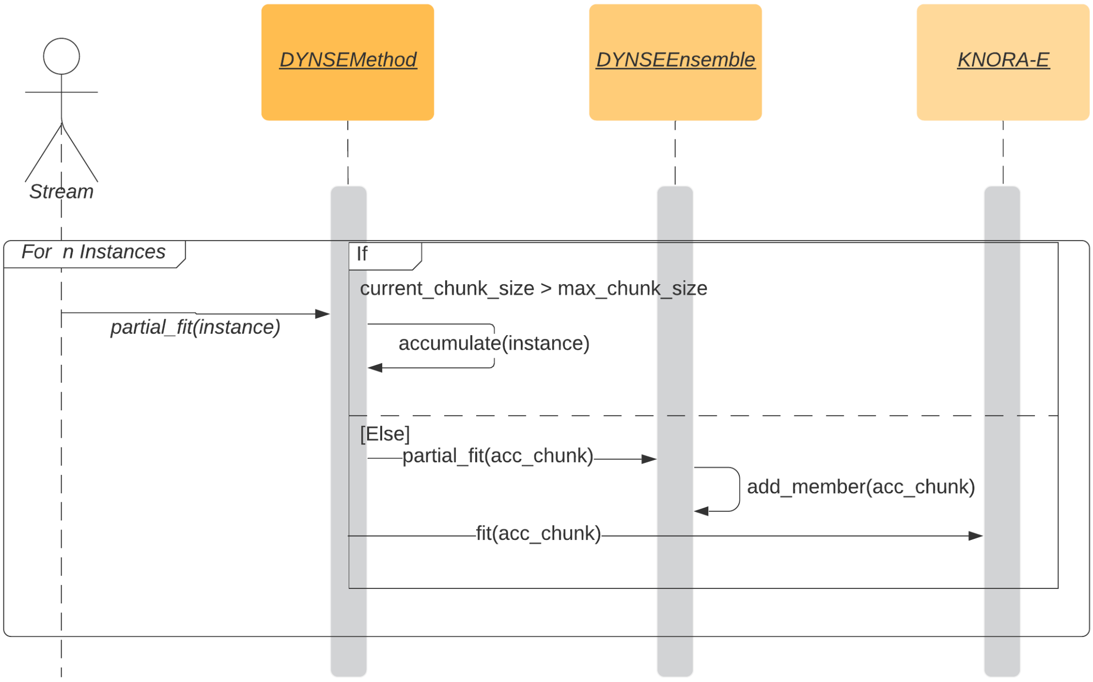


Figure 4.2: DYNSE Training Phase

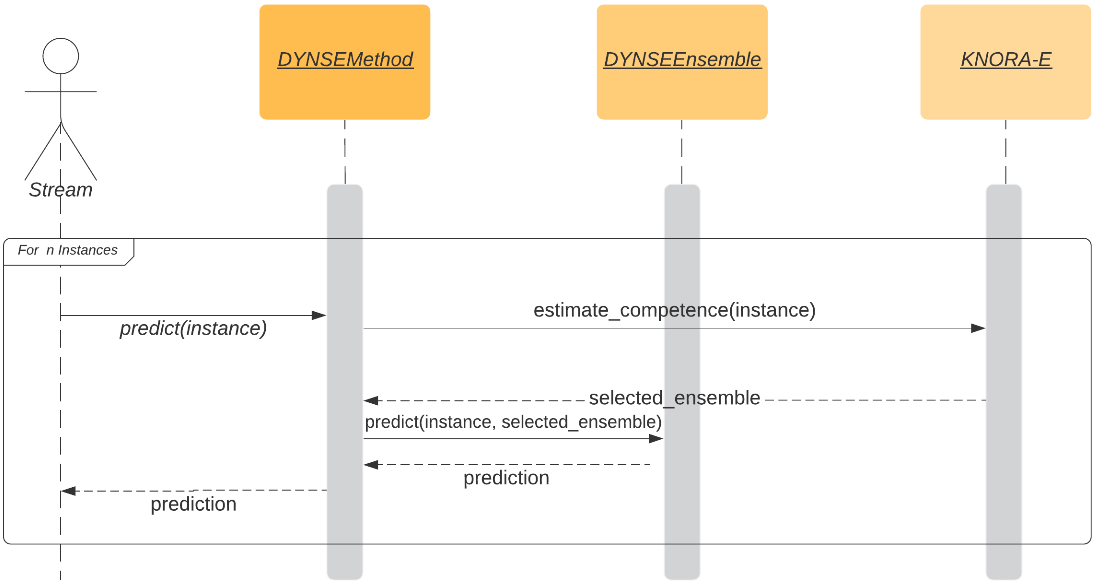


Figure 4.3: DYNSE Testing Phase

```
def partial_fit(self, X, y):
  for x_i, y_i in zip(X, y):
    if len(self.buffer_x) < self.chunk_size:
      self.buffer_x.append(x_i)
      self.buffer_y.append(y_i)
    else:
      self.ensemble.partial_fit(self.buffer_x, self.buffer_y)
      self.dcs_method.fit(self.buffer_x, self.buffer_y)
      self.buffer_x = []
      self.buffer_y = []
```

Figure 4.4: Implementation of the `partial fit` on DYNSEMethod

```
def predict(self, X):
    if len(self.ensemble) > 0:
        all_predictions = self.ensemble.predict(X)
        selected_indexes =
            self.dcs_method.estimate_competence(self.ensemble, X)
        masked_predictions =
            ma.masked_array(all_predictions, selected_indexes)
        final_predictions = np.max(masked_predictions, axis=1)
        return final_predictions
    else:
        return np.array([])
```

Figure 4.5: Implementation of the `predict` on DYNSEMethod

is still less than the size defined for a data chunk `max_chunk_size`, the instance is only accumulated into an internal buffer. Otherwise, a call to `partial_fit` from the class `DYNSEEnsemble` is made, containing the whole data chunk. Upon this call, `DYNSEEnsemble` will create a new member of its pool of classifiers trained on the recently passed chunk. Finally, the internal validation of the DCS method, in this case, KNORA-E, is replaced with the same chunk used to train the new classifier, using the `fit` function.

For the testing phase, displayed in Figure 4.3, a similar loop of instances is applied. For each instance, the `predict` method is called. Again, this first step is the same for predicting using a Scikit-Multiflow classifier. Then, the `DYNSEMethod` will call the `estimate_competence` in KNORA-E, which will use its internal validation set to select and return the members to predict that given instance With the members of the ensemble selected, the `predict` method of `DYNSEEnsemble` will be called only with the selected members, and a prediction is returned.

Figures 4.4 and 4.5 illustrates the implementation on the framework of the methods `partial fit` and `predict` respectively.

Chapter 5

Behavior of Dynamic Selection on Data Stream Mining

The application of dynamic selection of classifiers in batch machine learning is very well studied and documented (CRUZ; SABOURIN; CAVALCANTI, 2018; JR; SABOURIN; OLIVEIRA, 2014). It is observed that DCS techniques achieve good performance when some key points are present. Each of these points is discussed as follows.

- The pool of classifiers must be diverse. As already explained in Section 3, the mistakes made by each base classifier should be different from each other. The way they react to the instances should not be the same, otherwise, the selection will not make sense. This is not only valid for selection purposes, but also for the whole purpose of ensembles.

- The problem must not be linearly separable, i.e., the dataset must not be too simple or else selection is not useful. In simple problems, multiple classifier systems with traditional voting methods, or even single classifiers, achieve good performance. Thus, using selection only inserts an additional complexity layer into the model, which is potentially unnecessary or even prejudicial.

- The pool must be composed of weak classifiers. Dynamic selection tends to work better when applied in a pool where the base classifiers are not too complex. Using weak classifiers, each member tends to be a specialist on a small part of the data, not capable of learning the whole concept alone. Hence, the selection aims to find these experts for each instance.

An experiment performed in the documentation of DESLIB (CRUZ et al., 2018), which can be accessed in <https://deslib.readthedocs.io/en/latest/auto_examples/plot_example_P2.html> (accessed in February 10, 2020), demonstrates the last point on top of a static dataset.

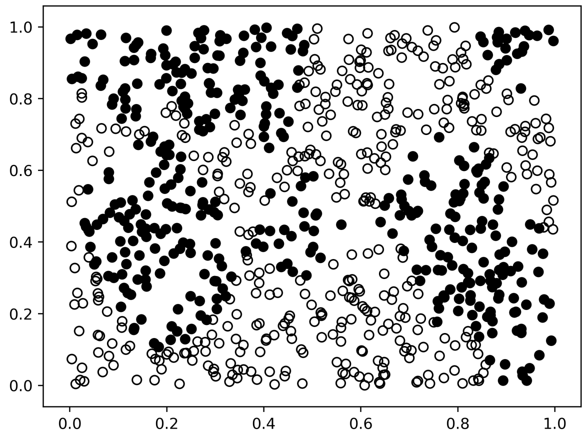


Figure 5.1: Decision Space of P2 Generator

An ensemble of five weak classifiers was generated using the Boosting (FRIEDMAN, 2000) technique. The base classifiers were Decision Trees, and they were weak because the maximum depth that the trees could achieve was limited at one, thus practically, the trees contained simply the root and its children. This is also called a Decision Stump. The ensemble was combined with three selection techniques: KNORA-E (KO; SABOURIN; BRITTO JR., 2008), OLA (WOODS; JR; BOWYER, 1996) and Modified Rank (WOODS; JR; BOWYER, 1996). The experiment also included a Random Forest (BREIMAN, 1996) with ten trees for comparison with the DCS techniques.

The dataset used for the experiment was the generator P2, proposed by Valentini (2006). It is a complex non-linear problem in which the classes are defined by multiple decision boundaries in different regions. The equations that determine the classes of the instances are defined in Equation 5.1. P2 is a binary class generator with only two features. One possible decision space of randomly generated P2 instances are defined in Figure 5.1, with one of the classes represented as full circles and the other as empty circles.

$$\begin{aligned}
E1(x) &= \sin x + 5 \\
E2(x) &= (x-2)^2 + 1 \\
E3(x) &= -0.1 \times x^2 + 0.6\sin(4x) + 8 \\
E4(x) &= \frac{(x-10)^2}{2} + 7.902
\end{aligned} \tag{5.1}$$

The results of the experiments are displayed in Table 5.1. The boosting ensemble without selection methods performed poorly when compared to the same pool combined with selection methods. The selection methods were also better than the non-limited Random Forest model, even though it used the double of base classifiers than boosting.

5.1 Dynamic Selection in Data Stream Mining

The objective of this chapter is to present an analysis of the appliance of traditional dynamic selection methods, for the batch environment, in traditional data stream classifiers. If there is an accuracy gain, in which cases and so on. The objective is not to compare if the selection methods will outperform the state-of-art methods, but to verify when DCS is welcomed in dynamic ensembles.

Each of the topics mentioned above must be carefully analyzed when dealing with dynamic environments, as some characteristics of data stream mining might seem contrary to some requirements for dynamic selection to be useful.

5.1.1 Weak Classifiers

In batch machine learning, weak classifiers are easy to produce. Classifiers' growth potential is pruned, or the quantity of data that it receives for training is lowered. However, in online machine learning, this is not so simple. That is because, in most dynamic ensemble construction methods, the instances keep arriving for each base classifier to update its model.

Thus, the trend is that all the base classifiers will be able to predict most types of instances and not an expert on a specific type.

Figure 5.2 is the plot of the accuracy overtime of an execution of a single Hoeffding Tree using the prequential process. Two million instances were generated by the Random Tree Generator, set up to five numerical and five categorical features and three classes. The evaluation of the model, i.e., data points in the chart, was done every 10 thousand instances, totaling 200 hundred evaluations.

The values between each evaluation vary from worse to better than the last evaluation, but overall, an increasing trend is observed in the results. That means that as the

Table 5.1: Results of DESLIB Experiment

Method	Accuracy	Method	Accuracy
Boosting - No Selection	79.50%	Boosting - KNORA-E	94.80%
Random Forest - No Selection	92.50%	Boosting - Modified Rank	94.80%
Boosting - OLA	93.20%		

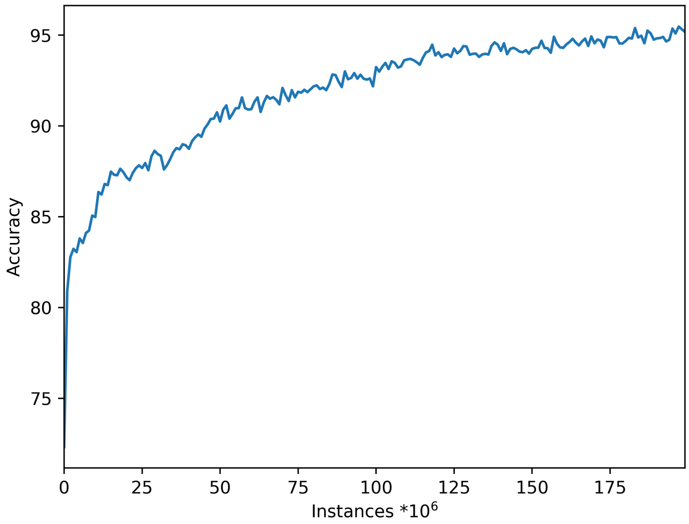


Figure 5.2: Evolution of the Accuracy Overtime

model receives more and more instances, it becomes more accurate. The accuracy trend will eventually reach a plateau, which is when the maximum that the model can extract from that data, without drift, is exceeded.

As explained, this characteristic is the opposite of what is expected for satisfactory results with dynamic selection in the batch environment, as the classifiers tend to not remain weak for long. This is not the ideal environment for DCS because, as previously explained, the selection mechanism only makes sense if there are experts in different areas of the data distribution. When the classifiers are strong, DCS starts to lose its appeal.

5.1.2 Complex Datasets

Another problem that needs to be dealt with when applying dynamic classifier selection (DCS) in data stream mining is the complexity of the datasets. Most generators present simple problems, normally linearly separable. For example, as explained in Section 2.5.1, the SEA Generator has three features, but one of them is not descriptive (noise). If the non-descriptive feature is ignored and the other ones are plotted, it is clear that the problem is linear. This is illustrated by Figure 5.3.

Another example of the lack of complexity of data stream mining generators regards their functions to determine the class of the instance. Even though some generators have many features, some of the functions use only a few of them. For example, the Agrawal Generator explained in Section 2.5.2 has nine features. However, its first func-

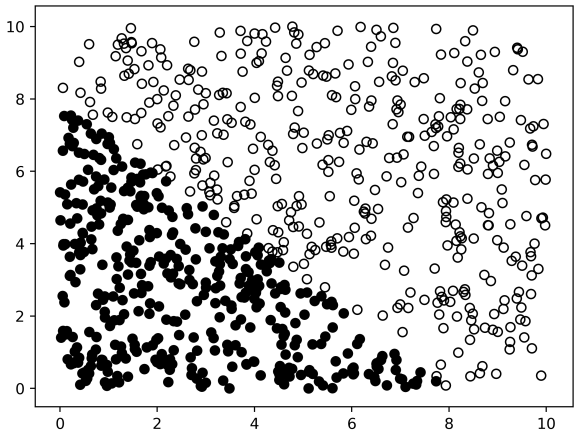


Figure 5.3: Decision Space of SEA Generator - Function 0

tion consists of:

$$\begin{cases} 1 & \text{if age} < 40 \text{ or } 60 \leq \text{ age} \\ 0 & \text{otherwise} \end{cases}$$

where only the "age" feature is used and all the other features act as noise in the learning process. Even though this function is simple, it is not linear, since it needs two comparisons for the class to be determined. This causes problems for some linear classifiers, but not for more complex ones. The noise surely adds some level of difficulty to the problem, but not as much as a complex (non-linear) function. The second classification function of the Agrawal Generator is slightly more complex:

$$\begin{cases} 1 & \text{if ((age} < 40) \text{ and } (50000 \leq \text{ salary } \leq 100000)) \text{ or} \\ & ((\ 40 \leq \text{ age} < 60) \text{ and } (75000 \leq \text{ salary } \leq 125000)) \text{ or} \\ & ((\text{age } \geq 60) \text{ and } (25000 \leq \text{ salary } \leq 75000)) \\ 0 & \text{otherwise} \end{cases}$$

As shown in Figure 5.4, the decision space of this function is not linear, however, its decision boundaries are clear and not challenging to many ensemble methods.

In a simple experiment, two thousand instances were generated using the aforementioned functions from the SEA and Agrawal, respectively. Half of the instances were used to train two classifiers: an offline Random Forest classifier with 100 trees and an offline bagging with 100 Perceptrons, which individually are linear classifiers. The other half of the instances was used to evaluate the accuracy of the trained models. These classifiers were chosen because the objective of this experiment was to evaluate the behavior

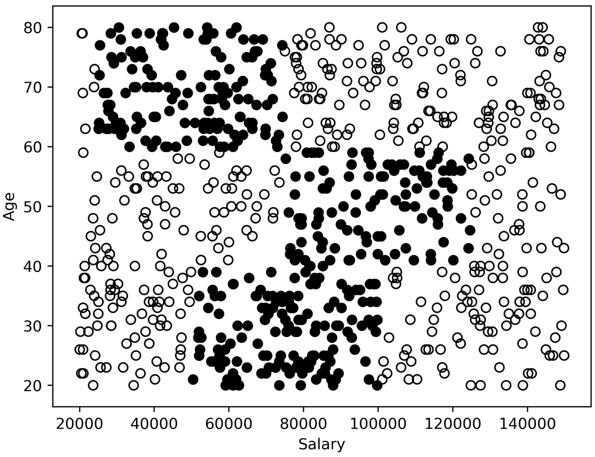


Figure 5.4: Decision Space of Agrawal's Second Function

Table 5.2: Results of Generators with Batch Random Forest

Dataset	Accuracy
Random Forest - Agrawal (first function)	100.00%
Random Forest - Agrawal (second function)	99.00%
Random Forest - SEA	98.90%
Bagging with Perceptrons - Agrawal (first function)	47.80%
Bagging with Perceptrons - Agrawal (second function)	61.50%
Bagging with Perceptrons - SEA	98.80%

of data stream generators in two different situations: with a state-of-art ensemble method and with an ensemble of weak classifiers.

As seen in Table 5.2, the batch version of Random Forest, implemented in Scikit-Learn (PEDREGOSA et al., 2012), achieved more than 98% of accuracy for all generators. For the ensemble of weak classifiers, the results for the simple but non-linear functions from the Agrawal generator were significantly worse. However, for the linear SEA generator, the results were practically the same, achieving more than 98%.

This implies that the generators are indeed very simple and easily predictable for the state-of-art method (Random Forest). For the ensemble of weak linear classifiers, for the non-linear functions, the results were not satisfactory, but for the linear function from the SEA generator, it got close to the results of the Random Forest.

These results lead to the conclusion that there is a considerable gap in terms of accuracy for non-linear classifiers that could benefit from selection. However, the potential of gain using the state-of-art method is very small if not nonexistent. Hence, at least in the batch environment, it would not make sense to apply dynamic selection on these datasets.

5.2 Experimental Protocol

With the aforementioned challenges in mind, a series of experiments were executed to check how dynamic selection behaves with existing online ensembles and commonly used data stream mining generators.

The application of the dynamic selection methods in online learning could not be made without some adaptations. The logic for creating and updating the validation set, from where the neighbors from each test instance would be extracted, needed to be defined. For that, an adaptation of the Interleaved Chunks validation scheme available on MOA (BIFET et al., 2010) was used. It works similarly to Prequential, explained in Section 2.4.2, but instead of testing then training a single instance at a time, it uses chunks of instances with n size. In order to define the validation set, instances were randomly drawn according to a probability p from each chunk during training, i.e., after they were already used for testing. Thus, the validation set defined on round x is used for testing the data made available in round $(x + 1)$.

The dynamic ensemble algorithms, DCS methods, and the generators used in the experiments are depicted in Table 5.3. All the generators were balanced, except P2 and the real-world datasets, were executed with and without concept drifts, the concept functions used were always one (default) and two (alternate). The parameter of the ensembles and the generators were left as default as given in the Massive Online Analysis (MOA) framework (BIFET et al., 2010), with the exception of the ensemble size, which was that to 100 base learners. The k for the nearest neighbor search on the DCS methods was set to 7.

The combination of all methods, datasets, and hyper-parameters, totaled 500 runs. The size n of the chunk was set to 1000, the number of the instances generated each time was 250,000, totaling 250 chunks. The evaluation of the model was performed at the end of each chunk. The drift added was in the middle of the execution, after 125,000 instances had passed.

Additionally, experiments with the base classifiers alone were executed for comparison.

5.2.1 Statistical Testing

Two statistical tests were applied in the results in order to better visualize the effect of DCS on the ensembles. The first was the combination of the Friedman test followed by the Nemenyi post-hoc test, as suggested in (DEMŠAR, 2006). The objective of the

Table 5.3: Ensemble Algorithms, DCS Methods and Generator used in the Experiments

Ensembles	DCS Methods	Generators
Adaptive Random Forest (ARF) with Hoeffding Tree	KNORA-E	SEA
Kappa Updated Ensemble (KUE) with Hoeffding Tree	KNORA-U	Agrawal
Kappa Updated Ensemble (KUE) with Naive Bayes	LCA	Asset
Kappa Updated Ensemble (KUE) with Perceptron	OLA	Random Tree Generator (RTG)
Ozabag with Hoeffding Tree	Oracle	P2
Ozabag with Naive Bayes		Spam Corpus
Ozabag with Perceptron		Electricity
OzabagAdwin with Hoeffding Tree		NOMAO
OzabagAdwin with Naive Bayes		
OzabagAdwin with Perceptron		

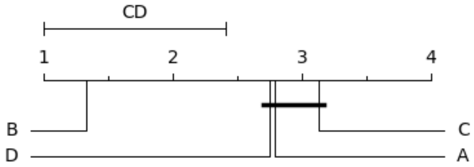


Figure 5.5: Critical Distance Plot Example

Friedman test, in this context, is to determine if there is a significant difference between the performance of three or more groups of classifiers on multiple datasets, respecting a significance level α. This significance level is basically the probability of a significant difference outputted by the test being false. α is usually set to 0.05. Friedman test alone, however, does not specify which classifiers' performances are significantly different than others, that is why the pairwise Nemenyi test needs to be applied after.

The most common way to output the results of the Nemenyi test is the critical distance plot, such as the example in Figure 5.5. In the example, the classifiers A, B, C, and D are being compared. The first thing to look at is the order that the classifiers appear, that is the mean rank of their performances on the datasets. In this case, the classifier B was the best placed one, followed by D, A, and C. All the classifiers that the same black bar touches, are not significantly different from each other. In this case, the only significantly different classifier is B. It is noteworthy that this test only takes into account the mean ranking of the classifiers, it does not consider how wide the gap between different classifiers is.

The second test applied was the Bayesian Analysis (BENAVOLI et al., 2017). This is a pairwise analysis that outputs the probability of each classifier being better than the others. It works by creating a normal distribution based on the results of two classifiers. Then, it draws n random samples of the distribution and based on these samples, determines the probability of classifier A being better than classifier B and vice-versa. It also outputs the probability of the classifiers being better than each other lying in the region of practical equivalence, or *rope*. The size of this region is set by a

Table 5.4: Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - Without Drift and Real-World datasets

	NO SELECTION	LCA	OLA	KNORA-U	KNORA-E
ARF - P2	98.53%	94.28%	95.48%	**98.55%**	98.41%
ARF - SEA	87.41%	85.87%	83.43%	**87.48%**	84.99%
ARF - RTG	**98.72%**	91.28%	93.52%	98.51%	98.59%
ARF - Asset	**93.65%**	81.78%	91.43%	93.35%	92.73%
ARF - Agrawal	**93.35%**	79.89%	83.82%	92.84%	91.84%
ARF - Spam Corpus	70.05%	66.49%	67.79%	**70.09%**	69.24%
ARF - Electricity	68.52%	65.33%	63.87%	68.93%	**69.04%**
ARF - NOMAO	71.28%	71.28%	74.92%	71.29%	**74.92%**
KUE - P2	94.40%	85.94%	94.00%	94.66%	**96.05%**
KUE - SEA	**86.53%**	77.73%	82.27%	86.21%	82.67%
KUE - RTG	96.03%	86.53%	**96.36%**	96.26%	95.77%
KUE - Asset	**93.65%**	73.02%	92.47%	93.64%	92.41%
KUE - Agrawal	94.31%	87.96%	92.62%	**94.40%**	93.48%
KUE - Spam Corpus	65.10%	63.15%	60.30%	63.45%	**65.12%**
KUE - Electricity	**44.16%**	43.46%	43.65%	44.19%	44.10%
KUE - NOMAO	70.29%	70.29%	70.47%	70.35%	**70.74%**

hyperparameter. For instance, if the $rope = 1\%$, any difference between classifiers that is less than 1% is not considered a difference. Unlike Friedman/Nemenyi, this test takes into account the performance values of the classifiers and not the mean ranking. In this experiments, we set $rope = 1\%$

5.3 Results

For clarity, the results reported below were separated by the ensemble types that had similar behavior, only the most relevant results will be presented in this chapter, but the full table of results is given in Appendix A.

5.3.1 Adaptive Random Forest and Kappa Updated Ensemble using Hoeffding Trees

These are the state-of-art methods amongst those tested. Therefore, the expectation was that the potential of gain with selection would be insignificant. Table 5.4 displays the results of these classifiers, combined with all selection methods and generators without drift and the real-world datasets. It is observed that the expectation was confirmed. Even though in some datasets the selection methods were better than without any selection, the improvements were minor.

When we look at the critical distance plot, in Figure 5.6, the methods without selection were placed in the first position, as expected, followed by methods with KNORA-U. LCA executions were placed in the last position with a wide margin. It also presented

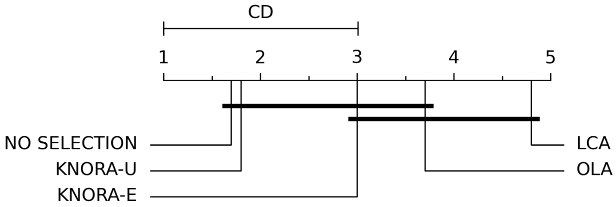


Figure 5.6: Critical Distance Plot for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - Without Drift and Real-World Datasets

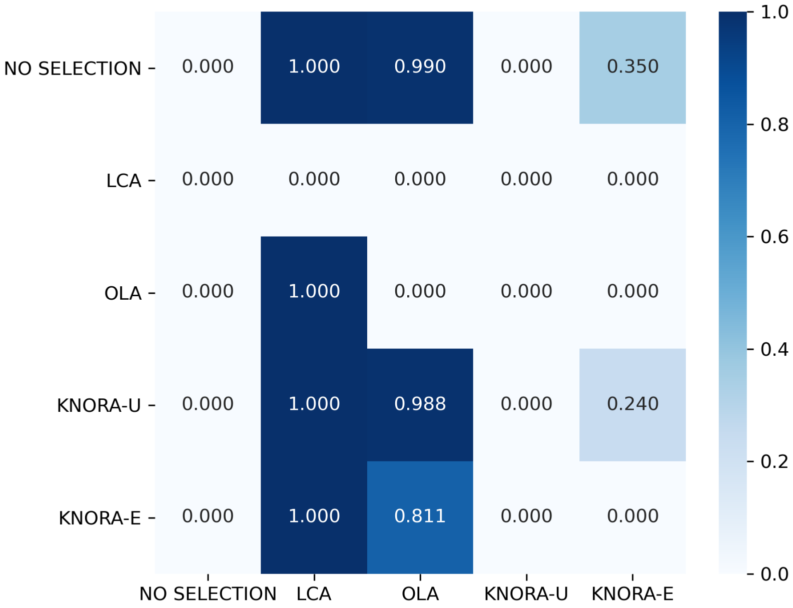


Figure 5.7: Bayesian Analysis Matrix for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - Without Drift and Real-World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

Table 5.5: Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - With Drift

	NO SELECTION	LCA	OLA	KNORA-U	KNORA-E
ARF - RTG	85.08%	79.05%	79.82%	**85.44**%	84.26%
ARF - Asset	**93.82**%	83.62%	92.04%	93.53%	93.15%
ARF - Agrawal	**73.29**%	62.07%	66.66%	71.92%	70.10%
ARF - SEA	86.53%	84.86%	82.33%	**86.62**%	83.82%
KUE - RTG	81.76%	68.02%	79.58%	**82.77**%	81.65%
KUE - Asset	93.15%	61.87%	92.62%	**93.28**%	92.80%
KUE - Agrawal	**79.15**%	61.44%	76.05%	75.96%	77.71%
KUE - SEA	**85.57**%	79.71%	81.01%	85.52%	82.08%

all methods but LCA as being statistically equivalent.

For understanding the Bayesian analysis, in Figure 5.7, one must understand that it means the pairwise matrixes of the Bayesian analysis of all methods pairwise, with *rope* = 1%. The matrix should be interpreted as follows: the intersection between a method *a* on the vertical axis with the method *b* on the horizontal axis is the probability of *a* being better than *b*. The opposite is also true, the intersection of *a* on the horizontal axis with *b* on the vertical axis is the probability of *a* being worse than *b*. If these numbers do not sum to 1 (or 100%), that means that the difference lied in the *rope*.

Thus, analysing Figure 5.7, in the vertical lines of the methods without selection, the probability of them being worse than any other method is zero. This means that there is no possible gain using any selection method in these experiments. Methods without selection, however, also have probability zero of being better than KNORA-U, meaning that these methods are equivalent for the *rope* set (1%). The executions without selection were clearly superior to DCS-LA methods and somewhat superior to KNORA-E, even though in the latter the majority of the probability lied in the *rope*.

The analysis of both statistical tests confirmed the initial intuition. There are no methods that can top the “normal” execution without selection. KNORA-U performed very close to without selection, being virtually equivalent, however, there is no advantage in that, because a significant overhead is added to the normal execution overflow.

Figure 5.8 illustrates the accuracy of these ensembles overtime for the SEA experiment, without drift, across different DCS methods, including the Oracle. It is observed that during the whole execution, the KNORA-U method accuracy was very close to the accuracy with no selection. This is probably due to KNORA-U, as explained in Section 3.1.4, being a more “inclusive” selection method, so more trees are selected.

When drift is present, as displayed in Table 5.5, the observed behavior remains unchanged, as the gain of the selection is minimal or inexistent. In some cases, the executions with some selection methods are not as affected by the drift as the one without

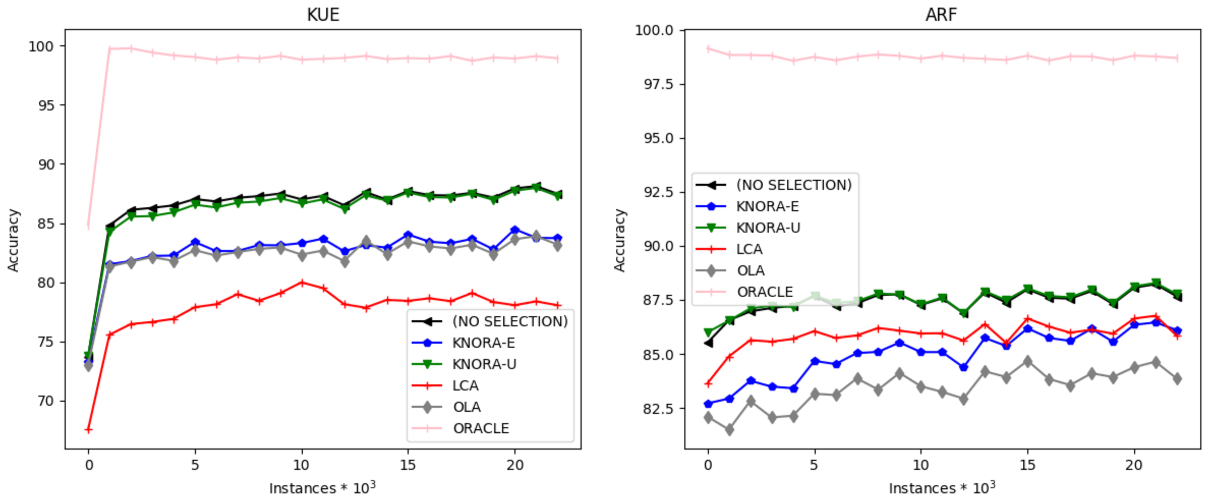


Figure 5.8: Accuracy Evolution Over Time of ARF and KUE with SEA Generator - Without Drift and Real-World datasets

selection. This can be seen in the Figure 5.9 on the left. KUE, when executed with the methods KNORA-E, KNORA-U, and OLA, presented this behavior. With ARF, on the right, however, the selection methods did not present any benefit.

The statistical tests in Figures 5.10 and 5.11 presented little change in the interpretation of the results. The Bayesian analysis confirmed what was observed in Table 5.5, the difference between methods without and with selection was wider. However, KNORA-U remained virtually equivalent to methods without selection, with only 9% of chance of being worse than it.

5.3.2 OzaBag using Hoeffding Trees

OzaBagging composed by Hoeffding trees usually yields slightly worse results than Adaptive Random Forest.

However, since a Hoeffding Tree alone is a quite strong base classifier, with great learning capacity, the dynamic selection results were expected to be better than with ARF or KUE, but still not provide a significant gain.

Table 5.6 displays the results of this classifier without drift and real-world datasets, again combined with all the selection methods. It is observed for all the common generators that the accuracy improvements were small, much like the results with ARF and KUE. However, for the more complicated generator P2, the selection methods KNORA-E and OLA presented a significant increase in the accuracy, 8.59% and 7.85%, respectively. Figure 5.12 shows the accuracy overtime on this generator. It is clear that the selection methods not only performed better but were also able to learn the concept much quicker.

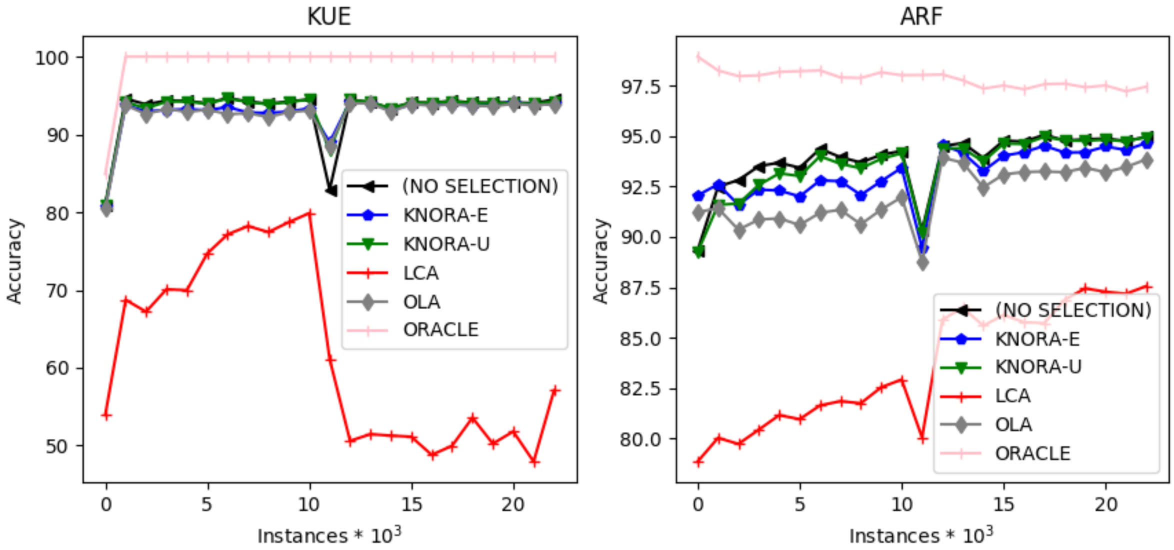


Figure 5.9: Accuracy Evolution Over Time of ARF and KUE with Asset Generator - With Drift

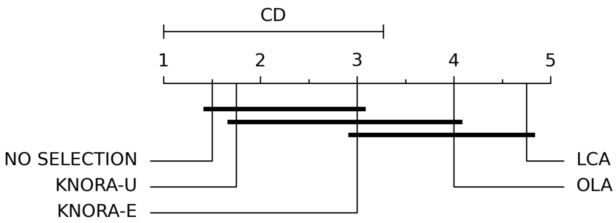


Figure 5.10: Critical Distance Plot for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - With Drift

Table 5.6: OzaBag with Hoeffding Tree - Without Drift and Real-World Datasets

	NO SELECTION	LCA	OLA	KNORA-U	KNORA-E
OzaBag - Agrawal	93.53%	92.15%	93.70%	93.70%	**93.84%**
OzaBag - Asset	94.15%	93.08%	94.02%	**94.16%**	94.08%
OzaBag - SEA	86.10%	85.56%	85.92%	**86.16%**	86.12%
OzaBag - P2	84.87%	79.50%	92.72%	88.06%	**93.46%**
OzaBag - RTG	94.68%	93.50%	95.08%	94.20%	**95.45%**
OzaBag - Spam Corpus	67.15%	63.11%	64.56%	68.34%	**68.46%**
OzaBag - Electricity	**62.54%**	43.48%	43.77%	44.24%	44.36%
OzaBag - NOMAO	**71.28%**	**71.28%**	**71.28%**	**71.28%**	**71.28%**

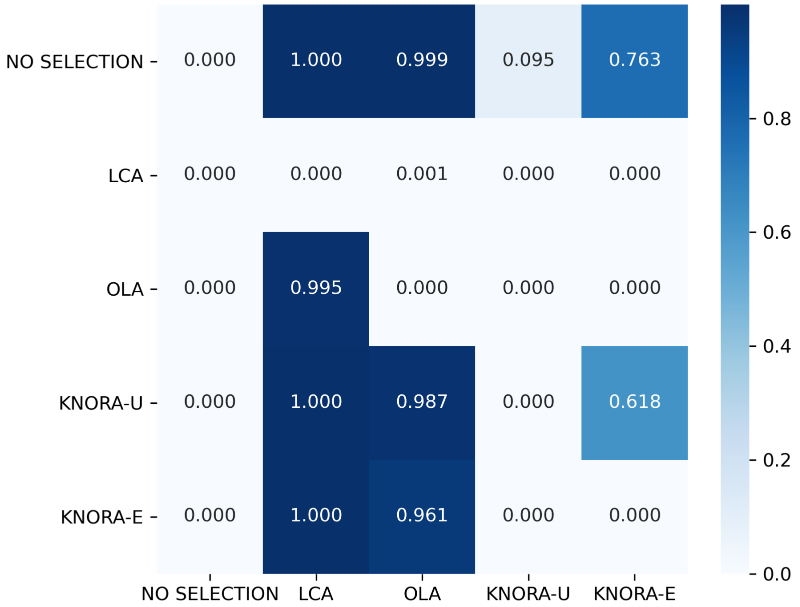


Figure 5.11: Bayesian Analysis Matrix for Adaptative Random Forest and Kappa Updated Ensemble with Hoeffding Tree - With Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

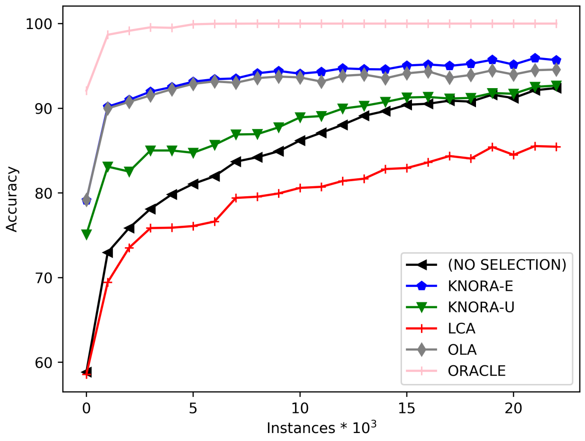


Figure 5.12: Accuracy Evolution Over Time of Ozabag with Hoeffding Tree on the P2 Generator - Without Drift

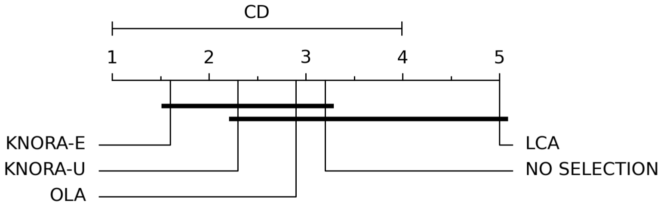


Figure 5.13: Critical Distance Plot for OzaBag with Hoeffding Tree - Without Drift and Real-World Datasets

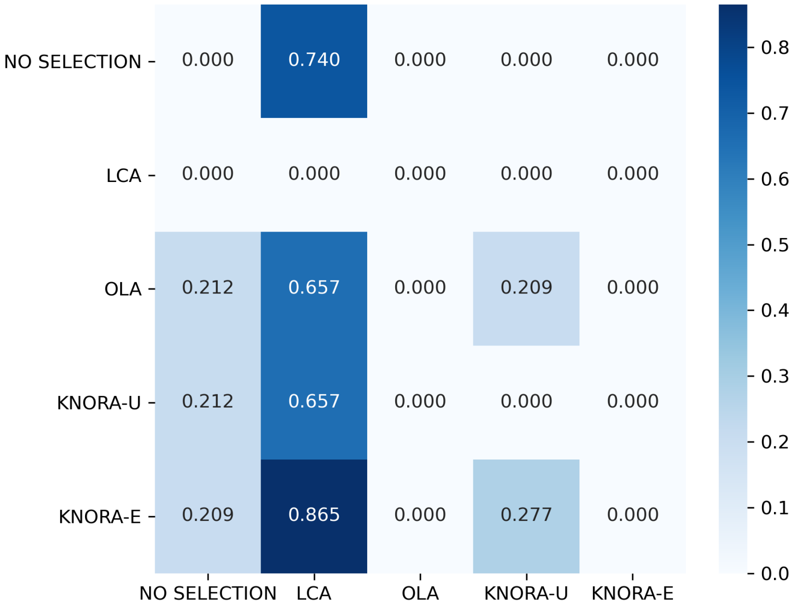


Figure 5.14: Bayesian Analysis Matrix for OzaBag with Hoeffding Tree - Without Drift and Real-World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

Regarding the statistical methods, we can see a change in the observed patterns. In the critical distance plot, in Figure 5.13, methods with KNORA-E were placed in the first position. Methods without selection appeared in the fourth position, winning only against LCA. However, all the methods but LCA were again considered as not significantly different.

Regarding the Bayesian analysis, in Figure 5.14, the only method that methods without selection present any probability of being better is against LCA. On the other hand, the chances of selection methods (OLA, KNORA-U, and KNORA-E) being better than the traditional execution are low, around 20%. This means that they have about 80% of chance of being equivalent.

While the results did not show a clear advantage of selection methods, they can be interpreted as a step, relative to the last results, towards an environment where selection methods are useful.

When drift is added, the results are similar to the ones achieved with ARF and KUE in terms of gain with selection methods, as presented in Table 5.7. As seen with KUE, with some streams such as RTG, in the right side of the Figure 5.7, the selection method KNORA-E was not as affected by the drift as without selection, and also presented a better recovery, keeping the accuracy higher than the execution without selection. For streams like SEA, presented on the right side of Figure 5.15, this behavior was not present.

Looking at the critical distance plot in Figure 5.16, the advantage of selection

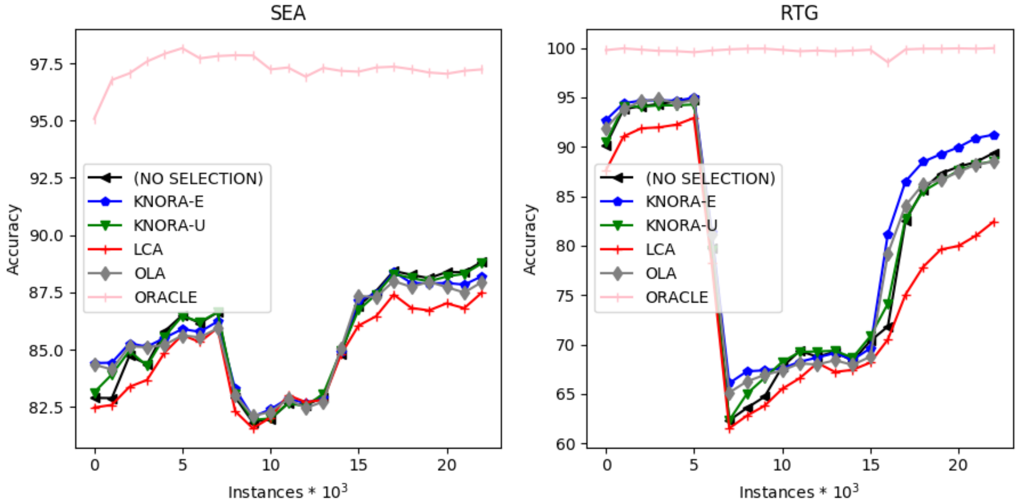


Figure 5.15: Accuracy Evolution Over Time of Ozabag with Hoeffding Tree on the SEA and RTG Generator - With Drift

Table 5.7: OzaBag with Hoeffding Tree - With Drift

	NO SELECTION	LCA	OLA	KNORA-U	KNORA-E
OzaBag - Agrawal	77.67%	76.82%	76.09%	74.19%	**77.93%**
OzaBag - RTG	79.81%	76.56%	80.40%	80.02%	**81.47%**
OzaBag - Asset	93.92%	92.72%	93.79%	**93.92%**	93.91%
OzaBag - SEA	85.38%	84.66%	85.32%	**85.41%**	85.47%

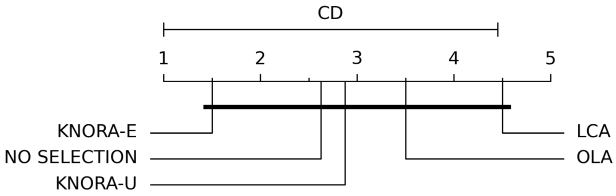


Figure 5.16: Critical Distance Plot for OzaBag with Hoeffding Tree - With Drift

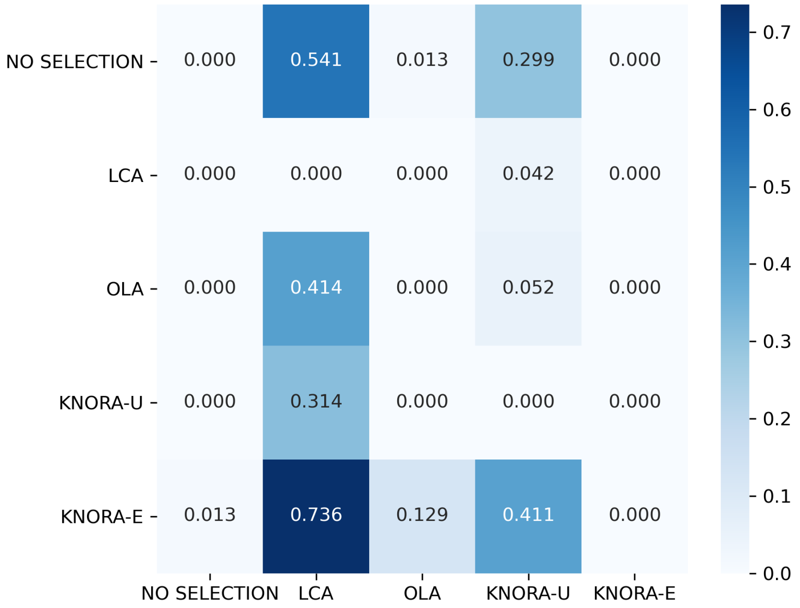


Figure 5.17: Bayesian Analysis Matrix for OzaBag with Hoeffding Tree - With Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

Table 5.8: OzaBag with Percepton - Without Drift and Real-World Datasets

	NO SELECTION	LCA	OLA	KNORA-U	KNORA-E
OzaBag - Asset	51.56%	51.93%	82.00%	72.67%	**82.12%**
OzaBag - Agrawal	**49.97%**	**49.97%**	**49.97%**	**49.97%**	**49.97%**
OzaBag - P2	54.73%	54.56%	86.95%	82.88%	**87.10%**
OzaBag - SEA	78.51%	74.34%	83.92%	82.33%	**84.23%**
OzaBag - RTG	86.34%	85.23%	86.68%	86.41%	**86.86%**
OzaBag - Spam Corpus	45.60%	45.42%	**52.36%**	47.48%	46.91%
OzaBag - Electricity	53.62%	42.83%	**57.17%**	42.38%	54.95%
OzaBag - NOMAO	**71.28%**	**71.28%**	**71.28%**	**71.28%**	**71.28%**

methods decreased. Methods without selection are now placed in the second position. KNORA-E remained in first. All the methods were set as not significantly different.

Concerning the Bayesian analysis in Figure 5.17, the vertical line of methods without selection shows that there are virtually no chance of selection methods presenting any gain. No selection methods also did not present significant advantage over selection methods, except for LCA.

5.3.3 OzaBag using Perceptrons

Perceptron, by default, is a simple classifier that learns linear separations between classes. Thus, if a problem is not linear, it is expected to have a bad performance. When OzaBagging is learned using Perceptron as the base learner, such ensemble can be viewed as an ensemble of weak classifiers. Therefore, the gain of dynamic selection is expected

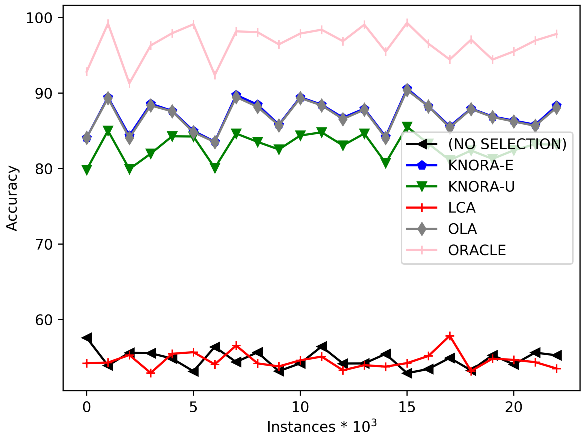


Figure 5.18: Accuracy Evolution Over Time of OzaBag with perceptron on the P2 Generator

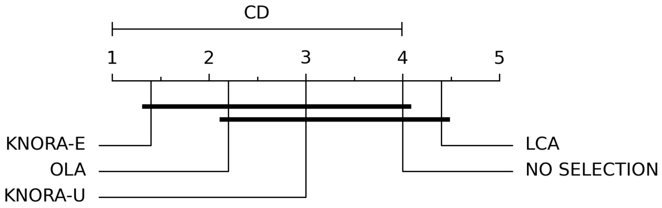


Figure 5.19: Critical Distance Plot for OzaBag with Perceptron - Without Drift and Real World Dataset

to be much more visible in this experiment.

As seen in Table 5.8, the effects of the dynamic selection are clear, especially in the Asset generator, in which the accuracy rate improved in more than 30%.

The P2 experiment also presented a clear improvement in accuracy, and the accuracy over time for this experiment is displayed in Figure 5.18. Selection methods like KNORA-E and OLA presented significantly higher accuracy during the whole time of execution. KNORA-U also presented significant improvement over the execution without selection. However it performed slightly worse than KNORA-E and OLA. The Oracle showed that there is still some potential gain to be achieved using the selection methods.

Regarding the critical distance plot, in Figure 5.19, the expected advantage of selection methods is confirmed, with only LCA being placed in a worse position than without selection. However, KNORA variants and OLA were set as not significantly different than without selection.

When looking at the Bayesian analysis, in 5.20, a clear advantage of selection methods can be seen, with all of them but LCA having around 88% of probability of being better than without selection.

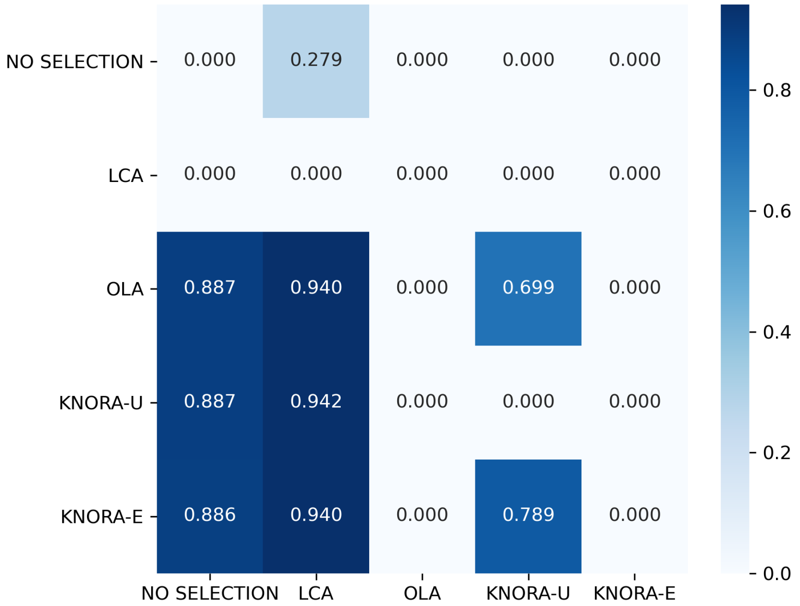


Figure 5.20: Bayesian Analysis Matrix for OzaBag with Perceptron - Without Drift and Real-World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

Table 5.9: OzaBag with Percepton - With Drift

	NO SELECTION	LCA	OLA	KNORA-U	KNORA-E
OzaBag - SEA	77.56%	76.10%	83.57%	81.83%	**83.82%**
OzaBag - Agrawal	**49.97%**	**49.97%**	**49.97%**	**49.97%**	**49.97%**
OzaBag - Asset	51.50%	51.67%	72.85%	68.94%	**72.99%**
OzaBag - RTG	68.59%	67.33%	73.44%	70.31%	**73.59%**

These results, as expected, showed that in such an environment of an ensemble of weak classifiers, selection methods are much more welcomed than in the previous environments. However, they do not top the state-of-art results for the same streams.

With drift present, the improvement was also very clear, as seen in Table 5.9. However, there was not a clear difference in terms of impact when the drift occurred, as seen with the other classifiers.

Regarding the statistical tests, virtually no change happens in the critical distance plot, in Figure 5.21. But in the Bayesian analysis, in Figure 5.22, the advantage of OLA

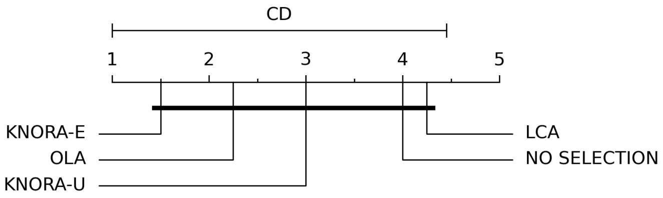


Figure 5.21: Critical Distance Plot for OzaBag with Perceptron - With Drift

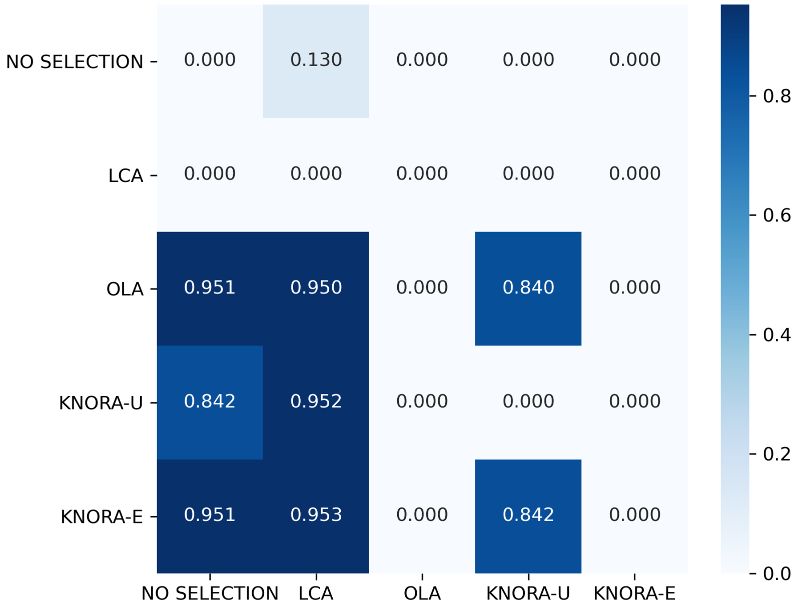


Figure 5.22: Bayesian Analysis Matrix for OzaBag with Perceptron - With Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

and KNORA-E are increased, with they having around 95% of chance of being better than methods without selection. These results are interesting because they show a potential of selection methods being useful in drifts contexts.

5.3.4 Single Base Classifiers

For comparison reasons, Table 5.10 presents the accuracy of the base classifiers used with the ensembles alone. The Hoeffding Tree results were very close to the state-of-art method ARF and KUE, without selection, with all streams but P2. Similar thing happened with the Perceptron, compared to Ozabag composed of this base classifier.

No statistical tests were run in the base classifiers results because they were added only as reference, and not to compare against selection methods.

5.4 Conclusion

The experiments reported in this chapter showed that the same characteristics usually required for dynamic selection to be useful in batch machine learning are also required for the online environment.

When dynamic selection is applied to robust classifiers such as Adaptive Random Forest and Kappa Updated Ensemble, the impact in accuracy is often small, and some

Table 5.10: Accuracy of the Base Classifiers in the Streams

Stream	Hoeffding Tree	Perceptron	Naive Bayes
P2	**83.50%**	54.49%	56.79%
SEA	**85.91%**	77.35%	82.48%
Asset	**93.14%**	50.79%	85.53%
Agrawal	93.28%	45.00%	**93.68%**
RTG	**94.10%**	86.24%	85.73%
Spam Corpus	72.76%	47.85%	76.42%
Electricity	65.73%	55.83%	64.00%
NOMAO	69.10%	67.28%	68.42%
SEA - Concept	**85.07%**	76.70%	82.84%
Asset - Concept	**92.98%**	50.79%	84.81%
Agrawal - Concept	**84.75%**	45.00%	83.76%
RTG - Concept	**85.60%**	67.85%	71.75%

times worse. Each base classifier of these ensembles, especially when they are composed of Hoeffding Trees, are not specialists in specifics regions of the decision space. Instead, they are capable of predicting most parts of it, hence, selection does not make a difference.

When the slightly less robust ensemble, i.e., OzaBag with Hoeffding Trees as base classifiers, was used, the results started to become more apparent. In the complex P2 generator, selection significantly improved the accuracy of the ensemble. However, as shown in Table 5.10, Hoeffding Tree is a strong classifier on its own, achieving satisfactory results on most streams without the need for an ensemble. Thus, this is not the ideal environment for classical dynamic selection to work.

With the ensemble of weak linear classifiers, i.e., OzaBag with Perceptrons, the positive impact of dynamic classifier selection in accuracy rates was clearly observed for the P2, SEA, and Asset generators. In the Asset and P2 generators, the improvement rates observed were over 30%.

After all, the dynamic selection was proved to be effective in data stream mining when applied under similar conditions to those observed for batch environments. However, as explained in Section 5.1, the objective of this experiment was not to compare dynamic selection techniques with state-of-art data stream mining techniques. The objective was to understand if the gains observed by dynamic selection in the offline environment were also true in the online environment.

Another point observed was that there is a gap for more complex data stream generators. The currently used ones are mostly linear or simple. This was shown in Table 5.10, where the base classifiers such as the Hoeffding Tree alone achieved results close to the state-of-art of dynamic ensembles. More complex generators could represent real-world functions that are not easily learned from the current classifiers. Dynamic selection could also present some potential gain in these cases.

Chapter 6

DCS Method Focused on Data Stream Mining

As discussed in Section 2.3, time is an important limitation when dealing with data stream mining. Krempl et al. (2014) explained that in this area, three main challenges are faced: volume, velocity, and volatility. Thus, in problems with a great amount of data, expected to run in a reasonable time, and with the possibility of concept drifts, proposals must have this concern well addressed in order for them to be useful.

Although a satisfactory statistical accuracy is important, it must be weighted with how much time it took to be achieved in comparison with other methods. For example, assuming that a model achieved 95% accuracy using the technique “a” taking 100 seconds to run. Another model using another technique “b” achieved 93% taking 10 seconds. In a careful analysis, the technique “a” should not be automatically chosen. Even though the technique “b” was not as accurate as “a”, it can handle larger amounts of data in a shorter time.

However, when analyzing DCS methods, most of them rely on the application of the traditional K-Nearest Neighbor algorithm for every instance to be predicted. This causes the necessity of computing a distance for all instances in a validation set, for each new test instance that arrives. In batch machine learning, this is not a very concerning problem since the number of instances available is usually lower. But in dynamic environments, added to the fact that the validation set might change over time, this might be considered very costly as it induces a significant computational overhead.

Thus, different approaches are needed in this case, such as using KD-Tree, as explained in Section 2.7.1, instead of traditional KNN. With KD-Tree, the search space on the validation set is drastically reduced to only the instances that are most likely to be similar to the test instance. The computational complexity of traditional KNN is $O(1)$ for training and $O(k \times n \times d)$ for testing, while KD-Tree is $O(d \times n \times \log n)$ and $O(k \times \log n)$

for training and testing, respectively.

In this chapter, we propose the Double Dynamic Classifier Selection (DDCS). It was named as is because we are dealing with dynamic selection with dynamic classifiers, culminating in the “double dynamic” term.

6.1 Double Dynamic Classifier Selection

The structure of Double Dynamic Classifier Selection (DDCS) is similar to DYNSE and PDCS (I and II). Algorithm 1 depicts its training process. First, when a chunk arrives to be trained, in line 1, the current ensemble E is retrieved (even if it is still empty). Next, from lines 2 to 4, if E is empty, and the hyperparameter `init_all` is set, all the ensemble is pre-initialized with empty instances of the selected base classifier. The effect of `init_all` is to define whether the ensemble is going to be complete from the beginning or if it is going to be filled one by one as new chunks arrive.

In line 5, a new base classifier is created. In lines 6 to 8, if the maximum size N of the ensemble E is reached, the member with the current lowest accuracy is removed. Line

Algorithm 1: Training Step of DDCS

```
    input  : BaseClassifier: base classifier to add to the ensemble
    input  : $(X, Y)$: chunk to train
    input  : $N$: maximum possible size of $E$
    input  : use_bagging: if set, train using online bagging
    input  : init_all: if set, initialize all the classifiers of the ensemble on first run
 [1] $E \leftarrow$ get_ensemble();
 [2] if $|E| == 0$ & init_all then
 [3]     $E \leftarrow \{e_i |$ such that $1 \leq i \leq (N-1)$ and $e_i$ is a BaseClassifier instance$\}$;
 [4] end
 [5] new_classifier $\leftarrow$ new BaseClassifier();
 [6] if $|E| = N$ then
 [7]     Remove from $E$ the member $e_i$ with lowest accuracy;
 [8] end
 [9] $E \leftarrow E \cup \{$new_classifier$\}$;
[10] if use_bagging then
[11]     For $i = 1, \ldots, |E|$, train $e_i$ using $(X, Y)$ using Online Bagging;
[12] else
[13]     For $i = 1, \ldots, |E|$, train $e_i$ using $(X, Y)$;
[14] end
[15] $V \leftarrow (X, Y)$;
[16] KDTree $\leftarrow$ get_searcher();
[17] KDTree.clear();
[18] KDTree.train(V);
```

Algorithm 2: Prediction Step of DDCS

```
    input  : $x$: instance to predict
    input  : dcs: DCS method to use
    input  : $k$: number of neighbors to gather
    output: Prediction $\hat{y}$ of $x_i$
[1]KDTree ← get_searcher();
[2]$E$ ← get_ensemble();
[3]neighbors ← KNNSearcher.find_neighbors(x, k);
[4]selected_members ← apply_dcs(neighbors, dcs, E);
[5]$\hat{y}$ ← predict(x, selected_members);
[6]return $\hat{y}$;
```

9 adds the newly created classifier to the ensemble. In lines 10 to 14, if the hyperparameter `use_bagging` is set, the whole ensemble is trained using the online bagging schema, explained in Section 2.8.1. If `use_bagging` is not set, then the ensemble is trained with each member receiving each instance of the chunk once. Finally, the chunk is assigned to the validation set V, in line 15. Then, in lines 16, 17, and 18, the KD-Tree algorithm is queried, reset so it can be trained again and trained with the new validation set.

The process for testing with DDCS is strictly the same as with DYNSE and PDCS I and II. This process is displayed on Algorithm 2. For a test instance x, in lines 1 and 2 the KD-Tree and the ensemble E are retrieved. Then, in line 3 the nearest neighbors of x in the validation set are queried. Line 4 selects the members of the ensemble that are most competent to predict x, using any traditional DCS method. Finally, in line 5, the prediction is made using the members selected and returned in line 6.

It is noteworthy to state that unlike DYNSE and PDCS, the whole ensemble of DDCS is always updated, thus, providing stronger classifiers that are trained on older and newer data. Because of this characteristic, DDCS requires that only online classifiers are used.

6.2 Experimental protocol

6.2.1 Experimental protocol

In our experiments, the ensembles were tested following the interleaved chunks test-then-train process. This can be seen as a variation of the prequential schema that works with chunks. The stream is divided into chunks of one thousand instances, then each chunk is firstly used for testing and later for training, with the exception is the first chunk, which is used solely for training.

Table 6.1: Stream Generators and Datasets Used in the Experiments

Name	# Instances	# Features	# Numeric Features	# Categoric Features	# Classes	Imb. Ratio
SEA	100,000	3	3	0	2	0.47
Agrawal	100,000	9	6	3	2	0.48
P2	100,000	2	2	0	2	1.00
Asset	100,000	5	0	5	2	1.00
NOMAO	34,465	118	118	0	2	0.39
Spam Corpus	9,324	39,917	0	39,917	2	0.34
Electricity	45,312	8	8	0	2	0.73

The experiments conducted and depicted in Table 6.1 included synthetic and real-world datasets. The synthetic generators were: Agrawal (Agr) (AGRAWAL; IMIELINSKI; SWAMI, 1993), Asset Negotiation (An) (ENEMBRECK et al., 2007; BARDDAL et al., 2016), SEA (STREET; KIM, 2001), and Valentini P2 (P2) (VALENTINI, 2006). The real-world datasets were Electricity (Elec) (RODRIGUES; GAMA; PEDROSO, 2008), Nomao (CANDILLIER; LEMAIRE, 2013), and Spam Corpus (KATAKIS; TSOUMAKAS; VLAHAVAS, 2006). All of the synthetic streams were executed with 100,000 instances. The streams were set to have zero, one, two, and three equally distributed concept drifts, abrupt and gradual, the gradual drift happened over a window of 1000 instances.

DDCS was compared against the OzaBag (OZA, 2005), Adaptive Random Forest (ARF) (GOMES et al., 2017) and the Dynamic Selection Based Drift Handler (DYNSE) (ALMEIDA et al., 2016). All the ensembles were set to have 100 Hoeffding trees as base classifiers. It is noteworthy that this number does not necessarily mean an optimal size for any of the methods, it is just to ensure a fair comparison. All ensembles but OzaBag were executed with 4 threads, OzaBag current state-of-art implementation does not provide a multithread implementation.

In this analysis, different hyper-parameter values (`true` or `false`) in DDCS for `use_bagging` and `init_all` were used. As for the dynamic selection algorithms, KNORA-E (KNE), KNORA-U (KNU) (KO; SABOURIN; BRITTO JR., 2008) and no selection method (majority vote) were chosen for the comparison. These algorithms were selected because of their smaller computational cost when compared to more recent approaches such as META-DES (CRUZ et al., 2015). The value of K for the K-Nearest neighbors for both DDCS and DYNSE was set to 7, as suggested in (CRUZ; SABOURIN; CAVALCANTI, 2018)

Since execution time and memory are of vital importance for data stream mining settings, an important hyper-parameter in this sense of ARF was optimized to provide a fair comparison, `max_features`. This parameter sets the size of the feature subset that ARF will use when splitting a node. In some cases, changes in this parameter can lead to changes in time of execution without losing its accuracy, thus, it is important to optimize it. The values used in this parameter varied from 2 to 6, with step 1, in addition to the square root of the total number of features. In the following results, we show two variants for ARF.

The remainder of the hyper-parameters were set as the default provided in the Massive Online Analysis (MOA) framework (BIFET et al., 2010). The experiments were executed 20 times and accuracy, execution time (CPU time), and memory use were assessed. We applied two statistical tests on the results, i.e., the Friedman and Nemenyi combination proposed in (DEMŠAR, 2006), and the pairwise Bayesian comparison (BENAVOLI et al., 2017).

Regarding accuracy, the results are the average of the 20 executions. Regarding processing time and memory consumption, all of the values were presented both as the absolute value and as a relative value to the slowest method and the one that consumed more memory.

In the following results, we present only two variants for ARF. The first regards the optimized version of ARF towards accuracy (ARF_ACC), which means that it is the execution of ARF with the value for `max_features` that presented the highest accuracy. The second focuses on the execution with `max_features` that required less processing time (ARF_TIME). Regarding DDCS results, only the executions with both parameters `use_bagging` and `init_all` set to *false* are reported, as these were the most positive results. The complete list of results with all the variants is displayed in <https://docs.google.com/spreadsheets/d/1PZSygbHw7gGV4lcHkdu-Hvna9l8vGTdUNxduYcToN5Y/edit?usp=sharing>. The results were subdivided into 4 sections, no drift, gradual drift, abrupt drift, and real-world.

6.3 Results

6.3.1 No Drift

For the results regarding the streams without any concept drift. In Table 6.2, we can see that the winner methods were always ARF_ACC and Ozabag. However, in the streams Agrawal and Asset, all DDCS variants outperformed ARF_ACC, with more than

Table 6.2: Results Comparing DDCS with Streams Without Drift

	DYN_KNE	DYN_KNU	DYN_NO_SEL	OzaBag	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
Agr	86.667	78.556	85.793	**93.716**	92.600	93.027	93.314	88.506	68.556
An	92.989	92.630	92.851	**94.005**	93.613	93.796	93.763	93.472	91.662
P2	76.919	79.175	57.872	80.920	87.892	86.757	78.456	**97.951**	97.902
SEA	88.528	88.136	87.929	88.025	88.386	88.343	87.840	**89.350**	88.775

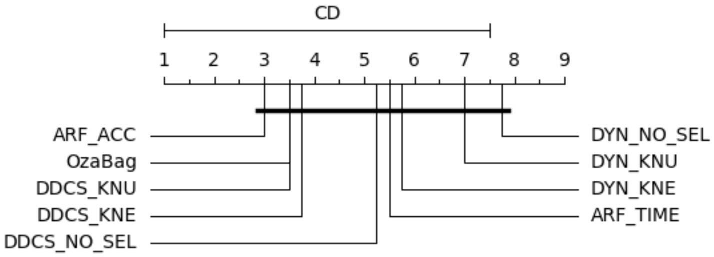


Figure 6.1: Critical Distance Comparing DDCS with Streams Without Drift

4% with Agrawal. DDCS methods also outperformed DYNSE methods in all streams but SEA. It is noteworthy that ARF_TIME was able to keep relatively competitive results, relative to ARF_ACC, in the streams Asset, P2, and SEA, however in Agrawal, it was greatly outperformed by ARF_ACC.

In the critical distance plot, in Figure 6.1, the dominance of ARF_ACC is confirmed, with they appearing in first and second position respectively. However, Ozabag is closely followed by DDCS with KNORA-U and KNORA-E. All DYNSE variants were placed in the last positions. All of the methods were defined as not significantly different according to this test.

Regarding the Bayesian analyisis, for the *rope* = 0.5%, in Figure 6.2 the horizontal line of ARF_ACC presents the highest and most consistent values, meaning that it is the method with the highest probability of being better than the others. Comparing ARF_ACC to DDCS methods with selection, ARF_ACC has between 50% and 57% of being better than DDCS, the opposite (probability of DDCS being better than ARF_ACC) is only about 25%, meaning that around 25% is in the *rope*. It is worth to cite that ARF_ACC has around 84% of the probability of being better than ARF_TIME, while the opposite number is 0%, meaning that for this *rope*, they have only around 16% of being considered equivalent.

For the *rope* = 1%, in Figure 6.3, as expected, the values decreased in several places. It is noteworthy that the chance of ARF_ACC outperforming both DYNSE and DDCS with KNORA-E dropped about 30% and 13% respectively. With KNORA-U, this drop was quite smaller, with 6% for DYNSE and 5% for DDCS. This means that KNORA-E was able to achieve a higher margin in the cases where it outperformed ARF_ACC.

In our experiments with streams without any drift, ARF_ACC was the superior choice in most cases. However, DDCS presented results relatively close and even better in some streams. If the processing time and memory consumption is added to the equation, DDCS becomes a quite competitive option. Table 6.3 shows the processing time of all method with relation to ARF_ACC, which in this round of experiments was the

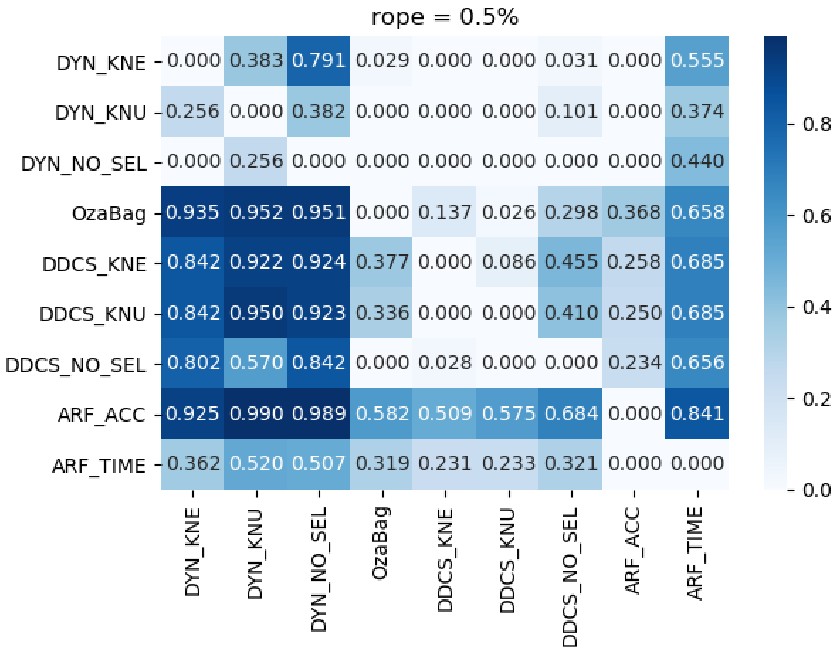


Figure 6.2: Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Streams Without Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

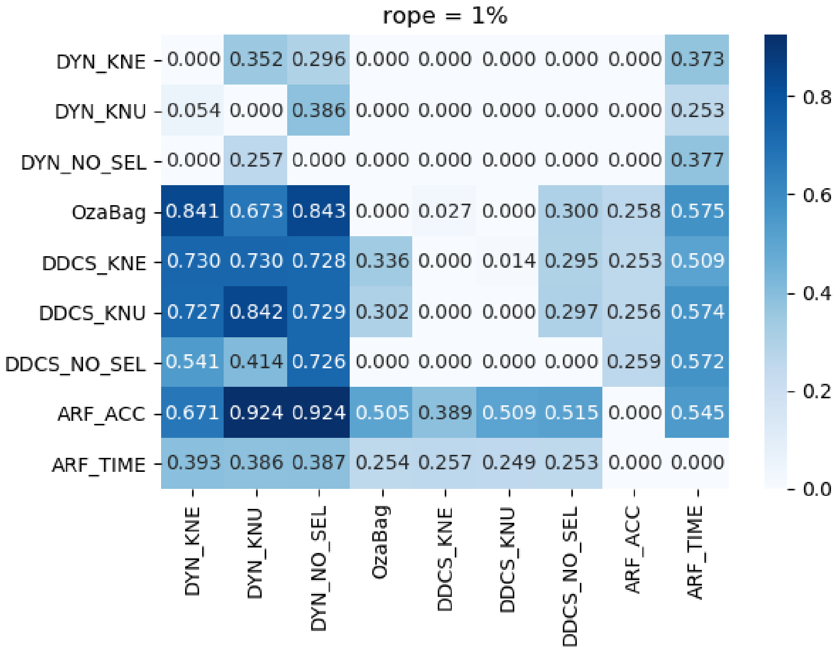


Figure 6.3: Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Streams Without Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

Table 6.3: Results Comparing Processing Time and Memory Consumption with Streams Without Drift Against ARF_ACC

Method	Processing Time (s)	Method	Memory Consumption (Ram-hours)
DYN_NO_SEL	0.308% (3.314)	DYN_NO_SEL	0.004% (1.358e-07)
DDCS_NO_SEL	1.014% (10.903)	DYN_KNU	0.018% (5.835e-07)
DYN_KNU	1.600% (17.204)	DYN_KNE	0.037% (1.212e-06)
DYN_KNE	1.738% (18.688)	DDCS_NO_SEL	0.040% (1.320e-06)
DDCS_KNU	2.332% (25.074)	DDCS_KNU	0.174% (5.748e-06)
DDCS_KNE	2.664% (28.643)	DDCS_KNE	0.217% (7.147e-06)
OzaBag	4.365% (46.927)	OzaBag	0.489% (1.613e-05)
ARF_TIME	25.147% (270.330)	ARF_TIME	12.208% (4.025e-04)
ARF_ACC	100.000% (1074.995)	ARF_ACC	100.000% (3.297e-03)

Table 6.4: Results Comparing DDCS with Streams with Gradual Drift

	DYN_KNE	DYN_KNU	DYN_NO_SEL	OzaBag	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
Agr-2X	83.527	73.768	78.949	89.969	90.890	90.949	**91.029**	84.762	69.507
Agr-3X	80.191	71.289	72.882	86.250	**89.864**	89.203	89.357	81.960	70.318
Agr-4X	80.745	73.383	73.303	83.880	**90.157**	88.811	88.513	84.155	72.743
An-2X	92.729	91.422	85.568	93.712	93.545	**93.785**	93.583	93.484	91.456
An-3X	92.517	91.072	79.923	93.445	**93.663**	93.579	93.392	93.346	91.295
An-4X	92.031	90.421	78.858	92.623	**93.583**	93.321	93.071	93.193	91.191
SEA-2X	88.440	87.482	87.231	87.473	88.222	87.946	87.551	**89.178**	88.535
SEA-3X	88.254	87.080	86.519	86.929	88.065	87.761	87.159	**88.963**	88.043
SEA-4X	88.104	86.856	86.086	86.150	87.964	87.740	87.125	**88.668**	88.018

worst, and the mean absolute time. DDCS with KNORA-U for instance (which featured in the third position in the critical distance plot), took only about 2.3% of the time ARF_ACC took to run and consumed only around 0.1% of the consumed memory. Comparing ARF_TIME with ARF_ACC, it is clear that the optimization focusing on time provided a great reduction in processing time (25%) and memory consumption (12%).

In conclusion, when dealing with datasets without drift, the use of DDCS is justifiable only when processing time and memory consumption are a must. Otherwise ARF_ACC was the obvious choice for such scenarios, with a lower probability of being outperformed. Other methods such as DYNSE variants also presented a much lower processing time and memory consumption (even lower than DDCS), but the accuracy gap of these methods in such scenario is in general too wide, except for the SEA generator.

6.3.2 Gradual Drift

Regarding the results on the streams set to have gradual drifts, Table 6.4 shows a different pattern than that observed in environments without any drift, DDCS variants, especially with KNORA-E, was able to outperform the other methods in all streams but SEA. With SEA, DDCS presented close results to ARF_ACC, but as it happened in the past results, it was again outperformed by DYNSE. Comparing ARF_ACC to ARF_TIME, the biggest discrepancies in accuracy were again observed in the Agrawal stream.

DDCS with KNORA-E and KNORA-U were placed in the first and second posi-

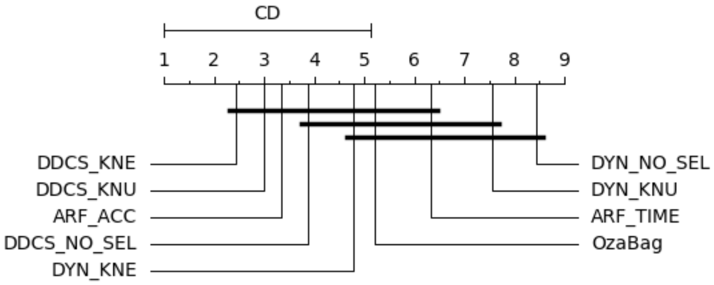


Figure 6.4: Critical Distance Comparing DDCS with Streams with Gradual Drift

tions respectively, as shown in the critical distance plot in Figure 6.4. They were closely followed by ARF_ACC. DYNSE with KNORA-E performed better in this scenario than without drift, lying in the fifth position. This was expected as DYNSE was focused on dealing with concept drifts. Only DYNSE without selection and with KNORA-U were marked as significantly different from the other methods.

Looking at the matrix of the Bayesian analysis, for *rope* = 0.5%, in Figure 6.5, the horizontal line of DDCS with KNORA-E was the method that presented the highest values for most columns, meaning that it is the method that has more probability of being better than the others. Looking at the vertical line of the same method, we can observe most values zero or close to zero, meaning that it is also the method least “threatened’ by the others. I.e, if DDCS with KNORA-E is not better than the method *a*, they must be equivalent (in the *rope*).

In Figure 6.6, for *rope* = 1%, an interesting difference is that all DDCS variants became virtually equivalent among each other, meaning that there is a chance of almost 100% of the difference between them lies in the *rope*. ARF_ACC was the most competitive method against DDCS variants, yet, with probability values close to zero.

Regarding processing time, in Table 6.5, the relative difference of DDCS methods against ARF_ACC dropped. For instance, DDCS with KNORA-E now took 7.1% of the time ARF_ACC used to process (against 2.6% in scenarios without drift). The same happen for memory consumption, with DDCS with KNORA-U now using 0.6% of the memory used by ARF_ACC (against 0.1% without drift). These values, however, are still significantly low, compared to ARF_ACC.

In conclusion, in a gradual drift scenario, DDCS both with KNORA-E and KNORA-U presented a strong basis for it to be considered a choice for dealing with such scenarios. It presented virtually no chance of being outperformed by other methods, however, it is important to highlight that this does not mean that can not be equivalent. This comes together with a much lower processing time and memory consumption. As for DYNSE, although it did perform better than without drift, it is still no match for the current

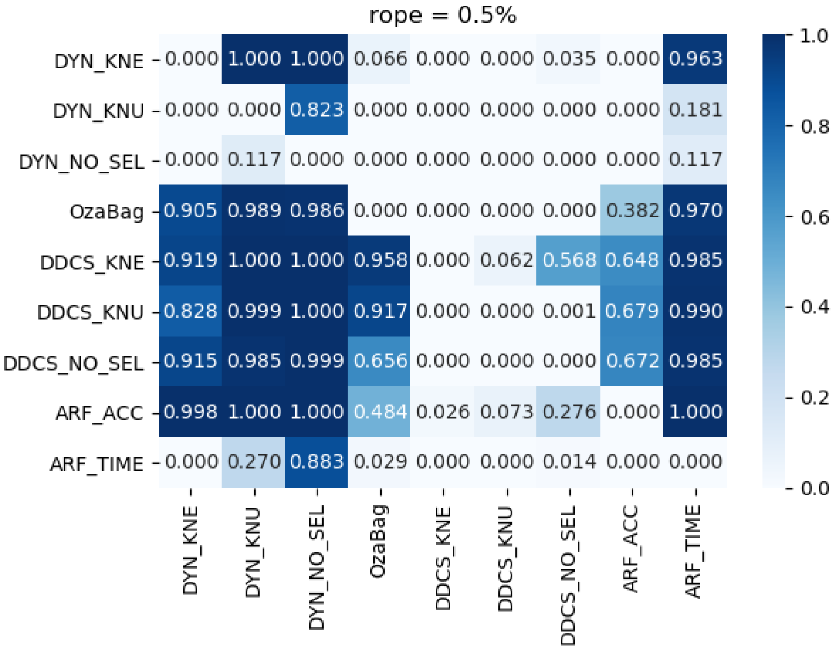


Figure 6.5: Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Streams with Gradual Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

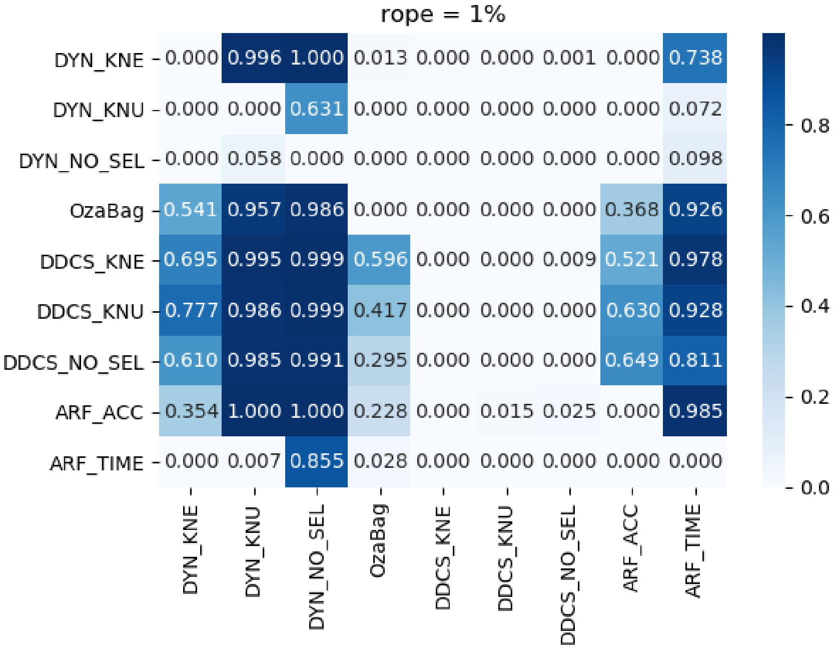


Figure 6.6: Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Streams with Gradual Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

state-of-art ARF_ACC.

Table 6.5: Results Comparing Processing Time and Memory Consumption with Streams with Gradual Drift Against ARF_ACC

Method	Processing Time (s)	Method	Memory Consumption (Ram-hours)
DYN_NO_SEL	0.539% (3.543)	DYN_NO_SEL	0.005% (1.362e-07)
DYN_KNU	2.997% (19.704)	DYN_KNU	0.052% (1.295e-06)
DDCS_NO_SEL	3.192% (20.991)	DYN_KNE	0.056% (1.395e-06)
DYN_KNE	3.244% (21.330)	DDCS_NO_SEL	0.073% (1.818e-06)
DDCS_KNU	6.359% (41.811)	DDCS_KNU	0.674% (1.686e-05)
DDCS_KNE	7.106% (46.724)	DDCS_KNE	0.787% (1.966e-05)
OzaBag	13.134% (86.360)	OzaBag	2.386% (5.963e-05)
ARF_TIME	25.873% (170.129)	ARF_TIME	10.836% (2.709e-04)
ARF_ACC	100.000% (657.556)	ARF_ACC	100.000% (2.500e-03)

Table 6.6: Results Comparing DDCS with Streams with Abrupt Drift

	DYN_KNE	DYN_KNU	DYN_NO_SEL	OzaBag	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
Agr_2X	83.129	74.448	79.070	90.617	91.053	91.148	**91.763**	84.595	68.808
Agr_3X	80.554	71.501	71.317	87.976	90.113	90.228	**90.347**	82.174	69.783
Agr_4X	81.185	74.364	72.753	87.337	**90.507**	89.394	89.438	84.925	72.401
An_2X	92.736	91.144	85.589	**93.704**	93.428	93.485	93.513	93.267	91.472
An_3X	92.469	90.558	79.702	**93.489**	93.407	93.464	93.317	93.286	91.266
An_4X	91.714	90.204	78.636	92.752	92.839	**92.947**	92.916	92.810	90.876
SEA_2X	88.388	87.486	87.214	87.524	88.178	88.053	87.552	**89.148**	88.527
SEA_3X	88.220	87.032	86.544	86.890	88.047	87.771	87.146	**88.966**	88.106
SEA_4X	88.234	86.956	86.129	86.213	88.046	87.741	87.126	**88.723**	88.077

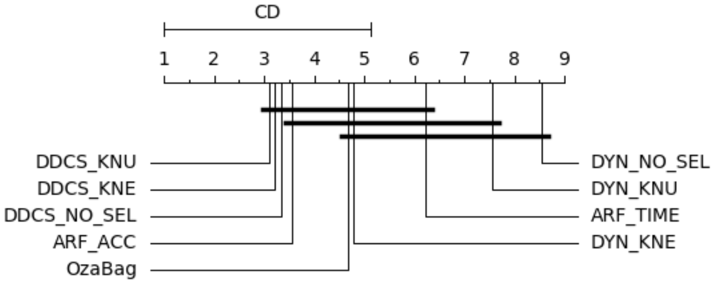


Figure 6.7: Critical Distance Comparing DDCS with Streams with Abrupt Drift

6.3.3 Abrupt Drift

In the abrupt drift results, in Table 6.6, DDCS methods were the winner in 4 streams, however, in two of them DDCS without any selection method presented the best results. The same pattern in SEA repeated, ARF_ACC was the winner and DYNSE methods were better than DDCS.

The critical distance plot in Figure 6.7 presented an unexpected result, which was DDCS without selection being better positioned than ARF_ACC. DDCS with KNORA-U and KNORA-E was placed in first and second position. Following the same pattern as the results with gradual drifts, the only methods that were considered to be significantly different than the others were DYNSE with KNORA-U and without selection.

For the Bayesian analysis matrix, with *rope* = 0.5%, in Figure 6.8, the results were very similar to the ones obtained in the gradual drift experiments. Looking at the vertical line of DDCS with KNORA-E and KNORA-U, we can see that the probabilities of any other method being better than DDCS is virtually zero. DDCS also presented probabilities close to 100% when compared to DYNSE variants, and DDCS with KNORA-E presented probabilities around 64% of being better than ARF_ACC (the remainder of the probability lied most in the *rope*).

Regarding the matrix for the *rope* = 1%, the results did not drastically changed. The probability of DDCS methods being better than some other methods (especially ARF_ACC) decreased a little. However, the probability of DDCS with KNORA-E being worse than any other method is literally 0%.

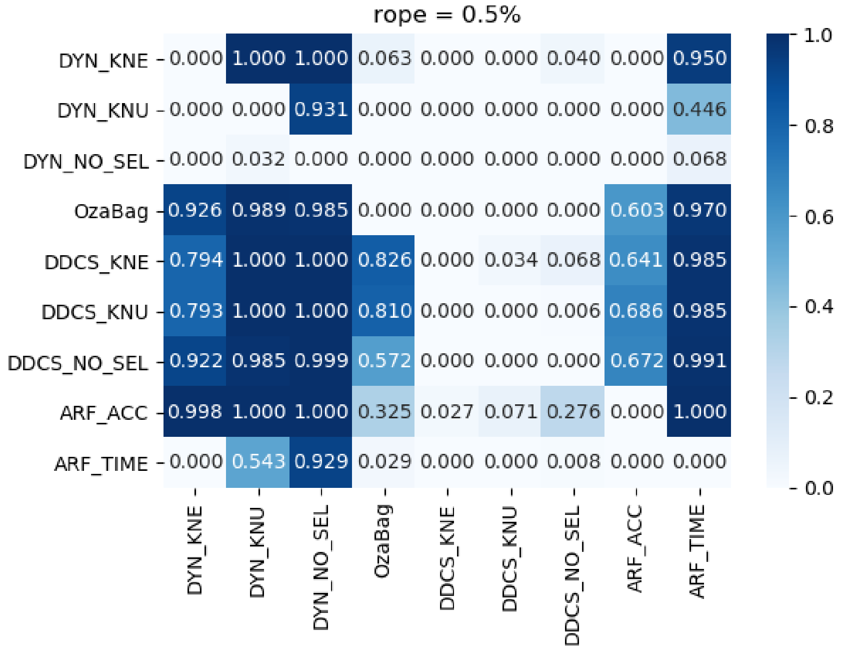


Figure 6.8: Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Streams with Abrupt Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

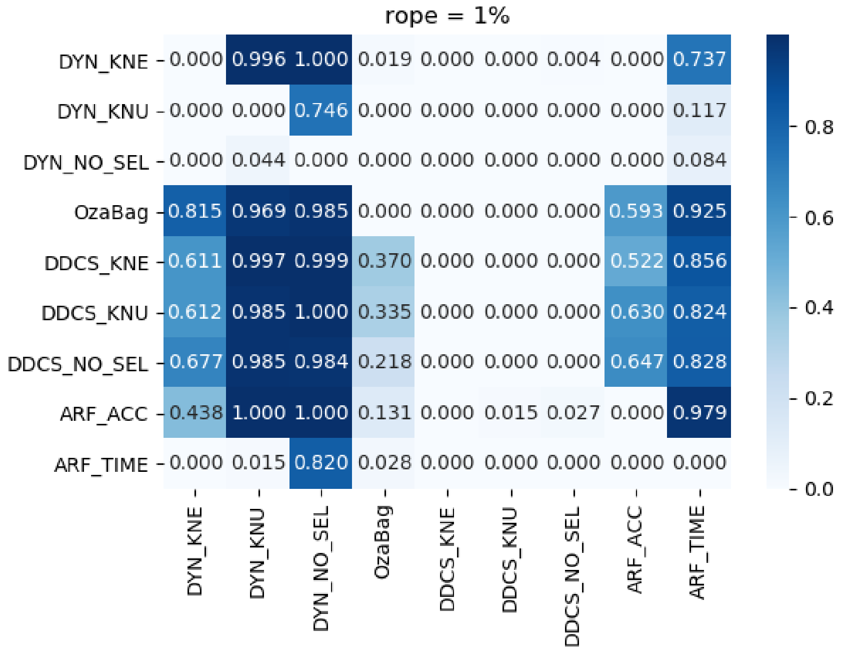


Figure 6.9: Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Streams with Abrupt Drift. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

Table 6.7: Results Comparing Processing Time and Memory Consumption with Streams with Abrupt Drift Against ARF_ACC

Method	Processing Time	Method	Memory Consumption
DYN_NO_SEL	0.560 (3.606s)	DYN_NO_SEL	0.006 (1.342e-07 ram-hour)
DDCS_NO_SEL	2.862 (18.435s)	DYN_KNU	0.053 (1.290e-06 ram-hour)
DYN_KNU	3.062 (19.723s)	DYN_KNE	0.057 (1.395e-06 ram-hour)
DYN_KNE	3.307 (21.301s)	DDCS_NO_SEL	0.059 (1.438e-06 ram-hour)
DDCS_KNU	5.892 (37.950s)	DDCS_KNU	0.552 (1.344e-05 ram-hour)
DDCS_KNE	6.658 (42.878s)	DDCS_KNE	0.665 (1.619e-05 ram-hour)
OzaBag	11.266 (72.559s)	OzaBag	1.605 (3.907e-05 ram-hour)
ARF_TIME	25.828 (166.345s)	ARF_TIME	10.764 (2.620e-04 ram-hour)
ARF_ACC	100.000 (644.059s)	ARF_ACC	100.000 (2.434e-03 ram-hour)

Table 6.7, following the pattern observed with the Gradual Drift, the relative difference in processing time of DDCS methods with DCS methods against ARF_ACC dropped to around 7%. The same thing happened regarding memory consumption, the same happened. With DDCS methods using around 0.7% of the used memory by ARF_ACC.

In conclusion, in an abrupt drift scenario, DDCS methods presented consistent results for being a satisfactory option for dealing with this type of context. It showed an extremely low probability of being outperformed by other methods, while keeping the processing time and memory consumption to a minimum. Another DCS method, DYNSE, was also largely outperformed by DDCS.

Table 6.8: Results Comparing DDCS with Real World Datasets

	DYN_KNE	DYN_KNU	DYN_NO_SEL	OzaBag	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
elec	76.689	76.912	78.714	76.249	78.113	79.091	**79.351**	78.905	61.707
nomao	93.341	91.853	87.532	92.371	**93.735**	93.416	93.206	93.451	91.001
spam_corpus	75.796	75.688	61.963	**75.997**	75.298	75.360	74.934	75.908	68.259

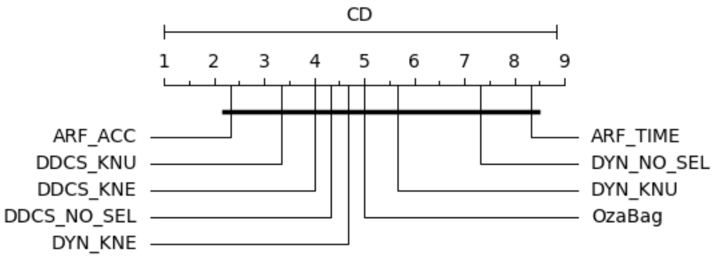


Figure 6.10: Critical Distance Comparing DDCS with Real World Datasets

6.3.4 Real World

The winner methods in real world datasets were Ozabag, DDCS with KNORA-E and without any selection method. This was the only round of experiments that ARF_ACC was not the winner in any case.

Looking at the critical distance plot in Figure 6.10, as in the results in the streams without any drift, no method was marked as significantly different from each other. ARF_ACC was placed in the first position, followed by all DDCS variants. In this datasets, ARF_TIME was the worst positioned method.

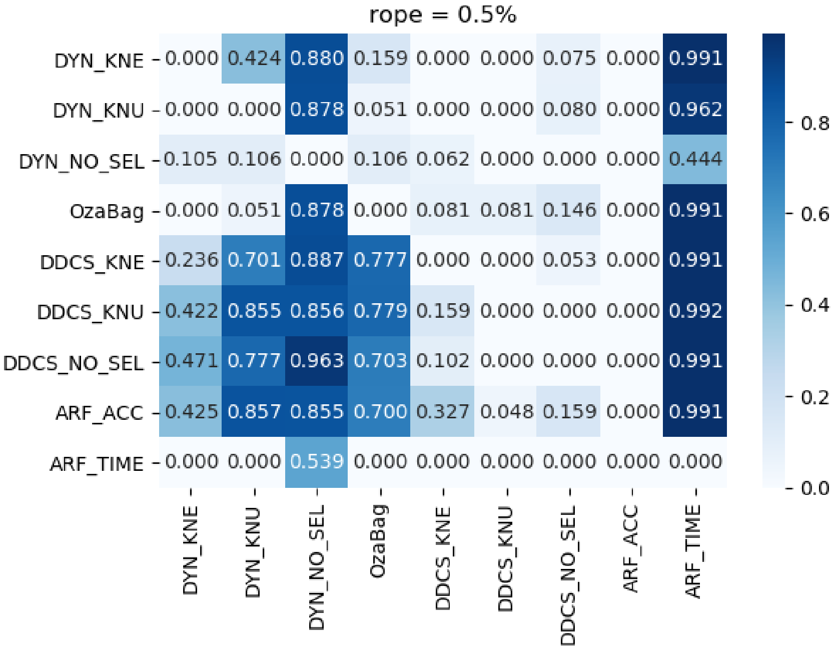
Regarding the Bayesian analyisis for *rope* = 0.5%, in Figure 6.11, ARF_ACC presented the horizontal line with the highest values. It also presented the vertical line with all the values equal to zero, this means ARF_ACC has the biggest probability of being better than the other methods, however, the majority of the probability against DDCS methods lied in the *rope*.

For *rope* = 1%, in Figure 6.12, ARF_ACC again presented the highest values in its horizontal line, however, the values against DDCS variants was zero.

Regarding processing time and memory consumption, in Table 6.9, this round of

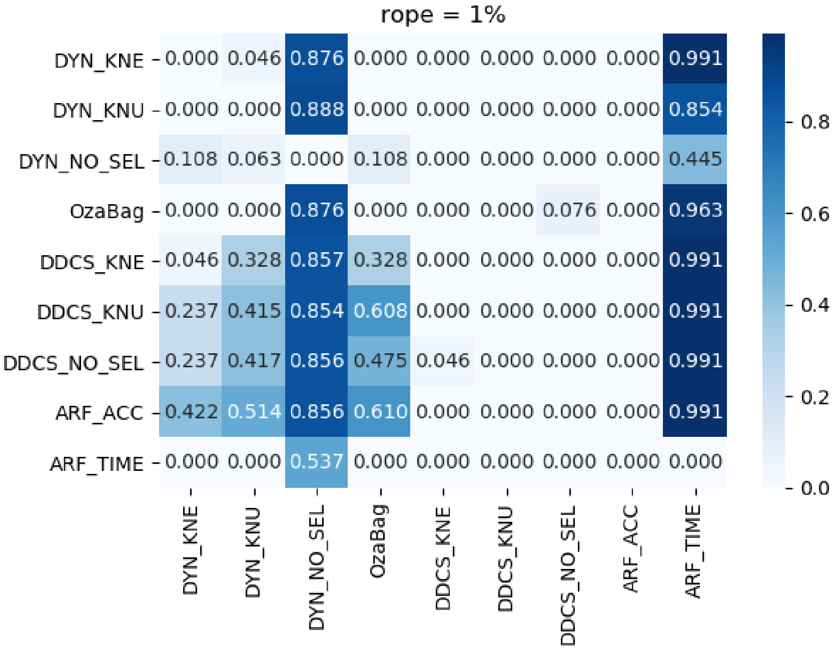
Table 6.9: Results Comparing Processing Time and Memory Consumption with Streams for Real World Datasets Against OzaBag

Method	Processing Time (s)	Method	Memory Consumption (Ram-hours)
DYN_NO_SEL	10.341% (5.941)	DDCS_NO_SEL	1.687% (2.458e-06)
DDCS_NO_SEL	14.894% (8.558)	DYN_NO_SEL	2.191% (3.192e-06)
ARF_TIME	28.941% (16.628)	ARF_TIME	7.451% (1.086e-05)
DYN_KNU	57.879% (33.255)	DYN_KNORAU	72.049% (1.050e-04)
DYN_KNE	58.209% (33.444)	DYN_KNORAE	72.618% (1.058e-04)
DDCS_KNU	69.378% (39.862)	DDCS_KNORAE	83.506% (1.217e-04)
DDCS_KNE	69.599% (39.989)	DDCS_KNORAU	83.547% (1.217e-04)
ARF_ACC	96.331% (55.347)	ARF_ACC	87.862% (1.280e-04)
OzaBag	100.000% (57.455)	OzaBag	100.000% (1.457e-04)



	DYN_KNE	DYN_KNU	DYN_NO_SEL	OzaBag	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
DYN_KNE	0.000	0.424	0.880	0.159	0.000	0.000	0.075	0.000	0.991
DYN_KNU	0.000	0.000	0.878	0.051	0.000	0.000	0.080	0.000	0.962
DYN_NO_SEL	0.105	0.106	0.000	0.106	0.062	0.000	0.000	0.000	0.444
OzaBag	0.000	0.051	0.878	0.000	0.081	0.081	0.146	0.000	0.991
DDCS_KNE	0.236	0.701	0.887	0.777	0.000	0.000	0.053	0.000	0.991
DDCS_KNU	0.422	0.855	0.856	0.779	0.159	0.000	0.000	0.000	0.992
DDCS_NO_SEL	0.471	0.777	0.963	0.703	0.102	0.000	0.000	0.000	0.991
ARF_ACC	0.425	0.857	0.855	0.700	0.327	0.048	0.159	0.000	0.991
ARF_TIME	0.000	0.000	0.539	0.000	0.000	0.000	0.000	0.000	0.000

Figure 6.11: Bayesian Analysis of All the Methods Pairwise with Rope = 0.5% for Real World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.



	DYN_KNE	DYN_KNU	DYN_NO_SEL	OzaBag	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
DYN_KNE	0.000	0.046	0.876	0.000	0.000	0.000	0.000	0.000	0.991
DYN_KNU	0.000	0.000	0.888	0.000	0.000	0.000	0.000	0.000	0.854
DYN_NO_SEL	0.108	0.063	0.000	0.108	0.000	0.000	0.000	0.000	0.445
OzaBag	0.000	0.000	0.876	0.000	0.000	0.000	0.076	0.000	0.963
DDCS_KNE	0.046	0.328	0.857	0.328	0.000	0.000	0.000	0.000	0.991
DDCS_KNU	0.237	0.415	0.854	0.608	0.000	0.000	0.000	0.000	0.991
DDCS_NO_SEL	0.237	0.417	0.856	0.475	0.046	0.000	0.000	0.000	0.991
ARF_ACC	0.422	0.514	0.856	0.610	0.000	0.000	0.000	0.000	0.991
ARF_TIME	0.000	0.000	0.537	0.000	0.000	0.000	0.000	0.000	0.000

Figure 6.12: Bayesian Analysis of All the Methods Pairwise with Rope = 1% for Real World Datasets. The results are given according to the probability of a being better than b, where a and b are the methods on the x and y axis, respectively.

experiments presented the most different results. The slowest method and the one that consumed most memory was Ozabag, and not ARF_ACC. The gain in speed and memory consumption that DDCS provided against Ozabag and ARF_ACC was significantly smaller. That is because as the number of dimensions increase, the cost of KD-Tree increases.

In conclusion, on the tested real-world datasets, DDCS loses some of its appeal. It did not present an advantage in accuracy over the state-of-art ARF_ACC, and when taking into account time processing and memory consumption, DDCS did not present the same advantage as it did in previous scenarios.

6.4 Conclusion

As important as is to analyze traditional statistical performance metrics, such as accuracy, processing time and memory consumption can not be left out of the equation when dealing with data streams. In this chapter we presented Double Dynamic Classifier Selection (DDCS) as a alternative to tackling the concept drift problem using dynamic classifier selection (DCS). DDCS has a lightweight structure, which takes advantage of KD-Tree for speeding up the selection process. We compared DDCS with Ozabag and Adaptive Random Forest (ARF), the latter is considered to be in the state-of-art for general purpose and concept drift classifiers for data streams. Besides, we also compared with Dynamic Selection Based Drift Handler (DYNSE), another DCS based method for tackling the concept drift problem.

We run experiments several times and gathered their accuraccy, processing time and memory consumption. The results show that for streams with both gradual and abrupt concept drift, DDCS variants outperformed other methods in most cases, while keeping a significantly low memory consumption and processing time, especially with relation to ARF, which is extremely performant, statistically speaking, but it currently state-of-art implementation has very high computational cost.

For streams with no drift and real-world datasets, DDCS is not as justifiable as it is in drift scenarios. For the no drift context, DDCS could not outperform ARF_ACC, but it still presented a satisfactory low use of processing time and memory. For real-world datasets, DDCS also did not outperform ARF_ACC, and did not run in significantly less time than it or consumed much less memory.

PART III

CONCLUSIONS

Chapter 7

Conclusions

Dynamic selection of classifiers (DCS) is a widely studied area of batch machine learning. In data stream mining, however, its behavior has not been deeply studied and analyzed by the scientific community. Also, most works in this area do not use online classifiers, but batch classifiers adapted to receive streams of data. This is an important gap to be filled since DCS methods were very successful in batch machine learning. Thus it has the potential to also contribute to data stream mining.

This work's analysis that allows a partial conclusion about the first hypothesis of the project: "The application of dynamic selection methods positively impact the performance of dynamic ensemble methods". Yes, there were cases where the dynamic selection positively impacted data stream mining ensembles. However, it was also detected that an environment with state-of-art ensembles and commonly used generators, is not an ideal environment for satisfactory performance with dynamic selection. Thus, further analysis is required to better determine the specific cases, if they exist, where the DCS methods are welcomed or not when applied to traditional dynamic ensembles.

As for the second hypothesis, "Dynamic selection methods tailored for data streams positively impact the classification accuracy in the data stream mining while keeping a low use of memory and processing time", we conclude that it proved to be true. DDCS not only provided better results, but used significantly less time and memory than the current state-of-art, the Adaptive Random Forest.

As future work, it is worthy investigating if traditional DCS methods present gains for dynamic ensembles in scenarios other than concept drift, such as imbalanced and noisy datasets. DDCS is also to be tested in such scenarios. New DCS methods focused on tackling these problems also need to be designed. Although there are DCS methods for imbalanced datasets, for instance, we still believe there is a gap to be filled, since most DCS methods do not support online classifiers.

A deeper analysis of the experimental protocol of DDCS should also be evaluated. There is a need to check if the use of a chunk-based evaluation did not interfere with the results of instance-based algorithms, such as ARF and OzaBag. For that, an adaptation of DDCS for it to work without the need of chunks should be done. The impact of the size of the chunks and the dimension of data in the quality of the prediction should also be assessed.

Bibliography

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, v. 5, n. 6, p. 914–925, Dec 1993. ISSN 2326-3865.

AHA, D. W.; KIBLER, D.; ALBERT, M. K. Instance-based learning algorithms. *Mach. Learn.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 6, n. 1, p. 37–66, jan. 1991. ISSN 0885-6125.

ALBUQUERQUE, R. A. S.; COSTA, A. F. J.; SANTOS, E. M. dos; SABOURIN, R.; GIUSTI, R. *A Decision-Based Dynamic Ensemble Selection Method for Concept Drift*. 2019.

ALMEIDA, P. R. L. D.; OLIVEIRA, L. S.; BRITTO, A. D. S.; SABOURIN, R. Handling concept drifts using dynamic selection of classifiers. In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2016. p. 989–995. ISSN 2375-0197.

ANTOSIK, B.; KURZYNSKI, M. New measures of classifier competence - heuristics and application to the design of multiple classifier systems. In: BURDUK, R.; KURZYŃSKI, M.; WOŹNIAK, M.; ŻOŁNIEREK, A. (Ed.). *Computer Recognition Systems 4*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 197–206. ISBN 978-3-642-20320-6.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F.; PFAHRINGER, B.; BIFET, A. On dynamic feature weighting for feature drifting data streams. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016*. [S.l.]: Springer, 2016. (Lecture Notes in Computer Science, v. 9852), p. 129–144.

BARROS, R. S.; CABRAL, D. R.; GONÇALVES, P. M.; SANTOS, S. G. Rddm: Reactive drift detection method. *Expert Systems with Applications*, v. 90, p. 344 – 355, 2017. ISSN 0957-4174.

BENAVOLI, A.; CORANI, G.; DEMŠAR, J.; ZAFFALON, M. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, v. 18, n. 77, p. 1–36, 2017.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 18, n. 9, p. 509–517, set. 1975. ISSN 0001-0782.

BIFET, A.; GAVALDÀ, R. Learning from time-changing data with adaptive windowing. In: . [S.l.: s.n.], 2007. v. 7.

BIFET, A.; GAVALDÀ, R.; HOLMES, G.; PFAHRINGER, B. *Machine Learning for Data Streams with Practical Examples in MOA*. [S.l.]: MIT Press, 2018.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KIRKBY, R.; GAVALDÀ, R. New ensemble methods for evolving data streams. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2009. (KDD '09), p. 139–148. ISBN 9781605584959.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KRANEN, P.; KREMER, H.; JANSEN, T.; SEIDL, T. Moa: Massive online analysis, a framework for stream classification and clustering. In: DIETHE, T.; CRISTIANINI, N.; SHAWE-TAYLOR, J. (Ed.). *Proceedings of the First Workshop on Applications of Pattern Analysis*. Cumberland Lodge, Windsor, UK: PMLR, 2010. (Proceedings of Machine Learning Research, v. 11), p. 44–50.

BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

BREIMAN, L. *Bias, variance, and arcing classifiers*. [S.l.], 1996.

BURKOV, A. *The Hundred-page Machine Learning Book*. [S.l.]: Andriy Burkov, 2019. ISBN 9781999579517.

CANDILLIER, L.; LEMAIRE, V. Active learning in the real-world design and analysis of the nomao challenge. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. [S.l.]: IEEE, 2013.

CANO, A.; KRAWCZYK, B. Kappa updated ensemble for drifting data stream mining. *Machine Learning*, 08 2019.

CAVALHEIRO, L. P.; BARDDAL, J. P.; BRITTO ALCEU, J. de S.; HEUTTE, L. scikit-dyn2sel – A Dynamic Selection Framework for Data Streams. *arXiv e-prints*, p. arXiv:2008.08920, ago. 2020.

CAVALIN, P.; SABOURIN, R.; SUEN, C. Dynamic selection approaches for multiple classifier systems. *Neural Computing and Applications*, v. 22, 03 2013.

CRUZ, R.; SABOURIN, R.; CAVALCANTI, G. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, v. 41, 05 2018.

CRUZ, R.; SABOURIN, R.; CAVALCANTI, G.; REN, T. I. Meta-des: A dynamic ensemble selection framework using meta-learning. *Pattern Recognition*, v. 48, 05 2015.

CRUZ, R. M. O.; HAFEMANN, L. G.; SABOURIN, R.; CAVALCANTI, G. D. C. DESlib: A Dynamic ensemble selection library in Python. *arXiv preprint arXiv:1802.04967*, 2018.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, v. 7, n. 1, p. 1–30, 2006.

DIETTERICH, T. G. Ensemble methods in machine learning. In: *Proceedings of the First International Workshop on Multiple Classifier Systems*. Berlin, Heidelberg: Springer-Verlag, 2000. (MCS '00), p. 1–15. ISBN 3540677046.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: . [S.l.]: ACM Press, 2000. p. 71–80.

DUONG, Q.-H.; RAMAMPIARO, H.; NØRVÅG, K. Applying temporal dependence to detect changes in streaming data. *Applied Intelligence*, Springer Science and Business Media LLC, v. 48, n. 12, p. 4805–4823, jul. 2018.

ENEMBRECK, F.; ÁVILA, B.; SCALABRIN, E.; BARTHÈS, J.-P. Learning drifting negotiations. *Applied Artificial Intelligence*, v. 21, p. 861–881, 10 2007.

FIX, E.; HODGES, J. L. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, JSTOR, v. 57, n. 3, p. 238, dez. 1989.

FOUCHÉ, E. *Blog*. 2017.

FRIAS-BLANCO, I.; CAMPO-AVILA, J. del; RAMOS-JIMENEZ, G.; MORALES-BUENO, R.; ORTIZ-DIAZ, A.; CABALLERO-MOTA, Y. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge* *and Data Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 27, n. 3, p. 810–823, mar. 2015.

FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, v. 29, p. 1189–1232, 2000.

GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. Learning with drift detection. In: BAZZAN, A. L. C.; LABIDI, S. (Ed.). *Advances in Artificial Intelligence – SBIA 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 286–295. ISBN 978-3-540-28645-5.

GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. On evaluating stream learning algorithms. *Machine Learning*, v. 90, p. 317–346, 10 2013.

GARCÍA, S.; ZHANG, Z.-L.; ALTALHI, A.; ALSHOMRANI, S.; HERRERA, F. Dynamic ensemble selection for multi-class imbalanced datasets. *Information Sciences*, v. 445-446, p. 22 – 37, 2018. ISSN 0020-0255.

GIACINTO, G.; ROLI, F. Methods for dynamic classifier selection. In: *Proceedings 10th International Conference on Image Analysis and Processing*. [S.l.: s.n.], 1999. p. 659–664. ISSN null.

GOMES, H. M.; BIFET, A.; READ, J.; BARDDAL, J. P.; ENEMBRECK, F.; PFHARINGER, B.; HOLMES, G.; ABDESSALEM, T. Adaptive random forests for evolving data stream classification. *Machine Learning*, v. 106, n. 9, p. 1469–1495, Oct 2017. ISSN 1573-0565.

GUANGA, A. *Machine Learning Series Day 7 (Decision Tree Classifier)*. [S.l.]: Becoming Human: Artificial Intelligence Magazine, 2019.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. The WEKA data mining software: an update. *SIGKDD Explorations*, v. 11, n. 1, p. 10–18, 2009.

HARRIES, M. B.; SAMMUT, C.; HORN, K. *Machine Learning*, Springer Science and Business Media LLC, v. 32, n. 2, p. 101–126, 1998.

HO, T. K. Random decision forests. In: *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*. Washington, DC, USA: IEEE Computer Society, 1995. (ICDAR '95), p. 278–. ISBN 0-8186-7128-9.

HO, T. K. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, USA, v. 20, n. 8, p. 832–844, ago. 1998. ISSN 0162-8828.

HUANG, Y. S.; SUEN, C. Y. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 17, n. 1, p. 90–94, Jan 1995. ISSN 1939-3539.

JAPKOWICZ, N. Assessment metrics for imbalanced learning. In: _____. *Imbalanced Learning*. [S.l.]: John Wiley & Sons, Ltd, 2013. cap. 8, p. 187–206. ISBN 9781118646106.

JOSE, M. B.-G.; CAMPO-ÁVILA, J. D.; FIDALGO, R.; BIFET, A.; GAVALDÀ, R.; MORALES-BUENO, R. *Early Drift Detection Method*. 2006.

JR, A.; SABOURIN, R.; OLIVEIRA, L. Soares de. Dynamic selection of classifiers—a comprehensive review. *Pattern Recognition*, v. 47, p. 3665–3680, 11 2014.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Dynamic feature space and incremental feature selection for the classiflcation of textual data streams. *Proceedings of ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams*, 01 2006.

KO, A. H. R.; SABOURIN, R.; BRITTO JR., A. S. From dynamic classifier selection to dynamic ensemble selection. *Pattern Recogn.*, Elsevier Science Inc., USA, v. 41, n. 5, p. 1718–1731, maio 2008. ISSN 0031-3203.

KOURTELLIS, N.; MORALES, G. D. F.; BIFET, A.; MURDOPO, A. VHT: Vertical hoeffding tree. In: *2016 IEEE International Conference on Big Data (Big Data)*. [S.l.]: IEEE, 2016.

KOYCHEV, I. Gradual forgetting for adaptation to concept drift. *ECAI*, 08 2004.

KREMPL, G.; ZLIOBAITE, I.; BRZEZIUNDEFINEDSKI, D.; HüLLERMEIER, E.; LAST, M.; LEMAIRE, V.; NOACK, T.; SHAKER, A.; SIEVI, S.; SPILIOPOULOU, M.; STEFANOWSKI, J. Open challenges for data stream mining research. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 16, n. 1, p. 1–10, set. 2014. ISSN 1931-0145.

KUNCHEVA, L. Clustering-and-selection model for classifier combination. In: *KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No.00TH8516)*. [S.l.]: IEEE, 2000.

LANEY, D. *3D Data Management: Controlling Data Volume, Velocity, and Variety.* [S.l.], 2001.

MANAPRAGADA, C.; WEBB, G. I.; SALEHI, M. Extremely fast decision tree. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* New York, NY, USA: Association for Computing Machinery, 2018. (KDD '18), p. 1953–1962. ISBN 9781450355520.

MARKHAM, K. *Simple guide to confusion matrix terminology.* [S.l.]: Data School, 2019.

MATUSZYK, P.; KREMPL, G.; SPILIOPOULOU, M. Correcting the usage of the hoeffding inequality in stream mining. In: *Advances in Intelligent Data Analysis XII.* [S.l.]: Springer Berlin Heidelberg, 2013. p. 298–309.

MINKU, L. L.; WHITE, A. P.; YAO, X. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, n. 5, p. 730–742, May 2010. ISSN 2326-3865.

MONTIEL, J.; READ, J.; BIFET, A.; ABDESSALEM, T. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, v. 19, n. 72, p. 1–5, 2018.

NÚÑEZ, M.; FIDALGO-MERINO, R.; MORALES-BUENO, R. Learning in environments with unknown dynamics: Towards more robust concept learners. *Journal of Machine Learning Research*, v. 8, p. 2595–2628, 11 2007.

OLIPHANT, T. E. *A guide to NumPy.* [S.l.]: Trelgol Publishing USA, 2006. v. 1.

OMOHUNDRO, S. Five balltree construction algorithms. In: . [S.l.: s.n.], 2009.

OPITZ, D.; MACLIN, R. Popular ensemble methods: An empirical study. *J. Artif. Int. Res.*, AI Access Foundation, El Segundo, CA, USA, v. 11, n. 1, p. 169–198, jul. 1999. ISSN 1076-9757.

OZA, N. C. Online bagging and boosting. In: *SMC.* [S.l.]: IEEE, 2005. p. 2340–2345. ISBN 0-7803-9298-1.

PAGE, E. S. Continuous inspection schemes. *Biometrika*, JSTOR, v. 41, n. 1/2, p. 100, jun. 1954.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.;

VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E.; LOUPPE, G. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, v. 12, 01 2012.

PELLET, J.-P.; DAME, A.; PARRIAUX, G. How beginner-friendly is a programming language? a short analysis based on java and python examples. In: . [S.l.: s.n.], 2019.

PESARANGHADER, A.; VIKTOR, H. L.; PAQUET, E. Mcdiarmid drift detection methods for evolving data streams. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–9.

PFAHRINGER, B.; HOLMES, G.; KIRKBY, R. New options for hoeffding trees. In: ORGUN, M. A.; THORNTON, J. (Ed.). *AI 2007: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 90–99. ISBN 978-3-540-76928-6.

PINAGE, F.; SANTOS, E.; GAMA, J. A drift detection method based on dynamic classifier selection. *Data Mining and Knowledge Discovery*, 10 2019.

POWERS, D. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. *Mach. Learn. Technol.*, v. 2, 01 2008.

QUINLAN, J. R. Induction of decision trees. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1, p. 81–106, mar. 1986. ISSN 0885-6125.

RODRIGUES, P.; GAMA, J.; PEDROSO, J. Hierarchical clustering of time-series data streams. *IEEE Transactions on Knowledge and Data Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 20, n. 5, p. 615–627, maio 2008.

SABOURIN, M.; MITICHE, A.; THOMAS, D.; NAGY, G. Classifier combination for hand-printed digit recognition. In: *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR '93)*. [S.l.: s.n.], 1993. p. 163–166. ISSN null.

SAMMUT, C.; WEBB, G. I. (Ed.). *Encyclopedia of Machine Learning*. [S.l.]: Springer US, 2010.

SANTOS, E.; SABOURIN, R.; MAUPIN, P. A dynamic overproduce-and-choose strategy for the selection of classifier ensembles. *Pattern Recognition*, v. 41, p. 2993–3009, 10 2008.

SCHAPIRE, R. E. A brief introduction to boosting. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (IJCAI'99), p. 1401–1406.

SCHLIMMER, J. C.; GRANGER JR., R. H. Incremental learning from noisy data. *Mach. Learn.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 1, n. 3, p. 317–354, mar. 1986. ISSN 0885-6125.

SOARES, R. G. F.; SANTANA, A.; CANUTO, A. M. P.; SOUTO, M. C. P. de. Using accuracy and diversity to select classifiers to build ensembles. In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. [S.l.: s.n.], 2006. p. 1310–1316. ISSN 2161-4407.

STREET, N.; KIM, Y. A streaming ensemble algorithm (sea) for large-scale classification. In: . [S.l.: s.n.], 2001. p. 377–382.

TSYMBAL, A. The problem of concept drift: Definitions and related work. 05 2004.

TSYMBAL, A.; PECHENIZKIY, M.; CUNNINGHAM, P.; PUURONEN, S. Dynamic integration of classifiers for handling concept drift. *Inf. Fusion*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 9, n. 1, p. 56–68, jan. 2008. ISSN 1566-2535.

VALENTINI, G. An experimental bias-variance analysis of svm ensembles based on resampling techniques. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, v. 35, p. 1252–71, 01 2006.

VALIANT, L. G. A theory of the learnable. *Communications of the ACM*, v. 27, p. 1134–1142, 1984.

WANG, B.; PINEAU, J. Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 28, n. 12, p. 3353–3366, dez. 2016.

WOLOSZYNSKI, T.; KURZYNSKI, M. A probabilistic model of classifier competence for dynamic ensemble selection. *Pattern Recognition*, v. 44, n. 10, p. 2656 – 2668, 2011. ISSN 0031-3203. Semi-Supervised Learning for Visual Content Analysis and Understanding.

WOLOSZYNSKI, T.; KURZYNSKI, M.; PODSIADLO, P.; STACHOWIAK, G. W. A measure of competence based on random classification for dynamic ensemble selection. *Information Fusion*, v. 13, n. 3, p. 207 – 213, 2012. ISSN 1566-2535.

WOODS, K.; JR, W. P. K.; BOWYER, K. Combination of multiple classifiers using local accuracy estimates. In: *Proceedings of the 1996 Conference on Computer Vision* *and Pattern Recognition (CVPR '96)*. USA: IEEE Computer Society, 1996. (CVPR '96), p. 391. ISBN 0818672587.

ZYBLEWSKI, P.; KSIENIEWICZ, P.; WOŹNIAK, M. Classifier selection for highly imbalanced data streams with minority driven ensemble. In: RUTKOWSKI, L.; SCHERER, R.; KORYTKOWSKI, M.; PEDRYCZ, W.; TADEUSIEWICZ, R.; ZURADA, J. M. (Ed.). *Artificial Intelligence and Soft Computing*. Cham: Springer International Publishing, 2019. p. 626–635. ISBN 978-3-030-20912-4.

ZYBLEWSKI, P.; SABOURIN, R.; WOŹNIAK, M. Data preprocessing and dynamic ensemble selection for imbalanced data stream classification. In: *Machine Learning and Knowledge Discovery in Databases*. [S.l.]: Springer, 2020. p. 367–379.

ZYBLEWSKI, P.; SABOURIN, R.; WOŹNIAK, M. Preprocessed dynamic classifier ensemble selection for highly imbalanced drifted data streams. *Information Fusion*, v. 66, p. 138 – 154, 2021. ISSN 1566-2535.

Appendix A

Results DCS Techniques in Data Stream Mining

This appendix presents the complete set of results of the experiments described in Section 5.2. Table A.1 organizes them by making each line a combination of an ensemble with a stream generator, and each column a selection method.

Table A.1: Results for all experiments reported in Section 5.3

	No Sel.	LCA	OLA	KNORA-U	KNORA-E	ORACLE
ARF - P2	98.53%	94.29%	95.48%	98.55%	98.41%	100.00%
ARF - RTG	98.72%	91.28%	93.52%	98.51%	98.59%	100.00%
ARF - Agrawal - Drift	73.29%	62.07%	66.66%	71.92%	70.10%	100.00%
ARF - SEA - Drift	86.53%	84.86%	82.34%	86.62%	83.82%	98.78%
ARF - SEA	87.41%	85.88%	83.43%	87.49%	84.99%	98.75%
ARF - Agrawal	93.35%	79.89%	83.83%	92.84%	91.84%	100.00%
ARF - Asset - Drift	93.82%	83.62%	92.04%	93.53%	93.15%	97.85%
ARF - RTG - Drift	85.08%	79.05%	79.82%	85.45%	84.26%	99.99%
ARF - Asset	93.65%	81.78%	91.44%	93.35%	92.73%	97.85%
OzaBAd. - Percep. - P2	54.73%	54.57%	86.95%	82.88%	87.10%	96.60%
OzaBAd. - Percep. - RTG	86.34%	85.23%	86.68%	86.42%	86.86%	94.22%
OzaBAd. - Percep. - Agrawal - Drift	49.97%	49.97%	49.97%	49.97%	49.97%	49.97%
OzaBAd. - Percep. - SEA - Drift	76.93%	76.15%	83.54%	81.52%	83.77%	99.34%
OzaBAd. - Percep. - SEA	78.34%	74.08%	83.92%	82.41%	84.31%	99.39%
OzaBAd. - Percep. - Agrawal	49.97%	49.97%	49.97%	49.97%	49.97%	49.97%
OzaBAd. - Percep. - Asset - Drift	51.58%	64.89%	87.85%	86.87%	87.95%	100.00%
OzaBAd. - Percep. - RTG - Drift	68.60%	67.32%	73.62%	70.32%	73.78%	96.26%
OzaBAd. - Percep. - Asset	52.86%	61.39%	86.22%	83.43%	86.34%	100.00%
OzaB. - NB - P2	56.96%	57.72%	62.00%	68.80%	61.96%	86.08%
OzaB. - NB - RTG	85.62%	85.27%	85.70%	85.62%	85.69%	94.09%
OzaB. - NB - Agrawal - Drift	71.26%	70.42%	71.43%	71.31%	71.49%	87.44%
OzaB. - NB - SEA - Drift	84.14%	83.83%	84.28%	84.15%	84.35%	93.07%
OzaB. - NB - SEA	82.70%	82.42%	82.83%	82.72%	82.95%	92.04%
OzaB. - NB - Agrawal	94.14%	93.33%	94.04%	94.18%	94.22%	97.94%
OzaB. - NB - Asset - Drift	85.59%	85.66%	87.51%	86.30%	87.60%	94.02%
OzaB. - NB - RTG - Drift	63.47%	62.93%	63.65%	64.14%	63.69%	86.43%
OzaB. - NB - Asset	85.57%	85.36%	87.22%	86.39%	87.33%	94.87%
OzaB. - HT - P2	84.87%	79.50%	92.72%	88.06%	93.46%	99.54%
OzaB. - HT - RTG	94.68%	93.50%	95.08%	94.20%	95.45%	99.72%

OzaB. - HT - Agrawal - Drift	77.67%	76.83%	76.09%	74.19%	77.93%	97.51%
OzaB. - HT - SEA - Drift	85.38%	84.66%	85.32%	85.41%	85.47%	97.29%
OzaB. - HT - SEA	86.10%	85.56%	85.92%	86.16%	86.12%	97.64%
OzaB. - HT - Agrawal	93.53%	92.16%	93.70%	93.70%	93.85%	100.00%
OzaB. - HT - Asset - Drift	93.92%	92.72%	93.79%	93.92%	93.92%	96.84%
OzaB. - HT - RTG - Drift	79.81%	76.56%	80.40%	80.02%	81.47%	99.76%
OzaB. - HT - Asset	94.15%	93.08%	94.02%	94.16%	94.08%	97.03%
OzaBAd. - NB - P2	57.74%	59.00%	75.23%	71.39%	75.21%	91.72%
OzaBAd. - NB - RTG	85.62%	85.27%	85.70%	85.62%	85.69%	94.09%
OzaBAd. - NB - Agrawal - Drift	71.72%	69.66%	73.79%	71.46%	74.29%	95.97%
OzaBAd. - NB - SEA - Drift	84.14%	83.83%	84.28%	84.15%	84.35%	93.07%
OzaBAd. - NB - SEA	82.70%	82.42%	82.83%	82.72%	82.95%	92.04%
OzaBAd. - NB - Agrawal	94.14%	93.33%	94.04%	94.18%	94.22%	97.94%
OzaBAd. - NB - Asset - Drift	86.42%	87.25%	91.78%	87.82%	91.86%	96.39%
OzaBAd. - NB - RTG - Drift	65.08%	65.73%	69.24%	67.44%	69.25%	90.62%
OzaBAd. - NB - Asset	85.62%	84.57%	88.80%	86.39%	88.96%	95.43%
OzaBAd. - HT - P2	85.17%	79.49%	92.96%	88.54%	93.80%	99.68%
OzaBAd. - HT - RTG	94.78%	93.50%	95.08%	94.20%	95.45%	99.92%
OzaBAd. - HT - Agrawal - Drift	75.81%	70.58%	76.36%	76.28%	77.07%	99.71%
OzaBAd. - HT - SEA - Drift	85.39%	84.55%	85.29%	85.42%	85.44%	97.35%
OzaBAd. - HT - SEA	86.10%	85.56%	85.92%	86.16%	86.12%	97.64%
OzaBAd. - HT - Agrawal	93.55%	92.12%	93.69%	94.08%	94.12%	100.00%
OzaBAd. - HT - Asset - Drift	93.92%	92.94%	94.15%	93.92%	94.27%	97.24%
OzaBAd. - HT - RTG - Drift	79.88%	77.20%	80.76%	80.07%	81.71%	99.79%
OzaBAd. - HT - Asset	94.15%	93.08%	94.02%	94.16%	94.08%	97.03%
KUE - HT - P2	94.40%	85.95%	94.00%	94.66%	96.05%	99.39%
KUE - HT - RTG	96.03%	86.53%	96.36%	96.26%	97.14%	99.39%
KUE - HT - Agrawal - Drift	79.15%	61.44%	76.05%	75.96%	77.71%	99.39%
KUE - HT - SEA - Drift	85.57%	79.71%	81.01%	85.52%	82.08%	98.50%
KUE - HT - SEA	86.53%	77.73%	82.27%	86.21%	82.67%	98.48%
KUE - HT - Agrawal	94.31%	87.96%	92.62%	94.40%	93.48%	99.39%
KUE - HT - Asset - Drift	93.15%	61.87%	92.62%	93.28%	92.80%	99.39%
KUE - HT - RTG - Drift	81.76%	68.02%	79.58%	82.77%	81.65%	99.39%
KUE - HT - Asset	93.65%	73.02%	92.47%	93.64%	92.41%	99.39%
OzaB. - Percep. - P2	54.73%	54.57%	86.95%	82.88%	87.10%	96.60%
OzaB. - Percep. - RTG	86.34%	85.23%	86.68%	86.42%	86.86%	94.22%
OzaB. - Percep. - Agrawal - Drift	49.97%	49.97%	49.97%	49.97%	49.97%	49.97%
OzaB. - Percep. - SEA - Drift	77.56%	76.10%	83.58%	81.83%	83.82%	98.77%
OzaB. - Percep. - SEA	78.51%	74.35%	83.92%	82.33%	84.23%	98.90%
OzaB. - Percep. - Agrawal	49.97%	49.97%	49.97%	49.97%	49.97%	49.97%
OzaB. - Percep. - Asset - Drift	51.50%	51.67%	72.86%	68.94%	72.99%	100.00%
OzaB. - Percep. - RTG - Drift	68.59%	67.33%	73.44%	70.31%	73.59%	95.95%
OzaB. - Percep. - Asset	51.56%	51.93%	82.00%	72.67%	82.13%	100.00%
KUE - NB - P2	57.24%	57.04%	81.62%	74.58%	80.82%	87.71%
KUE - NB - RTG	85.18%	83.19%	85.56%	83.67%	85.56%	93.58%
KUE - NB - Agrawal - Drift	72.05%	52.76%	71.56%	52.27%	71.41%	91.93%
KUE - NB - SEA - Drift	83.75%	79.07%	83.59%	81.07%	83.30%	92.78%
KUE - NB - SEA	82.27%	76.95%	82.10%	79.53%	81.88%	91.71%
KUE - NB - Agrawal	93.60%	68.30%	93.45%	58.04%	93.27%	97.51%
KUE - NB - Asset - Drift	85.55%	80.90%	92.14%	89.85%	92.36%	97.58%
KUE - NB - RTG - Drift	64.46%	60.10%	70.59%	65.67%	71.07%	89.20%
KUE - NB - Asset	84.97%	81.70%	91.95%	90.26%	92.12%	97.30%

KUE - Percep. - P2	56.69%	54.34%	88.08%	83.82%	87.59%	97.43%
KUE - Percep. - RTG	86.13%	83.70%	86.66%	86.13%	86.82%	97.89%
KUE - Percep. - Agrawal - Drift	49.98%	49.97%	50.11%	49.97%	50.03%	51.13%
KUE - Percep. - SEA - Drift	77.90%	70.93%	82.92%	82.82%	83.16%	99.06%
KUE - Percep. - SEA	78.89%	70.57%	83.41%	83.18%	83.60%	99.07%
KUE - Percep. - Agrawal	49.87%	49.97%	49.97%	50.00%	50.36%	51.18%
KUE - Percep. - Asset - Drift	76.08%	70.57%	88.43%	88.17%	88.58%	99.39%
KUE - Percep. - RTG - Drift	68.00%	64.47%	74.83%	70.91%	75.22%	98.96%
KUE - Percep. - Asset	77.36%	76.97%	86.66%	85.74%	86.55%	99.39%