Pós-Graduação em Ciência da Computação

Laura María Palomino Mariño

Variants of the Fast Adaptive Stacking of Ensembles algorithm

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Laura María Palomino Mariño

Variants of the Fast Adaptive Stacking of Ensembles algorithm

A M.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Concentration Area: Computational Intelligence
Advisor: Dr. Germano Crispim Vasconcelos

Recife
2019

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2105

M339v Mariño, Laura María Palomino
Variants of the Fast Adaptive Stacking of Ensembles algorithm/ Laura María Palomino Mariño – 2019.
119 f.: il., fig., tab.

Orientador: Germano Crispim Vasconcelos
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação. Recife, 2019.
Inclui referências e apêndices.

1. Inteligência artificial. 2. Mudanças de conceito. 3. Fluxo de dados. 4. Métodos de Combinação de Classificadores. I. Vasconcelos, Germano Crispim (orientador). II. Título.

006.31 CDD (22. ed.) UFPE-MEI 2019-163

Laura María Palomino Mariño

"Variants of the Fast Adaptive Stacking of Ensembles algorithm"

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 26/07/2019.

BANCA EXAMINADORA

Prof. Dr. Cleber Zanchettin
Centro de Informática / UFPE

Prof. Dr. Antonio Cezar de Castro Lima
Departamento de Engenharia Elétrica / UFBA

Prof. Dr. Germano Crispim Vasconcelos
Centro de Informática / UFPE
(Orientador)

To my mother, thank you for your infinite dedication.

ACKNOWLEDGEMENTS

Firstly, I would like to thank professor Germano Crispin Vasconcelos for the professional opportunity and the support in the present research. In the same way, to José Luis Martínez and Isvanis Frías for the help received to start this master's degree.

To Bruno Maciel, Silas Garrido, Juan Isidro and especially to professor Roberto Souto Maior from Concept Drift team, for their important assistance at the beginning of this work.

With all my heart to all people who sustained me, especially to my parents and my life partner Agustín Alejandro Ortíz Díaz, who is also an excellent professional, professor, and researcher; for his assistance, the interesting discussions, the valuable ideas, and advice.

For friendship and accompaniment at all times since I came here to Yarima, José, Juan, and Sonia, in general to all my friends, although not all have been mentioned, for the invaluable company and teachings. I also thank you.

To the professors and employees of the postgraduate program in Computer Science of the CIN-UFPE. In addition to the CAPES that granted a scholarship for master's studies.

ABSTRACT

The treatment of large data streams in the presence of concept drifts is one of the main challenges in the fields of machine learning and data mining. This dissertation presents two families of ensemble algorithms designed to quickly adapt to concept drifts, both abrupt and gradual. The families Fast Stacking of Ensembles boosting the Old (FASEO) and Fast Stacking of Ensembles boosting the Best (FASEB) are adaptations of the Fast Adaptive Stacking of Ensembles (FASE) algorithm, designed to improve run-time and memory requirements, without presenting a significant decrease in terms of accuracy when compared to the original FASE. In order to achieve a more efficient model, adjustments were made in the update strategy and voting procedure of the ensemble. To evaluate the proposals against state of the art methods, Naïve Bayes (NB) and Hoeffding Tree (HT) are used, as learners, to compare the performance of the algorithms on artificial and real-world data-sets. An extensive experimental investigation with a total of 70 experiments and application of Friedman and Nemenyi statistical tests showed the families FASEO and FASEB are more efficient than FASE with respect to execution time and memory in many scenarios, often also achieving better accuracy results.

Keywords: Concept Drift. Data Stream. Ensemble Methods. Non-stationary Data.

RESUMO

O tratamento de grandes fluxos de dados na presença de mudanças de conceito é um dos principais desafios nas áreas de aprendizado de máquina e mineração de dados. Essa dissertação apresenta duas famílias de algoritmos de combinação de classificadores projetados para se adaptar rapidamente a mudanças de conceitos, tanto abruptos quanto graduais. As famílias Fast Stacking of Ensembles boosting the Old (FASEO) e Fast Stacking of Ensembles boosting the Best (FASEB) são adaptações do algoritmo Fast Stacking of Ensembles (FASE), projetadas para melhorar seu tempo de execução e consumo de memória, sem apresentar uma diminuição significativa de desempenho em termos de acurácia em comparação com o algoritmo original. Para obter um modelo mais eficiente, foram feitos ajustes na estratégia de atualização e no processo de votação de FASE. Para avaliar as propostas em relação ao estado da arte, usamos o Naive Bayes (NB) e o Hoeffding Tree (HT) como classificadores base para comparar o desempenho dos algoritmos em conjuntos de dados reais e sintéticos. Um avaliação experimental extensa, com um total 70 experimentos e emprego dos testes estatísticos de Friedman e Nemenyi, mostraram que as famílias FASEO e FASEB são mais eficientes que FASE com respeito a tempo de execução e memória em vários cenários, as vezes alcançando também melhores resultados na acurácia dos algoritmos.

Palavras-chaves: Mudanças de conceito. Fluxo de dados. Métodos de Combinação de Classificadores. Dados não-estacionários.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ADOB	Adaptable Diversity-based Online Boosting
AI	Artificial Intelligence
BOLE	Boosting-like Online Learning Ensemble
DDD	Diversity for Dealing with Drifts
DDM	Drift Detection Method
DM	Data Mining
DWM	Dynamic Weighted Majority
EDDM	Early Drift Detection Method
FASE	Fast Adaptive Stacking of Ensembles
FASEB	Fast Stacking of Ensembles boosting the Best
FASEB$_{wv1}$	Fast Ensemble boosting the Best with Combined Weighting Voting$_1$
FASEB$_{wv2}$	Fast Ensemble boosting the Best with Combined Weighting Voting$_2$
FASEB$_{wv3}$	Fast Ensemble boosting the Best with Combined Weighting Voting$_3$
FASEO	Fast Stacking of Ensembles boosting the Old
FASEO$_{wv1}$	Fast Ensemble boosting the Old with Combined Weighting Voting$_1$
FASEO$_{wv2}$	Fast Ensemble boosting the Old with Combined Weighting Voting$_2$
FASEO$_{wv3}$	Fast Ensemble boosting the Old with Combined Weighting Voting$_3$
HDDM$_A$	Hoeffding-based Drift Detection Method$_{A-test}$
HDDM$_W$	Hoeffding-based Drift Detection Method$_{W-test}$
HT	Hoeffding Tree
ML	Machine Learning
MOA	Massive Online Analysis
NB	Naïve Bayes
OzaBag$_D$	Oza and Russell's Online Bagging with Detector
VFDT	Very Fast Decision Tree classifier
WMA	Weighted Majority Algorithm

LIST OF SYMBOLS

$\mathbf{x}$	d-dimensional vector $\mathbf{x} = (x_1, \ldots, x_d)$.
x_j	The j-th attribute value of vector $\mathbf{x}$.
y	Discrete value, category or class label taken from a finite set Y of class labels.
$\mathbf{z}$	The description of an instance: $\mathbf{z} = (\mathbf{x}, y)$.
S	Data Streams $S = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_i, \ldots\}$, where $\mathbf{z}_i = (\mathbf{x}_i, y_i)$.
x_{ij}	The attribute value of $\mathbf{x}_i$.
D	It is a distribution which represents the joint probability $P(X, y)$.
$P(y)$	Represent the prior class probability.
$P(X\|y)$	Represent the class-conditional probability.
$P(y\|X)$	Represent the posterior probability for a class y.
$P(X)$	It is the evidence factor which is considered as a scale factor.
k	Number of classifiers in the ensemble.
n	Number of samples or instances.
c	Number of classes.
$H(\mathbf{x})$	Represent the final output of the classifier ensemble.
$h_j(\mathbf{x})$	Prediction of the classifier h_j.
h_j	Represent the j_{th} classifier in the model.
$h^{'}$	It is the second-level classifier in the Stacking.
M	Correspond to the meta-instance and can be represented as $(\mathbf{x}^{'}, y)$ or $(\hat{y}_1, \ldots, \hat{y}_j, \ldots, \hat{y}_k; y)$.
$\hat{y}_j$	Correspond to predicted class label provided by the classifier h_j, also can be represented by $h_j(\mathbf{x})$.
$LC(\mathbf{x})$	Represent the largest sum of probabilities to determine the final output of the classifier ensemble using weighted voting.
w_j	Represent the weight values for the j_{th} classifier.

CONTENTS

1 INTRODUCTION

In recent years, data generated by different sources such as cell phones, sensors, networks and satellites has increased significantly. There are numerous examples of technologies used in many areas that generate large amounts of data very quickly. Part of these data can be viewed as a sequence of examples that arrive at high rates and can often be read only once using a small amount of processing time and memory (GAMA; GABER, 2007). In the literature, such data are known as data-streams.

According to (AGGARWAL, 2014), in the streaming scenario, two primary issues have to be dealt with in the construction of training models: (i) One-pass Constraint, which regards the need for all processing algorithms to perform their computations in one single pass over the data, since volume in data-streams is very large and, (ii) Concept Drift, which refers to the fact that the typical generating process of the data-streams may change over time. In this case, learning from data-streams is directly related to Concept Drift because it is an inherent feature of the data arrival process itself over time (VERDECIA-CABRERA; BLANCO; CARVALHO, 2018).

The presence of Concept Drift affects the performance of the classification algorithms, because models become stale over time. Therefore, it is crucial to adjust the model in an incremental way in order for the algorithm to achieve high accuracy over current unknown instances (AGGARWAL, 2014).

According to (FRÍAS-BLANCO, 2014), Concept Drift involves two different concepts: initial concept (P_I) and final concept (P_F). Attending to the time it takes to change from P_I to P_F (t_{ch}), this change can be abrupt (sudden) ($t_{ch} \sim 0$) or gradual ($t_{ch} > 0$). Depending on the taxonomy of the involved aspects in the distribution change, different types of Concept Drift can be analyzed: virtual Concept Drift (the distribution of instances change but the underlying concept does not), real Concept Drift (there exist a change in the class boundary), recurrence of the concepts (when previously active concept reappears after some time) (ORTIZ-DIAZ et al., 2015), and some other types which will be further discussed in chapter 3 .

Giving the difficulty of processing data-streams with concept drifts with traditional data mining algorithms, irrespective of their nature, there is a growing interest in the efficient processing of data-streams with the least amount of resources possible.

However, most of the studies conducted so far have focussed on accuracy, although it is well known that following an *error-rate-only* approach is not beneficial in all situations because concept drifts occur in many scenarios where other parameters should also be taken into account.

For example, in an emergency response context, the time required to present a model to users may be the most important criteria, i.e., users may be willing to accept less

accuracy for speed and partial information. Further, regarding the area of mobile data mining, which has application in defense and environmental impact assessment. Here, the memory resources may be limited, due to connection issues, and thus reducing the memory footprint is also of importance (PESARANGHADER; VIKTOR; PAQUET, 2018).

Based on previous works (BARROS; SANTOS, 2019) that analyzed the performance of several algorithms, it was observed that Fast Adaptive Stacking of Ensembles (FASE) (FRIAS-BLANCO et al., 2016) presents good accuracy results, but its memory and run-times indicate there is room for improvement. FASE is an ensemble method which uses a meta-learner to combine the predictions of the classifiers. It also maintains a set of adaptive classifiers in order to deal with changes of concepts in the data-stream. The method can be regarded as a three-level ensemble (FRIAS-BLANCO et al., 2016), one of the reasons to impact its performance.

Since in many scenarios, the processing of data-streams demands constraints on the amount of memory and run-time used. This dissertation considers a few different strategies to improve the original FASE algorithm with respect to these other metrics, yet trying to preserve, at the same time, its accuracy rate.

The following question is raised and investigated in this *research*: how to handle more efficiently both abrupt and gradual changes to perform the learning task in the context of data streaming.

In this research "efficiency" has a particular meaning: it is a suitable balance among accuracy, memory and run-time highlighting that "suitable" is associated with the degree to which these aspects have relevance in a given context. That is, producing the desired result without excessive effort or expense preventing the wasteful use of these particular resources (STEVENSON, 2010).

To accomplish that, this research proposes and experimentally investigates two main modifications in FASE, regarding (i) the update strategy and (ii) the voting procedure of the ensemble.

The update strategy FASE employs is based on the use of a set of adaptive learners. Each adaptive learner also uses Hoeffding-based Drift Detection Method$_{A-test}$ (HDDM$_A$) (FRÍAS-BLANCO et al., 2015) to monitor the error rate of the classifier induced for stable concepts. When the warning level is raised, the adaptive learner trains an alternative classifier that replaces the main classifier if the warning level is followed by a drift signal. Adaptive learners can thus have at most two learners and their predictions are combined by weighted voting (FRIAS-BLANCO et al., 2016). In addition, FASE's voting procedure is based on a meta-classifier that combines the predictions of the classifiers in the ensemble (FRIAS-BLANCO et al., 2016).

This work proposes two families of methods obtained from FASE: Fast Stacking of Ensembles boosting the Best (FASEB) and Fast Stacking of Ensembles boosting the Old (FASEO). They partially adopt FASE's model but make some changes. For example, the

concept of adaptive classifier is eliminated, because the design of adaptive learners allows them to have two classifiers (the main and the alternative classifier), and this structure is part of both the ensemble of classifiers and the meta-classifier. Instead, it was introduced an alternative ensemble where a new classifier is activated and begins to be trained when the warning level is reached by a classifier in the main ensemble. Once a drift is detected in the main ensemble, the corresponding classifier is replaced by the classifier with greater accuracy (FASEB methods) or by the classifier with more time in the alternative ensemble (FASEO methods).

The second modification regards the voting strategy of the ensemble. The first two variants with the same name of FASEB and FASEO families, respectively, adopt a similar concept to the employed in FASE, maintaining a meta-classifier, but the design of the classifiers in our model is simpler than an adaptive classifier, only formed by learners that include a drift detector, by default HDDM$_A$, in order to monitor its error rate. This mechanism allows us to determine when the learner triggered a warning and if it reaches a drift.

The following variants use Weighted Majority Voting to combine the classifiers' predictions instead a Meta-Classifier:

- Fast Ensemble boosting the Best with Combined Weighting Voting$_1$ (FASEB$_{wv1}$)
- Fast Ensemble boosting the Best with Combined Weighting Voting$_2$ (FASEB$_{wv2}$)
- Fast Ensemble boosting the Best with Combined Weighting Voting$_3$ (FASEB$_{wv3}$)
- Fast Ensemble boosting the Old with Combined Weighting Voting$_1$ (FASEO$_{wv1}$)
- Fast Ensemble boosting the Old with Combined Weighting Voting$_2$ (FASEO$_{wv2}$)
- Fast Ensemble boosting the Old with Combined Weighting Voting$_3$ (FASEO$_{wv3}$)

1.1 OBJETIVE

The objective of this research is to develop ensemble systems based on FASE improving its run-time and consumed memory in order to handle more efficiently both abrupt and gradual changes during the learning task from data-streams.

1.2 HYPOTHESES

Ensemble algorithms based on FASE that improving its run-time and consumed memory can handle more efficiently both abrupt and gradual changes during the learning task from data-streams.

1.3 METODOLOGY

The methodologies and research techniques were used at a theoretical and empirical level to achieve the objective. For this purpose, an analysis of the concepts related to the problem of Concept Drift in data-streams and a survey of the state of the art research in the area were conducted. This allowed to determine the strengths and weaknesses of existing approaches, showing new research needs.

In addition, a detailed study was carried out of the procedures to be followed to address these weaknesses in order to improve the selected existing methods in some aspect. The proposed methods are experimentally evaluated using appropriate techniques on an extensive well-known data-set and compared with the benchmarks in different scenarios.

To achieve the proposed objective, the following *research tasks* were developed:

- Theoretical study of the state of art focused on classification ensemble algorithms designed to work on non-stationary data-stream;
- Characterization of the benchmark classification ensemble algorithms designed to work on non-stationary data-streams, considering their main strengths and weaknesses;
- Analysis of some of the most representative data-sets (synthetic and real) used in the evaluation and comparison of classification ensemble algorithms, designed to work on non-stationary data-stream;
- Selection and proposal of new ideas, starting from the FASE method, to obtain new ensemble algorithms in order to more efficiently handle abrupt and gradual concept changes on large data-streams;
- Design and implementation of the new methods on the framework Massive Online Analysis (MOA) based on the proposed ideas;
- Employment of evaluation methodologies to compare the proposed methods with the benchmark in the context of classification ensembles designed to work in non-stationary data-streams.

This research work is structured as follows:

1.4 STRUCTURE

- *Chapter 2* presents several issues related to the research area that support the present work such as data-stream concept, classification tasks, and the classifiers ensemble model. This chapter also discuss the several measures and methodologies used in the evaluation and comparison of the proposed methods to the benchmarks models.

- In *Chapter 3* the Concept Drift problem is discussed and some related works are also reported.
- *Chapter 4* presents the implementation of the proposed families of algorithms FASEB and FASEO. Each method is explained through a general description of its pseudo-code emphasizing the differences and similarities between FASEB and FASEO.
- In *Chapter 5* an analysis of the spatial and temporal complexity of the proposed methods is presented and the experimental setup is described, providing the performance evaluation of the proposed algorithms against previous approaches, discussing the results and main findings obtained with the proposed methods.
- Finally, *Chapter 6* concludes the dissertation, points out the main contributions of the research carried out, discusses limitations and raises possible ideas for future works.

2 THE CLASSIFICATION CHALLENGES IN DATA-STREAM CONTEXTS

2.1 INTRODUCTION

Within the area of Artificial Intelligence (AI), the learning component refers to the techniques and mechanisms for generalizing from examples and updating the predictive models from evolving data (GAMA et al., 2014). Learning can also be seen as an update on behavior, skills or knowledge in general in order to improve performance (ORTIZ-DIAZ, 2014). In this context, Machine Learning (ML) involves the study and development of computational models capable of improving their performance with experience and of acquiring knowledge on their own (MINKU; WHITE; YAO, 2010).

When designing a learning system, several elements must be taken into account: the type of training experience, the measurement of performance, the target function and its representation as well as the algorithm to approximate it. The target function can be understood as the definition of the type of knowledge that must be learned. Machine Learning (ML) seeks to describe an target function in order to use some algorithms to compute it. Commonly, the description of this function is non-operational, that is, it cannot be computed efficiently. As a consequence, learning systems seek an operational description of it, which is called approximation of functions. Depending on the kind of experience used, Machine Learning (ML) can be divided into two main types: Supervised, when the aim is learning a function from examples of inputs and outputs, that is, the *a priori* classes are previously known and based on them the data is classified; and Unsupervised, when there are no specified output values of *a priori* classes on the data, and the aim is learning patterns of inputs and creating differentiated groups (ORTIZ-DIAZ, 2014).

Thus, the classification within supervised learning tries to infer a target function that maps feature values into class labels from training data, and apply the function to data with unknown class labels. In general, a classification algorithm defines a hypothesis space to search for the correct model making certain assumptions about hypothesis space. The algorithm also defines a criterion to determine the quality of the model. Thus, the model that reached the best measure in the hypothesis space will be returned. The classification aims to find the model that achieves good performance when predicting the labels of future unseen data. To improve the generalization ability of a classification model, it should not overfit the training data; instead, it should be general enough to cover unseen cases (AGGARWAL, 2014).

In the following sections, related topics will be addressed in more detail.The present chapter discusses how classification algorithms process the data-stream whether stationary or not, whereas the next chapter will address the problem of Concept Drift.

2.2 DATA-STREAM CONTEXT

Advances in hardware technology have led to the increasing popularity of data-streams. Many simple operations of everyday life, such as using a credit card or the phone, often lead to automated creation of data. Since these operations often involve a large numbers of participants, they lead to massive data-streams. Similarly, telecommunications and social networks often contain large amounts of network or text data-streams (AGGARWAL, 2014). A data-stream is a potentially unbounded, ordered sequence of data items which arrive over time. The time intervals between the arrival of each data item may vary. These data items can be simple attribute-value pairs like relational database tuples, or more complex structures (KRAWCZYK et al., 2017).

Thus, data-streams can be seen as stochastic processes in which events occur continuously and independently from each another. Querying data-streams is quite different from querying in the conventional relational model. A key idea is that operating on the data-stream model does not preclude the use of data in conventional stored relations: data might be transient (GAMA, 2010). What makes processing data-streams different from the conventional relational model?

Traditional data mining is able to scan data-sets many times; execute with unlimited time and memory; has only one concept; and needs to produce fairly accurate results. Moreover, data-stream mining may produce approximate results and has to satisfy constraints, such as single-pass, real-time response, bounded memory, and concept-drift detection (RASTIN, 2018):

- Single-pass: Unlike traditional data mining that may read static data-sets repetitively many times, each sample in a data-stream is examined at most once and cannot be backtracked.
- Real-time response: Many data-stream applications such as stock market prediction require real-time response. The amount of time for processing the data and providing decision must be fast.
- Bounded memory: The amount of arriving data is extremely large. As we may only compute and store a small summary of the data-streams and possibly throw away the rest of the data; approximate results are acceptable.
- Concept Drift: In data-streams, concept drifts refer to the situation when the discovered patterns (or the underlying data distribution) change over time.

Data-streams pose new challenges for Machine Learning (ML) and Data Mining (DM) as the traditional methods have been designed for static data-sets and are not capable of efficiently analyzing fast growing amounts of data because this context grants new features in the data, for example (GAMA, 2010):

- The data elements in the stream arrive sequentially over time.
- The system has no control over the order in which data elements arrive, either within a data-stream or across data-streams.
- Data-streams are potentially unbound in size, it is usually impossible to store all the data from the data-stream in memory.
- Once an element from a data-stream has been processed, it is discarded or archived (only one scan). It cannot be retrieved easily unless it is explicitly stored in memory, which is small relative to the size of the data-streams.
- The data items arrival rate is rapid (relatively high with respect to the processing power of the system),
- data-streams are susceptible to change (data distributions generating examples may change on the fly),
- The data labeling may be very costly (or even impossible in some cases), and may not be immediate.

Data-Streams may be described by a sequence (possibly infinite) of examples (also known by instances or experiences) $S = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_i, \ldots, \mathbf{z}_t\}$ that are obtained in time, usually one at a time. A training example $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ is formed by a d-dimensional vector $\mathbf{x}_i = (x_{i1}, \ldots, x_{ij}, \ldots, x_{id})$ and a discrete value or category y_i, that is its corresponding class label taken from a finite set Y of class labels. Each vector $\mathbf{x}_i$ has the same number of dimensions d, where each dimension corresponds to an attribute whose attribute value (numeric or nominal) is x_{ij} (FRÍAS-BLANCO, 2014). These data can be stationary, where examples are drawn from a fixed, albeit unknown, probability distribution, and non-stationary, where data can evolve over time. In the second case, target concepts (classes of examples) and/or attribute distributions change. In other words, the concept from which the data-stream is generated shifts after a minimum stability period. This phenomenon is called Concept Drift.

The change of concept are reflected in the incoming instances and deteriorate the accuracy of models learned from past training instances (KRAWCZYK et al., 2017). Some real-life scenarios of the non-stationary data-stream are (ZLIOBAITE, 2010): computer or telecommunication systems (attackers look for new ways of overcoming security systems), traffic monitoring (patterns may change over time), weather predictions (climate changes and natural anomalies may influence the forecast), system following personal interests like personal advertisement (users may change their preferences), medical decision aiding (disease progression may be influenced and changed in response to applied drugs or natural resistance of the patients), and others.

2.3 THE CLASSIFICATION PROBLEM

An important challenge in the streaming scenario is data classification. In this problem, the data instances are associated with labels, and it is desirable to determine the labels on the unknown instances. Typically, it is assumed that both the training data and the test data may arrive in the form of a stream. In the most general case, the two types of instances are mixed with one another. Since the test instances are classified independently of one another, it is usually not difficult to perform the real-time classification of the test stream (AGGARWAL, 2014).

In the classification scenario, it is assumed that exist a target function $f(\mathbf{x}) = y$ and the goal is to obtain a model that approximates $f(\mathbf{x})$ as $h(\mathbf{x})$ to classify or predict the label of unlabeled examples, such that $h(\mathbf{x})$ maximizes the prediction accuracy (MITCHELL, 1997). In others words, during the classification task, a learning model $h(\mathbf{x})$ attempts to predict the class label of the incoming instance (KHAMASSI et al., 2018).

The problem of data classification is one of the most widely studied in the DM and ML communities. Classification is a rather diverse topic, and the underlying algorithms depend on the data domain and problem scenario. This is because the problem try to learn the relationship between a set of feature variables and a target classes. Since many practical problems can be expressed as associations between feature and target variables, this provides a broad range of applicability of this task. Applications of classification include a wide variety of problem domains such as (AGGARWAL, 2014):

- Medical Disease Diagnosis: The use of data mining methods in medical technology has gained increasing attraction in recent years.The features may be taken out from medical records and the goal is to predict whether or not a patient may pick up a disease in the future.

- Customer Target Marketing: This method is extremely popular for the problem of customer target marketing. In such cases, feature variables describing the customer are used as previous training examples in order to predict their interests. The target variable may encode the buying interest of the customer.

- Supervised Event Detection: Class labels may be related with time stamps corresponding to unusual events in a large number of scenarios. For example, an intrusion activity may be considered as a class label. In such cases, time-series classification methods can be very useful.

- Social Network Analysis: Many forms of social network analysis, such as collective classification, associate labels with the underlying nodes. These are then used in order to predict the labels of other nodes. Such applications are very useful for predicting useful properties of actors in a social network.

- Biological Data Analysis: Biological data is often represented as discrete sequences, in which it is desirable to predict the properties of particular sequences. In some cases, the biological data is also expressed in the form of networks. Therefore, classification methods can be applied in a variety of different ways in this scenario.

- Multimedia Data Analysis: It is often desirable to perform classification of large volumes of multimedia data such as photos, audio and videos. Many times multimedia data analysis can be challenging, because of the complexity of the underlying feature space and the semantic gap between the feature values and corresponding inferences.

- Document Categorization and Filtering: Large numbers of applications require the classification of many documents in real time. This application is referred to as document categorization, and is an important area of research in its own right.

The problem of classification may be stated as follows: Given a training data set along with associated labels, determine the class label for an unlabeled data. These kind of algorithms typically contain two phases (AGGARWAL, 2014):

(i) Training Phase: In this phase, a model is constructed from the training instances.

(ii) Testing Phase: In this phase, the model is used to assign a label to an unlabeled test instance

In the first phase, the model is constructed tuple to tuple, describing in each case the attributes of the data-set that is analyzed. It is assumed that each tuple belongs to a predefined class, which is determined by one of the attributes called class. The tuples within the classification are also known as examples, instances or experiences. The set of examples analyzed to build the model form the training data-set. The individual examples that make up the training set are randomly selected from the sample population by assigning to each one a class label.

In the second phase, the model obtained in the previous phase is used in the prediction and the model (or classifier) accuracy can be estimated (ORTIZ-DIAZ, 2014). The output of a classification algorithm may be presented for a test instance in one of two ways (AGGARWAL, 2014):

- Discrete Label: In this case, a label is returned for the test instance.

- Numerical Score: In this case, a numerical score is returned for each class label and test instance combination.The numerical score can be converted to a discrete label for a test instance and its advantage is that it now becomes possible to compare the relative propensity of different test instances to belong to a particular class of importance, and rank them if needed

As a summary, we can say that, the classification aims at the construction of concise models that represent the distribution of the dependent attribute (class) as a function of the predictor attributes (attributes). The resulting model will be used, mainly, to determine the class to which the observations belong, of which all the values of their attributes are known, with the exception of the class value, that is, their task will be preferably predictive (YE; CHEN, 2003).

2.3.1 Popular classification algorithms

ML classifiers have been successfully used in a range of different applications. Classifiers widely used DM include statistical and probabilistic classifiers (i.e., the Naïve Bayes algorithm), decision tree and rule-based classifiers (i.e., ID3 and C4.5), regression methods (i.e., logistic regression), neural networks, support vector machines, and instance based learning classifiers (i.e., kNN algorithm).

The identification of optimal base learners is very important for the predictive performance of any model that involves the use of classifiers as base learners. The following sections describe the selected base classifiers, HT and NB in the proposed approaches.

2.3.1.1 Naïve Bayes classifier

The Naïve Bayes classifier(NB) (JOHN; LANGLEY, 1995) algorithm is a statistical classification algorithm based on Bayes's theorem. It allows probabilistic knowledge to be used, learned, and represented clearly, since the statistical components of the NB algorithm can be unambiguously and formally expressed. Based on the class conditional independence assumption, the algorithm is a simple, yet computationally efficient classification algorithm that can scale well in several fields. The algorithm shows high predictive performance and obtains results comparable to other classification techniques, such as decision trees and neural networks (ONAN; KORUKOĞLU; BULUT, 2016). As shown in the 2.1 equation for each unlabelled instance, the NB predicts a class y, based on the posterior probability of class y, given the *instance* $\mathbf{x} = (x_1, \ldots, x_j, \ldots, x_d)$:

$$\hat{y} = \arg\max_{y \in Y} P(y) \prod_{j=1}^{d} P(x_j|y) \tag{2.1}$$

where $P(y)$ is the prior probability of class y and $P(x_j|y)$ is the conditional probability of x_j given y.

2.3.1.2 Hoeffding Tree classifier

Very Fast Decision Tree classifier (VFDT) is based on the Hoeffding Tree classifier (HT) (HULTEN; SPENCER; DOMINGOS, 2001), a decision tree learning method. The intuition idea of the HT is that to find the best splitting attribute it is sufficient to consider

only a small portion of the training items available at a node. To achieve this goal, the Hoeffding bound (HOEFFDING, 1963) is utilized. Basically, given a real-valued random variable r having range R, if we have observed n values for this random variable, and the sample mean is $\bar{r}$, then the Hoeffding bound states that, with probability $1 - \delta$, the true mean of r is at least $\bar{r} - \varepsilon$, where:

$$\varepsilon = \sqrt{\frac{R^2(\ln 1/\delta)}{2n}} \tag{2.2}$$

Based on the above analysis, if at one node we find that $\overline{G}_l(x_a) - \overline{G}_l(x_b) \geq \varepsilon$ where $\overline{G}_l$ is the splitting criterion, and x_a and x_b are the two attributes with the best and second best $\overline{G}_l$ respectively, then x_a is the correct choice with probability $1 - \delta$ (AGGARWAL, 2014).

What makes Hoeffding Bound attractive is the ability to achieve the same results regardless of the probability distribution that generates the observations. However, the number of observations required to reach certain values of δ and ϵ to obtain the best performance of the classifier must be adjusted according to the probabilities distribution in which it is intended to apply. For a better understanding of the algorithm, the Hoeffding's Inequality Theorem and the pseudo-code are presented in *Appendix. A*.

2.4 ENSEMBLE OF CLASSIFIERS IN DATA-STREAM CONTEXT

Ensemble algorithms are classification methods that re-uses one or more currently existing classification algorithm by applying either multiple models, or combining the results of the same algorithm with different parts of the data. Ensemble methods aim to obtain more robust results by combining the output from multiple training models either sequentially or independently. Different forms of ensemble analysis attempt to reduce the bias and variance because the overall error of a classification model depends upon the bias and variance, besides the intrinsic noise present in the data. The bias of a classifier depends upon the fact that the decision boundary of a particular model may not correspond to the true decision boundary (AGGARWAL, 2014).

These methods have emerged as a powerful technique for improving the accuracy of base models (AGGARWAL, 2014) combining the predictions of multiple classifiers to obtain a classification model with better predictive performance. Thus, by combining classifiers, the variance and bias of classification can be reduced and the dependency of results on characteristics of a single training set eliminated (KUNCHEVA, 2004).

Moreover, they are particularly popular methods for stream classification, because of the fact that classification models frequently cannot be built robustly in a fast data-stream. Ensemble algorithms can be regarded as a family of classification algorithms, which are developed to improve the generalization abilities of classifiers. It is hard to get a single classification model with good generalization ability, which is called a strong

classifier, but ensemble learning can transform a set of weak classifiers into a strong one by their combination. Formally, we training k classifiers $h_1, ..., h_j, ..., h_k$ each of which maps vectors of feature values $\mathbf{x}$ into a class label y. We combine them into an ensemble classifier H with the hope that it achieves better performance (AGGARWAL, 2014).

There are two important factors that contribute to the success of ensemble learning (ORTIZ-DIAZ, 2014; AGGARWAL, 2014):

(i) The expected error in ensemble models is smaller than the expected error in a single model.

(ii) Ensemble models have the ability to overcome the limitation of the hypothesis space made by single models.

Intuitively, if we know in advance that $h_1(\mathbf{x})$ was the best prediction performance on unknown data, then without hesitation, we should discard the other classifiers and choose h_1 for future predictions. However, we do not know the true labels of unknown data, and thus we are unable to know in advance which classifier performs the best. Therefore, our best bet should be the prediction obtained by combining multiple models.

On the other hand, a single-model classifier usually searches for the best model within a hypothesis space that is assumed by the specific learning algorithm. It is very likely that the true model does not reside in the hypothesis space, and then it is impossible to obtain the true model by the learning algorithm. However, by combining multiple classifiers, ensemble methods can simulate the true boundary. The reason is that ensemble approaches combine different hypotheses and the final hypothesis is not necessarily contained in the original hypothesis space. Therefore, ensemble methods have more flexibility in the hypothesis they could represent.

Due to these advantages, many ensemble approaches have been developed to combine complementary predictive power of multiple models (AGGARWAL, 2014). There are two components in ensemble learning: Training base models and learning their combinations.

From the majority voting, the base classifiers should be accurate and independent to obtain a good ensemble. In general, we do not require the base models to be highly accurate—as long as we have a good amount of base classifiers, the weak classifiers can be boosted to a strong classifier by combination. However, in the extreme case where each classifier is extremely wrong, the combination of these classifiers will give even worse results. Therefore, at least the base classifiers should be better than random guessing. Independence among classifiers is another important issue we want to see in the collection of base classifiers. If base classifiers are highly correlated and make very similar predictions, their combination will not improve anymore.

In contrast, when base classifiers are independent and make diverse predictions, the independent errors have better chances to be canceled out. Typically, the following techniques have been used to generate a good set of base classifiers (AGGARWAL, 2014):

- Obtain different bootstrap samples from the training set and train a classifier on each bootstrap sample;
- Extract different subsets of examples or subsets of features and train a classifier on each subset;
- Apply distinct learning algorithms on the training set;
- Incorporate randomness into the process of a particular learning algorithm or use different parametrization in order to obtain different predictions.

To further improve the accuracy and diversity of base classifiers, people have explored various ways to prune and select base classifiers (AGGARWAL, 2014). Classifier selection involves the identification of an optimal set of classifiers however, identifying an optimal subset of classifiers can require enumerating many different cases. In order to overcome this computational burden, heuristic methods and selection metrics can be used in classifier selection (ONAN; KORUKOĞLU; BULUT, 2016).

2.4.1 Ensemble combination strategy

Once the base classifiers are obtained, the important question is how to combine them. The combination strategies used by ensemble learning methods roughly fall into two categories: Stacking and Voting (Un-weighted and weighted).

2.4.1.1 Stacking

Stacked Generalization (Stacking) learns a high-level classifier on top of the base classifiers. It can be regarded as an ensemble learning method which has a two-level structure (WOLPERT, 1992). In that strategy, the base classifiers are called first-level classifiers and a second-level classifier (meta-classifier) is learn to combine the first-level classifiers. The general procedure of Stacking has the following three major steps:

(i) First-level classifiers are trained to obtain predictions based on training instances.

(ii) Construct a meta-level training data-set with the same classes of the original. To obtain meta-level data, a meta-instance is generated by combining the outputs of the k first-level classifiers and true class labels, and the meta-classifier is trained with those meta-instances. In others words, the output predicted labels of the first-level classifiers are regarded as new features, and the original class labels are kept as the labels in the new data-set based on the outputs of these classifiers. Formally, is assumed that each example in D is $(\mathbf{x}, y)$ and is constructed a corresponding example $(\mathbf{x}^{'}, y)$ in the new data-set, where $\mathbf{x}^{'} = (h_1(\mathbf{x}), ..., h_j(\mathbf{x}), ..., h_k(\mathbf{x}))$ (AGGARWAL, 2014). Because in the learning process, each classifier h_j, can map vectors of feature

values $\mathbf{x}$ into a class label y, the meta-instance can also be represented as $M = (\mathbf{x}', y) = (\hat{y}_1, \ldots, \hat{y}_j, \ldots, \hat{y}_k; y)$ where each $\hat{y}_j = h_j(\mathbf{x})$ is the predicted class label provided by classifier h_j and y is its corresponding class label.

(iii) Learn a second-level classifier based on the newly constructed data-set. Any learning method could be applied to learn the second-level classifier. Once the second-level classifier is generated, it can be used to combine the first-level classifiers. For an unseen example, its predicted class label of stacking is $h'(h_1(\mathbf{x}), ..., h_j(\mathbf{x}), ..., h_k(\mathbf{x}))$, where $h_1, ..., h_j, ..., h_k$ represent the first-level classifiers and h' is the second-level classifier. We can see that Stacking is a general framework. We can plug in different learning approaches or even ensemble approaches to generate first or second level classifiers. Compared with Bagging and Boosting, Stacking "learns" how to combine the base classifiers instead of voting (AGGARWAL, 2014).

Is important to consider what types of feature to create for the second-level classifier's training set, and what type of learning methods to use in building the second-level classifier (WOLPERT, 1992). Besides using predicted class labels of first-level classifiers, we can consider using class probabilities as features (TING; WITTEN, 1999). The advantage of using conditional probabilities as features is that the training set of second-level classifier will include not only predictions but also prediction confidence of first-level classifiers. Some choices of second-level classification algorithms are eligible, for example, (i) Feature-Weighted Linear Stacking combines classifiers using a linear combination of meta features, (ii) Multi-response linear regression, a variant of least-square linear regression, etc (AGGARWAL, 2014).

2.4.1.2 Voting Strategy

Voting is the simplest form of combining individual classification algorithms, in which choosing the combination rule is critical for designing classifier ensembles. In general, voting strategy include: Un-weighted and Weighted voting:

Un-weighted voting

In classifier ensembles, simple majority voting ranks among the most widely used combination rules. It is a typical un-weighted combination strategy, in which we count the number of votes for each predicted label among the base classifiers and choose the one with the highest votes as the final predicted label (AGGARWAL, 2014), in other words, the binary outputs of the k base classifier are combined so that the highest number of votes is determined as the output of the ensemble. This approach treats each base classifier as equally accurate and thus does not differentiate them in the combination. However, several other linear algebraic combination rules guide classifiers as well: minimum probability, maximum probability, product of probability, and the average of probabilities (ONAN; KORUKOĞLU; BULUT, 2016).

Let $\mathbf{z} = (\mathbf{x}, y)$ be an instance and let $h_j\,(1 \leq j \leq k)$ be a classifier.

Let $h_j(\mathbf{x}) = (P_j(y_1|\mathbf{x}), \ldots, P_j(y_C|\mathbf{x}))$ where $P_j(y_c|\mathbf{x})\,(1 \leq c \leq C)$ is the posterior probability of y_c given $\mathbf{x}$. Let $LC(\mathbf{x}) = (\sum_{j=1}^{k} P_j(y_1|\mathbf{x}), \ldots \sum_{j=1}^{k} P_j(y_C|\mathbf{x}))$, where $LC_c(\mathbf{x}) = \sum_{j=1}^{k} P_j(y_c|\mathbf{x}))$, then $H(\mathbf{x})$ is the component of $LC(\mathbf{x})$ which has the largest sum of probabilities in Equation. 2.3 to determine the final output of the classifier ensemble (ONAN; KORUKOĞLU; BULUT, 2016):

$$H(\mathbf{x}) = \underset{1 \leq c \leq C}{\operatorname{argmax}}(LC_c(\mathbf{x})) \tag{2.3}$$

in which c denotes the number of classes. Based on that notation, average, product, minimum, and maximum of probabilities and majority voting combination rules can be computed using Equation. 2.4-2.7, respectively (ONAN; KORUKOĞLU; BULUT, 2016):

$$LC_c(\mathbf{x}) = \frac{1}{k} \sum_{j=1}^{k} P_j(y_c|\mathbf{x}) \tag{2.4}$$

$$LC_c(\mathbf{x}) = \frac{1}{k} \prod_{j=1}^{k} P_j(y_c|\mathbf{x}) \tag{2.5}$$

$$LC_c(\mathbf{x}) = \min_{1 \leq j \leq k} P_j(y_c|\mathbf{x}) \tag{2.6}$$

$$LC_c(\mathbf{x}) = \max_{1 \leq j \leq k} P_j(y_c|\mathbf{x}) \tag{2.7}$$

Weighted voting

Weighted combination usually assigns a weight to each classifier with the hope that higher weights are given to more accurate classifiers so that the final prediction can bias towards the more accurate classifiers. The weights can be inferred from the performance of base classifiers or the combined classifier on the training set (AGGARWAL, 2014).

In simple weighted voting, the weight values w_j for each classifier are determined based on the predictive performance of a particular classifier with the training set using Equation.2.8

$$w_j = \frac{a_j}{\sum_k a_k} \tag{2.8}$$

in which a_j denotes the estimated accuracy of the j_{th} classifier on the training set and a_k refers to the estimated accuracy of the k_{th} classifier.

In rescaled weighted voting, a zero weight is assigned to classifiers that have the worst performance with the training data. Consequently, classifiers with predictive performance inferior than the threshold value are excluded from the classifier ensemble. The weight values are scaled proportionally using Equation.2.9, in which e_j denotes the number of

errors obtained by a particular classifier h_j, n denotes the number of samples, and c denotes the number of classes (MORENO-SECO et al., 2006):

$$w_j = \max\left\{0, 1 - \frac{c.e_j}{n(c-1)}\right\} \tag{2.9}$$

In best–worst weighted voting, classifiers in the ensemble with the best and worst performance are identified. The weight value of the best classifier is one, whereas the weight value of the worst classifier is zero. As such, the classifier with the worst predictive performance is removed from the classifier ensemble. The weight value of the $j-th$ classifier is determined using Equation.2.10, in which e_j is the error of the $j-th$ classifier, e_B denotes the minimum error among the classifiers and e_w the maximum error (MORENO-SECO et al., 2006):

$$w_j = 1 - \frac{e_j - e_B}{e_w - e_B} \tag{2.10}$$

The quadratic best–worst weighted vote aims to assign higher weight values to classifiers with higher predictive performance, namely by using Equation.2.11:

$$w_j = \left(\frac{e_w - e_j}{e_w - e_B}\right)^2 \tag{2.11}$$

Once the weights for each classifier decision have been computed, the class receiving the highest score in the voting is the final class prediction, then the prediction of the ensemble can be computed as:

$$H(\mathbf{x}) = \underset{1\leq c\leq C}{\operatorname{argmax}}(w_j LC_c(\mathbf{x})) \tag{2.12}$$

2.4.2 Popular ensemble algorithms

Some examples of popular and representative ensemble algorithms follow below (AGGARWAL, 2014):

- Boosting: Boosting (FREUND, 1995) is a common technique used in classification. The idea is to focus on successively difficult portions of the data-set to create models that can classify the data points in these portions more accurately, and then use the ensemble scores over all the components. A hold-out approach is used in order to determine the incorrectly classified instances for each portion of the data-set. Thus, the idea is to sequentially determine better classifiers for more difficult portions of the data, and then combine the results in order to obtain a meta-classifier, which works well on all parts of the data.

- Bagging: Bagging (BREIMAN, 1996) is an approach that works with random data samples, and combines the results from the models constructed using different sam-

ples. The training examples for each classifier are selected by sampling with replacement. These are referred to as bootstrap samples. This approach has often been shown to provide superior results in certain scenarios, though this is not always the case. This approach is not effective for reducing the bias, but can reduce the variance, because of the specific random aspects of the training data.

- Model Averaging and Combination: This is a typical model used in ensemble analysis. In fact, the random forest method is a special case of this idea. In the context of the classification problem, many Bayesian methods (DOMINGOS; HULTEN, 2000) exist for the model combination process. The use of different models ensures that the error caused by the bias of a particular classifier does not dominate the classification results.

- Random Forests: Random forests (BREIMAN, 2001) are a method that use sets of decision trees on either splits with randomly generated vectors, or random sub-sets of the training data, and compute the score as a function of these different components. Frequently, the random vectors are generated from a fixed probability distribution. Therefore, random forests can be created by either random split selection, or random input selection. Random forests are related to bagging, and in fact bagging with decision trees can be considered a special case of random forests, in terms of how the sample is selected (boot strapping). In the case of random forests, it is also possible to create the trees in a lazy way, which is tailored to the particular test instance at hand.

- Stacking: Methods such as stacking (WOLPERT, 1992) also combine different models in a variety of ways, such as using a second-level classifier in order to perform the combination. The output of different first-level classifiers is used to create a new feature representation for the second level classifier. These first level classifiers may be chosen in a variety of ways, such as using different bagged classifiers, or by using different training models. In order to avoid over fitting, the training data needs to be divided into two subsets for the first and second level classifiers.

2.5 EVALUATION IN DATA-STREAM CONTEXT

Proper evaluation of the different models is a important issue in ML. Thus, many evaluation measures, techniques for their experimental estimation and approaches to compare algorithms have already been proposed for static data scenarios. On the other hand, researchers agree that the evaluation of data-stream algorithms is a complex task. Especially in non-stationary environments, fewer solutions are presented (KRAWCZYK et al., 2017). This fact is due to many challenges like the presence of Concept Drift, limited processing time in real-world applications and the need for time-oriented evaluation, amongst oth-

ers (GAMA et al., 2004). The next section presents some out of the most popular evaluation measures and then their experimental estimation procedures.

2.5.1 Evaluation measures

According to (KOHAVI; PROVOST, 1998) confusion matrix contains information about actual and predicted classifications done by a binary classification system where:

- **True Positive (TP)** are the number of positive instances correctly classified.
- **False Positive (FP)** are the number of instances that are falsely classified to the positive class.
- **True Negative (TN)** are the number of negative instances correctly classified.
- **False Negative (FN)** are the number of instances that are falsely classified to the negative but are positive.

The metrics above are defined for binary classification problems, but they can easily be used for multi-class problems. A common practice is to calculate the measure for each class separately and then to average the metrics over all classes (AGGARWAL, 2014).

From the classification outputs, it is possible to obtain several metrics that may be more or less useful in certain contexts. Some of these metrics will be discussed below.

Accuracy (ACC): Represent the overall correctness of the learner. The accuracy is most often used as the defining measure of the classification performance for evaluating learning algorithms in most streaming studies (HULTEN; SPENCER; DOMINGOS, 2001; GAMA et al., 2004; BAENA-GARCIA et al., 2006; BIFET; GAVALDÀ, 2007; HUANG et al., 2015). It is calculated incrementally using either the prequential or hold-out evaluation procedures (BIFET; KIRKBY, 2009) that is, as the sum of correctly classified instances divided by the total instances. The accuracy is computed using the equation:

$$ACC = \frac{TN + TP}{TN + FN + FP + TP} \tag{2.13}$$

Generally, in online learning, is also important to consider the equation 2.14, that computes the accuracy given a time t (BAENA-GARCIA et al., 2006; DU; SONG; JIA, 2014). It represents the average accuracy obtained by the prediction of each example before it is learned, where acc_{ex} is 1 if the current example is correctly classified and 0 otherwise. f represent the first timestamp of every calculation, ie, the timestamp of each detected concept drift. Thus, the highest value is the best, suggesting better performance in prequential accuracy (MINKU; YAO, 2012; HIDALGO; MACIEL; BARROS, 2019).

$$acc(t) = \begin{cases} acc_{ex}(t), \ \ if \ \ t = f \\ acc(t-1) + \frac{acc_{ex}(t) - acc(t-1)}{t-f+1}, \ \text{otherwise.} \end{cases} \tag{2.14}$$

An interesting feature in the dynamic environments is the fact that usually the data is not stationary, i.e., the probability distribution of the data may change over time (HIDALGO; MACIEL; BARROS, 2019). When analyzing the performance of classifiers dedicated to non-stationary data, other factors should be also taken into consideration as their adaptation to change abilities, i.e., evaluating the maximum performance deterioration and restoration time. Apart from the predictive accuracy or error, the following performance metrics should be monitored and taken into account during properly executed evaluation of streaming algorithms (KRAWCZYK et al., 2017):

Memory consumption: Memory consumption should not only monitor the average memory requirements of each algorithm, but also their change over time with respect to actions being taken.

Update time: Update time refers to the amount of time that an algorithm requires to update its structure and accommodate new data from the stream. In an ideal situation, the update time should be lower than the arrival time of a new example (or chunk of data).

Decision time: Decision time represent the time that a model needs to make a decision regarding new instances from the stream. This phase usually comes before the updating procedure takes place. So, any decision latency may result in creating a bottleneck in the stream processing. This is especially crucial for algorithms that cannot update and make predictions regarding new instances at the same time. Nevertheless, in order to compute reaction times and other adaptability measures, usually a human expert needs to determine moments when a drift starts and when a classifier recovers from it. Alternately, such evaluations are carried out with synthetic data generators (KRAWCZYK et al., 2017).

Besides, the interplay between the accuracy and other factors, such as memory usage and run-time considerations, has received limited attention (PESARANGHADER; VIKTOR; PAQUET, 2016). Thus, more complex measures have also been proposed to evaluate other properties of algorithms. Some proposals follow:

Bifet et al. (2009) considered the **Memory**, **Time** and **Recall** measures separately, in order to compare the performances of ensembles of classifiers.

Bifet et al. (2010) further introduced the **RAM-Hour** measure, where every **RAM-Hour** equals to 1 GB of RAM occupied for one hour, to compare the performances of three versions of perceptron-based Hoeffding Trees.

In (SHAKER; HüLLERMEIER, 2015) it was proposed a complete framework for evaluating the recovery rate of the algorithm once a change has occurred in the stream. They consider not only how well the model reduced its error in the new decision space, but also what was the time necessary to achieve this.

In (PESARANGHADER; VIKTOR; PAQUET, 2016) it was introduced the **EMR** measure which combines error-rate, memory usage and run-time for evaluating and ranking learning algorithms.

In (ZLIOBAITE; BUDKA; STAHL, 2015) it was proposed the return on investment **ROI** measure to determine whether the adaptation of a learning algorithm is beneficial. Through the notion of cost-sensitive update is evaluated the potential gain from the cost (understood as time and computational resources). The authors argue that this allows to check if the actual update of the model was a worthwhile investment. They concluded that adaptation should only take place if the expected gain in performance, measured by error-rate, exceeds the cost of other resources (i.e., memory and time) required for adaptation. In their work, the **ROI** measure was used to indicate whether an adaptation to a Concept Drift is beneficial, over time.

In (OLORUNNIMBE; VIKTOR; PAQUET, 2015) it was extended the above mentioned **ROI** measure, in order to dynamically adapt the number of base learners in online bagging ensembles.

In (PESARANGHADER; VIKTOR; PAQUET, 2016) it was proposed an approach to count true positive (**TP**), false positive (**FP**), and false negative (**FN**) of drift detection, in order to evaluate the performances of Concept Drift detectors. They introduced the acceptable delay length notion as a threshold that determines how far a detected drift could be from the real location of drift to be considered as a true positive.

In (PESARANGHADER; VIKTOR; PAQUET, 2018) it was introduced the **CAR** measure which not only considers the classification error-rates, memory usages, and run-times but also the drift detection delays, false positives and false negatives. The **CAR** measure, consists of three components namely Classification, Adaptation, and Resource Consumption. The classification part consists of the error-rate ($\mathbf{E}_C$) of classifier C, the adaptation part represents the detection delay ($\mathbf{D}_D$), false positive ($\mathbf{FP}_D$) and false negative ($\mathbf{FN}_D$) of drift detector D, while the resource consumption part is associated with the memory consumption ($\mathbf{M}_{(C,D)}$) and run-time ($\mathbf{R}_{(C,D)}$) of the (Classifier, Detector) pair.

2.5.2 Techniques used for evaluation

Any classification algorithms require training and testing phases, in which the test examples are cleanly separated from the training data and the model is constructed on these examples (AGGARWAL, 2014). In order to do this, several strategies can be applied, such as holdout, bootstrapping, cross-validation and Prequential methodology (DAWID, 1984).

Bootstrapping: Bootstrapping used sampling with replacement for creating the training examples. The most typical scenario is that n examples are sampled with replacement, as a result of which the fraction of examples not sampled is equal to $(1 - 1/n)^n \approx 1/e$, where e is the basis of the natural logarithm. The class accuracy is then evaluated as a weighted combination of the accuracy acc_1 on the test examples, and the accuracy acc_2 on the full labeled data. The full accuracy can be computed by the equation: $ACC = (1-1/e).acc_1+(1/e).acc_2$ This procedure is repeated over multiple bootstrap sam-

ples and the final accuracy is reported. The component acc_2 tends to be highly optimistic, as a result of which the bootstrapping approach produces highly optimistic estimates. It is most appropriate for smaller data-sets (AGGARWAL, 2014).

Cross-validation: The training data is divided into a set of s disjoint subsets. One of the s subsets is used for testing, whereas the other $(s-1)$ subsets are used for training. This process is made periodically by using each of the s subsets as the test set, and the error is averaged over all possibilities. This has the advantage that all examples in the labeled data have an opportunity to be treated as test examples. Furthermore, when s is large, the training data size approaches the full labeled data. Therefore, such an approach approximates the accuracy of the model using the entire labeled data well. A special case is "leave-one-out" cross-validation, where s is chosen to be equal to the number of training examples, and therefore each test segment contains exactly one example. This is, however, expensive to implement (AGGARWAL, 2014).

In the context of static and batch learning the most often used scenario for estimating prediction measures is *cross validation*. However, in the context of online learning with computationally strict requirements and concept drifts, it is not directly applicable. Two main approaches are used depending whether the stream is stationary or not (KRAWCZYK et al., 2017), as shown below.

Holdout evaluation: A fixed percentage of the training examples are not used in the training. These examples are then used for evaluation. It is arranged that, at any given moment of time when we want to conduct model evaluation, we have at our disposal a holdout set not previously used by our model. Since only a subset of the training data is used, the evaluation tends to be pessimistic with the approach. Some variations use stratified sampling, in which each class is sampled independently in proportion. This ensures that random variations of class frequency between training and test examples are removed (AGGARWAL, 2014). By testing the learning model on such a continuously updated set (it must be changed after each usage to ensure that it represents the current concept well), we obtain an unbiased estimator of the model error. When conducted in a given time or instance interval, it allows us to monitor the progress of the model (KRAWCZYK et al., 2017).

Prequential evaluation: Methodology for the performance evaluation of classifiers on data-streams with stationary and non stationary distributions. The prequential (combination of words predictive and sequential) evaluation use each instance first to test the model, and then to train the model. This scheme has the advantage that no holdout set is needed for training, making maximum use of the available data, even so, knowing that the holdout methodology provides alower error-rate during early stages of learning (BIFET et al., 2009). It performs a sequential analysis where the sample size is not fixed in advance. Instead, data are evaluated as they are collected. Predictive sequential evaluation, or prequential, follows the online learning protocol. Whenever an example is observed,

the current model makes a prediction; when the system receives feedback from the environment, we can compute the loss function (KRAWCZYK et al., 2017). The prequential evaluation methodology has three common variations regarding the portion of the data considered in the calculations: (i) Basic Window (BW), (ii) Sliding Window (SW), and (iii) Fading Factors (FF) (HIDALGO; MACIEL; BARROS, 2019):

(i) BW variation, also known as Interleaved Test-Then-Train, uses all processed instances of the stream to builds the decision model. This feature might be seen as a disadvantage of the method because can difficult the classification task at a given point in time, since the success rate is influenced by errors in previous concepts.

(ii) SW variation was proposed later to provide a forgetting mechanism aiming to minimize possible issues using very large numbers of instances. In this case, the decision model is updated based on a portion of instances, represented by a SW of size W.

(ii) FF variation also adopts a forgetting mechanism. In addition, FF uses a fading factor which applied a function in order to decrease weight, giving more importance to the recent examples.

2.6 FINAL CONSIDERATIONS

This chapter has presented several inherent definitions of Machine Learning (ML) and more specifically the classification problem in data-streams context. As was discussed, ML involves the study and development of computational models and seeks to describe a target function in order to use some algorithms to compute it. The target function represents the knowledge that must be learned. The classification problem tries to infer this function mapping feature values into class labels. In traditional data mining, it is assumed datasets can be scanned many times, execution can go on with unlimited time and memory, and yet fairly accurate results have to be produced. Moreover, data-stream mining may produce approximate results and has to satisfy constraints, such as single-pass, real-time response, bounded memory, and concept-drift detection. Data classification is an important challenge in the streaming scenario. The way to address the classification problem depends on the data domain. As discussed in this chapter, there are many scenarios in which classification algorithms can be applied, such as Medical Disease Diagnosis, Customer Target Marketing, Social Network Analysis, and others. Classifiers used include statistical and probabilistic classifiers, decision tree and rule-based classifiers, regression methods, neural networks, support vector machines, and instance-based learning classifiers. Two examples of classifiers widely applied in data-stream mining are NB and HT. On the other hand, Ensemble methods emerged as a powerful technique that aims to obtain more robust results by combining the output from classifiers. An important question is how to do this. The combination strategies used by ensemble learning methods

roughly fall into two categories: Stacking and Voting. As results of these approaches, popular ensemble algorithms are implemented such as Boosting, Bagging, Model Averaging and Combination, Random Forests, Stacking. Other important issue in ML is the proper evaluation of the different models. Researchers agree that the evaluation of data-stream algorithms is a complex task. Thus, some of the most frequent approaches in the literature to address this challenge were also discussed in the present chapter, highlight the interplay between the accuracy and other factors, such as memory usage and run-time. Finally, it is very important the experimental estimation procedures. In order to do this, several strategies can be applied, such as holdout, bootstrapping, cross-validation and Prequential methodology. The last one is the most used on data-streams with stationary and non-stationary distributions, in this sense, the Prequential methodology was more widely addressed.

3 CONCEPT DRIFT & RELATED WORKS

In the previous chapter, the data-streams concept and some approaches over data-stream classification through the classifiers ensemble methods were discussed. However, when the data-stream is not stationary, a new challenge when carrying out the classification task emerge. The present chapter makes a study of the problem of the Concept Drift, associated definitions, types of Concept Drift and already existing approaches. Moreover, some of these approaches are discussed, as part of the related works used as a comparative basis during the empirical study.

3.1 INSIDE OF CONCEPT DRIFT

According to (WEBB et al., 2016) the classical definition of a concept is "a concept is a set of objects"; by extension two variants are possible: (i) a concept is a set of vectors of values such that any object with any vector of values in the set belongs to the concept. However this definition does not allow that different objects with identical attribute values might belong to different concepts. The other definition: (ii) concept is a set of object instances. This is coherent, but does not seem useful in the context of stream learning, because in many applications each object instance will appear only once. In order to deepen these definitions and adapt them for data-stream environments, this section will characterize the Concept Drift problem, main categorizations, and study approaches.

3.1.1 Concept Drift definitions

According to (NARASIMHAMURTHY; KUNCHEVA, 2007) the term "concept" is used to mean slightly different things. It is sometimes used to refer to the class of interest. Concept Drift or concept change, refers to a non-stationary learning problem over time. The training and the application data often mismatch in real life problems (HAND, 2006). A concept change in this case would mean a deviation of the description of the class from the original description.

In this work, the use of "concept" refer to the whole distribution of the problem at time moment. This concept can be characterized by the distribution D which represents the joint probability $P(X, y)$, where X denotes a random variable over vectors of attribute values $\mathbf{x}$ and y is a corresponding class label taken from a finite set Y of class labels. Hence, when referring to a particular distribution D_t at time t (i.e. a particular joint probability $P_t(X, y)$ at time t) we define it as a concept $D_t = \{P_t(X, y_1), P_t(X, y_2), ..., Pt(X, y_C)\}$. In this way, a change in the probability distribution function of the problem (also known as context (GAMA et al., 2004) brings about a change of concept. Thus, a Concept Drift

occurs when there is a change in the joint probability between two time points t_0 and t_1 (FRÍAS-BLANCO, 2014).

Formally, Concept Drift between time point t_0 and time point t_1 can be defined as $\exists X : P_{t_0}(X, y) \neq P_{t_1}(X, y)$ (GAMA et al., 2014)) where $P_{t_0}(X, y) = P_I$ (initial concept) and $P_{t_1}(X, y) = P_F$ (final concept)

Let us deeply understand the origin of Concept Drift. To estimate $P(X, y)$ at any time point, we consider the prior class probability $P(y)$ and the class-conditional probability $P(X|y)$ as follows: $P(X, y) = P(y)P(X|y)$. Then, to make classification decision for instance $\mathbf{z}$, the learner considers the maximal posterior probability $P(y|X)$ for a class y according to the Bayesian Decision (DUDA; STORK; HART, 2001): $P(y|X) = (P(y)P(X|y))/P(X)$ where $P(X) = \sum_{i=1}^{C} P(y)P(X|y)$ for all classes $y \in Y$.

Notice that $P(X)$ is the evidence factor which is considered as a scale factor that guarantees that the posterior probabilities sum is equal to 1. Thus, it is constant for all the classes y_i (KHAMASSI et al., 2018).

3.1.2 Categorization of Concept Drift

The type of changes can be roughly summarized as follows. These categories are by no means mutually exclusive and various combinations are possible (ZLIOBAITE, 2010).

They are briefly discussed in the following section.

3.1.2.1 Attending to the affected class

According to (GAMA et al., 2014), Concept Drift can occur with respect to any of the three major variables in in the context of Bayes theorem,

- the prior probabilities of classes $P(y)$ may change,
- the class conditional probabilities $P(X|y)$ may change, and as a result
- the posterior probabilities of classes $P(y|X)$ may change, affecting the prediction.

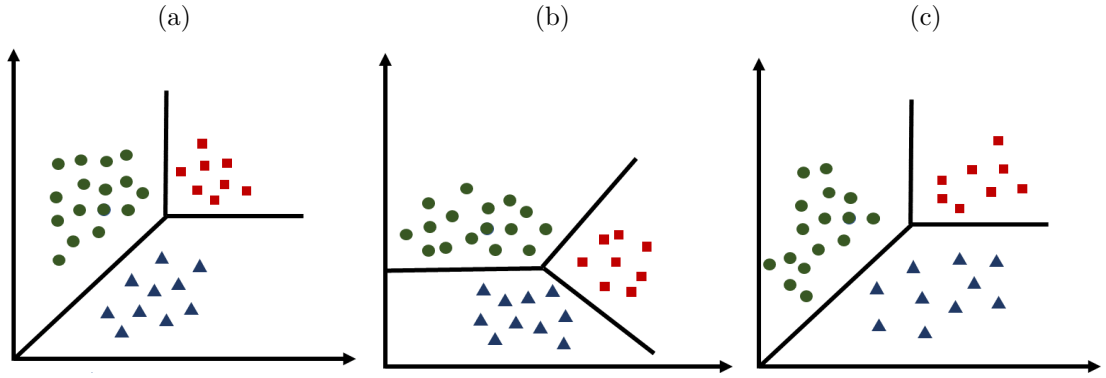
In (HOENS; POLIKAR; CHAWLA, 2012) was explained Real and Virtual Concept Drift like:

Virtual Concept Drift: In virtual Concept Drift, while the distribution of instances may change (corresponding to a change in the class priors or the distribution of the classes), the underlying concept (i.e., the posterior distribution) does not, as shown in Figure 1(c). It was defined more formally in (DITZLER et al., 2015) like the evidence or the marginal distribution of the data, $P(X)$ changes without affecting the posterior probability of classes $P(y|X)$. This may cause problems for the learner, as such changes in the probability distribution may change the error of the learned model, even if the concept did not change. Additionally, while previously portions of the target concept may

have gone unseen by the learner, due to a change in the distribution such instances may become more prevalent.

Real Concept Drift: Real Concept Drift is defined as a change in the class boundary (or, more formally, a change in the posterior distribution). (DITZLER et al., 2015) has defined it mathematically like the posterior probability $P(y|X)$ varies over time, independently from variations in the evidence $P(X)$. Figure 1(b) illustrates this definition.

Figure 1 – Types of Concept Drift. **(a)** Stationary data-stream, **(b)** Real Concept Drift, **(c)** Virtual Concept Drift.



Source: Adapted from Khamassi et al. (2018)

Class prior Concept Drift: Attending to the researches (KHAMASSI et al., 2015; KHAMASSI et al., 2018) other distinct drift type is considered when changes take place in the class prior probability $P(y)$. This kind of drift can exhibit class imbalance, novel class emergence or existing class fusion, and is named Class prior Concept Drift.

3.1.2.2 Attending to the speed of drift

If the period in which a change of concept occurs is considered, it can be classified as gradual or abrupt Concept Drift. According to (MINKU; WHITE; YAO, 2010; KHAMASSI et al., 2018), speed is the inverse of the drifting time. In the sense that a higher speed is related to a lower number of time steps and a lower speed is related to a higher number of time steps. Gradual drift: Gradual drift occurs when the drifting time is considered large. However, in fact there are two types being mixed under this term. The first type of gradual drift is referring to a period when two sources that provide the distributions D_0 and D_1 respectively, are active. As time passes, the probability of sampling from source D_0 decreases, probability of sampling from source D_1 increases. Note, that at the beginning of this gradual drift, before more instances are seen, an instance from the source D_1 might be easily mixed up with random noise. Another type of drift also referred as gradual includes more than two sources, however, the difference between the sources is very small, thus the drift is noticed only when looking at a longer time period (ZLIOBAITE, 2010). Figure

2(b) illustrates this definition, where P_I and P_F represent the initial and final concept respectively.

Abrupt drift: Abrupt drift, as know as sudden drift too, occurs when the new concept, rapidly, replaces the old one in short drifting time. This kind of drift immediately deteriorates the learner performance, as the new concept quickly substitutes the old one (KHAMASSI et al., 2018). If D_0 generates the original concept, D_1 generates the new concept, and t is a definite time period at which D_0 ceases to be used to generate the concepts; at this point D_1 is used instead. This is the simplest case of Concept Drift. Since sudden Concept Drift, is defined as having a sharp boundary between generating functions, it is often the easiest to detect, as future data no longer resembles the past data (HOENS; POLIKAR; CHAWLA, 2012). Figure 2(a) illustrates this definition.

3.1.2.3 Attending to the severity of drift

The criterion severity was introduced in (MINKU, 2010). Attending this work, drifts can be divided into Severe or Global and Intersected or Local.

Severe or Global drift: Drifts are severe when no example maintains its target class in the new concept or if the probabilities associated to the whole input space are modified.

Intersected or Local: If part of the input space has the same target class in the old and new concepts, or if part of the input space maintains the same probability, the drift is intersected. They suggested 2 different measures to characterize drifts according to class severity:

Percentage of the input space which has its target class changed after the drift is complete. Maximum percentage of the input space associated to a particular class in the old concept that has its target class changed in the new concept. Although class severity can reflect mainly changes in the prior and posterior probabilities, it does not reflect well changes in the unconditional and class-conditional pre-change probability distributions function (pdfs). That mean, the feature severity represents the amount of changes in the distribution of the input attributes. This criterion can be measured, by calculating the area between the curves of the old and new unconditional pdfs.

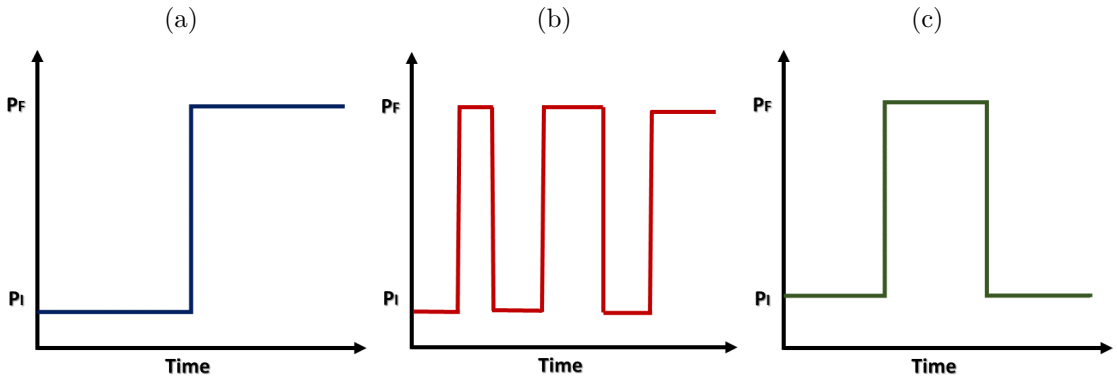
3.1.2.4 Attending to the recurrency of concept

Another big type of drift referred as reoccurring context. That is when previously active concept reappears after some time. Figure 2(c) represents this context where P_I and P_F represent the initial and final concept respectively. According to (ZLIOBAITE, 2010) it differs from common seasonality notion in a way that it is not certainly periodic, it is not clear when the source might reappear. According to (MINKU; WHITE; YAO, 2010) and (KHAMASSI et al., 2018) the recurrent drift can be divided into cyclic and acyclic.

Cyclic recurrent drift: The cyclic recurrent drift may occur according to a certain periodicity or due to a seasonable trend.

Acyclic recurrent drift: The acyclic recurrent drift may not be certainly periodic, i.e., it is not clear when the concept may reappear.

Figure 2 – Types of Concept Drift according to speed. **(a)** Abrupt Concept Drift, **(b)** Gradual Concept Drift, **(c)** Recurring Concept Drift.



Source: Adapted from Pérez (2018)

3.1.3 Popular approaches to handle Concept Drift

According to Khamassi et al. (2018), most of the approaches can be analyzed taking into account some dimensions like *how the data is processed*, *how is Concept Drift monitored*, *how the Concept Drift is handled* and *how the learning task is processed*. The section will introduce some approaches that meet these criteria.

3.1.3.1 Data processed

One possible categorization of this methods is like (GAMA et al., 2014) proposed. They categorized Concept Drift methods into three general groups, as follows:

Sequential Analysis based Methods: Sequentially evaluate prediction results as they become available, and alarm for drifts when a pre-defined threshold is met. Consider a sequence of examples. The null hypothesis H_0 is that X is generated from a given distribution D_0, and the alternative hypothesis H_1 is that X is generated from another (known) distribution D_1. The logarithm (log) of the likelihood ratio for the two distributions is calculated. Two thresholds, α and β are defined depending on the target error rates. If $\log < \alpha$, H_0 is accepted, else if $\log > \beta$, H_1 is accepted. In the case where $\alpha \leq \log \leq \beta$, the decision is postponed, the next example in the stream is added to the set, and is calculated and compared with the thresholds (FAITHFULL; RODRÍGUEZ; KUNCHEVA, 2019). This strategy is applicated for detecting significant change in the mean of input data. The Cumulative Sum (CUSUM), Page-Hinkley (PH) (PAGE, 1954) and Geometric Moving Average (GMA) (ROBERTS, 1959) are exemplars of this group.

Statistical based Approaches: Also know as Control charts, are a category of methods that are based upon statistical process control (SPC). Assume that we monitor classifica-

tion error. This error can be interpreted as a Bernoulli random variable with probability of "success" (where error occurs) p. The probability is unknown at the start of the monitoring, and is re-estimated with every new example as the proportion of errors encountered thus far. At example i, we have a binomial random variable with estimated probability p_i and standard deviation σ_i. The method follows a set of rules to place itself into one of three possible states: *in-control*, *warning*, and *out-of-control* (FAITHFULL; RODRÍGUEZ; KUNCHEVA, 2019). The Drift Detection Method (DDM) (GAMA et al., 2004), Early Drift Detection Method (EDDM) (BAENA-GARCIA et al., 2006), Exponentially Weighted Moving Average (EWMA) (ROSS et al., 2012), are members of this second group.

Windows based Methods: The methods in this category monitor the distributions of two windows of data. The basic construction involves a reference window composed of old data, and a detection window composed of new data. This can be achieved with a static reference window and a sliding detection window, or a sliding pair of windows over consecutive observations. The old and new windows can be compared with statistical tests, with the null hypothesis being that both windows are drawn from the same distribution (FAITHFULL; RODRÍGUEZ; KUNCHEVA, 2019). A significant difference between the distributions of these two windows suggests the occurrence of a drift. Kifer's (KIFER; BEN-DAVID; GEHRKE, 2004), Nishida's (NISHIDA; YAMAUCHI, 2007), Bach's (BACH; MALOOF, 2008), the Adaptive Windowing (ADWIN) (BIFET; GAVALDÀ, 2007), SeqDrift detector (SAKTHITHASAN; PEARS; KOH, 2013), (HDDM A-test and HDDM W-test) (FRÍAS-BLANCO et al., 2015), and Adaptive Cumulative Windows Model (ACWM) (SEBASTIÃO; GAMA; MENDONÇA, 2017) are examples of the successful application of this approach.

3.1.3.2 Monitoring process

These methods can also be classified according to whether the class tag is present or not, if the class tag is present, a supervised approach is followed. Depending on the percentage of data with tagged classes that the data-set has, an unsupervised or semi-supervised approach can be followed.

Methods based on supervised indicators: Methods based on supervised indicators are useful for detecting changes when the prediction feedback is immediately available. Very often, these methods focus on preserving the learner performance by handling Real Concept Drifts. The main key for handling this type of drift relies on monitoring the learner feedback. Some measures that are widely used for the feedback are Accuracy, Recall, Precision, Sensitivity. The works (GAMA; CASTILLO, 2006), (BIFET; GAVALDÀ, 2007), (BAENA-GARCIA et al., 2006), (KHAMASSI et al., 2015) used this approach.

Methods based on semi-supervised indicators: Methods based on semi-supervised indicators are useful for detecting Concept Drift in data-streams with scarcely labeled instances. They utilize a large number of unlabeled data with the limited amount of labeled data (KIM; PARK, 2017). The work (LUGHOFER et al., 2016), (KIM; PARK, 2017) are ex-

amples of works that used this kind of approach.

Methods based on unsupervised indicators: Methods based on unsupervised indicators are useful for detecting changes when the prediction feedback is delayed. These approaches must be robust in the face of unknown context (FAITHFULL; RODRÍGUEZ; KUNCHEVA, 2019). The work of (REIS et al., 2016; SONG et al., 2016) are representative of the use of this kind of approach.

3.1.3.3 Adapting process

Inside this category, there are two main approaches (GAMA et al., 2004): strategies that adapt the learning at regular intervals without considering that the change really occurred (implicit change detection), known as Blind methods or also Passive Approaches Adaptation algorithms; and strategies that first detect the Concept Drift, and then execute a learning process to adapt the model (explicit change detection), know as Informed methods or also Active Approaches Adaptation.

Blind methods or Passive Approaches Adaptation algorithms: Algorithms following the passive approaches continuously update the model every time new data are presented, without any strategy to explicitly detect Concept Drift. Typically it uses techniques as fixed-size sliding windows that take a window size W as a parameter and periodically retrain the model with the latest W examples. A special case of Passive Approaches is incremental and online learning where the model evolves with data. These strategies are proactive; they update the model based on the loss function, forget old concepts at a constant speed independently of whether changes are happening or not. When changes are happening, it may be more beneficial to discard old data faster, and to discard old data slower or not discard at all at times when changes are not happening. The main limitation of this methods is slow reaction to Concept Drift (GAMA et al., 2014). Ditzler et al. (2015) concluded that passive approaches have been shown to be quite effective in prediction settings with gradual drifts and recurring concepts. In addition, they are generally better suited for batch learning. Are member of this group the works (MUTHUKRISHNAN; BERG; WU, 2007), (PINTO; GAMA, 2007), (KRAWCZYK; WOŹNIAK; SCHAEFER, 2014), (VORBURGER; BERNSTEIN, 2006) and (KUNCHEVA, 2004).

Informed methods or Active Approaches Adaptation: Algorithms following the active approach specifically aim at detecting Concept Drift. This approach generally include methods that are more complex and two main techniques can be distinguished: instance selection and instance weighting. Active approaches work quite well in settings where the drift is abrupt and have been shown to work well in online settings as well. This method are reactive; their actions depend on whether a trigger has been flagged (BIFET; GAVALDA, 2006). Triggers can be either change detectors or specific data descriptors that we will see in the reoccurring concept management techniques (WIDMER; KUBAT, 1996). The reaction to a drift signal might apply to the model as a whole or might explore characteristics

of the language used to represent generalization of examples (GAMA et al., 2004). The researches (GAMA; CASTILLO, 2006), (BAENA-GARCIA et al., 2006), (KHAMASSI; SAYED-MOUCHAWEH, 2014), (GONÇALVES et al., 2014), (CIESLAK; CHAWLA, 2009), (LICHTEN-WALTER; LUSSIER; CHAWLA, 2010), (DITZLER; POLIKAR, 2011), (CAUWENBERGHS; POG-GIO, 2001) are used this approach.

Both active and passive approaches intend to provide an up-to-date model; however, the mechanisms used by each to do so are different. (DITZLER et al., 2015) emphasize that both approaches can be successful in practice; however, the reason for choosing one over the other is typically specific to the application. In fact, before choosing a specific algorithm for learning in a non-stationary environment, it is important to consider the dynamics of the learning scenario, computational resources available and any assumptions that can be made about the distributions of the data.

3.1.3.4 Learning process

It alludes to how the learning process is more efficient if using a single classifier or an ensemble of classifiers.

Single learner: In non-stationary environment, single adaptive (or dynamic) learning algorithms are widely used for handling concept changes. They can be considered as an extension of incremental algorithms especially when they can incorporate forgetting mechanisms in order to discard data from outdated concepts. Another possible extension of the incremental learning algorithms is to be self-adaptive. The objective is to monitor the learning process, adapt the model, and interpret the encountered changes in order to response to the new environment requirement. These approaches are not recommended for handling recurrent drift. An example of this work is the approach of (CAUWENBERGHS; POGGIO, 2001).

Ensemble learners: Ensemble learners have received great interest as they have shown better generalization abilities than single learner in the last years. The success of the ensemble methods for handling Concept Drift relies on two primordial points: diversity and adaptability. There are many related works, some of them are (POLIKAR, 2001), (KOLTER; MALOOF, 2007), (OZA; RUSSELL, 2001), (BIFET et al., 2010), (MINKU; WHITE; YAO, 2010), (WOŹNIAK; KRAWCZYK, 2012), (KUNCHEVA, 2004), (KOLTER; MALOOF, 2007), (SANTOS et al., 2014), (BARROS; SANTOS; GONÇALVES JR., 2016), (FRIAS-BLANCO et al., 2016).

3.2 RELATED WORKS

As was mentioned above, Gama et al. (2004) distinguishes two categories where strategies are located to face the problem of changing the concept: (i) strategies that adapt learning at regular time intervals without considering that a change in the concept has

occurred or not and (ii) strategies that first detect the change of concept and then the learning model is adapted to the change.

Ensemble system is usually included within the first strategy since they have mechanisms that allow them to evolve without having to directly detect points exchange. However, today there are several researches that insert mechanisms for direct detection of changes of concepts into the ensemble, which also includes them within the second strategy. The advantage of incorporating change detectors is to take advantage of the ability of ensemble system in order to adapt to gradual changes combined with the work of the change detector for abrupt changes (ORTIZ-DIAZ, 2014). In the next sections, some of the algorithms, used as a comparative basis are described, which covers ensemble methods and their respective detection methods, which are available in the Massive Online Analysis (MOA) (BIFET et al., 2010).

3.2.1 Related ensemble methods

In this section, a bibliographic study of the ensemble methods with or without detection mechanisms will be carried out aiming to highlight the methods related to our proposal in this dissertation. Later, Section 5.2 will provide a comparison between the methods proposed in this dissertation and some of the methods described below.

3.2.1.1 Fast Adaptive Stacking of Ensembles

Fast Adaptive Stacking of Ensembles (FASE) (FRIAS-BLANCO et al., 2016) is based on the online bagging algorithm (OZA; RUSSELL, 2001), and uses HDDM$_A$ as a drift detection mechanism to estimates the error. It has a set of adaptive learners to handle Concept Drift explicitly by detecting the changes and updating the model if a Concept Drift is estimated. The adaptive learners estimate error rates (by the corresponding change detectors) with a predictive sequential approach (test-then-train). FASE uses weighted voting to combine the predictions of the main and alternative models. It uses a meta-classifier too, combining the predictions of the adaptive learners. For that, it generates a training meta-instance $M = (\widehat{y}_1, \ldots, \widehat{y}_j, \ldots, \widehat{y}_k; y)$ where each $\widehat{y}_j$ is an attribute value and y is its corresponding class label. Each attribute value $\widehat{y}_j$ of the meta-instance M corresponds to the prediction from classifier h_j for the example $\mathbf{z}$. The class label of the meta-instance M is the same label of the original training example (FRIAS-BLANCO et al., 2016).

3.2.1.2 Boosting-like Online Learning Ensemble

Boosting-like Online Learning Ensemble (BOLE) (BARROS; SANTOS; GONÇALVES JR., 2016) is based on simple heuristic modifications to Adaptable Diversity-based Online Boosting (ADOB) (SANTOS et al., 2014). To obtain Boosting-like Online Learning Ensemble (BOLE), two modifications were made with respect to the original algorithm (i) weak-

ening the requirements to allow the experts to vote and (ii) changing the Concept Drift detection method internally used, improving the ensemble accuracy in most situations, especially when concept drifts are frequent and/or abrupt. In addition, BOLE delivers very strong performance in most situations, irrespective of the auxiliary drift detection method used (BARROS; SANTOS, 2019), however the authors considered the comparison of Concept Drift detectors (GONÇALVES et al., 2014) to concluded that Drift Detection Method (DDM) (GAMA et al., 2004) is the best default detector method *overall*. Thus, the variant used (*BOLE4*) to carry out the experiment, incorporates DDM as default detection mechanism.

3.2.1.3 Dynamic Weighted Majority

Dynamic Weighted Majority(DWM) (KOLTER; MALOOF, 2007) is an ensemble method to cope with Concept Drift based on Weighted Majority Algorithm (WMA) (BLUM, 1997), which aggregates and eliminates classifiers considering its local and global performance. It can be regarded as a general method since it can be used any online learning algorithm as a base learner.

Dynamic Weighted Majority (DWM) obtains a classification from each member of the ensemble. If one's prediction is incorrect, then DWM decreases the learner's weight by multiplying it by a parameter β. Regardless of the correctness of the prediction, DWM uses each learner's prediction and its weighth to compute a weighted sum for each class. The class with the greastest weight is set as the global prediction. Since DWM always decreases the weights of experts, it normalizes the weights by scaling them uniformly so that, after the transformation, the maximum weight is 1. This prevents newly added experts from dominating predictions. DWM also removes poorly performing experts by removing those with a weight less than the threshold θ, although it will not remove the last expert in the ensemble. If the global prediction is incorrect, DWM adds a new expert to the ensemble with a weight of 1. Finally, after using the new example to train each learner in the ensemble, DWM outputs the global prediction, which is the weighted vote of the expert predictions. In order to cope better with noisy or many examples, a parameter p defines the period over which DWM will not update learners' weights and will not remove or create experts. During this period, however, DWM still trains the learners.

3.2.1.4 Oza and Russell's Online Bagging(OzaBag)

Oza and Russell's Online Bagging (OzaBag) (OZA; RUSSELL, 2001) is a version online for Bagging ensemble learning methods. The traditional Bagging algorithms combine multiple learned base models with the aim of improving generalization performance. It works by resampling the original training set of size n to produce m bootstrap training sets of size n, each of which is used to train a base model. The online version trains m base models online. It simulates the bootstrap process by sending zK copies of each new

example to update each base model, where zK is a suitable Poisson random variable. This simple trick yields learning behavior similar to that of bath bagging. Online Bagging using ADWIN (BIFET et al., 2009) is implemented as ADWIN Bagging where the Bagging method is the online bagging method of Oza and Rusell with the addition of the ADWIN algorithm (BIFET; GAVALDÀ, 2007) as a change detector. The base classifiers used are Hoeffding Trees. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble. The variant used in this work, called Oza and Russell's Online Bagging with Detector (OzaBag_D) is an adaptation of ADWIN Bagging where the detection mechanism can be any, by default it is HDDM_A.

3.2.1.5 Diversity for Dealing with Drifts

Diversity for Dealing with Drifts (DDD) (MINKU; YAO, 2012) is also a variation of Online Bagging and uses four ensembles of classifiers with high and low diversity, before and after a Concept Drift is detected by a configurable auxiliary drift detector (default is Early Drift Detection Method (EDDM)). Prior to drift is detected, the learning system is composed of two ensembles: an ensemble with lower diversity (h_{nl}) and an ensemble with higher diversity (h_{nh}). Both ensembles are trained with incoming examples, but only the low diversity ensemble is used for system predictions. After a drift is detected, new low diversity and high diversity ensembles are created. The ensembles before the drift detection are kept and denominated old low and old high diversity. The old high diversity ensemble starts to learn with low diversity in order to improve its convergence to the new concept. Maintaining the old ensembles allows not only a better exploitation of diversity and the use of information learned from the old concept to aid the learning of the new concept, but also helps the approach to be robust to false alarms. Both the old and the new ensembles perform learning and the system predictions are determined by the weighted majority vote of the output of 1) the old high diversity, 2) the new low diversity, and 3) the old low diversity ensemble.

3.2.2 Related detection methods

The detection methods can indicate whether there has been a change of concepts or not, generally observing the number of errors sequentially committed by the base classifier (BRZEZINSKI; STEFANOWSKI, 2016). In general, a detector can be in one of these three states: in-control, warning, and out-of-control (drift), where drift represents that a change of concept occurred.

The following methods represent the detection mechanisms that integrate the ensemble algorithms used as a comparative basis to evaluate our proposals.

3.2.2.1 Drift Detection Method

Drift Detection Method (DDM) (GAMA et al., 2004) uses the prediction error of an incremental learning algorithm as a random variable corresponding to Bernoulli's experiments. A binomial distribution is assumed and it is considered that for a large number of instances this is close to the normal distribution, where the error in the prediction (p_i) and its standard deviation $s_i = \sqrt{p_i(1-p_i)/i}$ are calculated for each instance (i). It is established that the error of the learning algorithm (p_i) will decrease if the number of examples increases and the distribution of the examples remains stationary. On the other hand, a significant increase in the number of algorithm errors suggests that the class distribution is changing and therefore the current decision model is inappropriate.The method for detecting changes stores two variables in training the learning algorithm,(p_{min}) and (s_{min}), which are updated when a new instance causes $p_i + s_i < p_{min} + s_{min}$.

DDM presents three states:

- *in-control*: when $p_i + s_i < p_{min} + w \times s_{min}$, to be declared stable, the authors assume that the same distribution contains the generated examples where w represents an error tolerance factor to reach the warning level.

- *warning*: when $p_i + s_i \geq p_{min} + w \times s_{min}$, the level warns that the error is increasing but has not yet reached the level considered significantly high to declare the change.

- *drift*: the change is detected, since $p_i + s_i \geq p_{min} + d \times s_{min}$ is satisfied, so the values p_{min} e s_{min} are restarted. In this context, d represents an error tolerance factor to reach the drift level.

The DDM parameters by default in MOA for the warning and drift levels are: $w = 2.0$, $d = 3.0$ respectively, and $n = 30$, where n is the minimum number of instances before change detection is allowed. This detector has good detection efficiency in scenarios where abrupt and gradual changes are not very slow, but in scenarios where gradual changes are very slow, it presents some difficulties (BAENA-GARCIA et al., 2006).

3.2.2.2 Early Drift Detection Method

Early Drift Detection Method (EDDM) (BAENA-GARCIA et al., 2006) was constructed with the purpose of improving the difficulties in detecting gradual changes while maintaining the same results as DDM in detecting abrupt changes. Basically this detector considers the distance between two classification errors instead of exclusively considering the number of errors as done by DDM.

Therefore, the detector assumes that as long as the learned distribution remains stationary or while the learning method is learning, it will improve the predictions and the distance between two errors will increase. The method for each instance (i) calculates the

mean of the distances between two errors ($p_i^{'}$) and its standard deviation ($s_i^{'}$). When $p_i^{'}$ and $s_i^{'}$ reach their maximum values ($p_i^{'} + 2 \times s_i^{'}$ reaches its maximum value), they will be stored in $p_{max}^{'}$ and $s_{max}^{'}$, being overwritten each time a new maximum is found. Two thresholds are defined:

- *warning* is reported when $\frac{p_i^{'}+2\times s_i^{'}}{p_{max}^{'}+2\times s_{max}^{'}} < w$, beyond this level, the examples are stored in advance of a possible change of context.
- *Concept Drift* is reached when $\frac{p_i^{'}+2\times s_i^{'}}{p_{max}^{'}+2\times s_{max}^{'}} < d$, being the values $p_{max}^{'}$ and $s_{max}^{'}$ restarted. The model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered.

It is necessary that $w > d$ for the creation of previous states. As default values, the authors leave fixed 0,95 e 0,90 to alert and change states, respectively.

3.2.2.3 Hoeffding-based Drift Detection Methods

Hoeffding-based Drift Detection Methods (HDDM) authors (FRÍAS-BLANCO et al., 2015) propose to monitor the performance of the base learner by applying "some probability inequalities that assume only independent, univariate and bounded random variables to obtain theoretical guarantees for the detection of such distributional changes". HDDM$_A$ "involves moving averages and is more suitable to detect abrupt changes" and the second Hoeffding-based Drift Detection Method$_{W-test}$ (HDDM$_W$) "follows a widespread intuitive idea to deal with gradual changes using weighted moving averages". For both cases, the Hoeffding inequality (HOEFFDING, 1963) is used to set an upper bound to the level of difference between averages. They have three common parameters, the confidence values for drifts ($\alpha_D = 0.001$) and warnings ($\alpha_W = 0.005$), and the direction of the error, which can be one-sided ($t = 0$, only increments), default for HDDM$_W$, or two-sided ($t = 1$, error increments and decrements), default for HDDM$_A$. Finally, HDDM$_W$ has an extra parameter (λ =0.05) which is used to control how much weight is given to more recent data in comparison to older data.

3.3 FINAL CONSIDERATIONS

In this chapter, were discussed the Concept Drift problem. Concept Drift (CD) or Concept Change refers to a non-stationary learning problem over time. There are diverse categories of changes. Attending to the affected class, Virtual and Real CD can be distinguished. If the period in which a change of concept occurs is considered, it can be classified as Gradual or Abrupt CD. Recurrency of the concept is when previously active concept reappears after some time. Attending to the severity of drift the change can be Global or Local. Whatever the nature of CD, it may cause problems for the learner even if the class label did not change. Normally, if the change is abrupt, the learner performs is

more strongly affected. Popular approaches to handle CD were discussed considering how the data is processed (Sequential analysis based methods, Statistical based approaches, Windows-based methods), how is CD monitored (Methods based on Supervised, Semi-supervised and Unsupervised indicators), how the CD is adapted (Blind and Informed methods), and how the learning task is processed (Using a single classifier or an ensemble of classifiers). The related works presented the benchmarks that have been taken as a comparative basis in order to perform the experiments. They were separated take into consideration: (i) the strategies that adapt learning at regular time intervals without considering a change in the concept has occurred, such as DWM and (ii) strategies that first detect the change and then the learning model is adapted, such as DDM, EDDM and HDDM$_A$. Ensemble systems are usually included within the first strategy since they have mechanisms that allow them to evolve without having to directly detect points of change. However, there are several researches that insert mechanisms for direct detect CD, then these particular systems are include within the second strategy such as FASE, BOLE, OzaBag$_D$ and Diversity for Dealing with Drifts (DDD).

4 PROPOSED METHODS

This chapter introduces two families of classifier ensemble methods derived from FASE: FASEO and FASEB. Algorithm families are originated using different change adaptation strategies and methods to combine the predictions of the classifiers that make up the ensemble. An overview of the different proposed methods is described below, followed by a detailed description of the strategies used. In particular, it is explained the general algorithm for updating the ensemble methods step by step.

4.1 OVERVIEW OF THE METHODS

According to (ORTIZ-DIAZ, 2014), when designing ensemble of classifiers two main points must be considered: (i) how the base classifiers in the ensemble are updated and (ii) how they are combined to make a joint prediction.

Taking into account above assumption, variations were introduced in FASE to search for a better balance accuracy and necessary resources (memory and run-time) for its operation. Given FASE can be viewed as a three-level ensemble of classifiers (FRIAS-BLANCO et al., 2016), we argue that the complexity introduced by its structure is the main reason to influence its high demand of memory and run-time.

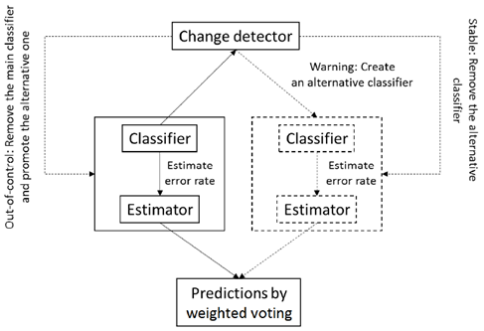
The original algorithm is composed of a set of adaptive classifiers (see Figure 3(a) (FRIAS-BLANCO et al., 2016)). Each one is formed by a base classifier and an alternative classifier (both classifiers include a drift detection mechanism) that is generated each time the base classifier issues a *warning* state. Both classifiers, the main and the alternative, process each instance and by weighted voting determines its class. This strategy is followed in the adaptive classifiers that form the ensemble, but also in the level of the meta-classifier (see Figure 3(b) (FRIAS-BLANCO et al., 2016)) that receives as input the predictions of each classifier in the form of a meta-instance.

Considering this scenario, our proposals aim to handle resources more efficiently while keeping accuracy at similar levels. Thus, the main proposed modifications made on FASE are (i) the update strategy and (ii) the voting procedure of the ensemble. As a result of these modifications, two families of ensemble algortithms derived from FASE were devised: FASEO and FASEB.

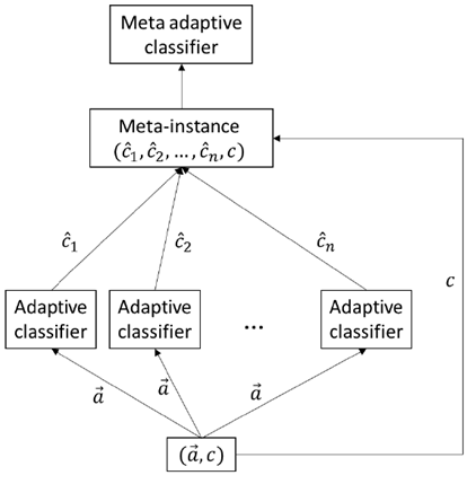
Differently, FASE variants eliminate the use of adaptive classifiers and create, in the structure of the general model, a parallel ensemble where a classifier is activated and begins to train once one of the classifiers of the main ensemble reaches the *warning* level. When a Concept Drift is detected, then one of two variants is followed, (i) the first classifier, the one that stayed longer (the oldest) in the alternative classifier ensemble, is promoted or (ii) the classifier with the best accuracy of the alternative classifiers is promoted. Such

Figure 3 – FASE s´ details of the structure and performance.

(a) Adaptative Classifier



(b) Meta-Adaptative-Classifier



Source: Frias-Blanco et al. (2016)

strategies promote the creation of FASEO and FASEB families, respectively.

Each family of algorithms is based on classifiers that integrate detection mechanisms. So, the associated detector triggers each of the three different drift signals manipulated in the model. The first two methods, (FASEO and FASEB), maintained the meta-classifier proposed in FASE in order to perform class voting while the others combine weighted voting for final decision. To determine the whole weight of each classifier, accuracy, entropy degree and class probabilities are combined in different ways.

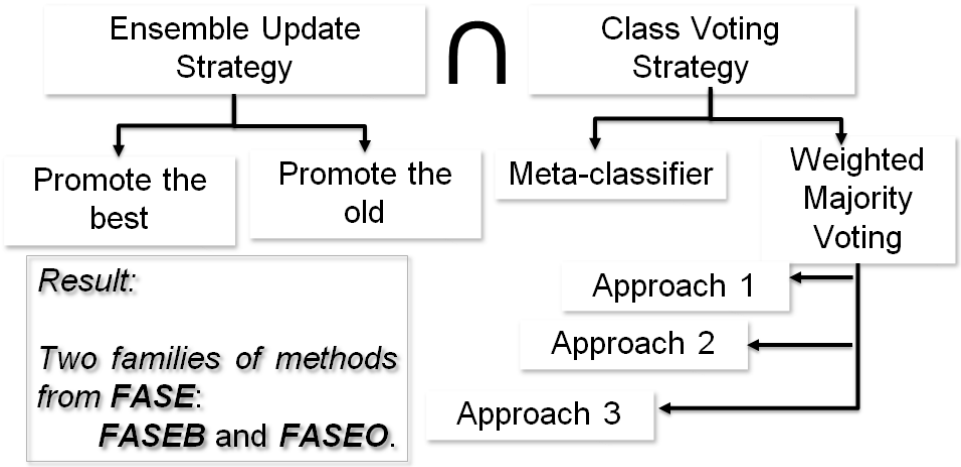
In this work are considered two update strategies. The first, updates the ensemble with the alternative classifier that has greater accuracy, the second updates the ensemble with the oldest alternative classifier. Moreover, it is also considered two class voting strategies. The first provides a combined voting using a meta-classifier. The second gives a combined voting using weighted majority voting in different ways.

Table 1 – All proposed methods.

ENSEMBLE UPDATE STRATEGY	**CLASS VOTING STRATEGY**			
	Meta-classifier	***Weighted Majority Voting***		
		Approach 1	**Approach 2**	**Approach 3**
Promote the best	**FASEB**	**FASEB$_{wv1}$**	**FASEB$_{wv2}$**	**FASEB$_{wv3}$**
Promote the old	**FASEO**	**FASEO$_{wv1}$**	**FASEO$_{wv2}$**	**FASEO$_{wv3}$**

Source: Own elaboration (2019)

Figure 4 – Combining strategies.



Source: Own elaboration (2019)

The update strategies together the class voting strategies, give rise 8 methods, which are: FASEO, FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB, FASEB$_{wv1}$, FASEB$_{wv2}$ and FASEB$_{wv3}$. Figure 4 and Table 1 summarize this explanation.

The description of each algorithm follows below:

- FASEO: To update the main ensemble, the classifier with more training time in the set of alternative classifiers is promoted. To determine the final class, a meta-classifier is used with its inputs being the meta-instances formed by the predictions of each classifier in the main ensemble.

- FASEO$_{wv1}$: As in FASEO, to update the main ensemble, the oldest classifier in the set of alternative classifiers is promoted. To vote the final class, the weight of each classifier is computed taking into account accuracy, entropy degree, and the class probability vector.

- FASEO$_{wv2}$: As in FASEO and FASEO$_{wv1}$, to update the main ensemble the classifier with more training time in the set of alternative classifiers is promoted. To vote the final class, the weight of each classifier is computed taking into account only accuracy and the class probability vector.

- FASEO$_{wv3}$: As in the three former cases, to update the main ensemble, the classifier with more training time in the set of alternative classifiers is promoted. To vote the final class, the weight of each classifier is computed taking into account only accuracy and entropy degree.

- FASEB: To update the main ensemble, the classifier with the best accuracy among the alternative classifiers is promoted. Decision on the final class is given by a meta-

classifier, whose inputs are the meta-instances formed by the predictions from each classifier in the main ensemble.

- FASEB$_{wv1}$: As in FASEB, to update the main ensemble, the classifier with the best accuracy among the alternative classifiers is promoted. To vote the final class, the weight of each classifier is computed taking into account accuracy, entropy degree, and the class probability vector.

- FASEB$_{wv2}$: As in FASEB and FASEB$_{wv1}$, to update the main ensemble, the classifier with the best accuracy among the alternative classifiers is promoted. To vote the final class, the weight of each classifier is computed taking into account only accuracy and the class probability vector.

- FASEB$_{wv3}$: As in the three last cases, to update the main ensemble, the classifier with the best accuracy is promoted. To vote the final class, the weight of each classifier is computed taking into account only accuracy and entropy degree.

The proposed algorithms were implemented in Java using IDE Eclipse 4.6.0 (<https://www.eclipse.org/>). The main benefits of Java, are portability, where applications can be run on any platform with an appropriate Java Virtual Machine (JVM), and the strong and well-developed support libraries. Use of the language is widespread, and features such as automatic garbage collection help to reduce programmer burden and error. Also, this guarantees the ease of integration into MOA.

As a summary, all proposed methods include base learner models (classifiers + concept drift detectors), that trigger *warnings* and *drifts* when changes in the concept have likely occurred and have been detected, respectively. Besides they have a parallel ensemble formed by alternative learners, activated once any learner models of the main ensemble is in the *warning* level. Otherwise, the alternative classifiers remain inactive. As a consequence, when a classifier of the main ensemble detects a concept drift without previous *warning* (abrupt changes), it is immediately replaced by an already trained classifier. In certain situations, this should contribute to time optimization.

4.2 THE UPDATE STRATEGY

This section provides a detailed description of update strategy used in the FASEO and FASEB families. In a general way, the proposed ensemble algorithms are updated once one of the learners (classifier with change detection mechanism) that compose the main ensemble, experiment any of the following change of states:

(i) A classifier initially *in-control*, triggered a *warning* (through its detection mechanism)

(ii) A classifier suddenly reaches the *drift* level from *in-control* state

(iii) A classifier reaches the *drift* level from state of *warning*

(iv) A classifier, currently in *warning*, return to the (by-default state) *in-control*

In (i), an alternative classifier is activated and placed in a parallel set (ensemble of alternative classifiers). When no *drift* is detected, the learning process is carried out by the learners of the main set.

When one of the classifiers of the main set reached an out-of-control signal (drift), (case (ii) or (iii)) the main set is updated, firstly the drifted classifier is deleted and then is promoted to the main ensemble a) the alternative classifier with the greatest accuracy (FASEB methods) or b) the oldest alternative classifier (FASEO methods). Once the alternative classifier is promoted, it is deleted from the parallel ensemble. Then, in (ii), the model activates a new alternative classifier, since, due to a sudden change, it was not created and, therefore, it was taken "borrowed" from a classifier that triggered warning, therefore, should be "returned to it".

In order to handle the final states there is an arrangement of states: initially, it is assumed that each value corresponding to the status of each classifier that is part of the main ensemble is *in-control* and therefore takes value 0; whenever a classifier of the ensemble enters *warning*, its status has value 1. When a classifier of the main ensemble detects a drift, either by going from *in-control* to *drift* or from *warning* to *drift*, the algorithm quickly updates the ensemble and the state is again *in-control*. Therefore, once the algorithm removes the classifier from the array of alternatives, it updates the state corresponding to the new classifier that became part of the main ensemble to *in-control*.

In the following, we present the general update strategy step by step. The proposed methods (see algorithm 1) receive as input the size of the main ensemble and training instances.

Lines 1 to 3 initialize the size of the active-classifier parallel ensemble, the main ensemble (with a learner and an associated detector in each position), and the state of the classifier. In line 4 the data-stream is read. Lines 5 to 37 represent the processing cycle of the ensemble. Lines 6 and 9 update the state of the classifiers of the ensemble before and after processing an instance, respectively. In line 7 the Poisson function is applied to the distribution in order to create diversity in the training instances. Line 8 represents the training of the main ensemble using the weight for training with the examples. Lines 10 to 36 check the four cases that may happen during the processing of instances of the data-stream.

The first case (lines 10 to 13) regards a stable (*in-control*) classifier that changes to the warning level. If this is the case, the status is updated and one classifier is activated in the ensemble of alternative classifiers.

The second case (lines 14 to 22) refers to stable classifiers that change to a drift state. Firstly, the existence of active alternative classifiers is verified: if available, the best one

is promoted and the ensemble of alternative classifiers is updated. When the drift signal is triggered without warning, it is necessary to activate an alternative classifier (line 18). When no alternative classifier is active, the classifier in the main ensemble that reached drift is restarted (line 21). The function in lines 16 and 25 promotes one alternative classifier to the main set. In Case 3 (lines 23 to 28), the warning level is followed by a drift signal. In this case, at least one active alternative classifier exists and the best one is promoted. The last condition (lines 29 to 33) captures the case of a classifier that returns to the *in-control* state after a *warning*. The number of active alternative classifiers is decremented (line 31) and the state is updated (line 32).

Algoritmo 1: General algorithm that represents the update strategy.

```
   Input: main ensemble size k, training instances z_i.
 1 nClassWarn ← 0
 2 InitializeMainEnsemble (OnlineBaseLearningAlg, d_j)
 3 stateNow ← in − control
 4 Next z_i
 5 for j ← 1 to k do
 6     stateBefore ← stateNow
 7     zk_j ← Poisson (z_i)
 8     TrainingMainEnsemble (zk_j)
 9     stateNow ← State(d(zk_j))
10     if stateBefore = in − control & stateNow = warning then
11         UpdateState (stateNow)
12         ++nClassWarn
13     else
14         if stateBefore = in − control & stateNow = drift then
15             if (nClassWarn > 0) then
16                 PromoteAlternativeClassifier(nClassWarn, j)
17                 UpdateAlternativeEnsemble(−−nClassWarn)
18                 ++nClassWarn
19                 UpdateState (stateNow)
20             else
21                 main-ensemble[j].ResetLearning()
22         else
23             if stateBefore = warning &
24                     stateNow = drift then
25                 PromoteAlternativeClassifier(·)
26                 UpdateState (stateNow);
27                 UpdateAlternativeEnsemble(−−nClassWarn)
28             else
29                 if stateBefore = warning &
30                         stateNow = in − control then
31                     UpdateAlternativeEnsemble (−−nClassWarn)
32                     UpdateState (stateNow)
```

The implementation of the FASEO and FASEB families methods has as a main difference the *PromoteAlternativeClassifier(·)* function. The former function in FASEO methods only receives as a parameter the number of classifiers that must be active in the parallel set (nClassWan), then the classifiers in the parallel set are reorganized until the

first position is occupied. So this function guarantees that always the first value of the parallel set is the learner that has more training time (the oldest classifier of the set). If the concept change occurs, the first learner of the parallel set is taken to replace the drifting learner in the main ensemble and then it is deleted from the parallel set.

In FASEB methods the *PromoteAlternativeClassifier(·)* function receives as a parameter the index of the classifier in the main ensemble that changed to drift (j) and also the number of active alternative classifiers (nClassWan). The *PromoteAlternativeClassifier (nClassWan, j)* function search in the parallel set, among the learners that are active until the one with the most accuracy (the best) is found; once this happens, it replaces the learner who reached the drift level in the main ensemble and later it is deleted from the parallel set. The number of active classifiers is also updated. If there are two classifiers with the same accuracy the most recent of them will be promoted. In the exceptional case that none of the active classifiers in the alternative ensemble had an accuracy greater than 0, then the last classifier of the ensemble (the most recent of all) will be promoted. This could positively influence on the adapt-to-the change process.

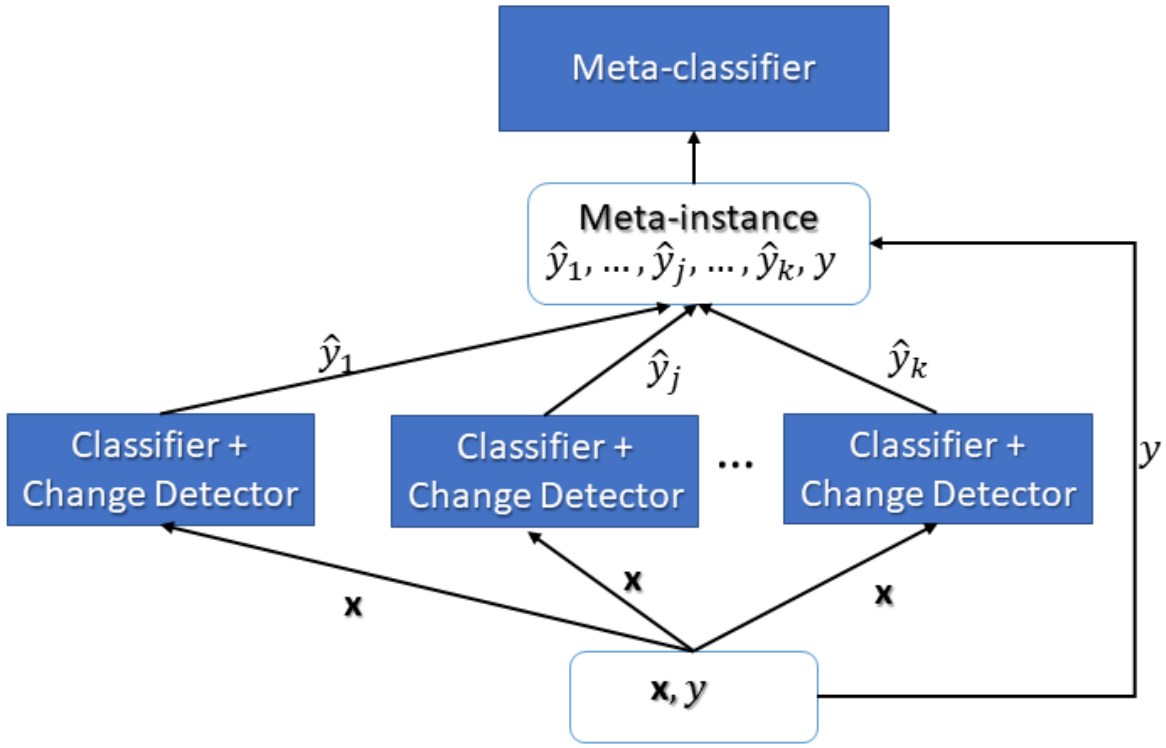
4.3 CLASS VOTING STRATEGIES

As previously explained, an ensemble of classifiers $H(\mathbf{x})$ are models learned from a set of classifiers $h_1(\mathbf{x}), ..., h_j(\mathbf{x}), ..., h_k(\mathbf{x})$. During the training process, it receives as an input a labeled instance $(\mathbf{x}_i, y_i)$, and the model $H(\mathbf{x})$ aims to predict the class $\widehat{y}_i$ of each instance in unlabeled data-set. To achieve this task, there is different ways of ensemble combination methods like stacking, voting schemes, unweighted voting schemes. Thus, in this work the classification is done using the stacking and the weighted-voting strategies, the particular way in which these strategies are applied is explained below.

4.3.1 Meta-classifier strategy

As was exposed before, the two first methods represent and have the same name that the algorithm families proposed: FASEO and FASEB, both use a meta-learning as a class voting mechanism like FASE. Meta-learning is the process of learning from ensemble members. Thus, the input of the meta-learner are the outputs of the ensemble-member classifiers. The goal of meta-learning process is to train a meta-classifier (meta-learner), which will combine the ensemble members' predictions into a single prediction. In this process, both the ensemble members and the meta-classifier need to be trained. The meta-classifier is the ensemble's combiner (MENAHEM; ROKACH; ELOVICI, 2013), thus, it is responsible for producing the final prediction. Similar to the ensemble-members, the meta-classifier of these methods is a one class classifier; it learns a classification model from meta-instances, whose attributes are nominal. As FASE, both methods uses a Prequential methodology to generate meta-instances. Thus, for each original training instance $\mathbf{z} =$

Figure 5 – Meta-learning strategy.



Source: Own elaboration (2019)

$(\mathbf{x}, y)$ it is generated a training meta-instance $M = (\hat{y}_1, \ldots, \hat{y}_j, \ldots, \hat{y}_k; y)$, where each attribute value $\hat{y}_j$ of the meta-instance M corresponds to the prediction from the base classifier j in the main ensemble for the original example $\mathbf{z}$ (see Figure 5). The class label of the meta-instance M is the same label of the original training example. The main set of base classifiers can change over time, since these classifiers can be substituted in response to Concept Drift. Thus, the meta-classifier can be affected by changes in the target concept that relates the predictions of the base classifiers with the true class label of a given instance (FRIAS-BLANCO et al., 2016). Similar to FASE, these two methods use the same kind of learner in the meta-classifier and in ensemble members. The ensemble members are classifier with change detector. During the ensemble's training, the ensemble-members, as part of a test-then-train process, are generating the meta-classifier's input.

4.3.2 Weighted voting strategies

A weighted voting is a system in which not all learners have the same amount of influence over the outcome because their votes have a different weight. In the classification task, an ensemble classifier can combines the decision of a set of classifiers by weighted voting to classify unknown examples. The weighting methods are best suited for problems where individual classifiers perform the same task (SHEN et al., 2018). Therefore, that

is the reason why in this work was adopted the weighted majority vote to obtain the final prediction of the class label combining hard and soft voting in different ways. In hard voting, is predicted the final class as the class label that has been predicted most frequently by the classification models. In soft voting, is predicted the class labels based on the predicted probabilities. In the proposed methods the vector of class probabilities stores in "soft" or "hard" way the predicted vote that each classifier of the main ensemble does by determining the class label for each instance in the data-stream.

As seen in the chapter 2, the weighted majority voting can be computed as in Equation (2.12): $H(\mathbf{x}) = \underset{1 \leq c \leq C}{\operatorname{argmax}}(w_j LC_c(\mathbf{x}))$. Then, if $LC_c(\mathbf{x}) = \sum_{j=1}^{k} P_j(y_c|\mathbf{x})$ the weighted majority voting can be computed as:

$$H(\mathbf{x}) = \underset{1 \leq c \leq C}{\operatorname{argmax}} \left\{ \sum_{j=1}^{k} w_j P_j(y_c|\mathbf{x}) \right\} \tag{4.1}$$

where w_j is the weight with which the classifier h_j predicts the class y_c.

In order to improve the performance of each classifier, the historical information of the training data and the recent information of the test data are combined. A similar idea was previously proposed in (SONG et al., 2016). To determine the weight of each classifier, the accuracy (a component of the weight derived from historical performance) and the degree of entropy in its classification (the component of the weight coming from the current behavior of the classifier) are taken into account. The weighted voting components are:

- Accuracy:

  To compute the weight component from historical information is adopted the simple weighted voting model where the weight values w_{acc_j} for each classifier are determined based on the predictive performance of a particular classifier with the training set using Equation.2.8.

  In this equation is considered the prequential accuracy that represents the average accuracy obtained by the prediction of each example before it is learned, calculated online. The computation of the prequential accuracy at timestamp t is 1 if the current example is correctly classified and 0 otherwise (HIDALGO; MACIEL; BARROS, 2019).

- Entropy:

  In Information Theory, entropy is defined as a way of measuring the average degree of uncertainty regarding information sources, which consequently allows the quantification of the present information flowing in the system. In simple terms, the concept of entropy is associated with the idea that, the more uncertain the result of a random experiment, the more information is obtained by observing its occurrence. In this context, a high degree of uncertainty is understood as bad performance, being unable to discern between classes, and thus its corresponding weight will be low.

The uncertainty is calculated as follows (HOLUB; PERONA; BURL, 2008):

$$E = -\sum_{c=1}^{C} P_j(y_c|\mathbf{x}) log_2 P_j(y_c|\mathbf{x}) \tag{4.2}$$

where $P_j(y_c|\mathbf{x})$ is the posterior probability of the instance $\mathbf{z}$ being a member of the c_{th} class, which takes into account all possible labels.

The value that is used in the weighting of voting by entropy is given by $w_E = 1 - E$ and, thus, when E increases, the value of w_E decreases.

These component are combined in different ways originating three weighting approaches:

- Approach 1: it is used the classifier accuracy weight, class probability vector and entropy weight. In this approach the vector of class probabilities stores in "soft" way the predicted vote. Besides, w_j is computed as: $w_j = w_E * w_{acc_j}$ where w_j is the weight with which the classifier j predicts the class y_i.
- Approach 2: it is used the classifier accuracy weight and class probability vector. In this approach the vector of class probabilities also stores in "soft" way the predicted vote. Besides, w_j is computed as: $w_j = w_{acc_j}$
- Approach 3: it is used the classifier accuracy weight and entropy weight. In this approach the vector of class probabilities stores in "hard" way the predicted vote. Besides, w_j is computed as: $w_j = w_E * w_{acc_j}$

4.4 FINAL CONSIDERATIONS

The chapter provided the description and implementation of the strategies that give rise to the FASEO and FASEB families derived from FASE. The first family boosting to the main ensemble the oldest alternative classifier. The FASEB family promote the classifier with the best accuracy result out of the alternative classifiers. The first two methods of these families, also named respectively FASEO and FASEB, at the same time that modify the update procedure as was described before, they maintain the voting system (meta-learner) from the original ensemble algorithm. That is, combining the prediction of the main models in a meta-instance in order to train the meta-classifier. The main different regarding FASE is the base-learner structures. FASE have an adaptive classifier in all their structures, instead the new proposals have a classifier with detection mechanism, reducing its complexity. The other 6 methods FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB$_{wv1}$, FASEB$_{wv2}$ and FASEB$_{wv3}$ adopt an approach based on weighted majority voting to predict the class label of each instance, and also modify the update procedure and the base-learner structures reducing its complexity. The resulting weight of each main classifier is computed by the combined weight between the entropy degree and accuracy in

3 different ways. The first approach considers the historical performance of the classifier (accuracy weight), the actual performance of the classifier (entropy weight) and uses soft voting in order to predict the final class label. The second approach only considers the historical performance of the classifier and also uses soft voting in order to predict the final class label. Finally, the third approach also considers the historical performance of the classifier, the actual performance of the classifier but uses hard voting in order to predict the final class label.

5 THEORETICAL AND EMPIRICAL STUDY

This chapter presents a theoretical and empirical study of the proposed methods in comparison to previous works. The former aims to analyze the temporal and spatial complexities of the methods with respect to the original FASE algorithm. The latter examines the experimental performance of those algorithms with respect to other references in the area. The experimental environment, configurations and data-set used to test the algorithms are described. Finally, the empirical results are discussed and statistically analyzed.

5.1 THEORETICAL ANALYSIS

In order to compare FASE algorithm with their derived methods, FASEB and FASEO families, it is necessary to perform an analysis of spatial and time complexity of all these algorithms. To do this, it is considered both the complexity of the base learners that make up its structure as well as the structures of the algorithms themselves.

5.1.1 Space complexity

If NB (JOHN; LANGLEY, 1995) is used as base learner, the space complexity analysis appear in the literature as $O(dvc)$ (ZHENG; WEBB, 2005). In the case of HT (HULTEN; SPENCER; DOMINGOS, 2001), the worst case space complexity is given in terms of the current number of leaves l as $O(ldvc)$ (MANAPRAGADA; WEBB; SALEHI, 2018). For both cases, d is the number of attributes, with v values per attribute and c is the number of classes per instance.

On the other hand, the space complexity also depends on the number of classifiers in the ensembles (k). Particularly for FASE, when all adaptive classifiers in the ensemble are in the warning level, the dimension of these structures is $2k$, in the worst case. This happens because in the warning state each base structure of the ensemble is composed of two basic models, the main and the alternative one. Thus, the final space complexity for FASE is approximately $O(Nldvc)$ using HT as base learner and $O(Ndvc)$ using NB where $N = 2k$.

On the other hand, both FASEB and FASEO families have the same space complexity that FASE in the worst case, because these methods has two base arrays with a maximum of k elements completely full. When the methods use HT as base learner, the complexity by each array is closely to $O(kldvc)$ and $O(kdvc)$ if NB is used.

5.1.2 Time Complexity

In this case, we measure the run-time for each processed instance, since the methods are designed for continuous execution in data-stream. Therefore, to perform this analysis,

three different situations are taken into account:

- *Create/Activate a base learner:* FASE and the derived families algorithms FASEB and FASEO, have similar performance in this process where the warning alarm must have been triggered before. Theoretical time complexity for HDDM$_A$ (FRÍAS-BLANCO et al., 2015) is $O(1)$. Thus, the temporal complexity depends on the temporal complexity of the selected base classifier (NB or HT) to process each example.

  Theoretically the time complexity for NB is $O(nd)$ where n is the number of training examples and d is the number of attributes of each example (ZHENG; WEBB, 2005): for one example, the complexity is $O(d)$ because $n = 1$. On the other hand, there are two primary operations in the learning using HT: (i) incorporating a training example by incrementing leaf statistics and (ii) evaluating potential splits at the leaf reached by an example (MANAPRAGADA; WEBB; SALEHI, 2018). The final time complexity using HT for the algorithms is $O(ndGc)$ (MANAPRAGADA; WEBB; SALEHI, 2018) where G is the computation of information gains that requires $O(1)$ arithmetic operations and c classes. For one example, it is $O(dGc)$.

- *Update the base learners*: In the update process, the drift alarm must have been triggered by HDDM$_A$ and the theoretical time complexity for it is $O(1)$, as before. In FASE the temporal complexity is $O(kd)$. FASEB family has a function in order to find the classifier with a better accuracy in the alternative ensemble. This function also has order $O(k)$. Then, the temporal complexity is $O(k) + O(kd)$. The temporal complexity of FASEO family is $O(kd)$, where k is the number of base learners.

- *Update the meta-classifier*: In this process, the drift alarm must also have been triggered by HDDM$_A$ and the theoretical time complexity for it is $O(1)$, as above. The meta-learner has a similar structure of a base classifier but it is trained with meta-instances. The temporal complexity depends on the temporal complexity of the selected base learner (NB or HT) to process each meta-instance.

  The time complexity for NB is $O(nk)$ where n is the number of examples and k is the number of predictions made by the ensemble for each example, i.e., the meta-instances have as many attributes as base learners on the set. For a single example, it is $O(k)$ because $n = 1$.

  The final time complexity using HT for the meta-classifier is $O(nkGc)$. For a single example, it is $O(kGc)$. FASEB$_{wv1}$, FASEB$_{wv2}$, FASEB$_{wv3}$, FASEO$_{wv1}$, FASEO$_{wv2}$ and FASEO$_{wv3}$ have no meta-classifier as the voting process of the class is done by weighted voting. This process is order $O(k)$.

5.2 EMPIRICAL ANALYSIS

The section presents the elements taken into account to carry out the experimentation. The environment in which the experiments were made is described, reporting the characteristics of the server employed , the frameworks and the tools used in the development and evaluation of the methods. The synthetic and real data-sets are also described and the configuration with which each method was evaluated. Finally, the results obtained are presented and discussed and an statistical analysis of results is described.

5.2.1 Environment description

In order to evaluate all methods, experimentation was performed on an AMD Fesktop Server with RYZEN 5 1600 COOLER processor, and 16 GB dual-channel DDR4 with 2400 Mhz of RAM, a SSD KINSTON 2,5 480 GB A400 SATA III of hard disk, with the Ubuntu operating system 18.04.2 64 bits.

In addition, a framework for dealing with data-streams mining MOA (BIFET et al., 2010) was installed. MOA is used in order to compare the proposed methods against some of the previous algorithms explained in chapter 3. The framework was developed in Java language at the University of Waikato in New Zealand and is on GPL licensed. It has a lot of acceptance by the researchers and is strengthened by its easy integration with the traditional mining framework WEKA (HALL et al., 2009). It is available in: <https: //moa.cms.waikato.ac.nz/>

MOA is mainly used in classification and clustering tasks but can be also used in regression, outliers treatment, and Concept Drift detection tasks. Also, it has a good collection of synthetic data-set generators as well as integration of real scenarios databases. Some of the generators currently available are Random Tree Generator, SEA Concepts Generator, STAGGER Concepts Generator, Rotating Hyperplane, Random-RBF Generator, LED Generator, Waveform Generator, and Function Generator.

In addition MOA contains several classifier methods such as: Naive Bayes, Decision Stump, Hoeffding Tree, Hoeffding Option Tree, Bagging, Boosting, Bagging using ADWIN, and Bagging using Adaptive-Size Hoeffding Trees (BIFET et al., 2009).

In order to model Concept Drift, MOA combines two pure distributions which characterizes the target concepts before and after the drift. Within this framework, it is possible to define the probability of instances in the stream to belong to the new concept after the drift. It uses the sigmoid function as an elegant and practical solution (BIFET et al., 2010). This function determines the change from one concept to another.

Finally, MOA allows the evaluation of data-stream classification algorithms in large streams. Traditional evaluation techniques considered to build a picture of accuracy over time are included as Holdout and Interleaved Test-Then-Train or Prequential (BIFET et al., 2010).

In addition, the MOAManager tool (PÉREZ, 2018) was employed, which is an open-source Web tool to assist in the creation, execution, extraction, and edition of experiments in online environments using the MOA framework. It is available in: <https://github.com/brunom4ciel/moamanager/>

5.2.2 Experimental Settings

This section describes relevant aspects of the experimental settings used to evaluate the proposed methods. Thus, the implementations of the methods proposed were compared to FASE, BOLE, DWM, DDD and OzaBag$_D$ ensembles with similar objectives.

To evaluate the accuracy of the learning models, the Prequential methodology (DAWID, 1984) with a sliding window of size 1000 (default setting in the MOA framework) as its forgetting mechanism (GAMA; SEBASTIÃO; RODRIGUES, 2013) was used. This choice was performed based on the research (HIDALGO; MACIEL; BARROS, 2019) that empirically evaluates the Prequential methodology considering its three common strategies: Basic Window, Sliding Window, and Fading Factors. The carried-out experiments suggest that the use of Prequential with the default Sliding Window variation is the best alternative in data-stream environment with concept drift. In general, the Prequential methodology is commonly used for both, stationary and non-stationary data-stream. This technique guarantees each instance of the data-stream is exploited to the maximum, first tested and then used for training (HIDALGO; MACIEL; BARROS, 2019).

Given that FASEB and FASEO family methods aim to make improvements in *run-time* and *memory* without having significant losses in *accuracy*, the methods are analyzed with respect to those metrics. To compute the measures in the synthetic data-sets, the experiments were run 30 times and the mean results were computed. In section 5.2.5, this topic will be more thoroughly addressed.

5.2.3 Parametrization of the Methods

As the compared methods have common parameters, they were similarly configured for a fair evaluation of results. Thus, two base learners were chosen, HT and NB among other possible classifiers because they are fast algorithms, widely used (FRIAS-BLANCO et al., 2016; BARROS; SANTOS; GONÇALVES JR., 2016; BARROS et al., 2017; BARROS; SANTOS, 2019; HIDALGO; MACIEL; BARROS, 2019) in environments with data-stream and their implementations are available in the MOA framework. Also, these base learners are the base classifiers by default in the state of the art compared methods (DWM, DDD, FASE use NB and OzaBag$_D$, BOLE use HT).

The number of experts was set to 10 for all methods that require this setup.

The parameters of FASE are the significant level for the change detections of HDDM$_A$ and the number of classifiers in the ensemble (FRIAS-BLANCO et al., 2016). Based on its default configuration in the MOA framework (BIFET et al., 2010), the significant levels of

HDDM$_A$ were set to $\alpha_D = 0.001$ for drifts and $\alpha_W = 0.005$ for warnings. The meta-Learner was setup by default with HDDM$_A$ as change detector. FASEB and FASEO families have the same parameters of FASE and were set with the same default values.

The variant of BOLE used in the experiments was BOLE$_4$, its default parametrization in MOA, which uses the *50%-Continue* voting strategy: breakVotes = "n", errorBound=0.5, and weightShift=0.0 (BARROS; SANTOS; GONÇALVES JR., 2016). The base learner and the ensemble size were configured such as in the other methods.

DWM uses 3 basic parameters, set to their default values. The time needed to verify if any expert will be added or removed and to update the weights of classifiers that incorrectly classifies the actual instance ($p = 50$); the decrement applied to the weight of an expert when it makes a mistake ($\beta = 0.5$); and the minimum value an expert must have to stay in the ensemble ($\theta = 0.01$) (KOLTER; MALOOF, 2007; BARROS; SANTOS; GONÇALVES JR., 2016).

DDD uses 4 default parameters: $\lambda_l = 1$ for encouraging low diversity in the underlying ensemble learning algorithm, $\lambda_h = 0,1$ for encouraging hight diversity in the underlying ensemble learning algorithm, multiplier constant $W = 1$ for the weight of the old low diversity ensemble, and parameter υ, for the drift detection methods, which is by default EDDM.

OzaBag$_D$: such as in FASE and its variants, the parameters are given by the detector employed, in this case HDDM$_A$, which was left with its default level setting, $\alpha_D = 0.001$ for drifts and $\alpha_W = 0.005$ for warnings.

5.2.4 Data-Sets

This section presents the data-sets used, both synthetic and real-world data-sets. A total of 6 synthetic generators (5.2.4.1) and 11 real data-sets, easily integrated into the framework, were considered for experimentation (5.2.4.2).

Synthetic Data-sets are useful due to the flexibility to impose different testing scenarios such as amount, position, and width of the concept drifts. Real world data-sets are equally important since their structure is typically unknown and bring unpredictability. Most of the data sets employed are available in the MOA website. In addition, Appendix. B summarizes the characteristics of several synthetic and real data-sets commonly used in the area.

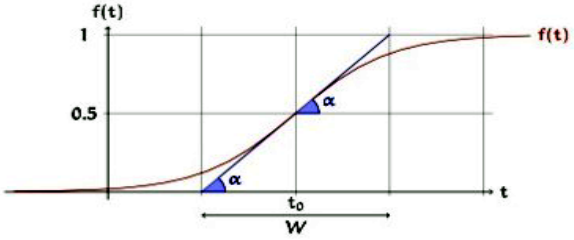
5.2.4.1 Synthetic Data-set Generators

In order to make adequate evaluation and comparisons between the methods, data-sets were built with known controlled concept drifts as the number of concept changes, the moment and frequency with which changes occur. Six (6) synthetic data-set were selected with two different sizes: 10000 and 50000 instances and for each of them, abrupt and gradual concept drifts were introduced.

In all generated synthetic data-sets, the concept drifts were distributed at regular intervals; for example, when the size of artificial data-set is 10000, the drifts are in instances 2000, 4000, 6000 and 8000.

For the gradual concept drifts, we simulated a window of change where the probability of a given instance belonging to the current concept or to the new concept follows the sigmoid function $f(t) = \frac{1}{(1+e^{-s(t-t_0)})}$ (see Fig. 6 (PÉREZ, 2018) present in MOA framework). This function determines the change from one concept to another.

Figure 6 – Sigmoid function



Source: Pérez (2018)

The parameter t_0 of the previous function, as well as α and W are used to setup the moment in which the Concept Drift occurs, the noise and the number of instances until the change occurs, respectively. At the beginning of the window W, the probability of processing an instance of the current concept is high; however, near the end of the window this probability decreases. After the window, the new concept is stable.The length of this window was set to 500 instances. In the abrupt concept drifts, the two concepts are simply concatenated.

Agrawal

The Agrawal generator stores information from people willing to receive a concession of a bank loan. To perform the classification task, the authors proposed 10 different functions with 9 attributes. Among the attributes, education level, car brand and zip code are categorical. Whereas salary, commission, age, value of the house, age of the house, and the desired loan amount are numerical. From this data, they should be classified as belonging to class A or B. The meaning of these groups varies according to each function. Agrawal is an excellent choice to generate concept changes since classification can be performed with up to ten different functions, and the label of each instance may vary according to the function used. In addition, it was used to show scalability in learning algorithms with decision trees. This data-set was previously tested in several researches (AGGARWAL; IMIELINSKI; SWAMI, 1993; MACIEL; SANTOS; BARROS, 2015).

LED

The LED generator represents the problem of predicting the digit shown by a seven-segment LED display, meant to simulate 7 light-emitting diodes. It has 24 boolean attributes (17 of which are irrelevant) and a categorical class, with ten possible values

(which is an integer ranging between 0 and 9). All attribute values are either 0 or 1, according to whether the corresponding light is on or off for the decimal digit. Each attribute has 10% chance of being inverted (noise). Concept drifts are simulated by simply changing the position of relevant attributes. This data-set was previously tested in several researches (BREIMAN et al., 1984; BIFET et al., 2010; BRZEZIŃSKI; STEFANOWSKI, 2011; GONÇALVES; BARROS, 2013; SANTOS et al., 2014).

Mixed

The Mixed generator aims at classifying an instance as positive or negative based on analysis of four attributes: two boolean (v, w) and two numerical (x, y). To do this, three conditions are checked for the attributes: $v, w, y < 0.5 + 0.3 \times \sin(3\pi x)$. If at least 2 of these conditions are satisfied, the example is considered positive. When a Concept Drift occurs, the classification is reversed. Concept Drift is made by gradually choosing examples from the old and new concepts, reducing the probability of selecting examples from the old context, while increasing the probability of the new context. In this data-set, we have an initial stable context that gradually changes to a new stable context. This is a noise free data-set and was previously tested in several researches (GAMA et al., 2004; BAENA-GARCIA et al., 2006; GONÇALVES; BARROS, 2013).

Sine

The Sine generator (GAMA et al., 2004) has two numeric attributes (x, y), each one having values distributed in the interval$[0, 1]$. Sine1 and Sine2 are the two possible context. In the first context, a given instance will be classified as positive if the point is below the curve $y = \sin(x)$. In the latter, the condition $y < 0,5 + 0,3\sin(3\pi x)$ must be satisfied. Concept drifts can be simulated either by reversing the aforementioned conditions or alternating between Sine1 and Sine2. This data-set was also previously tested by several authors: (GAMA et al., 2004; BAENA-GARCIA et al., 2006; ROSS et al., 2012).

Waveform

The Waveform generator represents the problem of differentiating between three wave classes, generated by combining two out of three base waves. A waveform has 40 numerical attributes, with the last 19 being irrelevant, only used to produce noise and also present three-classes. To perform changes, the positions of the attributes representing a certain context are reversed. This data-set was previously used in experiments in the works (BREIMAN et al., 1984; BIFET et al., 2009; SANTOS; BARROS; Gonçalves Jr., 2015)

Random-RBF

The Random-RBF (Radial Basis Function) generator uses a fixed number of random centroids with their centers. Each center has a random position, a single standard deviation, a class label, and a weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weights are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribu-

tion with standard deviation determined by the chosen centroid. This data-set generator has 6 classes, 40 attributes, and 50 centroids. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities which is very hard to learn. A Concept Drift is simulated by changing the positions of centroids with a constant speed. The Random-RBF set was already used in the following papers (BIFET et al., 2009; BIFET et al., 2010; MACIEL; SANTOS; BARROS, 2015).

5.2.4.2 Real Data-sets

In addition, real-world data-sets previously used in the area were experimented with different number of instances and complexities. In these data-sets, the number and position of the drifts are unknown. The databases are available at <http://archive.ics.uci.edu/ml/index.php>, belonging to the UCI Machine Learning Repository (DUA; GRAFF, 2017).

Connect-4

The Connect-4 contains 67557 instances without missing values and describes the all legal 8-play positions in the game of Connect-4 in which neither player has won yet, and in which the next move is not forced. "x" is the first player and "o" is the second. It has 42 categorical attributes and 3 classes. The outcome class is the game theoretical value for the first player that can be win, loss or draw. The database has been used in the works (BURTON; KELLY, 2006; ZHONG; TANG; KHOSHGOFTAAR, 2005; GODASE; ATTAR, 2012; WANKHADE; DONGRE; THOOL, 2012).

Covertype/CovSorted

The Covertype datset aims to predict the forest cover type from cartographic variables. Its contains information on the forest cover type for 30×30 meter cells obtained from US Forest Service (USFS) Region 2 data and contains 581,012 examples and 54 attributes (numeric and categorical). It has been used in several papers on data-stream classification, such as (BIFET et al., 2009; BIFET et al., 2010; GONÇALVES; BARROS, 2013; MACIEL; SANTOS; BARROS, 2015). There is another version referred to as Covertype-Sorted (IENCO et al., 2013) where the instances are sorted by the elevation attribute. It induces gradual concept drifts on the class distribution depending on the elevation, some types of vegetation disappear while others start to appear.

Lung-cancer

The *Lung-cancer*data-set records 3 types of pathological lung cancers. The authors give no information on the individual variables nor on where the data was originally used. This database has 32 instances and 56 integer attributes. All predictive attributes are nominal with integer values 0-3. The data present missing values and was previously used in (HONG; YANG, 1991) as well as other papers (DASH; LIU, 1998; JIN; AGRAWAL, 2003).

NSL-KDDCup99

The NSL-KDDCup99 data-set represents a network connection and is a refined version of the KDDCup99 data-set used in the 3^{rd} International Knowledge Discovery and Data Mining Tools Competition, modified to eliminate some of the limitations present in the original data-set. Many types of analysis have been carried out by researchers with the NSL-KDDCup99 data-set employing different techniques and tools with a universal objective to develop an effective intrusion detection system. It is available at the UCI Machine Learning Repository (DUA; GRAFF, 2017). The possible value of classes is 23, with 22 of them representing attack and the remaining 23rd representing normality (PERVEZ; FARID, 2014). In this research, a version of the database containing only two classes (normal and anomaly) was applied. The data-set has 125.973 instances with 41 discrete or continuous values input attributes. This data-set has been used in related works (REVATHI; MALATHI, 2013; KUMAR; CHAUHAN; PANWAR, 2013).

PokerHand/Pokerhand-1M

The Pokerhand represents the problem of identifying the value of a five-card hand in the game of Poker. It is constituted of five numeric and five categorical attributes and one categorical class with 10 possible values informing the value of the hand (one pair, two pairs, a sequence, a street flush, etc). In the original and harder to classify version of this data-set, with 1,000,000 instances, the cards are not ordered. This version is referred to Pokerhand-1M in this work. This data-set was previously used in (BIFET et al., 2009; SANTOS; BARROS; Gonçalves Jr., 2015). There is a modified version available at the MOA website (BIFET et al., 2010; GONÇALVES; BARROS, 2013) in which cards are sorted by rank, and suit and duplicates are removed, resulting in 829,201 instances. This version is comparatively much more used than the original version.

Wine/ Wine-Red/ Wine-Red-100

These data-sets are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determines the quantities of 13 constituents found in each of the three types of wines. The data-sets have been used first in (FORINA et al., 1991). Due to confidentiality and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (i.e. there is no data about grape types, wine brand, wine selling price, etc.). The classes are ordered and not balanced (i.e. there are much more normal wines than excellent or poor ones). It has 1599 instances, 11 real attributes an the class value are integers between 1 and 9. In particular, Wine-Red-100 is a version of Wine-Red database with 100 instances. Those data-sets have been used in classification and regression tasks in some related works (ZHONG; TANG; KHOSHGOFTAAR, 2005; FISCHER; POLAND, 2005).

Usenet

Usenet: The Usenet-1 and Usenet-2 data-sets are based on the 20 newsgroups collection. Usenet-2 (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008; KOHAIL, 2011; DUA; GRAFF, 2017) is a version of Usenet-1. This database is a flow of 1500 instances, divided into five

time periods. The periods contain 300 instances. Concept change happens after the end of each period. They simulate a stream of messages from different newsgroups sequentially presented to a user, who then labels them as interesting (+) or junk (-), according to his/her personal interest. It also has 100 categorical attributes and 2 classes. It is available at <http://qwone.com/~jason/20Newsgroups/>. It is noted the first Usenet1 database is a much more diverse data-set, with all categories changing the class in each period (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008). For its part, Usenet-2 is more moderate in concept changes (2 of 3 classes change concepts all the time). Both databases present abrupt and recurrent concept changes. This data-set has been used in some related works (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010; SUDHA; GOVINDARAJAN, 2017).

Sensor

Sensor Stream (ZHU et al., 2010) contains information collected from 54 sensors deployed in the Intel Berkeley Research Lab (temperature, humidity, light, and sensor voltage). It contains consecutive information recorded over a 2 months period, with one reading every 1-3 minutes. The sensor ID is the target attribute, which must be identified based on the sensor data and the corresponding recording time. This data-set is constituted of 2,219,803 instances, 5 attributes, and 54 classes. This is an interesting and intriguing data-set because, in addition to being much larger than the others, produces considerable variations in the accuracy performance of the methods.

5.2.5 Experimental Result and Analysis

This section compares algorithms FASEO, FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB, FASEB$_{wv1}$, FASEB$_{wv2}$, FASEB$_{wv3}$, FASE, BOLE, DWM, OzaBag$_D$ and DDD in 5 experimentation scenarios. Among those scenarios, four were designed with synthetic data and one with real data. The scenarios with synthetic data cover data-sets of size 10k and 50k built with abrupt and gradual changes. Both in the synthetic (24) and real (11) data-sets each algorithm is tested and trained using the classifiers HT and NB. In summary, 70 experiments were carried out to evaluate the performance of each method according to the three metrics considered.

Tables 2 to 11 present accuracy rates, run-times and memory values achieved by the algorithms in the 5 experimentation scenarios, using both NB and HT as base learners. In addition, the average rank reached by each method according to above metrics is presented, based on the statistical Friedman test.

The first values appearing in the tables refer to accuracy, then run-times and finally consumed memory. Each cell in the tables presents the values reached by the methods. The higher values are indicated in bold inside a cell with an edge and results indicating improvements with respect to FASE are in highlighted bold and italics. Note that higher values in accuracy indicate better performance whereas, for the other two metrics, the lower values the better. The average ranks by scenario where the best measurements

are determined by lower values are indicated in bold. Conversely, worst results are highlighted in italics. Thus, regarding the proposed algorithms, highlighted values indicate improvements with respect to FASE.

Below, a first general idea about the overall results achieved is clarified. Then, a more detailed evaluation of the methods is presented with both synthetic and real data-sets and different scenarios for each base classifier (NB and HT).

5.2.5.1 Overview of Results

In general, regarding accuracy FASE still presented the best average in the synthetic data-set, especially in the data-sets obtained from the Agrawal and Mixed generators. In almost all cases with synthetic data, it had better performance than the proposed variants, although the results with the proposed methods remained very close to those of FASE.

However, the FASEB variant, particularly, outperformed FASE in 25 experiments out of 70 carried out (35,71%). See Appendix.D Table 17.

Also, Using NB as base classifier, FASEB had the best performance in the gradual synthetic data-set whereas the same method performed better in the synthetic data-sets with abrupt changes when the HT classifier was used.

The data-set on which the developed variants performed better were those obtained from the Led generator, where the methods that use weighted voting to perform classification reached better behavior, especially FASEO$_{wv2}$ and FASEB$_{wv2}$.

Concerning the other methods, they are surpassed by the proposed methods in most scenarios, although in Scenario 5, corresponding to real data-sets, BOLE obtained the best result with both NB and HT as base classifiers. In this scenario, when the methods use the NB classifier, FASEO$_{wv2}$ and FASEB$_{wv2}$ performed equally or better than FASE in 6 out of 11 real data-sets (See Appendix.D Table 15). Similarly, FASEB improved or tied FASE in 6 out of the 11 real data-sets when HT was employed (See Appendix.D Table 16). Moreover, in each designed scenario, DWM has the worst performance.

Considering run-time OzaBag$_D$ was the best-ranked algorithm using both NB and HT as base classifiers in each designed scenario, while FASE and DDD were the worst, respectively, with those classifiers. The implemented variants of FASE had better performance than FASE, with better ranking positions in all scenarios. The exception was notived with classifier HT in Scenario 5, corresponding to real data-sets, where FASEO and FASEB performed slower than FASE. In conclusion, the variant of FASE with a meta-classifier demanded more run-time to perform the classification than the others with weighted voting. Particularly, FASEB$_{wv3}$ improvement over FASE occured in 66 out of 70 experiments (94.29 %), being the fastest among all proposed methods. Appendix.D Table 17 summarizes the results.

Considering memory consumption when the methods used both NB and HT as base classifiers, in the first four scenarios DWM presented the best performance followed by OzaBag$_D$. Diversely, in scenario 5, corresponding to real data-sets, OzaBag$_D$ was the algorithm with the best results. When the experiments were performed with classifier NB, FASE showed the worst results in all 5 scenarios. Particularly in scenario 3, corresponding to synthetic data-sets with abrupt changes of size 10k, FASE and FASEB performed the worst. On the other hand, when HT was considered as base classifier, DDD attained the worst results in all 5 scenarios.

Analysis of overall memory requirement results, indicated that FASEB$_{wv3}$ outperformed FASE in 66 out of 70 experiments (94.29 %) being the algorithm with the least demanding memory among all implemented variants. Only in the particular case of real data-sets with HT as base classifier FASEO and FASEB consumed more memory than FASE. Also, FASEO and FASEB consumed more memory than the other variants. Finally, the variants with the meta-classifier consumed more memory to perform classification than the others which use weighted voting.

5.2.5.2 Analysis with NB as base classifier

Here we analyse the five scenarios in terms of accuracy, run-time and memory assuming each algorithm used NB as a base classifier.

Scenario-1-NB: In the synthetic data-sets with gradual changes of size 10k, FASE obtains the best global accuracy, followed by FASEB and FASEO. FASE wins at the data-sets Agrawal and Random-RBF. The variant FASEB has better results than FASE in 3 data-sets (Led, Mixed and Sine) winning in Sine. Whereas FASEB$_{wv2}$ wins in Led. In total, out of the 8 variants, 7 (FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB, FASEB$_{wv1}$, FASEB$_{wv2}$ and FASEB$_{wv3}$) outperformed FASE in Led and 2 (FASEO, FASEB) in Sine. In the base Mixed, wins DDD while in Waveform it was BOLE. The worst result in global accuracy is seen with DWM. In this same scenario OzaBag$_D$ has the best result with respect to run time, followed by DWM, which has the best result in memory. All the FASE variants improved its performance in each one of the data-sets in run time and memory. FASE has the worst result in run time and memory in the data-sets Agrawal, Mixed and Sine, followed by DDD in the other data-sets. FASE was the method that demanded the most resources in terms of run time and memory. Complete results are shown in Table 2.

Scenario-2-NB: In the scenario of synthetic data-sets with gradual changes of size 50k, FASE obtains the best result in terms of global accuracy, followed by FASEB and FASEO. FASE wins in data-sets Agrawal and Random-RBF, while FASEB improved the FASE accuracy in the data-sets Led and Sine. Both reached the highest value in the Mixed base. Particularly, in the Led base, variants FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB$_{wv1}$ (highest accuracy), FASEB$_{wv2}$ (highest accuracy) and FASEB$_{wv3}$ also outper-

formed FASE. FASEO, FASEB$_{wv1}$ and FASEB$_{wv2}$ achieved the same accuracy as that of FASE in the Sine base. BOLE wins at the Sine and Waveform data-sets. The worst result in accuracy was reached by DWM. In the same scenario, OzaBag$_D$ had the best result in run time and DWM in memory. All variants of FASE improved its performance in each of the data-sets with respect to run time and memory. FASE had the worst time and memory results in Agrawal, Mixed and Sine. In the other data-sets, DDD shared with FASE similar results in terms of memory. Specifically in the base Waveform, it was the slowest method. DWM has the worst run time values in the Led and Random-RBF data-sets. In general, FASE was the method to demand more resources in terms of run time and memory also in this scenario. See in Table 3.

Scenario-3-NB: In the third scenario of 10k data-sets designed with abrupt changes, FASEB had the best result in accuracy, followed by FASEO and FASE. FASE achieved the highest results in comparison to the other methods in the Agrawal, Mixed and Random-RBF data-sets. While BOLE wins in the data-sets Sine and Waveform. In this last base, although the FASE variants are not winners, FASEO$_{wv3}$ has the same result as FASE and the other 7 variants surpass its performance in terms of accuracy. In Led, all FASE variants surpassed its performance and FASEO$_{wv2}$ reached the best result. The worst accuracy was achieved by DWM. Regarding run time and consumed memory, OzaBag$_D$ was the fastest algorithm while DWM showed lower memory consumption. Except in the Waveform database where only FASEB$_{wv2}$ and FASEB$_{wv3}$ improved FASE performance; in the rest of the data-sets, all variants of FASE had better results than FASE in run time and memory. FASE showed the worst result in Agrawal, Mixed and Sine, while DDD performed the worst in the other data-sets. Again, FASE was the algorithm to demanded more resources in terms of run time and memory (shared with FASEB) in this scenario. See Table 4 for full results.

Scenario-4-NB: In the 50k data-sets designed with abrupt changes FASE obtained the best ranking in accuracy followed by FASEB and FASEO. FASE wins in Agrawal, Mixed and Random-RBF. While BOLE wins in the data-sets Sine and Waveform. Variants FASEB$_{wv1}$ and FASEB$_{wv2}$ reached the highest value in Led, and with the exception of FASEO the remaining variants surpassed FASE in this data-set. The worst result in accuracy was given by DWM. Regarding the other metrics, run time and memory, OzaBag$_D$ was the fastest algorithm and DWM demanded the least memory in this scenario. All FASE variants outperformed the original method, except FASEO regarding run time in the base Sine. The worst run time results by data-set were presented by FASE (Agrawal and Mixed), DWM (Led and Random-RBF), FASEO (Sine) and DDD (Waveform). DDD also presented the highest values of consumed memory in data-sets Led, Waveform and Random-RBF, while FASE presented the highest values in the other data-sets. In this scenario, the overall worst results in run time and memory were presented by FASE. See Table 5.

Scenario-5-NB: Eleven (11) real data-sets were chosen for performance evaluation (Connect-4, Covertype, Covertype-Sorted, Lung-cancer, NslKdd99, Pokerhand, Pokerhand-1M, Sensor, Wine-Red, Wine-Red-100, and Usenet-2). Concerning accuracy, BOLE had the best overall result, whith winnings in most of the data-sets (Connect-4, Covertype, Covertype-Sorted, NslKdd99, Pokerhand, Usenet-2 and Sensor) followed by OzaBag_D and FASEB_{wv2}. OzaBag_D wons in the Wine-Red data-set while FASE performed better in Lung-cancer, as well as, the FASEO and FASEB variations. In the Pokerhand-1M data-set, the best methods were FASEO_{wv2}, FASEO_{wv3}, FASEB_{wv2}, FASEB_{wv3}, and DDD. In Wine-Red-100, the hightest result was reached by FASEO_{wv3}, FASEB_{wv3}.

In general, in this scenario, several of the proposed methods had similar results or outperformed FASE in 8 out of 11 data-sets, FASEO family methods: FASEO in (Lung-cancer, NslKdd99, Pokerhand-1M, Sensor), FASEO_{wv1} (Connect-4, Pokerhand-1M, Wine-Red-100, Sensor), FASEO_{wv2} (Connect-4, Covertype-Sorted, Pokerhand-1M, Wine-Red, Wine-Red-100, Sensor), FASEO_{wv3} (Connect-4, Covertype-Sorted, Pokerhand-1M, Wine-Red, Wine-Red-100). In addition, the methods of the family FASEB: FASEB in (Lung-cancer, NslKdd99, Pokerhand-1M, Sensor), FASEB_{wv1} (Connect-4, Pokerhand-1M, Wine-Red-100, Sensor), FASEB_{wv2} (Connect-4, Covertype-Sorted, Pokerhand-1M, Wine-Red, Wine-Red-100, Sensor) and FASEB_{wv3} (Covertype-Sorted, Pokerhand-1M, Wine-Red, Wine-Red-100, Sensor).

Regarding the other metrics, run-time and memory,OzaBag_D was the fastest algorithm and demanded the least memory. On the other hand, FASE was the algorithm to demand more of run time and memory. In particular, DWM was the most delayed algorithm in the data-sets Covertype, Covertype-Sorted, Pokerhand, Pokerhand-1M and Sensor, FASEO in Wine-Red-100, whereas DDD was the worst in the other data-sets. DDD together with FASEO_{wv1}, FASEO_{wv2}, FASEB, FASEB_{wv1}, FASEB_{wv2} and FASE were delayed when compared to the other algorithms in Lung-cancer. Similarly, regarding consumed memory, FASE had the worst performance in Lung-cancer and Wine-Red, FASEO in Wine-Red-100, DWM in Pokerhand, Pokerhand-1M and Sensor; DDD in the other data-sets.

In this scenario the proposed algorithms do not always improve FASE with respect to run time and memory. Concerning run time, FASEO did not improve FASE in (Covertype, Covertype-Sorted, Pokerhand, Pokerhand-1M, Wine-Red-100), FASEO_{wv1} (Lung-cancer, Usenet-2), FASEO_{wv2} (Lung-cancer, NslKdd99, Wine-Red-100); and FASEB did not outperformed FASE in (Lung-cancer, Wine-Red-100), FASEB_{wv1} (Lung-cancer), FASEB_{wv2} (Connect-4, Lung-cancer, Wine-Red-100). Regarding memory, FASEO did not improve FASE in (Covertype, Covertype-Sorted, Wine-Red-100), FASEO_{wv1} was not superior to FASE in (Covertype, Usenet-2), and FASEO_{wv2} was not superior to FASE in (NslKdd99). See Table 6.

Table 2 – Mean accuracies, run-times and memory in percentage (%) using NB, with 95% confidence intervals in scenarios of 10k size synthetic data-sets and gradual concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	61,06	60,57	60,60	59,93	61,16	60,56	60,58	59,92	**62,26**	60,85	56,92	61,91	59,74
LED	67,00	***67,77***	***67,79***	***67,51***	***67,21***	***67,79***	**67,81**	***67,51***	67,10	67,53	42,18	66,53	66,59
Mixed	83,68	83,34	83,30	83,04	***83,77***	83,42	83,39	83,09	83,69	81,89	81,68	82,73	**84,04**
Sine	***81,78***	81,52	81,52	81,35	**81,86**	81,60	81,59	81,35	81,71	80,59	79,97	79,43	81,80
Waveform	77,62	77,56	77,53	77,36	77,62	77,58	77,55	77,36	78,19	**80,71**	76,10	78,10	78,47
Random	30,90	30,81	30,79	30,92	30,94	30,84	30,82	30,85	**31,61**	20,03	23,97	31,28	30,48
AVG. Rank	5,25	7,25	7,33	8,58	3,75	5,92	6,17	8,92	**3,50**	7,83	*12,67*	7,33	6,50
Agrawal	***2,33***	***2,47***	***2,46***	***2,48***	***2,73***	***2,45***	***2,48***	***2,48***	**3,05**	1,55	0,45	0,75	2,23
LED	***2,67***	***2,47***	***2,34***	***2,45***	***2,65***	***2,30***	***2,33***	***2,27***	3,03	2,17	1,20	1,47	**4,66**
Mixed	***1,19***	***1,13***	***1,11***	***1,10***	***1,18***	***1,04***	***1,03***	***1,07***	**1,31**	0,49	0,10	0,24	0,59
Sine	***1,20***	***1,13***	***1,14***	***1,16***	***1,24***	***1,13***	***1,14***	***1,12***	**1,42**	0,50	0,10	0,25	0,67
Waveform	***1,88***	***1,74***	***1,69***	***1,68***	***1,96***	***1,62***	***1,65***	***1,70***	2,25	1,59	0,51	0,94	**2,85**
Random	***3,74***	***3,06***	***3,20***	***3,11***	***3,87***	***3,09***	***3,01***	***3,00***	4,40	3,42	4,06	2,49	**8,08**
AVG. Rank	***9,67***	***7,75***	***7,58***	***8,00***	***11,00***	***5,42***	***6,25***	***6,00***	*12,50*	3,83	2,67	**1,83**	8,50
Agrawal	***147,73***	***159,18***	***158,38***	***159,58***	***179,27***	***155,20***	***156,63***	***156,96***	**214,43**	47,82	1,76	11,03	105,32
LED	***279,74***	***256,67***	***243,10***	***254,05***	***276,99***	***224,63***	***227,46***	***221,73***	332,41	173,14	10,14	72,58	**862,98**
Mixed	***68,31***	***65,05***	***64,13***	***63,28***	***69,65***	***59,01***	***58,73***	***60,51***	**83,84**	9,66	0,20	2,03	12,66
Sine	***69,62***	***65,92***	***66,44***	***67,57***	***74,27***	***64,93***	***65,59***	***63,96***	**91,58**	10,48	0,21	2,28	16,38
Waveform	***146,20***	***136,39***	***132,66***	***132,08***	***155,73***	***124,79***	***126,48***	***130,34***	189,38	88,98	2,34	26,35	**318,33**
Random	***465,58***	***382,46***	***399,66***	***387,98***	***480,59***	***370,83***	***360,14***	***359,82***	560,91	433,33	38,36	170,84	**2423,28**
AVG. Rank	***9,67***	***8,67***	***8,33***	***8,50***	***11,33***	***5,33***	***5,67***	***5,50***	*12,50*	4,00	**1,00**	2,00	8,50

Source: Own elaboration (2019)

Table 3 – Mean accuracies, run-times and memory in percentage (%) using NB, with 95% confidence intervals in scenarios of 50k size synthetic data-sets and gradual concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	65,42	65,09	65,09	64,98	65,42	65,08	65,09	64,98	**65,68**	65,00	57,62	65,07	62,26
LED	72,41	***72,57***	***72,57***	***72,50***	***72,45***	**72,60**	**72,60**	***72,50***	72,42	72,48	42,93	70,81	70,15
Mixed	90,41	90,37	90,36	90,29	**90,44**	90,39	90,38	90,32	**90,44**	88,97	87,89	89,91	89,58
Sine	***86,69***	86,68	86,68	86,67	***86,72***	***86,69***	***86,69***	86,67	86,69	**87,77**	84,95	85,77	85,87
Waveform	80,10	79,80	79,80	79,75	80,11	79,86	79,86	79,86	80,20	**81,82**	77,09	79,68	79,30
Random	31,05	30,85	30,84	31,19	31,09	30,86	30,85	31,18	**32,25**	19,77	23,68	31,46	31,06
AVG. Rank	5,17	6,67	7,08	7,92	3,67	5,17	5,25	7,25	**3,17**	7,17	*12,83*	9,00	10,67
Agrawal	***6,60***	***6,63***	***6,60***	***6,57***	***7,03***	***6,39***	***6,22***	***6,21***	**8,09**	4,10	6,48	2,13	5,96
LED	***11,84***	***10,48***	***10,39***	***10,74***	***11,66***	***9,59***	***11,06***	***10,36***	12,67	9,46	**20,84**	6,03	20,62
Mixed	***5,01***	***4,76***	***4,66***	***4,79***	***5,19***	***4,62***	***4,63***	***4,60***	**5,65**	2,18	0,83	1,00	2,51
Sine	***5,07***	***4,83***	***4,63***	***4,45***	***5,01***	***4,31***	***4,37***	***4,31***	**5,34**	2,15	1,11	1,01	2,78
Waveform	***8,21***	***7,32***	***7,27***	***7,43***	***8,36***	***7,00***	***7,04***	***7,05***	9,38	7,01	8,23	3,80	**12,47**
Random	***16,09***	***13,62***	***13,40***	***13,57***	***16,07***	***12,63***	***12,32***	***12,86***	19,16	14,77	**82,72**	9,95	30,14
AVG. Rank	***10,25***	***8,33***	***7,08***	***7,83***	***10,67***	***4,25***	***5,50***	***4,58***	*12,17*	3,50	7,67	**1,17**	8,00
Agrawal	***415,66***	***428,93***	***427,29***	***425,29***	***460,36***	***403,37***	***393,04***	***391,85***	**567,80**	126,86	35,28	31,47	290,52
LED	***1238,65***	***1077,57***	***1068,83***	***1104,27***	***1215,83***	***934,75***	***1078,15***	***1010,00***	1390,34	754,75	214,42	298,68	**3957,23**
Mixed	***291,02***	***273,61***	***267,76***	***275,52***	***306,60***	***262,00***	***262,76***	***260,80***	**360,32**	43,57	1,96	8,37	55,58
Sine	***295,44***	***281,01***	***269,70***	***259,12***	***299,05***	***247,14***	***250,46***	***246,83***	**343,76**	45,77	2,69	9,04	70,05
Waveform	***643,10***	***588,75***	***584,71***	***597,54***	***662,78***	***535,89***	***538,51***	***539,29***	788,76	397,14	44,09	106,66	**1451,84**
Random	***1984,01***	***1751,55***	***1722,42***	***1745,09***	***1980,69***	***1503,15***	***1466,76***	***1530,84***	2435,11	1882,28	1001,35	682,58	**9163,15**
AVG. Rank	***10,33***	***8,83***	***7,67***	***8,67***	***11,17***	***5,17***	***6,00***	***5,17***	*12,50*	4,00	**1,33**	1,67	8,50

Source: Own elaboration (2019)

Table 4 – Mean accuracies, run-times and memory in percentage (%) using NB, with 95% confidence intervals in scenarios of 10k size synthetic data-sets and abrupt concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	62,91	62,09	62,13	61,43	63,03	62,21	62,25	61,63	**63,97**	61,80	57,57	63,06	60,40
LED	***69,13***	***69,68***	**69,72**	***69,44***	***69,13***	***69,66***	***69,70***	***69,45***	68,64	68,94	44,98	68,54	67,34
Mixed	89,89	89,68	89,63	89,51	89,91	89,69	89,64	89,50	**89,95**	88,50	86,96	78,54	87,86
Sine	86,29	86,22	86,23	86,12	86,32	86,25	86,25	86,15	86,42	**86,76**	84,58	78,44	85,04
Waveform	***78,63***	***78,53***	***78,50***	***78,38***	***79,00***	***78,65***	***78,53***	***78,50***	78,38	**80,95**	76,49	78,60	78,90
Random	30,96	30,84	30,82	30,95	30,87	30,85	30,83	30,95	**31,58**	20,06	24,02	31,06	30,59
AVG. Rank	***4,42***	6,58	6,92	8,50	**3,92**	5,08	5,83	7,83	4,42	7,17	*12,50*	7,83	10,00
Agrawal	***2,35***	***2,47***	***2,54***	***2,47***	***2,78***	***2,56***	***2,47***	***2,49***	**2,98**	1,58	0,44	0,73	2,30
LED	***2,76***	***2,44***	***2,38***	***2,37***	***2,66***	***2,31***	***2,32***	***2,30***	2,92	2,25	1,17	1,39	**4,64**
Mixed	***1,20***	***1,10***	***1,12***	***1,08***	***1,21***	***1,05***	***1,06***	***1,06***	**1,30**	0,51	0,09	0,23	0,58
Sine	***1,23***	***1,12***	***1,15***	***1,14***	***1,26***	***1,12***	***1,12***	***1,12***	**1,39**	0,54	0,09	0,26	0,63
Waveform	1,91	1,70	1,70	1,71	2,19	1,94	***1,63***	***1,66***	1,70	1,60	0,49	0,89	**2,88**
Random	***3,81***	***3,09***	***3,26***	***3,21***	***3,87***	***3,16***	***3,03***	***2,98***	4,43	3,47	4,11	2,50	**7,70**
AVG. Rank	***9,50***	***7,08***	***8,67***	***7,67***	***11,33***	***7,25***	***5,50***	***5,50***	*11,67*	3,83	2,67	**1,83**	8,50
Agrawal	***149,11***	***158,59***	***163,57***	***158,83***	***182,42***	***161,98***	***155,96***	***157,38***	**209,57**	48,67	1,71	10,68	108,87
LED	***290,98***	***247,33***	***241,24***	***240,72***	***276,92***	***225,32***	***225,79***	***223,66***	319,59	177,50	9,82	68,78	**861,25**
Mixed	***71,28***	***63,23***	***64,45***	***61,98***	***71,41***	***59,57***	***60,04***	***59,79***	**82,61**	10,17	0,18	1,96	12,35
Sine	***73,21***	***64,93***	***66,71***	***66,27***	***75,16***	***64,09***	***64,34***	***64,22***	**89,50**	11,42	0,18	2,32	15,22
Waveform	149,97	134,44	134,83	135,78	184,03	154,08	***124,83***	***127,39***	130,25	89,34	2,26	24,92	**319,68**
Random	***470,96***	***383,97***	***405,10***	***398,28***	***479,19***	***377,98***	***362,50***	***356,17***	563,74	438,38	38,84	171,39	**2303,60**
AVG. Rank	***9,67***	***7,83***	***9,17***	***8,17***	*11,50*	***6,83***	***5,67***	***5,17***	*11,50*	4,00	**1,00**	2,00	8,50

Source: Own elaboration (2019)

Table 5 – Mean accuracies, run-times and memory in percentage (%) using NB, with 95% confidence intervals in scenarios of 50k size synthetic data-sets and abrupt concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	65,72	65,54	65,55	65,42	65,74	65,54	65,55	65,44	**66,01**	65,14	57,81	65,40	62,19
LED	72,72	***72,86***	***72,86***	***72,76***	***72,75***	**72,87**	**72,87**	***72,78***	72,74	72,72	43,36	70,75	70,05
Mixed	91,50	91,41	91,40	91,35	91,52	91,42	91,41	91,36	**91,57**	89,58	89,10	79,46	89,94
Sine	87,18	87,14	87,15	87,21	87,18	87,14	87,15	87,20	87,26	**88,41**	85,95	79,40	85,37
Waveform	80,36	80,07	80,07	80,07	80,35	80,11	80,10	80,10	80,42	**81,91**	77,15	79,62	79,22
Random	31,03	31,01	31,00	31,29	31,03	31,00	30,99	31,27	**32,26**	19,78	23,68	31,36	31,14
AVG. Rank	5,08	7,00	6,83	6,50	4,50	6,00	6,08	5,92	**2,50**	7,75	*12,33*	10,00	10,50
Agrawal	***6,50***	***6,54***	***6,54***	***6,51***	***7,15***	***6,37***	***6,22***	***6,28***	**7,89**	4,13	6,58	2,09	5,88
LED	***11,94***	***10,40***	***10,15***	***10,65***	***11,79***	***9,65***	***10,87***	***9,95***	12,82	9,52	**26,77**	6,12	20,55
Mixed	***4,96***	***4,68***	***4,72***	***4,66***	***5,14***	***4,63***	***4,49***	***4,57***	**5,66**	2,17	0,79	0,99	2,62
Sine	**5,26**	***4,50***	***4,58***	***4,38***	***4,74***	***4,43***	***4,19***	***4,22***	5,23	2,19	1,04	1,00	2,78
Waveform	***8,11***	***7,38***	***7,37***	***7,37***	***8,44***	***7,02***	***7,03***	***7,15***	9,33	6,89	8,24	3,80	**12,34**
Random	***16,16***	***13,62***	***13,38***	***13,23***	***16,16***	***12,62***	***12,37***	***12,56***	18,93	14,37	**83,23**	10,06	29,71
AVG. Rank	***9,92***	***8,08***	***7,83***	***6,92***	***10,75***	***5,17***	***4,67***	***4,83***	*12,00*	3,33	8,33	**1,17**	8,00
Agrawal	***412,00***	***421,89***	***421,89***	***419,92***	***468,11***	***401,94***	***392,44***	***396,53***	**553,86**	127,88	35,83	30,87	285,76
LED	***1254,75***	***1060,11***	***1035,28***	***1086,97***	***1229,84***	***940,72***	***1060,15***	***970,64***	1406,25	763,59	275,07	303,38	**3883,18**
Mixed	***291,54***	***268,02***	***270,11***	***266,97***	***303,43***	***262,51***	***254,47***	***259,16***	**360,98**	43,49	1,83	8,34	58,05
Sine	***311,88***	***261,46***	***265,86***	***254,24***	***282,72***	***253,52***	***239,88***	***241,40***	**336,44**	46,55	2,48	8,96	69,17
Waveform	***629,51***	***587,11***	***586,15***	***586,06***	***669,30***	***537,12***	***538,17***	***547,13***	783,54	389,99	44,12	106,89	**1433,92**
Random	***1994,43***	***1739,19***	***1707,60***	***1688,42***	***1987,86***	***1499,16***	***1469,68***	***1491,73***	2407,22	1830,60	1009,18	690,25	**9030,07**
AVG. Rank	***10,50***	***8,75***	***8,58***	***7,83***	***11,00***	***5,67***	***5,17***	***5,50***	*12,50*	4,00	**1,33**	1,67	8,50

Source: Own elaboration (2019)

Table 6 – Mean accuracies, run-times and memory in percentage (%) using NB in real-world data-sets.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Connect-4	74,47	***75,10***	***75,10***	***74,66***	74,48	***74,92***	***74,92***	74,64	74,66	**77,29**	72,99	75,95	74,47
Covertype	85,51	85,12	84,52	83,69	85,58	85,16	84,53	83,73	88,13	**91,79**	85,16	86,05	84,02
Covertype-Sorted	68,10	68,61	***69,55***	***69,15***	68,27	68,67	***69,74***	***69,35***	69,06	**71,26**	68,33	70,70	67,52
Lung-cancer	**77,97**	65,57	66,82	65,57	**77,97**	65,57	66,82	65,57	**77,97**	51,98	64,38	67,24	67,24
NslKdd99	***89,83***	89,79	89,79	89,75	***89,83***	89,79	89,79	89,75	89,81	**94,51**	89,56	89,79	89,77
Pokerhand	74,43	75,97	75,82	75,11	74,73	76,02	75,89	75,15	76,55	**80,50**	75,84	77,58	77,47
Pokerhand-1M	***50,10***	***50,10***	**50,11**	**50,11**	***50,10***	***50,09***	**50,11**	**50,11**	49,87	50,05	46,42	50,00	**50,11**
WineRed	48,74	48,38	***52,86***	***54,09***	48,74	48,38	***52,86***	***54,09***	50,60	53,38	47,52	**55,59**	52,57
Wine-Red-100	41,63	***50,65***	***55,99***	**57,11**	41,63	***50,65***	***55,99***	***57,11***	44,48	47,37	46,66	55,78	55,79
Usenet-2	70,23	67,31	67,31	66,84	70,11	67,10	67,10	66,27	72,86	**72,96**	68,25	69,15	71,33
Sensor	***86,41***	***89,36***	***88,06***	86,01	***87,15***	***89,86***	***88,30***	***87,00***	86,23	**92,88**	29,36	86,52	86,36
AVG. Rank	8,00	7,32	***5,95***	8,14	7,41	7,45	***5,86***	7,64	6,41	**3,91**	*10,77*	4,77	7,36
Connect-4	***11,90***	***11,81***	***13,03***	***11,99***	***12,55***	***12,45***	15,88	***12,69***	13,58	12,35	13,30	6,94	**21,71**
Covertype	144,34	***135,11***	***127,09***	***126,04***	***126,91***	***114,97***	***115,36***	***114,58***	135,94	97,26	**1105,95**	72,12	234,25
Covertype-Sorted	156,96	***139,68***	***132,32***	***128,72***	***140,94***	***124,66***	***122,60***	***120,41***	149,64	148,38	**1967,69**	81,95	294,19
Lung-cancer	***0,05***	**0,06**	**0,06**	***0,05***	**0,06**	**0,06**	**0,06**	***0,05***	**0,06**	0,05	0,02	0,04	**0,06**
NslKdd99	***24,30***	***24,62***	30,45	***24,33***	***26,85***	***25,42***	***21,94***	***21,60***	27,96	24,27	31,30	14,46	**47,37**
Pokerhand	113,25	***92,13***	***90,55***	***87,55***	***95,93***	***84,43***	***81,82***	***83,54***	105,32	61,90	**689,22**	29,87	75,00
Pokerhand-1M	183,64	***129,78***	***125,88***	***126,00***	***163,13***	***123,44***	***126,20***	***130,43***	181,06	98,87	**4267,56**	64,24	159,40
WineRed	***0,56***	***0,53***	***0,45***	***0,51***	***0,56***	***0,52***	***0,54***	***0,49***	0,70	0,41	0,15	0,24	**0,71**
Wine-Red-100	**0,11**	***0,09***	0,10	***0,08***	0,10	***0,09***	0,10	***0,09***	0,10	0,09	0,03	0,06	0,08
Usenet-2	***0,89***	0,92	***0,91***	***0,86***	***0,86***	***0,85***	***0,90***	***0,82***	0,92	0,86	0,26	0,53	**1,34**
Sensor	***1028,25***	***693,5***	***678,79***	***695,9***	***1042,65***	***635,06***	***644,64***	***654,65***	1153,36	611,23	**86448,79**	462,04	1030,79
AVG. Rank	***9,00***	***7,64***	***7,86***	***5,55***	***9,00***	***5,59***	***6,86***	***4,73***	*10,82*	4,18	8,27	**1,36**	10,14
Connect-4	***1151,28***	***1212,34***	***1338,43***	***1228,13***	***1186,48***	***1143,03***	1457,71	***1164,75***	1346,06	929,01	164,17	295,55	**3198,56**
Covertype	14924,32	14824,90	***13925,86***	***13803,74***	***13168,50***	***11352,52***	***11409,60***	***11338,39***	14282,11	7261,71	23154,64	3680,14	**46966,48**
Covertype-Sorted	17255,42	***16073,27***	***15223,22***	***14817,15***	***16094,48***	***13672,63***	***13437,64***	***13208,21***	17184,39	16159,50	54221,50	4748,20	**76513,24**
Lung-cancer	***4,18***	***5,23***	***5,17***	***4,20***	***5,30***	***5,15***	***5,06***	***4,62***	**6,10**	2,87	0,30	1,71	3,48
NslKdd99	***2282,42***	***2280,51***	2819,95	***2253,88***	***2550,01***	***2354,43***	***2031,90***	***2001,02***	2786,20	2057,07	286,53	643,64	**8962,84**
Pokerhand	***7490,65***	***6199,10***	***6094,90***	***5894,00***	***6788,76***	***5478,53***	***5308,43***	***5419,50***	7692,58	2245,72	**15522,47**	484,08	3289,47
Pokerhand-1M	***14369,67***	***9799,80***	***9504,62***	***9516,03***	***13104,98***	***9266,67***	***9473,83***	***9792,67***	15053,26	4799,41	**233345,74**	1673,47	13552,92
WineRed	**41,57**	**37,42**	**31,58**	**36,12**	**42,66**	**37,05**	**38,44**	**35,04**	**55,65**	17,94	0,51	5,23	52,95
Wine-Red-100	**7,85**	***6,29***	***6,49***	***5,48***	***7,63***	***6,02***	***6,66***	***6,02***	7,82	3,35	0,10	1,21	2,96
Usenet-2	***107,76***	113,51	***111,60***	***106,24***	***101,32***	***98,59***	***103,92***	***95,24***	113,03	99,85	4,87	35,84	**257,24**
Sensor	***186224,84***	***82440,22***	***80698,63***	***82737,44***	***190114,02***	***65610,07***	***66605,46***	***67650,83***	194455,9	50198,99	**14559501,7**	24836,2	214243,76
AVG. Rank	***9,36***	***8,64***	***8,00***	***6,64***	***9,09***	***5,41***	***6,36***	***4,77***	*11,36*	3,64	6,27	**1,55**	9,91

Source: Own elaboration (2019)

5.2.5.3 Analysis with HT as base classifier

Now, we provide the evaluation of the methods considering the five scenarios in terms of accuracy, time and memory assuming that each algorithm used HT as base classifier.

Scenario-1-HT: FASE obtains the best global accuracy followed by FASEB and FASEO. FASE wins in the data-sets Agrawal, Sine and Random-RBF. All FASE variants outperformed FASE in Led and FASEO$_{wv2}$ is the winner in this base. In the base Mixed win DDD, while in Waveform BOLE reach the highest accuracy. The worst performance is presented by DWM. Regarding the other metrics, OzaBag$_D$ and DWM are respectively the fastest algorithm and that demand less memory consumption in this scenario. In these metrics the FASE variants improved its performance in each base, the exception was FASEB in Agrawal that presented greater memory consumption. In this same scenario, DDD has the worst result in relation to time in all the data-sets while in memory it maintains this result except in the Mixed base of 10k with gradual changes where FASE has the worst performance. So DDD is the one that demands more resources in this scenario. See these results in the table 7.

Scenario-2-HT: In this scenario, FASE obtains the best global accuracy followed by FASEB and FASEO. FASE wins in the data-sets Agrawal and Mixed while FASEB improved FASE and reached the best result in the base Random-RBF. Specifically in Led, variants FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB$_{wv1}$, FASEB$_{wv2}$ (winner) and FASEB$_{wv3}$ also outperformed FASE. BOLE wins in Sine and Waveform data-sets. DWM performs the worst in terms of accuracy. In the same way that in the previous scenario, OzaBag$_D$ and DWM are, respectively, the fastest algorithm and and the least in memory consumption. In time, except in Agrawal (FASEO, FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB, FASEB$_{wv1}$) the variants of FASE had better performance than FASE. Respect to the memory usage cost, except in Agrawal (FASEO, FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB, FASEB$_{wv1}$, FASEB$_{wv2}$) all variants of FASE improved its performance. In this same scenario, DWM has the worst results of time in the data-sets Led and Random-RBF and DDD in the others data-sets. In the same way, the experiments showed that it was the method that consumed most memory. In summary DDD is the method that demanded more time and memory in this scenario. See these results in the table 8.

Scenario-3-HT: In this scenario, FASE obtains the best accuracy followed by FASEB and FASEO. FASE wins in the data-sets Agrawal, Mixed and Sine. FASEB improved FASE and reached the best result in the base Random-RBF. In Led absolutely all variants of FASE outperformed its accuracy and FASEB$_{wv2}$ reaches the highest result while BOLE wins in Waveform base. The worst performance in accuracy is presented by DWM. In the same way that in the previous scenario, OzaBag$_D$ and DWM are respectively the fastest algorithm and the least in memory consumption. All variants improved the FASE performance in each base against time and memory. In these last metrics, DDD has the worst result in all data-sets. See these results in the table 9.

Scenario-4-HT: In this scenario, FASE obtains the best result winning in Agrawal and Mixed, followed by FASEB and FASEO. Besides, the variant FASEB$_{wv2}$ wins in Led, and with the exception of FASEO and FASEB the rest surpasses the results of FASE in this base. FASEB improved the results of accuracy in the data-sets Sine and Random-RBF. Regarding the same data-sets FASEO reached the same accuracy of FASE in the first and wins in Random-RBF. Bole wins in the data-sets Sine and Waveform. Regarding the other metrics also OzaBag$_D$ and DWM are respectively the fastest algorithm and the least in memory consumption. In time, except in Agrawal (FASEO, FASEO$_{wv1}$, FASEO$_{wv2}$, FASEO$_{wv3}$, FASEB, FASEB$_{wv1}$) and Sine (FASEO), the FASE variants had better performance than FASE. Regarding this metric, the worst results are DWM (Led and Random-RBF) and DDD in all the remaining data-sets. In memory, except in the Agrawal database, all FASE variants improved its performance in each base. DDD has the worst results in all data-sets. In summary DDD is the method that demanded more time and memory in this scenario. See these results in the table 10.

Scenario-5-HT: In relation to the accuracy, the best result is obtained by BOLE that wins in most data-sets (except Lung-cancer, Wine-Red and Wine-Red-100) followed by DDD and OzaBag$_D$. FASE method achieved the best result in the Lung-cancer base with FASEO and FASEB. DDD method wins in Wine-Red database. In Wine-Red-100 all methods had the same result, outperforming BOLE. In general, in this scenario, several of the proposed methods perform equally or improved FASE in 8 out of 11 chosen data-sets:

FASEO (Connect-4, Covertype-Sorted, Lung-cancer, Pokerhand-1M, Wine-Red-100), FASEO$_{wv1}$ (Covertype-Sorted, Wine-Red-100, Usenet-2, Sensor), FASEO$_{wv2}$ (Connect-4, Covertype-Sorted, Wine-Red, Wine-Red-100, Sensor), FASEO$_{wv3}$ (Wine-Red, Wine-Red-100, Usenet-2, Sensor), FASEB (Connect-4, Covertype-Sorted, Lung-cancer, Pokerhand-1M, Wine-Red-100, Sensor), FASEB$_{wv1}$ (Usenet-2, Sensor, Covertype-Sorted, Wine-Red-100), FASEB$_{wv2}$ (Connect-4, Covertype-Sorted, Wine-Red, Wine-Red-100, Sensor) and FASEB$_{wv3}$ (Wine-Red-100, Usenet-2, Sensor).

Regarding other metrics, OzaBag$_D$ is the fastest algorithm and the least in memory consumption whereas DDD demanded more out of these resources.Specifically, BOLE was the algorithm most delayed in the Wine-Red-100 and Usenet-2 data-sets, DWM in Covertype-Sorted, Pokerhand and Pokerhand-1M. DDD in the others data-sets. Concerning the memory consumption FASE had the worst performance in Wine-Red-100, FASEO in Sensor, FASEB in Pokerhand and DDD in the other data-sets. In this scenario, the proposed algorithms are not always the best in terms of time and memory consumed with respect to FASE. The algorithms and their respective list of data-sets where they do not improve FASE performance with respect to time are the following: FASEO (Covertype, Covertype-Sorted, Lung-cancer, NslKdd99, Pokerhand, Pokerhand-1M, Wine-Red-100, Sensor), FASEO$_{wv1}$ (Covertype-Sorted, Lung-cancer, NslKdd99, Pokerhand-1M), FASEO$_{wv2}$ (Covertype-Sorted, Lung-cancer, NslKdd99), FASEO$_{wv3}$ (Covertype-Sorted,

Lung-cancer, NslKdd99), FASEB (Connect-4, Covertype, Covertype-Sorted, Lung-cancer, Pokerhand, Pokerhand-1M, Usenet-2, Sensor), FASEB$_{wv1}$ (Connect-4, Covertype-Sorted, Lung-cancer), FASEB$_{wv2}$ (Covertype-Sorted, NslKdd99), FASEB$_{wv3}$ (Covertype-Sorted, Lung-cancer, nslKdd99, Pokerhand-1M).

Besides, the algorithms and their respective list of data-sets where they do not outperform FASE regarding memory consumption are the following: FASEO (Covertype, Covertype-Sorted, NslKdd99, Pokerhand, Pokerhand-1M, Sensor), FASEO$_{wv1}$ (Covertype-Sorted, NslKdd99, Pokerhand-1M), FASEO$_{wv2}$ (Covertype-Sorted, Lung-cancer, NslKdd99, Pokerhand-1M), FASEO$_{wv3}$ (Covertype-Sorted, NslKdd99, Pokerhand-1M), FASEB (Connect-4, Covertype, Covertype-Sorted, Lung-cancer, Pokerhand, Pokerhand-1M, Sensor), FASEB$_{wv1}$ (Connect-4, Covertype-Sorted, Lung-cancer, Pokerhand-1M), FASEB$_{wv2}$ (Covertype-Sorted, NslKdd99, Pokerhand-1M), FASEB$_{wv3}$ (Covertype-Sorted, NslKdd99, Pokerhand-1M). See these results in the table 11.

Table 7 – Mean accuracy, run-times and memory in percentage (%) using HT, with 95% confidence intervals in scenarios of 10k size synthetic data-sets and gradual concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	62,19	61,36	61,45	61,08	62,22	61,43	61,49	61,26	**62,80**	60,37	53,00	61,71	61,45
LED	***67,01***	***67,53***	**67,79**	***67,48***	***67,13***	***67,45***	***67,77***	***67,47***	66,90	67,37	37,49	66,32	66,50
Mixed	83,21	83,15	83,00	82,92	83,27	83,22	83,06	82,97	83,42	82,03	79,57	82,46	**83,72**
Sine	82,57	82,40	82,13	82,07	82,68	82,50	82,21	82,16	**82,93**	81,34	78,81	80,46	82,43
Waveform	77,87	77,65	77,69	77,53	77,87	77,67	77,70	77,54	78,15	**80,63**	69,72	78,16	78,49
Random	32,42	31,77	31,88	31,82	32,51	31,77	31,92	31,81	**32,59**	20,57	23,98	31,53	31,66
AVG. Rank	4,75	7,08	6,25	8,83	3,75	6,58	5,33	8,33	**3,17**	9,33	*12,83*	8,83	5,92
Agrawal	***4,81***	***6,30***	***6,40***	***6,38***	***6,65***	***6,27***	***6,36***	***6,37***	6,93	5,90	1,75	3,17	**8,99**
LED	***3,97***	***3,58***	***3,81***	***3,69***	***4,18***	***3,61***	***3,57***	***3,73***	4,50	4,27	2,34	2,02	**7,48**
Mixed	***2,33***	***2,21***	***2,27***	***2,25***	***2,64***	***2,59***	***2,62***	***2,63***	2,67	1,77	0,20	0,60	**2,75**
Sine	***2,46***	***2,39***	***2,40***	***2,42***	***2,55***	***2,45***	***2,39***	***2,40***	2,94	1,90	0,24	0,80	**3,17**
Waveform	***3,29***	***3,29***	***3,30***	***3,19***	***3,60***	***3,25***	***3,12***	***3,22***	3,93	4,58	1,02	1,64	**6,61**
Random	***5,96***	***5,19***	***5,07***	***5,41***	***5,88***	***5,20***	***5,13***	***5,02***	6,54	8,12	7,27	3,41	**12,30**
AVG. Rank	***7,58***	***5,17***	***7,08***	***6,50***	***10,17***	***6,50***	***5,08***	***6,42***	11,50	7,50	2,83	**1,67**	*13,00*
Agrawal	***742,24***	***999,84***	***1013,71***	***1011,16***	1126,49	***1032,65***	***1046,80***	***1046,97***	1093,55	723,93	49,05	302,30	**2372,29**
LED	***666,66***	***618,97***	***658,04***	***638,17***	***731,21***	***599,38***	***591,80***	***618,26***	840,52	584,39	24,29	149,59	**2255,90**
Mixed	***282,18***	***276,08***	***283,18***	***280,57***	***338,66***	***319,33***	***323,44***	***324,71***	**373,17**	131,40	1,14	17,10	359,85
Sine	***314,69***	***313,13***	***315,23***	***317,25***	***344,54***	***317,76***	***310,87***	***310,88***	429,57	147,94	1,62	29,52	**460,49**
Waveform	***495,74***	***518,02***	***520,40***	***503,29***	***575,33***	***502,71***	***483,07***	***498,54***	670,35	606,75	11,46	94,90	**1790,82**
Random	***1259,72***	***1112,31***	***1087,34***	***1160,19***	***1264,16***	***1083,33***	***1068,51***	***1046,00***	1478,47	1808,25	95,18	371,82	**6312,75**
AVG. Rank	***6,67***	***6,17***	***7,67***	***7,17***	***10,83***	***7,00***	***5,50***	***6,50***	11,83	5,83	**1,00**	2,00	*12,83*

Source: Own elaboration (2019)

Table 8 – Mean accuracies, run-times and memory in percentage (%) using HT, with 95% confidence intervals in scenarios of 50k size synthetic data-sets and gradual concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	71,03	69,34	69,72	69,18	70,99	69,52	69,80	69,34	**71,66**	67,60	53,14	70,67	66,71
LED	72,18	***72,52***	***72,57***	***72,49***	72,20	***72,52***	**72,58**	***72,50***	72,41	72,47	32,36	70,81	70,22
Mixed	90,97	90,77	90,60	90,51	91,01	90,83	90,65	90,59	**91,08**	90,59	82,55	90,44	90,08
Sine	90,80	90,20	90,21	89,84	90,84	90,21	90,22	89,84	90,91	**90,98**	82,42	90,41	89,45
Waveform	81,45	80,54	80,70	80,31	81,42	80,61	80,79	80,38	81,46	**82,18**	56,64	81,47	80,48
Random	***33,54***	32,55	32,64	32,74	**33,57**	32,53	32,67	32,71	33,38	19,97	22,61	33,35	32,67
AVG. Rank	4,17	7,50	6,42	8,92	3,83	6,83	5,25	8,25	**3,00**	6,92	*12,83*	6,17	10,92
Agrawal	29,20	29,92	29,08	27,23	28,01	27,23	***26,07***	***24,47***	26,24	21,82	11,25	19,29	**37,55**
LED	***16,39***	***15,97***	***15,81***	***14,87***	***17,32***	***14,41***	***15,14***	***15,16***	17,75	15,62	**36,76**	8,82	28,42
Mixed	***11,89***	***10,88***	***10,94***	***10,74***	***12,36***	***10,88***	***10,94***	***11,27***	12,66	7,46	2,16	4,21	**14,25**
Sine	***13,29***	***12,47***	***11,97***	***11,55***	***12,81***	***11,64***	***11,37***	***11,12***	13,58	8,01	2,39	5,00	**15,27**
Waveform	***18,43***	***16,77***	***16,53***	***17,42***	***18,93***	***16,51***	***17,03***	***17,04***	19,16	16,87	16,08	10,42	**33,95**
Random	***22,79***	***20,07***	***21,05***	***19,92***	***22,85***	***19,61***	***18,93***	***19,19***	24,94	26,65	**100,23**	14,36	46,63
AVG. Rank	***9,83***	***7,58***	***7,25***	***5,75***	***10,00***	***4,83***	***5,08***	***5,50***	10,50	5,33	5,17	**1,50**	*12,67*
Agrawal	8796,79	9075,73	8827,24	8250,35	8621,63	8202,73	7842,56	***7365,69***	7470,72	4533,28	393,31	4282,76	**19994,09**
LED	***2774,71***	***2739,29***	***2712,66***	***2550,98***	***3061,21***	***2385,58***	***2506,93***	***2509,83***	3308,56	2143,18	572,78	695,26	**8803,73**
Mixed	***1711,26***	***1542,66***	***1550,40***	***1522,58***	***1871,52***	***1532,70***	***1540,06***	***1587,85***	1988,15	610,02	27,50	206,48	**2392,06**
Sine	***2035,80***	***1882,49***	***1807,92***	***1743,26***	***2047,40***	***1740,57***	***1700,23***	***1661,92***	2251,01	725,40	40,04	289,99	**2831,16**
Waveform	***3727,69***	***3475,69***	***3426,64***	***3609,88***	***4033,69***	***3369,21***	***3474,93***	***3475,63***	4038,40	2553,51	387,29	1236,91	**15312,95**
Random	***4842,07***	***4438,78***	***4656,57***	***4406,39***	***4998,28***	***4068,04***	***3928,41***	***3981,27***	5630,01	5998,20	2008,13	1759,51	**24684,60**
AVG. Rank	***9,83***	***8,67***	***8,00***	***6,83***	***10,50***	***5,17***	***5,17***	***5,67***	10,67	4,50	**1,17**	1,83	*13,00*

Source: Own elaboration (2019)

Table 9 – Mean accuracy, run-times and memory in percentage (%) using HT, with 95% confidence intervals in scenarios of 10k size synthetic data-sets and abrupt concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	64,42	63,59	63,62	63,29	64,29	63,47	63,49	63,13	**64,88**	61,74	53,53	63,30	62,09
LED	***69,05***	***69,35***	***69,68***	***69,44***	***69,06***	***69,32***	**69,70**	***69,43***	68,47	68,71	40,39	68,31	67,39
Mixed	89,50	89,47	89,17	89,28	89,52	89,54	89,24	89,33	**89,88**	88,73	85,07	81,23	87,98
Sine	88,12	87,91	87,60	87,56	88,18	87,97	87,56	87,58	**88,50**	88,30	84,05	86,69	85,97
Waveform	78,78	78,57	78,59	78,45	78,75	78,55	78,57	78,43	78,94	**80,86**	70,64	78,63	78,84
Random	32,45	31,87	31,99	31,97	**32,58**	31,83	31,98	31,88	32,50	20,54	24,01	31,67	31,49
AVG. Rank	4,17	6,25	5,50	7,58	3,67	6,50	6,33	7,83	**2,83**	7,83	*12,67*	9,83	10,00
Agrawal	***3,94***	***5,21***	***6,10***	***6,01***	***6,47***	***5,94***	***5,92***	***5,89***	6,76	5,75	1,74	3,16	**8,36**
LED	***4,40***	***4,09***	***4,09***	***3,74***	***4,04***	***3,64***	***3,62***	***3,52***	4,42	4,39	1,96	2,12	**7,36**
Mixed	***2,49***	***2,30***	***2,36***	***2,29***	***2,40***	***2,39***	***2,23***	***2,25***	2,68	1,84	0,21	0,75	**3,20**
Sine	***2,97***	***2,86***	***2,49***	***2,55***	***2,73***	***2,51***	***2,49***	***2,46***	2,99	1,91	0,23	0,88	**3,23**
Waveform	***3,33***	***3,31***	***3,26***	***3,18***	***3,46***	***3,19***	***3,17***	***3,16***	3,92	4,69	0,96	1,59	**6,48**
Random	***5,83***	***5,34***	***5,14***	***5,16***	***5,98***	***5,11***	***5,24***	***5,01***	6,77	8,19	7,22	3,33	**12,88**
AVG. Rank	***8,83***	***7,42***	***7,17***	***6,50***	***9,33***	***6,33***	***5,08***	***3,83***	11,50	7,50	2,67	**1,83**	*13,00*
Agrawal	***552,20***	***762,04***	***892,51***	***879,17***	***966,40***	***858,18***	***854,45***	***851,18***	1051,06	654,80	44,54	284,38	**1568,18**
LED	***753,71***	***695,77***	***695,99***	***636,61***	***705,16***	***602,70***	***598,71***	***581,47***	821,48	596,07	20,26	158,04	**2222,48**
Mixed	***318,55***	***288,24***	***295,60***	***287,17***	***310,83***	***297,65***	***277,18***	***279,54***	376,20	138,09	1,14	24,12	**424,64**
Sine	***404,25***	***386,02***	***336,77***	***343,95***	***380,27***	***336,06***	***333,34***	***330,32***	447,57	151,29	1,55	36,37	**486,30**
Waveform	***503,28***	***521,60***	***514,14***	***500,89***	***549,44***	***491,31***	***486,95***	***486,78***	666,71	621,83	10,50	92,05	**1762,07**
Random	***1226,21***	***1145,13***	***1102,82***	***1106,19***	***1286,80***	***1065,64***	***1093,49***	***1045,67***	1530,75	1823,04	94,47	362,85	**6604,69**
AVG. Rank	***8,67***	***7,83***	***8,00***	***7,17***	***10,00***	***6,33***	***5,00***	***4,00***	11,83	6,17	**1,00**	2,00	*13,00*

Source: Own elaboration (2019)

Table 10 – Mean accuracies, run-times and memory in percentage (%) using HT, with 95% confidence intervals in scenarios of 50k size synthetic data-sets and abrupt concept drifts.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Agrawal	72,18	70,49	70,96	70,31	72,18	70,54	70,98	70,34	**72,87**	67,90	53,20	71,11	66,09
LED	72,52	***72,79***	***72,85***	***72,75***	72,52	***72,80***	**72,87**	***72,75***	72,73	72,70	32,61	70,78	70,04
Mixed	92,35	92,09	91,90	91,86	92,34	92,10	91,91	91,88	**92,56**	91,89	84,65	84,99	90,16
Sine	***91,98***	91,35	91,39	91,00	***92,01***	91,36	91,38	91,01	91,98	**92,69**	83,91	90,34	89,45
Waveform	81,48	80,81	80,97	80,63	81,53	80,82	80,97	80,64	81,59	**82,25**	56,50	81,68	80,43
Random	**33,64**	32,59	32,75	32,71	***33,60***	32,52	32,72	32,67	33,40	19,99	22,61	33,40	32,69
AVG. Rank	3,92	7,33	5,25	8,92	3,83	6,67	5,08	8,58	**3,17**	7,00	*12,83*	7,25	11,17
Agrawal	29,32	29,73	29,41	27,03	27,90	26,81	***25,13***	***24,82***	25,62	21,83	11,25	18,73	**36,35**
LED	***16,48***	***15,51***	***15,78***	***15,17***	***17,19***	***14,37***	***14,50***	***15,02***	17,65	16,14	**36,49**	9,11	28,52
Mixed	***12,57***	***11,06***	***11,13***	***10,97***	***12,37***	***10,99***	***11,19***	***10,92***	12,83	7,61	1,98	5,15	**13,42**
Sine	13,90	***12,64***	***12,45***	***11,77***	***12,88***	***11,54***	***11,67***	***11,59***	13,52	8,30	2,23	5,95	**14,79**
Waveform	***17,52***	***16,12***	***16,24***	***16,80***	***18,32***	***16,48***	***16,05***	***16,12***	19,32	17,42	16,21	9,76	**34,01**
Random	***22,86***	***20,10***	***19,90***	***19,68***	***22,28***	***19,78***	***19,32***	***18,56***	24,37	26,80	**100,11**	14,63	47,61
AVG. Rank	***10,17***	***7,42***	***7,67***	***6,17***	***9,67***	***5,17***	***4,67***	***3,75***	10,33	6,17	5,67	**1,50**	*12,67*
Agrawal	8722,60	8910,98	8827,89	8102,17	8559,66	8021,79	7506,66	7406,20	7217,44	4358,32	403,93	4040,18	**17308,40**
LED	***2828,22***	***2645,38***	***2691,20***	***2587,51***	***3039,36***	***2378,55***	***2401,54***	***2487,20***	3287,71	2221,89	566,47	786,18	**8770,38**
Mixed	***1861,65***	***1575,09***	***1584,41***	***1561,71***	***1886,63***	***1557,54***	***1586,12***	***1547,96***	2032,63	623,11	25,69	294,70	**2135,13**
Sine	***2191,53***	***1927,77***	***1899,02***	***1794,73***	***2087,42***	***1750,00***	***1770,13***	***1758,16***	2272,88	755,57	35,57	412,05	**2675,77**
Waveform	***3471,72***	***3214,97***	***3241,36***	***3351,41***	***3789,70***	***3253,71***	***3167,09***	***3182,34***	4005,91	2620,50	375,98	1083,58	**15300,57**
Random	***4844,64***	***4409,44***	***4365,99***	***4317,76***	***4865,37***	***4113,51***	***4016,34***	***3858,76***	5497,76	6026,92	2010,10	1781,49	**25189,45**
AVG. Rank	***10,00***	***8,33***	***8,33***	***7,17***	***10,33***	***5,50***	***5,67***	***4,67***	10,50	4,50	**1,17**	1,83	*13,00*

Source: Own elaboration (2019)

Table 11 – Mean accuracies, run-times and memory in percentage (%) using HT in real-world data-sets.

DATA-SET	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
Connect-4	***75,01***	74,89	***75,13***	74,50	***75,06***	74,78	***74,96***	74,44	74,94	**77,65**	70,43	75,44	75,14
Covertype	85,17	84,89	84,54	83,65	85,24	84,80	84,45	83,67	87,84	**92,58**	86,34	86,14	85,27
Covertype-Sorted	***74,08***	***72,13***	***72,71***	71,73	***73,70***	***72,46***	***73,28***	71,95	72,04	**80,80**	61,68	73,35	78,94
Lung-cancer	**74,74**	67,29	66,46	67,29	**74,74**	67,29	66,46	67,29	**74,74**	46,25	70,99	69,74	69,74
NslKdd99	98,62	98,35	98,47	98,41	98,62	98,35	98,47	98,41	98,67	**99,15**	89,50	98,67	98,67
Pokerhand	75,87	75,05	75,63	74,95	75,85	75,24	75,76	75,07	76,25	**80,39**	71,17	77,27	77,20
Pokerhand-1M	***54,40***	50,36	52,69	52,81	***54,30***	50,36	52,69	52,81	53,32	**54,64**	47,56	52,08	52,73
WineRed	48,57	50,13	***54,41***	***54,25***	49,06	50,66	***54,24***	53,79	54,02	34,17	51,37	55,25	**55,72**
Wine-Red-100	**60,46**	**60,46**	**60,46**	**60,46**	**60,46**	**60,46**	**60,46**	**60,46**	**60,46**	23,08	**60,46**	**60,46**	**60,46**
Usenet-2	66,53	***68,07***	67,79	***68,10***	66,56	***68,07***	67,79	***68,10***	67,95	**73,72**	69,61	70,06	71,62
Sensor	85,59	***89,39***	***88,43***	***86,71***	***86,06***	***89,55***	***88,84***	***87,71***	86,03	**93,30**	10,24	87,37	85,84
AVG. Rank	6,73	8,68	7,41	8,68	6,36	8,41	7,68	8,50	5,86	**4,27**	*9,41*	4,55	4,45
Connect-4	***21,87***	***22,60***	***23,61***	***22,21***	27,94	33,10	***26,91***	***23,55***	27,08	28,87	28,68	12,94	**44,40**
Covertype	305,10	***243,13***	***238,25***	***235,87***	300,65	***236,74***	***234,90***	***236,89***	291,42	289,93	715,33	139,81	**1306,58**
Covertype-Sorted	372,31	269,34	278,12	267,18	333,73	262,40	268,59	275,65	245,78	695,44	**2028,78**	194,93	1777,37
Lung-cancer	0,11	0,11	0,14	0,12	0,14	0,17	***0,10***	0,12	0,11	0,17	0,11	0,11	**0,19**
NslKdd99	1256,79	1218,50	1107,60	1137,01	***1052,11***	***1013,71***	1155,79	1109,51	1069,80	872,02	73,17	1022,49	**1800,30**
Pokerhand	250,11	***186,66***	***181,73***	***179,88***	305,92	***174,06***	***173,16***	***174,68***	219,25	195,44	**591,70**	65,64	281,77
Pokerhand-1M	605,85	***472,22***	***427,66***	***435,17***	559,44	***421,31***	***433,76***	445,22	438,03	602,82	**2722,52**	170,59	878,59
WineRed	***0,93***	***0,88***	***0,92***	***0,90***	***1,01***	***0,94***	***0,98***	***1,00***	1,09	1,19	0,33	0,50	**1,78**
Wine-Red-100	0,18	***0,15***	***0,15***	***0,17***	***0,14***	***0,15***	***0,17***	***0,17***	0,18	**0,24**	0,12	0,12	0,21
Usenet-2	***1,51***	***1,22***	***1,37***	***1,38***	1,70	***1,56***	***1,35***	***1,40***	1,70	**2,92**	0,46	1,16	2,83
Sensor	4697,27	***1085,96***	***1059,98***	***1076,79***	4547,49	***1093,21***	***1063,68***	***1078,59***	1384,75	780,05	**56546,83**	517,74	1443,71
AVG. Rank	8,77	***5,91***	***5,77***	***5,41***	8,91	***6,14***	***5,00***	***6,68***	7,73	9,23	7,50	**1,77**	*12,18*
Connect-4	***3760,40***	***4226,63***	***4416,89***	***4146,17***	5288,07	5786,37	***4733,45***	***4148,14***	5063,96	4032,48	1626,43	1083,89	**11990,23**
Covertype	72088,00	***47528,08***	***46554,03***	***46100,86***	74921,94	***42180,81***	***41826,03***	***42175,00***	56102,22	41423,64	45307,81	11704,97	**2570360,54**
Covertype-Sorted	146964,64	72541,26	74750,39	71849,16	106601,91	71232,33	72695,76	74807,24	53814,88	663628,84	182101,00	47578,00	**3901540,80**
Lung-cancer	***18,65***	***17,60***	21,21	***18,68***	22,65	26,27	***15,36***	***19,19***	19,24	20,85	1,53	6,93	**26,86**
NslKdd99	5166555,60	5028954,69	4570509,10	4675465,26	***4348380,11***	***4187345,36***	4770694,63	4577496,12	4446866,18	3299071,03	18472,74	4134736,86	**12202739,37**
Pokerhand	48497,48	***25811,27***	***25132,96***	***24873,41***	**68602,04**	***23175,13***	***23056,06***	***23258,02***	32996,33	19806,07	32433,41	2444,08	45235,16
Pokerhand-1M	207434,13	134939,75	122243,56	124264,47	199333,39	120143,08	123797,02	127182,35	116753,25	291623,18	219574,74	18407,00	**557537,17**
WineRed	***139,03***	***127,14***	***134,00***	***130,59***	***149,65***	***133,51***	***139,21***	***141,13***	175,56	124,48	1,73	23,00	**324,57**
Wine-Red-100	***26,46***	***20,95***	***20,27***	***24,14***	***19,81***	***20,54***	***23,75***	***22,94***	**27,81**	23,80	0,61	5,41	24,37
Usenet-2	***293,38***	***241,33***	***271,78***	***272,53***	***339,78***	***303,38***	***263,89***	***273,18***	355,48	556,74	9,79	113,28	**934,34**
Sensor	***15785186,5***	***200861,4***	***196070,07***	***199154,22***	***14604968***	***186546,63***	***181493,67***	***184029,13***	346770,05	108510,74	11013345,4	39746,14	581631,76
AVG. Rank	9,45	***6,64***	***6,64***	***6,45***	9,64	***6,18***	***5,73***	***6,64***	8,45	6,27	5,00	**1,55**	*12,36*

Source: Own elaboration (2019)

In the next sub-section we will analyze the particular behavior of FASE, its variants and the other methods, in detail, to determine if there are significant statistical differences between them in terms of the addressed metrics.

5.2.5.4 Statistical Analysis

In order to conduct an statistical analysis of the methods studied, the *Friedman* test (FRIEDMAN, 1937; DEMŠAR, 2006) was applied to identify which of the ensembles obtained better results regarding accuracy, run-time and memory. The tests were used with a significance level of 5% and the total of the experiment that was taken into consideration was 35, corresponding to the test performed by base classifier (NB or HT) in the 5 scenarios of experimentation. That is, to evaluate the methods in each out of 35 synthetic and real data-sets.

More precisely, the *Friedman* test is used to decide if the difference in the observed average rank of a specif metric provided by the algorithms are statistically significant at the level of 5%. The rank results established by the statistical method provide the criterion for evaluating the ensembles, considering the lower value of the calculated rank will be the better result in this approach.

Moreover, as the rejection of the null hypothesis of the *Friedman* test indicates that the observed differences between the methods are globally statistically significant but does not define which of them are statistically significant, the *Nemenyi* post-test (NEMENYI, 1963; DEMŠAR, 2006) was applied. According to this test, the performance of two algorithms is considered significantly different if their corresponding average ranks differ by at least a suitable critical difference (Appendix. C explains the tests with more detail). The critical difference (CD) determined by the *Nemenyi* post-test was 3,0842.

Table. 12 present the average ranks of accuracy, run time and memory consumed by each methods using the classifiers NB and HT in all scenarios of Concept Drift. The best measurements were determined by the lower values indicated in bold. Regarding the proposed methods in relation to FASE, the best measurements were indicated in bold and italic. On the other hand, the worst results were highlighted italics.

In addition, Figures 7 to 9 showed the best measurements located in the first positions.

Particularly, Figures 7(a) and 7(b) show a comparison of the methods accuracies using the *Friedman* and *Nemenyi* tests in all scenarios of concept drifts using the classifiers NB and HT respectively.

Figures 8(a) and 8(b) show a comparison of the methods run-time using the *Friedman* and *Nemenyi* tests in all scenarios of concept drifts using the classifiers NB and HT respectively.

Finally, Figures 9(a) and 9(b) show a comparison of the memory consumed by the methods using the *Friedman* and *Nemenyi* tests in all scenarios of concept drifts using the classifiers NB and HT respectively.

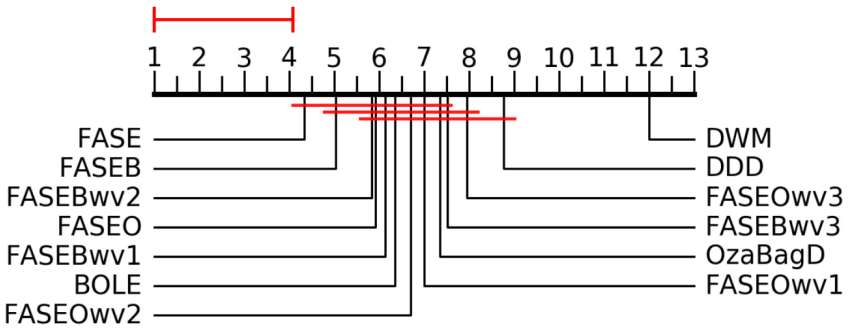
Table 12 – Summary of average ranks of accuracy, run time and memory consumed by each methods using both classifiers NB and HT in all scenarios.

AVG. Ranks	FASEO	FASEOwv1	FASEOwv2	FASEOwv3	FASEB	FASEBwv1	FASEBwv2	FASEBwv3	FASE	BOLE	DWM	OzaBagD	DDD
AVG. Ranks_acc (NB)	5,93	7,01	6,70	7,96	5,04	6,14	5,84	7,53	**4,34**	6,36	*12,01*	7,36	8,77
AVG. Ranks_acc (HT)	5,03	7,56	6,34	8,60	4,59	7,20	6,19	8,33	**3,93**	6,67	*11,73*	6,93	7,91
AVG. Ranks_time (NB)	***9,57***	***7,76***	***7,81***	***6,96***	***10,33***	***5,54***	***5,91***	***5,07***	*11,69*	3,80	6,26	**1,46**	8,84
AVG. Ranks_time (HT)	***9,00***	***6,59***	***6,81***	***5,97***	***9,51***	***5,84***	***4,99***	***5,44***	9,94	7,44	5,16	**1,67**	*12,63*
AVG. Ranks_memory (NB)	***9,83***	***8,56***	***8,30***	***7,77***	***10,57***	***5,64***	***5,86***	***5,16***	*11,97*	3,89	2,77	**1,74**	8,94
AVG. Ranks_memory (HT)	***9,00***	***7,40***	***7,57***	***6,89***	***10,17***	***6,06***	***5,46***	***5,66***	10,34	5,57	2,31	**1,80**	*12,77*

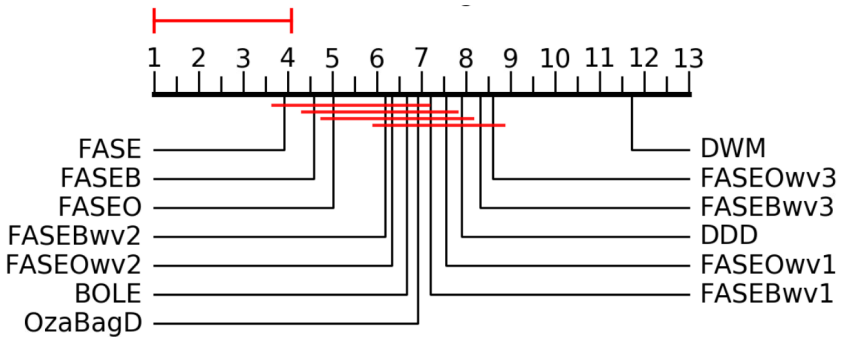
Source: Own elaboration (2019)

Figure 7 – Comparison of methods accuracies using the *Friedman* and *Nemenyi* tests with a 5% significance level.

(a) Acc_NB



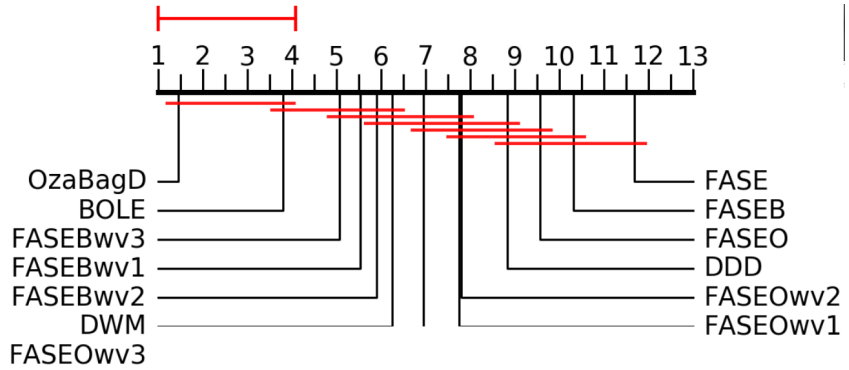
(b) Acc_HT



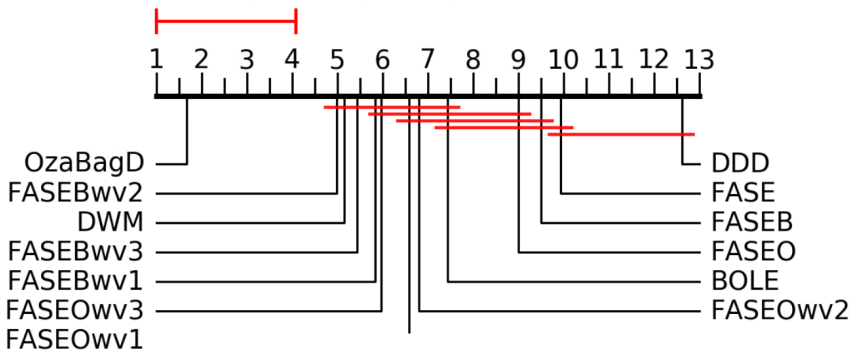
Source: Own elaboration (2019)

Figure 8 – Comparison of methods run-time using the *Friedman* and *Nemenyi* tests with a 5% significance level.

(a) Run-time_NB



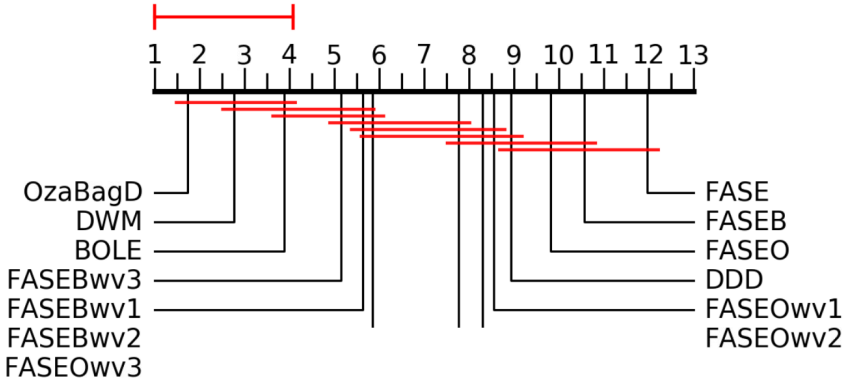
(b) Run-time_HT



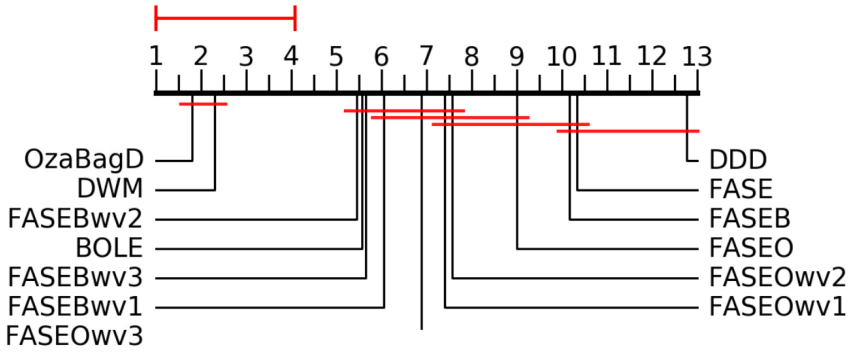
Source: Own elaboration (2019)

Figure 9 – Comparison of methods memory using the *Friedman* and *Nemenyi* tests with a 5% significance level.

(a) Memory_NB



(b) Memory_HT



Source: Own elaboration (2019)

Regarding accuracy, the best ranked method was FASE. With NB as base classifier, FASE is significantly better than FASEO$_{wv3}$, FASEB$_{wv3}$, DDD and DWM. With respect to the others methods the observed differences were not statistically significants. Besides, when HT was used as base classifier, FASE exceeded significantly the results of FASEO$_{wv1}$, FASEB$_{wv1}$, FASEO$_{wv3}$, FASEB$_{wv3}$, DWM and DDD. DWM is the worst ranked with both classifiers, and significantly inferior to all methods (except FASEO$_{wv3}$ with HT).

Concerning run-time, for both NB and HT as base classifier, the best ranked algorithm was OzaBag$_D$. With NB as base classifier, there was no significant difference between this method and BOLE, all the others observed differences were significant. Besides, with HT as base classifier, OzaBag$_D$ significantly outperformed all the others methods. Finally, also for both NB and HT as base classifier, FASEO, FASEB, FASE and DDD, are more time consuming methods. With the classifier NB, FASE is the worst ranked and significantly less fast with respect to all methods (except FASEO, FASEB and DDD). Moreover, with HT as base classifier, DDD is the worst ranked method and significantly less fast respect to the others (except FASE).

Memory consumption was also analyzed, OzaBag$_D$ and DWM were the algorithms with the least memory consumption with both classifiers. Specifically with NB, these algorithms also have no significant difference with respect to BOLE. FASE is the worst ranked with the classifier NB and significantly demanded more memory respect to all methods (except FASEO, FASEB and DDD). Besides, with HT as base classifier, DDD is the worst ranked method and significantly demanded more memory with respect to the others (except FASE and FASEB).

It can be observed that both FASEO and FASEB, which have the meta-classifier as a class voting mechanism, perform better in terms of accuracy, but they are slower and demand more memory than the other variants. With variants that use weighted voting the opposite happens, they present less favorable results in accuracy but they also demand fewer resources.

Thus, two main remarks can be made:

- FASEO, FASEB, FASEB$_{wv2}$,FASEO$_{wv2}$ did not have significant accuracy loss with respect to the original algorithm.
- The implemented versions of the FASE are significantly faster and require less memory, with the exception of FASEB and FASEO that did not present significant improvements concerning these metrics.

Therefore, the versions FASEB$_{wv2}$ and FASEO$_{wv2}$ at the same time that they did not present significant losses of accuracy are noticeably faster and consume less memory than the original algorithm.

5.3 FINAL CONSIDERATION

In this chapter, theoretical analysis of the FASEO and FASEB families was performed. The space complexity of both families was similar regarding FASE. The temporal complexity was computed taking into account three situations: Create/Activate a base leaner, Update the base learners and Update the meta-classifier. The result of the time complexity analysis of both families was similar regarding FASE, even when in order to Update the base learners FASEB family can be more complex, in the most of situations it is insignificant. The variants without meta-classifier have inferior time complexity than FASE. In addition, the MOA framework and Moa Manager tool were used in the empirical evaluation of the proposed methods. As part of the empirical study, it was described the characteristics of the 24 synthetic and 11 real data-sets used in order to evaluate the proposed methods against the previous state of art methods. The configuration of the experiments and the parameters used was detailed. In order to make a fair evaluation, all the methods were left with their default setting, at the same time they were compared with the same number and classifier algorithms. Finally, the results obtained by each scenario were presented and discussed. Overall, FASE presented the best accuracy in the synthetic data-set. FASEB, in Scenario 3 with NB was better ranked than FASE. In each scenario, DWM has the worst performance. The other methods (BOLE, DWM, OzaBag_D, DDD) are surpassed by the proposed methods in most scenarios. In Scenario 5, corresponding to the real data-sets, BOLE showed the best result with both NB and HT. In this scenario the methods FASEO_{wv2}, FASEB_{wv2} and FASEB performed better than FASE in 6 out of 11 data-set using NB (in the two first) and HT respectively. OzaBag_D was the algorithm that presented the best performance in terms of run-time using NB and HT as base classifiers in each scenario, while FASE and DDD were the worst, respectively. Implemented variants performed better than FASE, with a lower ranking in all scenarios. The exception was with HT classifier in Scenario 5, corresponding to real data sets, where FASEO and FASEB performed slower than FASE. Regarding consumed memory, when the methods used NB and HT, in the first 4 scenarios DWM presented the best performance followed by OzaBag_D. In Scenario 5, OzaBag_D was the algorithm with the best results. When the experiments were performed with the NB classifier, FASE presented the worst results in all 5 scenarios, in Scenario 3, the same as FASEB. When HT was the base classifier, DDD got the worst results in all scenarios. Only in Scenario 5, FASEO and FASEB consumed more memory than FASE. To determine the relevance of these results it was used the statistical test of Friedman and the Nemenyi post-test. As an interesting finding, FASEO_{wv2} and FASEB_{wv2}, although not showing significant accuracy losses were notably faster and consume less memory than the original algorithm FASE.

6 CONCLUSIONS

This work proposed FASEB and FASEO families of algorithms, a total of 8 new ensemble methods for operation in concept drift scenarios with full access to labeled classes. The new algorithms are variants of the FASE ensemble (FRIAS-BLANCO et al., 2016).

The main difference of FASEB and FASEO families as compared to FASE is the update strategy employed by the algorithms. While FASE uses adaptive classifiers to keep the ensemble updated, the implemented algorithms have in common a parallel ensemble formed by alternative classifiers, activated and set to be trained when one of the classifiers in the main ensemble issues a warning; otherwise, the alternative classifiers remain inactive.

When a Concept Drift is detected, algorithms in the FASEB family boosts the alternative classifier with the greatest accuracy. Algorithms in the FASEO family, instead, promote the oldest active alternative classifier.

On the other hand, classification is also performed differently in some of the proposed algorithms. FASE uses a meta-classifier to predict the final class of the unlabeled instance as well as FASEO and FASEB methods. The other proposed variants, however, FASEB$_{wv1}$, FASEB$_{wv2}$, FASEB$_{wv3}$ and FASEO$_{wv1}$, FASEO$_{wv2}$ FASEO$_{wv3}$ modify the voting strategy using weighted voting.

The proposed variants were compared to FASE and other state-of-the-art ensemble classifiers through similar parametrization and same testing conditions in order to accordingly evaluate their performance, using HT and NB as base learners. Twenty four (24) synthetic data-sets with abrupt and gradual drifts and eleven (11) real-world data-sets were employed for experimental evaluation. In terms of accuracy, FASE obtained the best results in most of the tested scenarios using HT and NB, but it was noticed a very close approximation of FASEB as compared to those of FASE. DWM had the worst performance both in the synthetic and real data-sets among all the algorithms.

In general, the proposed methods achieved a better result in synthetic data-sets generated from LED, with both abrupt and gradual changes with the two possibles sizes. In this case, the methods that use weighted majority voting to perform classification presented the best accuracy, especially FASEO$_{wv2}$ and FASEB$_{wv2}$. FASEB method had a good performance using NB as a base classifier in the synthetic data-sets with gradual changes. At the same time, it was better ranked than FASE in Scenario 3. Also, the proposed algorithms presented better accuracy results than FASE in most of the selected real data-sets. In particular, FASEO$_{wv2}$ and FASEB$_{wv2}$ when the methods use NB as base classifier and FASEB when the methods use HT as base classifier.

Among the new proposals, FASEB family performed better than FASEO methods

concerning the considered metrics. In most (artificial) scenarios, variants that use a meta-classifier (FASEO and FASEB) have better accuracy results, but are slower and consume more memory than other variants. However, variants that use Weighted Majority Vote (FASEO_{wv1}, FASEO_{wv2}, FASEO_{wv3}, FASEB_{wv1}, FASEB_{wv2}, FASEB_{wv3}) have less accuracy, but also require fewer resources (time and memory). Regarding the variants that use Weighted Majority Vote, FASEO_{wv2} and FASEB_{wv2} that apply approach 2, they have the best performance with respect to accuracy. However, FASEO_{wv3} and FASEB_{wv3} that apply approach 3, are faster and consume less memory.

Moreover, run-time and memory were improved by the proposed methods with respect to FASE in the most part of the sudden and gradual change scenarios, on synthetic data-sets. This was also the case with the real data-sets, although the improvements were less important. But, the better performance in this sense was reached by OzaBag_D method, while the worst results corresponded to the FASE and DDD methods.

The statistical significance of the results provided by the experiments were evaluated using the non-parametric *Friedman* test together with the *Nemenyi* post-test. Those tests confirmed the proposed algorithms were often significantly better than FASE with respect to run-time and memory. In particular, versions FASEB_{wv2} and FASEO_{wv2}, while not showing significant accuracy losses, were noticeably faster and consumed less memory than the original FASE algorithm. This can be very useful in contexts that require quick access to partial information, a high level of accuracy is still needed, but a very fast decision has to be made.

6.1 MAIN CONTRIBUTIONS

Based on above observations, the main contributions of this work can be summarized as follows:

- The proposed variants of FASE provide improvements with respect to run-time and memory consumption when compared to FASE, while keeping competitive performance regarding accuracy in some of the variations.

- Particularly, the adaptations in the strategy **of updating the set and the voting mechanism of the classes** were among the main contributions. This resulted in two families of algorithms FASEO and FASEB, designed to work on large data-streams and to adapt quickly to sudden and gradual changes of concepts more efficiently. We started from FASE, because it showed better accuracy results in most scenarios, in previous works. However, when observing other metrics, the opposite happened. Therefore, our objective was to balance the behavior between several metrics without having considerable losses in the accuracy achieved.

6.2 SECONDARY CONTRIBUTIONS

Also, secondary, but important contributions of this research, can be pointed out:

- The carry out study of the state of the art describing concepts of classification algorithms and ensemble methods designed to handle gradual and abrupt Concept Drifts on large data-streamings.
- Characterization of some of the main models of incremental learning adopted for the processing of large data-streams in presence of Concept Drift; their main strengths and weaknesses.
- An analysis of the most used data-sets (synthetic and real) in the evaluation and comparison of incremental algorithms designed to work on large data-streams in presence of Concept Drift. This allowed distinguishing the essential parameters in the evaluation of the algorithms treated: accuracy, run-time and memory used.
- Such as FASE, the families FASEO and FASEB were implemented on the framework MOA in order to make the process of evaluating the algorithms more feasible and less costly. This can be further used by other researchers in future works. This environment include several implementations of classification algorithms and also allows the monitoring of experiment executions in real time. An experimentation was designed taking into account a rigid controlled simulation of both gradual and abrupt Concept Drift. Two base learners were selected: *Naive Bayes* and *Hoeffding Tree*. These learners were chosen because given their broad application in the online classification. Also, several data-sets were constructed using synthetic data generators and real databases were also employed. The new algorithms achieved promising results in the tests, in comparison with state of the art methods also implemented in the MOA environment, taking into account the accuracy, run-time and memory.

An article, with two of the proposed methods (FASEB and FASEB$_{wv3}$), was accepted to be presented and published in the proceedings of the *2019 International Joint Conference on Neural Networks (IJCNN)*, which will take place in Budapest, Hungary on July 14-19, 2019.

- "*Improving Fast Adaptive Stacking of Ensembles*". **Laura M. P. Mariño, Juan I. G. Hidalgo, Roberto S. M. Barros, Germano C. Vasconcelos**.

Other papers will be shortly prepared for submission to other conferences and events.

6.3 FUTURE WORKS

This work can be extended in at least two main directions:

- First, one can explore other aspects of the FASE structure that may allow a better balance between accuracy and other important metrics, which gain relevance according to the context where the algorithm is applied.

- Besides, one can extend this study by introducing the Error-Memory-Run-time (**EMR**) (PESARANGHADER; VIKTOR; PAQUET, 2016) measure aiming to optimize and compare the proposed methods regarded FASE. **EMR** combines error-rate, memory usage and runtime for evaluating and ranking learning algorithms in order to select the "best" model in some domains. In (PESARANGHADER; VIKTOR; PAQUET, 2016) was defined the domain dependent cost weights for error-rate, memory usage and run-time to emphasize their importance to measure the cost of classifiers, where a classifier with the lowest **EMR** is preferred in a multi-strategy learning setting. In real world problems, the values of these three weights will be set to reflect the current domain of application. For instance, a classifier which has been shown to be very slow over the last period of time may be made to fade away.

- Finally, in a similar way one can consider the **CAR** measure (PESARANGHADER; VIKTOR; PAQUET, 2018) which not only considers the classification error-rates, memory usages, and run-times but also the drift detection delays, false positives and false negatives.

REFERENCES

AGGARWAL, C. C. *Data classification: algorithms and applications*. [S.l.]: CRC Press, 2014.

AGGARWAL, R.; IMIELINSKI, T.; SWAMI, A. N. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, v. 5, n. 6, p. 914–925, 1993.

BACH, S. H.; MALOOF, M. A. Paired Learners for Concept Drift. In: *Proceedings of 8th IEEE International Conference on Data Mining (ICDM'08)*. Pisa, Italy: [s.n.], 2008. p. 23–32.

BAENA-GARCIA, M.; Del Campo-Ávila, J.; FIDALGO, R.; BIFET, A.; GAVALDÀ, R.; MORALES-BUENO, R. Early Drift Detection Method. In: *International Workshop on Knowledge Discovery from Data Streams*. [S.l.: s.n.], 2006. p. 77–86.

BARROS, R. S. M.; CABRAL, D. R. L.; Gonçalves Jr, P. M.; SANTOS, S. G. T. C. RDDM: Reactive drift detection method. *Expert Systems with Applications*, Elsevier, v. 90, p. 344–355, 2017.

BARROS, R. S. M.; SANTOS, S. G. T. C. An overview and comprehensive comparison of ensembles for concept drift. *Information Fusion*, Elsevier, v. 52, n. C, p. 213–244, 2019.

BARROS, R. S. M.; SANTOS, S. G. T. C.; GONÇALVES JR., P. M. A Boosting-like Online Learning Ensemble. In: *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN)*. Vancouver, Canada: [s.n.], 2016. p. 1871–1878.

BIFET, A.; GAVALDA, R. Kalman filters and adaptive windows for learning in data streams. In: SPRINGER. *International Conference on Discovery Science*. [S.l.], 2006. p. 29–40.

BIFET, A.; GAVALDÀ, R. Learning from Time-Changing Data with Adaptive Windowing. In: *Proc.˜of 7th SIAM International Conference on Data Mining (SDM'07)*. Minneapolis, MN, USA: [s.n.], 2007. p. 443–448.

BIFET, A.; HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. {MOA}: Massive Online Analysis. *Journal of Machine Learning Research*, MIT Press, v. 11, p. 1601–1604, 2010.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KIRKBY, R.; GAVALDÀ, R. New ensemble methods for evolving data streams. In: *Proceedings of 15th ACM International Conference on Knowledge Discovery and Data Mining (KDD'09)*. Paris, France: [s.n.], 2009. p. 139–148.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; FRANK, E. Fast Perceptron Decision Tree Learning from Evolving Data Streams. In: *Advances in Knowledge Discovery and Data Mining*. [S.l.]: Springer, 2010, (LNCS, v. 6119). p. 299–310.

BIFET, A.; KIRKBY, R. Data stream mining a practical approach. Citeseer, 2009.

BLUM, A. Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning*, Springer, v. 26, n. 1, p. 5–23, 1997.

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

BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001.

BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. *Classification and Regression Trees*. Belmont, California: Wadsworth International Group, 1984. (Wadsworth Statistics / Probability series).

BRZEZIŃSKI, D.; STEFANOWSKI, J. Accuracy Updated Ensemble for Data Streams with Concept Drift. In: CORCHADO, E.; KURZYNSKI, M.; WOZNIAK, M. (Ed.). *Hybrid Artificial Intelligent Systems*. [S.l.]: Springer, 2011, (LNCS, v. 6679). p. 155–163.

BRZEZINSKI, D.; STEFANOWSKI, J. Stream classification. In: *Encyclopedia of Machine Learning*. [S.l.]: Springer, 2016.

BURTON, A. N.; KELLY, P. H. Performance prediction of paging workloads using lightweight tracing. *Future Generation Computer Systems*, Elsevier, v. 22, n. 7, p. 784–793, 2006.

CAUWENBERGHS, G.; POGGIO, T. Incremental and decremental support vector machine learning. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2001. p. 409–415.

CIESLAK, D. A.; CHAWLA, N. V. A framework for monitoring classifiers' performance: when and why failure occurs? *Knowledge and Information Systems*, Springer, v. 18, n. 1, p. 83–108, 2009.

DASH, M.; LIU, H. Hybrid search of feature subsets. In: SPRINGER. *Pacific Rim International Conference on Artificial Intelligence*. [S.l.], 1998. p. 238–249.

DAWID, A. P. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, JSTOR, p. 278–292, 1984.

DEMŠAR, J. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, JMLR.org, v. 7, p. 1–30, 2006.

DITZLER, G.; POLIKAR, R. Hellinger distance based drift detection for nonstationary environments. In: IEEE. *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*. [S.l.], 2011. p. 41–48.

DITZLER, G.; ROVERI, M.; ALIPPI, C.; POLIKAR, R. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, IEEE, v. 10, n. 4, p. 12–25, 2015.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proc. of 6th ACM SIGKDD Internat.˜Conf.˜on Knowledge Discovery and Data Mining*. Boston, USA: ACM, 2000. (KDD '00), p. 71–80.

DU, L.; SONG, Q.; JIA, X. Detecting concept drift: An information entropy based method using an adaptive sliding window. *Intelligent Data Analysis*, IOS Press, v. 18, n. 3, p. 337–364, 2014.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Available at: <http://archive.ics.uci.edu/ml>.

DUDA, R. O.; STORK, D. G.; HART, P. E. Pattern classification. Wiley, 2001.

FAITHFULL, W. J.; RODRÍGUEZ, J. J.; KUNCHEVA, L. I. Combining univariate approaches for ensemble change detection in multivariate data. *Information Fusion*, Elsevier, v. 45, p. 202–214, 2019.

FISCHER, I.; POLAND, J. Amplifying the block matrix structure for spectral clustering. In: CITESEER. *Proceedings of the 14th annual machine learning conference of Belgium and the Netherlands*. [S.l.], 2005. p. 21–28.

FORINA, M.; LANTERI, S.; ARMANINO, C. et al. Parvus-an extendible package for data exploration, classification and correlation, institute of pharmaceutical and food analysis and technologies, via brigata salerno, 16147 genoa, italy (1988). *Av. Loss Av. O set Av. Hit-Rate*, 1991.

FREUND, Y. Boosting a weak learning algorithm by majority. *Inform. and Computation*, v. 121, n. 2, p. 256–285, 1995.

FRÍAS-BLANCO, I. *Nuevos métodos para el aprendizaje en flujos de datos no estacionarios*. Phd Thesis (PhD Thesis) — Universidad de Granma, 2014.

FRÍAS-BLANCO, I.; CAMPO-ÁVILA, J. del; RAMOS-JIMÉNEZ, G.; MORALES-BUENO, R.; ORTIZ-DÍAZ, A.; CABALLERO-MOTA, Y. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. *IEEE Transactions on Knowledge and Data Engineering*, v. 27, n. 3, p. 810–823, 2015.

FRIAS-BLANCO, I.; VERDECIA-CABRERA, A.; ORTIZ-DIAZ, A.; CARVALHO, A. Fast adaptive stacking of ensembles. In: ACM. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.], 2016. p. 929–934.

FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, Taylor & Francis, v. 32, n. 200, p. 675–701, 1937.

GAMA, J. *Knowledge discovery from data streams*. [S.l.]: Chapman and Hall/CRC, 2010.

GAMA, J.; CASTILLO, G. Learning with local drift detection. In: SPRINGER. *International Conference on Advanced Data Mining and Applications*. [S.l.], 2006. p. 42–55.

GAMA, J.; GABER, M. M. *Learning from data streams: processing techniques in sensor networks*. [S.l.]: Springer, 2007.

GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. Learning with Drift Detection. In: *Advances in Artificial Intelligence: SBIA 2004*. [S.l.]: Springer, 2004, (LNCS, v. 3171). p. 286–295.

GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. On evaluating stream learning algorithms. *Machine Learning*, Springer, v. 90, n. 3, p. 317–346, 2013.

GAMA, J.; ZLIOBAITE, I.; BIFET, A.; PECHENIZKIY, M.; BOUCHACHIA, A. A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, v. 46, n. 4, p. 44:1—-37, 2014.

GODASE, A.; ATTAR, V. Classification of data streams with skewed distribution. In: *Proceedings of the IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*. [S.l.: s.n.], 2012. p. 151–156.

GONÇALVES, P. M.; BARROS, R. S. M. Speeding Up Statistical Tests to Detect Recurring Concept Drifts. In: LEE, R. (Ed.). *Computer and Information Science*. [S.l.]: Springer, 2013, (Studies in Computational Intelligence, v. 493). p. 129–142.

GONÇALVES, P. M.; SANTOS, S. G. T. C.; BARROS, R. S. M.; VIEIRA, D. C. L. A comparative study on concept drift detectors. *Expert Systems with Applications*, Elsevier, v. 41, n. 18, p. 8144–8156, 2014.

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

HAND, D. J. Classifier technology and the illusion of progress. *Statistical science*, JSTOR, p. 1–14, 2006.

HIDALGO, J. I. G. *Experiências com Variações Prequential para Avaliação da Aprendizagem em Fluxo de Dados*. Master's Thesis (Master's Thesis) — Universidade Federal de Pernambuco, Brazil, 2017.

HIDALGO, J. I. G.; MACIEL, B. I. F.; BARROS, R. S. M. Experimenting with prequential variations for data stream learning evaluation. *Computational Intelligence*, 2019. Available at: <https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12208>.

HOEFFDING, W. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, v. 58, p. 13–30, 1963.

HOENS, T. R.; POLIKAR, R.; CHAWLA, N. V. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, Springer, v. 1, n. 1, p. 89–101, 2012.

HOLUB, A.; PERONA, P.; BURL, M. C. Entropy-based active learning for object recognition. In: IEEE. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'08)*. Anchorage, AK, USA, 2008. p. 1–8.

HONG, Z.-Q.; YANG, J.-Y. Optimal discriminant plane for a small number of samples and design method of classifier on the plane. *pattern recognition*, Elsevier, v. 24, n. 4, p. 317–324, 1991.

HUANG, D. T. J.; KOH, Y. S.; DOBBIE, G.; BIFET, A. Drift detection using stream volatility. In: SPRINGER. *Joint European conference on machine learning and knowledge discovery in databases*. [S.l.], 2015. p. 417–432.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: *Proceedings of the Seventh ACM SIGKDD Intern.˜Conf.˜on Knowledge Discovery and Data Mining*. San Francisco, USA: [s.n.], 2001. (KDD '01), p. 97–106.

IENCO, D.; BIFET, A.; ŽLIOBAITĖ, I.; PFAHRINGER, B. Clustering based active learning for evolving data streams. In: SPRINGER. *International Conference on Discovery Science*. [S.l.], 2013. p. 79–93.

JIN, R.; AGRAWAL, G. Efficient decision tree construction on streaming data. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 571–576.

JOHN, G. H.; LANGLEY, P. Estimating continuous distributions in Bayesian classifiers. In: MORGAN KAUFMANN PUBLISHERS INC. *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. [S.l.], 1995. p. 338–345.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. *Incremental clustering for the classification of concept-drifting data streams*. [S.l.]: Citeseer, 2008.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, Springer, v. 22, n. 3, p. 371–391, 2010.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. P. An ensemble of classifiers for coping with recurring contexts in data streams. In: *ECAI*. [S.l.: s.n.], 2008. p. 763–764.

KHAMASSI, I.; SAYED-MOUCHAWEH, M. Drift detection and monitoring in non-stationary environments. In: IEEE. *2014 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*. [S.l.], 2014. p. 1–6.

KHAMASSI, I.; SAYED-MOUCHAWEH, M.; HAMMAMI, M.; GHÉDIRA, K. Self-adaptive windowing approach for handling complex concept drift. *Cognitive Computation*, Springer, v. 7, n. 6, p. 772–790, 2015.

KHAMASSI, I.; SAYED-MOUCHAWEH, M.; HAMMAMI, M.; GHÉDIRA, K. Discussion and review on evolving data streams and concept drift adapting. *Evolving systems*, Springer, v. 9, n. 1, p. 1–23, 2018.

KIFER, D.; BEN-DAVID, S.; GEHRKE, J. Detecting change in data streams. In: VLDB ENDOWMENT. *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. [S.l.], 2004. p. 180–191.

KIM, Y.; PARK, C. H. An efficient concept drift detection method for streaming data under limited labeling. *IEICE TRANSACTIONS on Information and Systems*, The Institute of Electronics, Information and Communication Engineers, v. 100, n. 10, p. 2537–2546, 2017.

KOHAIL, S. N. *Learning Concept Drift Using Adaptive Training Set Formation Strategy*. Phd Thesis (PhD Thesis) — The Islamic University of Gaza, 2011.

KOHAVI, R.; PROVOST, F. Glossary of terms: Machine learning. *30: 271*, v. 274, 1998.

KOLTER, J. Z.; MALOOF, M. A. Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts. *Journal of Machine Learning Research*, JMLR.org, v. 8, p. 2755–2790, 2007.

KRAWCZYK, B.; MINKU, L. L.; GAMA, J.; STEFANOWSKI, J.; WOŹNIAK, M. Ensemble learning for data stream analysis: A survey. *Information Fusion*, Elsevier, v. 37, p. 132–156, 2017.

KRAWCZYK, B.; WOŹNIAK, M.; SCHAEFER, G. Cost-sensitive decision tree ensembles for effective imbalanced classification. *Applied Soft Computing*, Elsevier, v. 14, p. 554–562, 2014.

KUMAR, V.; CHAUHAN, H.; PANWAR, D. K-means clustering approach to analyze nsl-kdd intrusion detection dataset. *International Journal of Soft Computing and Engineering (IJSCE)*, 2013.

KUNCHEVA, L. I. Classifier ensembles for changing environments. In: SPRINGER. *International Workshop on Multiple Classifier Systems*. [S.l.], 2004. p. 1–15.

LICHTENWALTER, R. N.; LUSSIER, J. T.; CHAWLA, N. V. New perspectives and methods in link prediction. In: ACM. *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2010. p. 243–252.

LUGHOFER, E.; WEIGL, E.; HEIDL, W.; EITZINGER, C.; RADAUER, T. Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances. *Information Sciences*, Elsevier, v. 355, p. 127–151, 2016.

MACIEL, B. I. F.; SANTOS, S. G. T. C.; BARROS, R. S. M. A Lightweight Concept Drift Detection Ensemble. In: *Proc. ˜of 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'15)*. Vietri sul Mare, Italy: [s.n.], 2015. p. 1061–1068.

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*. [S.l.: s.n.], 2018. p. 1953–1962.

MENAHEM, E.; ROKACH, L.; ELOVICI, Y. Combining one-class classifiers via meta learning. In: ACM. *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. [S.l.], 2013. p. 2435–2440.

MINKU, L.; YAO, X. {DDD}: A New Ensemble Approach for Dealing with Concept Drift. *IEEE Transactions on Knowledge and Data Engineering*, v. 24, n. 4, p. 619–633, 2012.

MINKU, L. L. *Online Ensemble Learning in the Presence of Concept Drift*. Phd Thesis (PhD Thesis) — School of Computer Science, The University of Birmingham, 2010.

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, 2010.

MITCHELL, T. *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

MORENO-SECO, F.; INESTA, J. M.; LEÓN, P. J. P. D.; MICÓ, L. Comparison of classifier fusion methods for classification in pattern recognition tasks. In: SPRINGER. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. [S.l.], 2006. p. 705–713.

MUTHUKRISHNAN, S.; BERG, E. van den; WU, Y. Sequential change detection on data streams. In: IEEE. *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. [S.l.], 2007. p. 551–550.

NARASIMHAMURTHY, A. M.; KUNCHEVA, L. I. A framework for generating data to simulate changing environments. In: *Artificial Intelligence and Applications*. [S.l.: s.n.], 2007. p. 415–420.

NEMENYI, P. Distribution-free multiple comparisons. *Princeton University*, 1963.

NISHIDA, K.; YAMAUCHI, K. Detecting concept drift using statistical testing. In: *Proceedings of 10th International Conference on Discovery Science (DS'07)*. [S.l.]: Springer, 2007. (LNCS, v. 4755), p. 264–269.

OLORUNNIMBE, M. K.; VIKTOR, H. L.; PAQUET, E. Intelligent adaptive ensembles for data stream mining: a high return on investment approach. In: SPRINGER. *International workshop on new frontiers in mining complex patterns*. [S.l.], 2015. p. 61–75.

ONAN, A.; KORUKOĞLU, S.; BULUT, H. A multiobjective weighted voting ensemble classifier based on differential evolution algorithm for text sentiment classification. *Expert Systems with Applications*, Elsevier, v. 62, p. 1–16, 2016.

ORTIZ-DIAZ, A.; CAMPO-ÁVILA, J. del; RAMOS-JIMÉNEZ, G.; BLANCO, I. F.; MOTA, Y. C.; HECHAVARRÍA, A. M.; MORALES-BUENO, R. Fast adapting ensemble: A new algorithm for mining data streams with concept drift. *The Scientific World Journal*, Hindawi, v. 2015, 2015.

ORTIZ-DIAZ, A. A. *Algoritmo multiclasificador con aprendizaje incremental al que manipula cambios de conceptos*. [S.l.]: Universidad de Granada, 2014.

OZA, N. C.; RUSSELL, S. Online Bagging and Boosting. In: *Artif. ˜Intellig. ˜and Stat.* [S.l.]: Morgan Kauf., 2001. p. 105–112.

PAGE, E. S. Continuous Inspection Schemes. *Biometrika*, v. 41, n. 1/2, p. 100–115, 1954.

PÉREZ, J. L. M. *Comitê de métodos estatísticos para detecção de mudanças de conceito.* Master's Thesis (Master's Thesis) — Universidade Federal de Pernambuco, Brazil, 2018.

PERVEZ, M. S.; FARID, D. M. Feature selection and intrusion classification in nsl-kdd cup 99 dataset employing svms. In: IEEE. *Software, Knowledge, Information Management and Applications (SKIMA), 2014 8th International Conference on*. [S.l.], 2014. p. 1–6.

PESARANGHADER, A.; VIKTOR, H.; PAQUET, E. Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams. *Machine Learning*, Springer, v. 107, n. 11, p. 1711–1743, 2018.

PESARANGHADER, A.; VIKTOR, H. L.; PAQUET, E. A framework for classification in data streams using multi-strategy learning. In: SPRINGER. *International conference on discovery science*. [S.l.], 2016. p. 341–355.

PINTO, C.; GAMA, J. Incremental discretization, application to data with concept drift. In: ACM. *Proceedings of the 2007 ACM symposium on Applied computing*. [S.l.], 2007. p. 467–468.

POLIKAR, R. The wavelet tutorial. 2001. *Disponıvel em:< http://users. rowan. edu/polikar/WAVELETS/WTtutorial. html>. Acesso em*, v. 12, 2001.

RAFTER, J. A.; ABELL, M. L.; BRASELTON, J. P. Multiple comparison methods for means. *Siam Review*, SIAM, v. 44, n. 2, p. 259–278, 2002.

RASTIN, P. *Automatic and Adaptive Learning for Relational Data Stream Clustering*. 150 p. Phd Thesis (PhD Thesis) — L'Université Sorbonne Paris, France, 2018.

REIS, D. M. dos; FLACH, P.; MATWIN, S.; BATISTA, G. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In: ACM. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.], 2016. p. 1545–1554.

REVATHI, S.; MALATHI, A. A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *International Journal of Engineering Research & Technology (IJERT)*, Citeseer, v. 2, n. 12, p. 1848–1853, 2013.

ROBERTS, S. Control Chart Tests Based on Geometric Moving Averages. *Technometrics*, v. 1, n. 3, p. 239–250, 1959.

ROSS, G. J.; ADAMS, N. M.; TASOULIS, D. K.; HAND, D. J. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, v. 33, n. 2, p. 191–198, 2012.

SAKTHITHASAN, S.; PEARS, R.; KOH, Y. S. One pass concept change detection for data streams. In: SPRINGER. *Pacific-Asia conference on knowledge discovery and data mining*. [S.l.], 2013. p. 461–472.

SANTOS, S. G. T. C.; BARROS, R. S. M.; Gonçalves Jr., P. M. Optimizing the Parameters of Drift Detection Methods Using a Genetic Algorithm. In: *Proc.˜of 27th IEEE Internatational Conference on Tools with Artificial Intelligence (ICTAI'15)*. Vietri sul Mare, Italy: [s.n.], 2015. p. 1077–1084.

SANTOS, S. G. T. C.; Gonçalves Jr., P. M.; SILVA, G.; BARROS, R. S. M. Speeding Up Recovery from Concept Drifts. In: *Machine Learning and Knowledge Discovery in Databases*. [S.l.]: Springer, 2014, (LNCS, v. 8726). p. 179–194.

SEBASTIÃO, R.; GAMA, J.; MENDONÇA, T. Fading histograms in detecting distribution and concept changes. *International Journal of Data Science and Analytics*, Springer, v. 3, n. 3, p. 183–212, 2017.

SHAKER, A.; HüLLERMEIER, E. Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. *Neurocomputing*, v. 150, p. 250 – 264, 2015. ISSN 0925-2312. Bioinspired and knowledge based

techniques and applications The Vitality of Pattern Recognition and Image Analysis Data Stream Classification and Big Data Analytics. Available at: <http://www.sciencedirect.com/science/article/pii/S0925231214013216>.

SHEN, H.; LIN, Y.; TIAN, Q.; XU, K.; JIAO, J. A comparison of multiple classifier combinations using different voting-weights for remote sensing image classification. *International journal of remote sensing*, Taylor & Francis, v. 39, n. 11, p. 3705–3722, 2018.

SONG, G.; YE, Y.; ZHANG, H.; XU, X.; LAU, R. Y. K.; LIU, F. Dynamic clustering forest: an ensemble framework to efficiently classify textual data stream with concept drift. *Information Sciences*, Elsevier, v. 357, p. 125–143, 2016.

STEVENSON, A. *Oxford dictionary of English*. [S.l.]: Oxford University Press, USA, 2010.

SUDHA, N.; GOVINDARAJAN, D. Opinion mining on news articles using feature reduction method. *International Journal of Latest Engineering and Management Research (IJLEMR)*, 2017.

TING, K. M.; WITTEN, I. H. Issues in stacked generalization. *Journal of artificial intelligence research*, v. 10, p. 271–289, 1999.

VERDECIA-CABRERA, A.; BLANCO, I. F.; CARVALHO, A. C. An online adaptive classifier ensemble for mining non-stationary data streams. *Intelligent Data Analysis*, IOS Press, v. 22, n. 4, p. 787–806, 2018.

VORBURGER, P.; BERNSTEIN, A. Entropy-based concept shift detection. In: IEEE. *Sixth International Conference on Data Mining (ICDM'06)*. [S.l.], 2006. p. 1113–1118.

WANKHADE, K.; DONGRE, S.; THOOL, R. New evolving ensemble classifier for handling concept drifting data streams. In: *Proceedings of the 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC)*. [S.l.: s.n.], 2012. p. 657–662.

WEBB, G. I.; HYDE, R.; CAO, H.; NGUYEN, H. L.; PETITJEAN, F. Characterizing concept drift. *Data Mining and Knowledge Discovery*, Springer, v. 30, n. 4, p. 964–994, 2016.

WIDMER, G.; KUBAT, M. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, Springer, v. 23, n. 1, p. 69–101, 1996.

WOLPERT, D. H. Stacked generalization. *Neural networks*, Elsevier, v. 5, n. 2, p. 241–259, 1992.

WOŹNIAK, M.; KRAWCZYK, B. Combined classifier based on feature space partitioning. *International Journal of Applied Mathematics and Computer Science*, Versita, v. 22, n. 4, p. 855–866, 2012.

YE, D.; CHEN, Z. Inconsistency classification and discernibility-matrix-based approaches for computing an attribute core. In: SPRINGER. *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*. [S.l.], 2003. p. 269–273.

ZHENG, F.; WEBB, G. A comparative study of semi-naive Bayes methods in classification learning. In: *Proceedings of Fourth Australasian Data Mining Workshop (AusDM)*. Sidney, Australia: [s.n.], 2005. p. 141–156.

ZHONG, S.; TANG, W.; KHOSHGOFTAAR, T. M. Boosted noise filters for identifying mislabeled data. *Department of Computer Science and engineering, Florida Atlantic University*, 2005.

ZHU, Q.; HU, X.; ZHANG, Y.; LI, P.; WU, X. A double-window-based classification algorithm for concept drifting data streams. In: IEEE. *2010 IEEE International Conference on Granular Computing*. [S.l.], 2010. p. 639–644.

ZLIOBAITE, I. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010.

ZLIOBAITE, I.; BUDKA, M.; STAHL, F. T. Towards cost-sensitive adaptation: When is it worth updating your predictive model? *Neurocomputing*, Elsevier, v. 150, p. 240–249, 2015.

APPENDIX A – HOEFFDING

This appendix presents the Hoeffding inequality theorem and the pseudo-code of the Hoeffding Tree base classifier.

A.1 HOEFFDING'S INEQUALITY THEOREM

Let $X_1, X_2, ..., X_n$, be independent random variables such that for every $X_i \in [a_i, b_i]$ where $0 \leq i \leq n$, and with maximum probability δ. Be the empirical average $\bar{X} = \sum_{i=1}^{n} \frac{X_i}{n}$ a random variable whose expected value is $E\left[\bar{X}\right]$. Then, for any $\varepsilon_H > 0$ can be applied equation A.1.

$$P_r(\left|\bar{X} - E\left[\bar{X}\right]\right| \geq \varepsilon) \leq 2e^{-2n^2\varepsilon^2 / \sum_{i=1}^{n}(b_i - a_i)_2} \tag{A.1}$$

The error can be estimated, $\varepsilon_\delta = \sqrt{\frac{1}{2n} ln \frac{1}{\delta}}$, it is known that the Hoeffding inequality assumes only independent random variables, but no probability function is assumed. $(\bar{X})$ and error rate (ε_δ) can be calculated with O (1) of temporal and spatial complexity, which makes the theorem applicable to learning in data stream (FRÍAS-BLANCO, 2014; PÉREZ, 2018).

A.2 PSEUDO-CODE OF THE HT CLASSIFIER

Follow the pseudo-code of Hoeffding Tree, one of the most used classifiers in the area of machine learning in non-stationary data stream.

Already exposed at the beginning of the research, the HT classifier is also known as VFDT (Very Fast Decision Trees for Mining High-Speed Data Streams). It allows the use of Information Gain or Gini Index as an evaluation measure (HULTEN; SPENCER; DOMINGOS, 2001; AGGARWAL, 2014).

As can be seen in the implementation, in the line **1** the algorithm begins with a leaf node, the root of the tree. When a new instance arrives, it is sorted to its corresponding sheet, sufficient statistics are collected from the data, and n_l is incremented, which is the number of instances observed in node l(lines **3-5**).

In the line **6** it is checked whether sufficient instances have been observed on the node in question to try to split it. This is done by using the n_min (minimum number of instances that must be read for the split attempt) and also checking that all data in the node up to that time belongs to the same class. If so, there is no need for division.

In the lines **7-9** the information gain heuristic and the Gini Index are used to choose the two best attributes to be used in the division of the node. To solve the problem of

Algoritmo 2: Hoeffding tree induction algorithm

Input: n_{min} (minimum number of examples of the tolerance period), τ (a tie threshold)
Output: HT decision tree

```
1  Let HT be a tree with a single leaft (the root)
2  for all training examples do
3      Sort example into leaft l using HT
4      Update sufficient statistics in l
5      Increment n_l the number of examples seen at l
6      if n_l mod n_min = 0 and examples seen at l not all of same class then
7          Compute Ḡ_l(x_i) for each attribute
8          Let x_a be attribute with highest Ḡ_l
9          Let x_b be attribute with second-highest Ḡ_l
10         Compute Hoeffding bound ε = sqrt(R²(ln 1/δ) / 2n)
11         if x_a ≠ x_∅ and (Ḡ_l(x_a) − Ḡ_l(x_b)) > ε or ε < τ then
12             Replace l with an internal node that splits on x_a
13             for all branches of the split do
14                 Add a new leaft with initialized sufficient statistics
```

deciding exactly how many instances are required for each node, the line **10** uses the Hoeffding bound (equation A.1).

In the line **11** are verified the next conditions:

- $x_a \neq x_\emptyset$: if at least one attribute was chosen, that is, if the best attribute is different from null;
- $(\bar{G}_l(x_a) - \bar{G}_l(x_b)) > \varepsilon$: if the difference between the two best attributes is greater than ε. This condition is tested to avoid dividing a node when two or more attributes have many values close since one attribute could become the best in the next iterations;
- $\varepsilon < \tau$: as n (number of instances observed on the node) increases, ϵ decrease. If the two best attributes have very close values in several iterations, ϵ would be as small as τ, which is a tiebreaker criterion.

If the condition of the line **11** is satisfied, in the line **12** the sheet l becomes an internal node that is divided using the attribute x_a. In the line **14** enough statistics are started for each sheet that results from node splitting (PÉREZ, 2018) .

APPENDIX B – DATA-SETS CHARACTERISTICS

The MOA framework contains a large number of data-set generators. It also allows the integration of real database (*.arff file). Table 13 show the most used data-set generators at the moment, each one allows to simulate the different kinds of concept drift. A reasonable number of real data-sets are also exposed (table 14) available on the internet.

B.1 SYNTHETIC DATA-SETS

Table 13 – Characteristics of the most commonly used synthetic data-set.

Data-set generators	Acronyms	Features	Classes
LED	LED	24	10
Sine	Sine	2	2
Mixed drift	Mixed	4	2
Wavefrom	Wave21	21	3
Wavefrom	Wave40	40	3
SEA Concept	SEA	3	2
Random RBF Simples	RBFS	10	2
Random RBF Complexa	RBFC	50	2
Random Tree Simples	RTS	20	2
Random Tree Complexa	RTC	100	2
Function Generator (Agrawal)	Agrawal	9	2
STAGGER Concept	STAGGER	3	2
Rotating Hyperplane	Hyperplane	10	2

Source: Pérez (2018)

B.2 REAL DATA-SETS

Table 14 – Characteristics of the most used real data-sets.

Database	Acronyms	Instances	Nominal	Numeric	Missing Values	Classes
Airlines	AIR	539.382	4	3	no	2
Adult	ADU	32.561	8	6	sim	2
Bank marketing	BAN	41.188	9	7	no	2
Connect-4	COM	67.557	21	0	no	3
Forest Covert	COV	581.012	44	10	no	7
Cars	CAR	1.728	6	0	no	4
EEG Eye State	EYE	14.980	0	14	no	2
Electricity	ELE	45.312	1	7	yes	2
Letter Recognition	LET	20.000	0	16	no	26
Mushroom	MUS	8.124	22	0	yes	2
NSL-KDD 99 joined	NSLK	148.561	7	34	no	2
Nursey	NUR	12.960	8	0	no	5
Outdoor	OUT	4.000	0	21	no	40
Poker Hand	POK	1.000.000	10	0	no	10
Rialto	RIA	82.250	0	27	no	10
Spam coprus 2	SPA	9.323	500	0	no	2
Segment	SEG	2.310	0	19	no	7
Usenet1	USE1	1.500	100	0	no	2
Usenet2	USE2	1.500	100	0	no	2
Usenet3	USE3	3.000	100	0	no	2
WineWhite	WINW	4.898	0	11	no	1
WineRed	WINR	1.599	0	11	no	1
Weather	WEA	18.159	0	8	no	2

Source: Pérez (2018)

APPENDIX C – STATISTICAL TEST

C.1 FRIEDMAN'S TEST

In the machine learning area, the *Friedman* (F_F) statistic is used to test the null hypothesis of performance equality between the different methods. F_F compares sample data in paired form, that is, when the same subject is evaluated more than once (HIDALGO, 2017). This statistical method does not use the metric score directly, it ranks the algorithms for each data set separately, the best performing algorithm getting the rank of 1, the second best rank 2, and so on. After the ordering is tested the hypothesis of equality of sum of the posts of each group (FRIEDMAN, 1937; PÉREZ, 2018)

In order to understand the statistic test and its use with the results of the investigation, we provide hereafter a detailed explanation of how the methods proposed in this dissertation were compared together with the selected methods of the state of the art.

The performance of the compared methods in the experiment can be represented by the matrix C.1,

$$\begin{array}{cc} & \text{Methods} \\ \text{Data-sets} & \begin{bmatrix} X_{11} & X_{12} & ... & X_{1n} \\ X_{21} & X_{21} & ... & X_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{m1} & X_{m2} & ... & X_{mn} \end{bmatrix} \end{array} \tag{C.1}$$

where X_{ij} denotes the performance of the j-th method on the i-th data set (for $i = 1, ..., m$ and $j = 1, ..., n$).

In the first step each method X_{ij} of the matrix C.1 are ranked according to the evaluation measure in ascending order by rows.

Then, each entry is replaced by the corresponding rank as it can be seen in the matrix expression C.2 where R_{ij} is the rank of the jth method in i data set (HIDALGO, 2017; PÉREZ, 2018).

$$R = \begin{bmatrix} R_{11} & R_{12} & ... & R_{1n} \\ R_{21} & R_{22} & ... & R_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ R_{m1} & R_{m2} & ... & R_{mn} \end{bmatrix} \tag{C.2}$$

Next the average rank for each methods over the different data set is calculated. As pointed out before, the best method gets rank 1, the second best method gets rank 2, and so on. In case of ties, average ranks are assigned. However, it should be noted that the

sum of the j-th column $R_j = \sum_{i=1}^{m} R_{ij}^2$, $\forall\ j = 1, ..., n$, depends on how the jth data set behaves in relation to the other data sets $(n-1)$(AGGARWAL, 2014).

Thus, under the null hypothesis, all methods are equivalent and hence the average ranks of the different classifiers should be similar. The test aim to detect a deviation from this and can detect if there are significant differences between at least two of the methods (AGGARWAL, 2014). The test statistic of the Friedman test is given by C.3, where n and m refer to the number of rows and columns in the matrix C.2 respectively, and R_j is the corresponding rank of each column(HIDALGO, 2017; PÉREZ, 2018).

$$\chi_F^2 = \frac{12n}{m\,(m+1)} \left[\sum_{j=1}^{n} R_j^2 - \frac{m\,(m+1)^2}{2} \right] \tag{C.3}$$

Finally, for a sufficient number of data set and methods (as a rule $n > 10$ and $m > 5$), χ_F^2 approximately follows a chi-square distribution with $m-1$ degrees of freedom.

For a small number of data sets and methods, exact critical values have been computed. Iman and Davenport (1980) derived a better statistic

$$F_F = \frac{(n-1)\chi_F^2}{n(m-1) - \chi_F^2} \tag{C.4}$$

that is distributed according to the F-distribution with $m-1$ and $(m-1)(Nn-1)$ degrees of freedom.

C.2 NEMENYI POST-TEST

When the test of *Friedman* rejects the null hypothesis, it is necessary to establish what are the significant differences between the adaptive methods. For this purpose, a Post-Test is performed (test performed after another global test has been performed). There are several alternatives to choose when you have to use a Post-test, the first is to choose which type of comparison fits the research context. The comparisons are grouped as follows:

- Non-parametric multiple groups One vs All (Comparison of one method against the others). The Bonferroni-Dunn (DEMŠAR, 2006) is used in the majority of researchs in the area when this type of comparison is needed.
- Non-parametric multiple groups All vs All. For this case, the usual Post-test found in the researchs is the *Nemenyi* (NEMENYI, 1963).

In this dissertation was used the Post-test of *Nemenyi* in order to make multiple comparisons between the algorithms. Therefore, the performance of two methods is significantly different if the ranks of the percentage differences between the corresponding means differ at least from the Critical Difference (CD) defined by equation C.5

$$CD = q_\alpha \sqrt{\frac{m\,(m+1)}{6n}}, \tag{C.5}$$

where the critical values q_α are based on the Studentized Range Statistic cite (RAFTER; ABELL; BRASELTON, 2002) divided by $\sqrt{2}$.

APPENDIX D – SUMMARIZED RESULTS

D.1 SUMMARIZED RESULTS REGARDING FASE BY SCENARIO AND BASE CLASSIFIER

Table 15 – Frequency of improvements reached by proposed methods in accuracy, runtime, and memory regarding FASE by scenario and base classifier (NB).

		NB							
		FASEO	**FASEOwv1**	**FASEOwv2**	**FASEOwv3**	**FASEB**	**FASEBwv1**	**FASEBwv2**	**FASEBwv3**
SC1	ACC	1	1	1	1	3	1	1	1
	TIME	6	6	6	6	6	6	6	6
	MEMORY	6	6	6	6	6	6	6	6
SC2	ACC	1	1	1	1	3	2	2	1
	TIME	6	6	6	6	6	6	6	6
	MEMORY	6	6	6	6	6	6	6	6
SC3	ACC	2	2	2	2	2	2	2	2
	TIME	5	5	5	5	5	5	6	6
	MEMORY	5	5	5	5	5	5	6	6
SC4	ACC	0	1	1	1	1	1	1	1
	TIME	5	6	6	6	6	6	6	6
	MEMORY	6	6	6	6	6	6	6	6
SC5	ACC	4	4	6	5	4	4	6	5
	TIME	6	9	8	11	9	10	8	11
	MEMORY	8	9	10	11	11	11	10	11
BC	**ACC**	8	9	11	10	13	10	12	10
	TIME	28	32	31	34	32	33	32	35
	MEMORY	31	32	33	34	34	34	34	35

Source: Own elaboration (2019)

Table 16 – Frequency of improvements reached by proposed methods in accuracy, run-time, and memory regarding FASE by scenario and base classifier (HT).

		HT							
		FASEO	**FASEOwv1**	**FASEOwv2**	**FASEOwv3**	**FASEB**	**FASEBwv1**	**FASEBwv2**	**FASEBwv3**
SC1	ACC	1	1	1	1	1	1	1	1
	TIME	6	6	6	6	6	6	6	6
	MEMORY	6	6	6	6	5	6	6	6
SC2	ACC	1	1	1	1	1	1	1	1
	TIME	5	5	5	5	5	5	6	6
	MEMORY	5	5	5	5	5	5	5	6
SC3	ACC	1	1	1	1	2	1	1	1
	TIME	6	6	6	6	6	6	6	6
	MEMORY	6	6	6	6	6	6	6	6
SC4	ACC	2	1	1	1	2	1	1	1
	TIME	4	5	5	5	5	5	6	6
	MEMORY	5	5	5	5	5	5	5	5
SC5	ACC	5	4	5	4	6	4	5	3
	TIME	3	8	8	8	3	8	9	7
	MEMORY	5	7	7	8	5	7	8	8
BC	**ACC**	10	8	9	8	12	8	9	7
	TIME	24	30	30	30	25	30	33	31
	MEMORY	27	29	29	30	26	29	30	31

Source: Own elaboration (2019)

D.2 SUMMARIZED RESULTS REGARDING FASE IN ALL EXPERIMENTS

Table 17 – Frequency and percentage of improvements reached by proposed methods in accuracy, run-time, and memory in all (70) experiments regarding FASE.

		FASEO	**FASEOwv1**	**FASEOwv2**	**FASEOwv3**	**FASEB**	**FASEBwv1**	**FASEBwv2**	**FASEBwv3**
$\mathbf{ACC}_T$		18	17	20	18	25	18	21	17
	%	25,71	24,29	28,57	25,71	35,71	25,71	30,00	24,29
$\mathbf{TIME}_T$		52	62	61	64	57	63	65	66
	%	74,29	88,57	87,14	91,43	81,43	90,00	92,86	94,29
$\mathbf{MEMORY}_T$		58	61	62	64	60	63	64	66
	%	82,86	87,14	88,57	91,43	85,71	90,00	91,43	94,29

Source: Own elaboration (2019)