JEAN PAUL BARDDAL

FEATURE ANALYSIS IN EVOLVING DATA STREAMS: ISSUES AND ALGORITHMS

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

Curitiba
2019

JEAN PAUL BARDDAL

FEATURE ANALYSIS IN EVOLVING DATA STREAMS: ISSUES AND ALGORITHMS

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

Major Field: Computer Science

Advisor: Fabrício Enembreck
Co-advisors: Albert Bifet and Bernhard Pfahringer

Curitiba
2019

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Luci Eduarda Wielganczuk – CRB 9/1118

B245f
2019

Barddal, Jean Paul

Feature analysis in evolving data streams : issues and algorithms / Jean Paulo Barddal ; advisor: Fabrício Enembreck ; co-advisors: Albert Bifet and Bernhard Pfahringer. – 2019.

xiv, 162 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2019

Bibliografia: f. 142-155

1. Mineração de dados (Computação). 2. Algoritmos. I. Enembreck, Fabrício. II. Bifet, Albert. III. Pfahringer, Bernhard. IV. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. V. Título.

CDD 22. ed. – 006.312

PUCPR
GRUPO MARISTA

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

DECLARAÇÃO

Declaro para os devidos fins que o aluno **JEAN PAUL BARDDAL**, defendeu sua tese de doutorado intitulada **"Feature Analysis in Evolving Data Streams: Issues and Algorithms"**, na área de concentração Ciência da Computação, no dia 27 de novembro de 2018, no qual foi aprovado.

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

Por ser verdade, firmo a presente declaração.

Curitiba, 23 de janeiro de 2019.

Programa de Pós-Graduação em Informática
PPGIa
PUCPR

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

This thesis was prepared with LaTeX. The bibliography created using BibTeX and the abnTeX2 style. Please respect the license required by the author and cite his works appropriately.

There is one kind of robber whom the law does not strike at, and who steals what is most precious to men: time.
Napoleon Bonaparte

Contents

List of Algorithms

List of Figures

List of Tables

Abstract

Feature drifts occur whenever a subset of features becomes, or ceases to be, relevant to the learning task. Even though this type of drift was mentioned in pioneer works on data stream mining, most of the offered algorithms assume that the same features are relevant to the concepts to be learned throughout the whole process and that drifts occur in partitions or ranges of each feature. This thesis is devoted to analyzing features in drifting scenarios. This work starts with a survey on the topic, including a formal definition to feature drifts and existing techniques capable of dealing with this trait. Furthermore, it introduces novel generators and inspects existing ones capable of synthesizing feature drifts. Regarding novel methods for handling feature drifts, the following are offered: **(i)** new scoring operators based on Information Theory, i.e., Conditional Entropy and Symmetrical Uncertainty, to keep track of features' relevance during the processing of data streams, **(ii)** the application of **(i)** as a weighting factor in both bayesian, instance-based and decision tree-based learning schemes, **(iii)** the use of **(i)** as the core of a merit-based feature selection method; and finally, **(iv)** a boosting-based strategy for dynamic feature selection on data streams. The results obtained for the novel techniques show that accuracy rates can be improved significantly across feature weighting, merit-guided feature selection, and the boosting-based feature selection methods, whereas at the expense of bounded computational resources for the weighting scheme and eventual improvements for the remainder of the proposals.

Keywords: Data Stream Classification; Concept Drift; Feature Drift; Feature Selection; Feature Weighting.

Resumo

Feature drifts ocorrem sempre que um subconjunto de atributos se torna, ou deixa de ser, relevante para a tarefa de aprendizagem. Por mais que este tipo de mudança tenha sido mencionado em trabalhos pioneiros da área de mineração de fluxos contínuos de dados, maior parte dos algoritmos propostos assume que o conjunto de atributos relevante para a aprendizagem é mantido durante todo o processo e que mudanças ocorrem em partições e intervalos de cada atributo. Esta tese objetiva a análise de atributos em cenários com mudanças de conceito. Este trabalho inicia com um levantamento de trabalhos existentes na área, incluindo uma definição formal de *feature drifts* e técnicas existentes para o tratamento destes. Ademais, este trabalho apresenta novos geradores de dados capazes de sintetizar este tipo de problema, assim como reporta os geradores existentes. Em termos de novos métodos para tratamento de *feature drifts*, o seguinte é proposto: **(i)** operadores de ranqueamento baseados na Teoria da Informação, i.e., Entropia Condicional e Incerteza Simétrica, para identificar o poder de discriminação de cada atributo durante o processamento de fluxos de dados, **(ii)** a aplicação de **(i)** como um fator de ponderação em métodos bayesianos, baseados em instância e em árvores de decisão, **(iii)** o uso de **(i)** como parte principal de um método de seleção baseado em mérito; e finalmente, **(iv)** um método baseado em *boosting* para seleção dinâmica de atributos em fluxos de dados. Os resultados obtidos para as novas técnicas mostram que melhorias de acurácia para diversos classificadores podem ser obtidas com os métodos de ponderação de atributos, seleção de atributos baseada em mérito e em *boosting*, contudo, com aumentos no custo computacional para o esquema de ponderação e eventuais ganhos para o restante dos métodos propostos.

Palavras-chave: Classificação de Fluxos Contínuos de Dados; Mudança de Conceito; *Feature Drift*; Seleção de Atributos; Ponderação de Atributos.

PART I

INTRODUCTION AND PRELIMINARIES

Chapter 1

Introduction

In the last decades, the interest in mining massive and potentially unbounded datasets which arrive at rapid rates, namely data streams, has grown substantially. Data Stream Mining has become of the utmost importance as most of the available data generators sequentially produce enormous amounts of data. Examples of data streams include, but are not limited to, consumer click streams, telephone usage flows, posts in social networks (BRAVO-MARQUEZ; FRANK; PFAHRINGER, 2015), multimedia data mining (GUHA, 2009; SILVA et al., 2013), computer networks intrusion detection (AGGARWAL et al., 2003) and stock market share exchanges (BARDDAL; GOMES; ENEMBRECK, 2015a). Aiming at extracting useful knowledge from these massive amounts of data, a variety of inductive learning techniques were developed and achieved concrete results in both supervised (BIFET et al., 2013; KOSINA; GAMA, 2012; GAMA et al., 2014) and unsupervised (AGGARWAL et al., 2003; KRANEN et al., 2011) learning fashions.

By far, the most common task in the streaming scenario is classification. In this task, instances[1] are associated with labels and the primary objective is to learn from labeled data how to accurately classify future instances. Data stream classification algorithms are presented to an enormous and possibly unbounded amount of data, each of which is made available to the algorithm in a serialized fast-paced fashion (GAMA et al., 2014). Moreover, due to the temporal and ephemeral characteristics of data streams, one must assume that the underlying concept is unstable, i.e., changes in the concept to be learned are expected to occur over time, a phenomenon named **concept drift** (TSYMBAL, 2004; WIDMER; KUBAT, 1996).

Even though current techniques for data stream classification handle most of the challenges posed by such environments, not much attention has been given to one of

[1] The term “instance” is often referred as “example” and “record” in other data mining and database works.

their characteristics: possible changes in the relevant subset of features, namely feature drift. This type of drift has been cited in pioneer studies on data streams (WIDMER; KUBAT, 1996), yet, they are virtually neglected, broadening both formalization and efficient proposals, and assessment procedures (BARDDAL et al., 2017).

The latter characteristic enforces classification algorithms to possess strategies to keep track or highlight the most discriminative set of features of the stream via feature selection and weighting methods. By performing feature selection as the stream progresses, classifiers are expected to compute faster, whilst requiring smaller memory space usage (due to lower dimensionality) and to present as good as or, sometimes, higher accuracy (NAIDU; DHENGE; WANKHADE, 2014). In addition to these expected results, several real-world scenarios require models that are intelligible and easy to explain, traits that can also be achieved with feature selection. Nevertheless, performing dynamic feature selection during the processing of streams is not straightforward, since this process must occur incrementally and adaptively, which is a current research gap.

1.1 Objectives

Although stated in pioneer works of data stream learning (WIDMER; KUBAT, 1996), recent works have shown that feature drift is, so far, a nearly neglected characteristic of such environments (BARDDAL et al., 2017). Due to the recent reminisce of the topic, this thesis provides an extensive study on feature drifts and techniques to excel at this problem. This study includes: (i) formalizations, (ii) surveys existing works, (iii) proposals of novel methods for tracking the relevance of features over time, thus allowing their selection or weighting; and finally (iv) assessment strategies for such techniques. The specific objectives of this project are presented as follows:

1. To provide formalizations for the feature drift problem.
2. To survey and discuss existing works that perform feature drift adaptation.
3. To identify how feature selection should be evaluated in streaming scenarios and whether assessment techniques developed for batch scenarios can be used in streaming settings.
4. To introduce dynamic scoring techniques based on concepts from Information Theory to track the relevance of features throughout data streams.
5. To propose and evaluate feature selection and weighting techniques that use the latter scoring operators to overcome feature drifts.

6. To develop and assess a boosting-based technique for dynamic feature selection from streaming scenarios.

1.2 Hypotheses

Due to the recentness of works aiming at feature drifts, this thesis encompasses several hypotheses, each one being discussed and evaluated throughout this work.

Hypothesis #1. It is possible to compute and keep track of features' relevance as a stream progresses.

Hypothesis #2. The tracking of features' relevance as a stream progresses allows swift model adaptation in the occurrence of feature drifts.

Hypothesis #3. Dynamic feature weighting improves instance-based, Bayesian and adaptive decision tree learners during both feature drifts and stationary streams when confronted with irrelevant features.

Hypothesis #4. Adaptive Information Theory scoring operators discover, keep track, and filter both irrelevant and redundant features as a stream progresses.

Hypothesis #5. Higher order interactions between features can be discovered by an adaptive boosting process, thus allowing the identification, tracking, and filtering of relevant features as a stream progresses.

1.3 Contributions

Besides merely providing techniques for dynamic feature selection, this thesis also contributes to feature drift formalization, thus filling an existing gap in the literature. The main contributions of this thesis are as follows:

- Feature drift is an unclear and nearly neglected research topic. In this thesis, a proper formalization is introduced, detailing its characteristics in theoretical and application ends (see Chapter 3).

- Until this very point, feature drifts are barely accounted for by existing data streams generators, therefore, some generators are extended (STREET; KIM, 2001; BIFET et al., 2010) and novel ones are proposed to synthesize feature drifts based on other works (ENEMBRECK et al., 2007; HALL et al., 2009). Additionally, all these generators are incremented with strategies to allow the addition of redundant features, and thus, making the experiments even more challenging (see Chapter 4).

- Most of the existing works on data streams that perform feature selection assume that the underlying data distribution is stationary, e.g., Very Fast Decision Tree (see Section 5.1) and Very Fast Decision Rules (see Section 5.2). Therefore, Chapter 5 is entirely devoted to surveying and benchmarking existing works on feature drift adaptation and detection (see Chapter 5).

- Tracking the relevance of features as a stream progresses is not straightforward. As one of the major contributions of this thesis, adaptive formulas for Information's Theory Entropy and Symmetrical Uncertainty are provided, thus allowing their adaptive computation following a sliding window approach (see Chapter 6).

- Instance-based learning algorithms often rely on which instances are deemed "close" to predict class labels. To determine how "close" two instances are, distance measures are commonly applied, being the Euclidian the most widely used. Nevertheless, the Euclidian distance allows irrelevant features to possess the same importance as those which are relevant. Another common approach for classification is Bayesian learning. Bayesian learning is often referred to as an appropriate approach for high-dimensional classification problems (CHEN; WANG, 2012), however, it has been recently shown to be prone to features drifts (BARDDAL; GOMES; ENEMBRECK, 2015b). In this thesis, extensions to the k-Nearest Neighbor and Naive Bayesian algorithms are proposed so that features' discriminative power are promptly computed given a sliding window, and this discriminative power is used as dynamic weights during classification. Additionally, these modified learners are later used at the leaves of Hoeffding Adaptive Trees (BIFET; GAVALDÀ, 2009), thus leading to higher accuracy rates (see Chapter 7).

- The introduction of a dynamic filter for selecting the relevant subset of features of a data stream is proposed along with different selection strategies. This filter allows the discovery and tracking of discriminative features as the stream progresses, and the removal of irrelevant and redundant ones, both based on the concepts of Symmetrical Uncertainty (WITTEN; FRANK, 2005), Predominant Correlation (YU; LIU, 2003) and the Hoeffding Bound (HOEFFDING, 1963) (see Chapter 8).

- The proposal of a dynamic boosting-based feature selection process for data streams hereafter referred to as *Adaptive Boosting for Feature Selection* (ABFS). ABFS chains decision stumps and drift detectors, and as a result, identifies which features are relevant to the learning task as the stream progresses with relative success. As a result, ABFS improves the classification rates of different types of learners and eventually improve computational resources usage (see Chapter 9).

1.4 Publications

The main results reported in this thesis are reported in the following publications.

- Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. *Analyzing the Impact of Feature Drifts in Streaming Learning.* In: Proceedings of the 22th International Conference on Neural Information Processing (ICONIP'15). pages 21–28. 2015.

- Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. *A Survey on Feature Drift Adaptation.* In: Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'15). pages 1053–1060. 2015.

- Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, and Bernhard Pfahringer. *A Survey on Feature Drift Adaptation: Definition, Benchmark, Challenges and Future Directions.* In: Journal of Systems and Software. pages 278–294. 2016.

- Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Bernhard Pfahringer, and Albert Bifet. *On Dynamic Feature Weighting for Feature Drifting Data Streams.* In: Proceedings of the 2016 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD'16). pages 129–144. 2016.

- Jean Paul Barddal, Heitor Murilo Gomes, Jones Granatyr, Alceu de Souza Britto Jr., and Fabrício Enembreck. *Overcoming Feature Drifts Through Dynamic Feature Weighted k-Nearest Neighbor Learning.* In: Proceedings of the 2016 International Conference on Pattern Recognition (ICPR'16). pages 2186–2191. 2016.

- Jean Paul Barddal, Heitor Murilo Gomes, Alceu de Souza Britto Jr. and Fabrício Enembreck. *A Benchmark of Classifiers on Feature Drifting Data Streams.* In: Proceedings of the 2016 International Conference on Pattern Recognition (ICPR'16). pages 2180–2185. 2016.

- Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Albert Bifet, and Bernhard Pfahringer. *Merit-guided Dynamic Feature Selection Filter for Data Streams.* In: Expert Systems with Applications. 2019.

- **(Accepted – To appear)** Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Albert Bifet, and Bernhard Pfahringer. *Boosting Decision Stumps for Dynamic Feature Selection on Data Streams.* In: Information Systems.

1.5 Financial Support

This thesis was financially supported by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) through the *Programa de Suporte à Pós-Graduação de Instituições de Ensino Particulares* (PROSUP) program program under the Finance Code 001, which was later replaced by the *Programa de Suporte à Pós-Graduação de Instituições Comunitárias de Ensino Superior* (PROSUC). Also, between March and August of 2016, this thesis was developed at the University of Waikato, New Zealand, under the supervision of Professor Bernhard Pfahringer, in which travel and accommodation costs were financed by the Computer Science department of the hosting university. Finally, between September and December of 2017, the development of this thesis took place at Télécom ParisTech, at Paris, France, under the supervision of Professor Albert Bifet, while its respective travel expenses have been financed by CAPES under the *Programa de Doutorado-sanduíche no Exterior* (PDSE) program.

1.6 Overview

This thesis project is partitioned into three parts. The first part introduces (i) basic concepts on data stream mining, (ii) the fundamentals regarding features that enable a proper formalization of feature drifts, (iii) the evaluation framework adopted, and (iv) a survey of the existing algorithms for feature drift detection and adaptation. This first part is divided into the following chapters:

- Chapter 2 states the data stream mining problem, focusing on the classification task and its conventionally tackled challenges, i.e., concept drifts, bounded processing time and memory space.

- Chapter 3 is devoted to features and the definition of relevance, redundancy, and selection. This chapter also provides a categorization for feature selection algorithms and discerns between feature selection and other related techniques of the area. Furthermore, based on previously introduced definitions, this chapter delivers formalizations to the main topic of this thesis: **feature drifts**.

- Chapter 4 introduces the evaluation framework adopted throughout this work. It broadens data stream generators capable of synthesizing feature drifts, real-world data and the experimental protocol adopted, i.e., processing time, memory usage, and feature selection specific metrics, such as selection accuracy and stability. It is relevant to mention at this point that the computation of both Selection Accuracy and Stability do exist for batch scenarios, but they can also be considered as contributions of this thesis since they have not been assessed before in streaming scenarios.

- Chapter 5 yields a survey on existing works on feature drift adaptation and highlighting open gaps in the area (BARDDAL; GOMES; ENEMBRECK, 2015c; BARDDAL; GOMES; ENEMBRECK, 2015b). Furthermore, this chapter evaluates the use of drift detectors and existing classifiers in feature drifting streams, as discussed in (BARDDAL et al., 2017) and (BARDDAL et al., 2016b).

The second part introduces the main contributions of this work, encompassing novel adaptive feature scoring operators and their usage as a feature weighting scheme and as the core of a novel filter for dynamic feature selection for data streams. Finally, a boosting-based technique for feature selection on data streams is also proposed. A fine-grained overview of the second part and its chapters is as follows.

- Chapter 6 introduces the proposed dynamic scoring operators. Moreover, it shows how these operators can be swiftly computed over a data stream. The foundations here provided refer back to the information theoretic Entropy (SHANNON, 1948) and Symmetrical Uncertainty (WITTEN; FRANK, 2005).

- Chapter 7 shows how the proposed dynamic feature scoring schemes (BARDDAL et al., 2016; BARDDAL et al., 2016a) can be used to boost both Bayesian and instance-based learning in feature drifting streams and real-world datasets. Furthermore, it shows how these two feature-weighted algorithms can improve the prediction accuracy of the state-of-the-art Hoeffding Adaptive Trees (BIFET; GAVALDÀ, 2009).

- Chapter 8 introduces a novel proposal for dynamic feature selection from data streams. Additionally, this chapter assesses several selection strategies in a variety of synthetic and real-world scenarios.

- Chapter 9 introduces an dynamic boosting-based feature selection method for data streams. As the chapter above, it also encompasses an analysis of the proposed method in different scenarios and classifiers.

Finally, the last part concludes this work, providing a discussion about the results obtained and the existing gaps that will be assessed in future work.

- Chapter 10 states the conclusions obtained and introduces envisioned future works.

Chapter 2

Data Stream Mining

Recent advances in hardware and software allowed the gathering and storage of massive amounts of data. However, as foreseen by Wirth's law (WIRTH, 1995), computational processing advances did not grow as swiftly as the latter ones. Therefore, dealing and extracting useful knowledge from these massive amounts of data has become a great challenge due to the physical constraints of current computers.

In addition to data being generated in monumental quantity, most generators do so sequentially, thus giving rise to data streams (AGGARWAL, 2006; GAMA, 2010; WIDMER; KUBAT, 1996). Streaming applications involve sequentially generated datasets that are too big to be stored in main memory, and as a result, are stored in secondary memory. Since performing random access in secondary memory to retrieve data is a costly process, the only feasible data access mode is according to the arrival of data, a technique named *single-pass processing* (GUHA, 2009).

Extracting useful knowledge from data streams is a challenge per se. Most of the data mining techniques assume that there is a static set of data, which probability distribution is stationary and that can be analyzed in a multi-step fashion by a batch-like algorithm. Nevertheless, none of the above conditions can be verified in streaming scenarios, mainly due to their ephemerality, i.e., a stream's data underlying distribution is likely to change with time, giving rise to *concept drifts*.

This chapter probes the data stream mining problem. First, Section 2.1 reviews the most common approach for extracting useful knowledge from data streams: classification. The problem of concept drift is detailed in Section 2.2, and existing drift detectors are briefly discussed in Section 2.3. Finally, Section 2.4 concludes this chapter, focusing on its main aspects for the remainder of this work.

2.1 Data Stream Classification

Classification is the task that distributes a set of instances into discrete classes according to relations or affinities. Given a set of possible classes $Y = \{y_1, \dots, y_c\}$, a classifier builds a model that predicts for every unlabeled instance $\vec{x}$ its corresponding class y with accuracy.

Definition 1. The batch classification task can be formalized as follows: a set of n training instances in the $(\vec{x}, y)$ form, where $y \in Y$ is a discrete class label and $\vec{x}_i$ is a d-dimensional vector of attributes belonging to a feature set (dimensions) $\mathcal{X}$, that can be categorical, ordinal, numeric or mixed. A classifier produces from this training set a model $f : \vec{x} \to Y$ that is used to classify future unlabeled instances.

Definition 2. According to the Bayesian theory, classification can also be posed as the prior probabilities of the classes $P[y]$ and the class conditional probability density functions (*pdfs*) $P[\vec{x}|y]$ for all possible classes $y \in Y$ (DUDA; HART; STORK, 2001). The classification decision is performed given the posterior probabilities of the classes, where Equation 2.1 states the posterior probability for an arbitrary class y.

$$P[y|\vec{x}] = \frac{P[y] \times P[\vec{x}|y]}{P[\vec{x}]} \tag{2.1}$$

Data stream classification, or online classification, is a variant of the traditional batch classification. The difference between these two approaches regards how data is presented to the learner. In the batch configuration, a static and entirely accessible dataset is provided to the learning algorithm, which returns a model f to predict future instances. Conversely, in streaming environments, instances are not readily available to the classifier for training, instead, these are presented sequentially over time, and the learner must update its model according to the arrival of instances from the stream (BIFET, 2010; GAMA, 2010).

Definition 3. Let $\mathcal{S} = [(\vec{x}^t, y^t)]_{t=0}^{\infty}$ define a data stream providing instances (x^t, y^t), each of which arriving at a timestamp t, where $\vec{x}^t$ is a d-dimensional attribute[1] vector belonging to a feature set $\mathcal{X}$ and y^t is the corresponding ground-truth label (class) of $\vec{x}^t$.

Definition 4. The feature set of a data stream $\mathcal{S}$ can be described as $\mathcal{X} = [X_j]_{j=1}^{d}$ that can be continuous, ordinal, categorical, or mixed with cardinality d. In order to access the value of a feature X_i in an instance $\vec{x}^t$, the $\vec{x}^t[X_i]$ notation is used. Furthermore, the values of an attribute X_i are expressed as a single letter variable, as for instance, $q \in X_i$.

[1] Throughout this work, the words “attribute”, “dimension” and “feature” are used interchangeably.

In traditional batch machine learning, most of the techniques assume that there is a static dataset generated by an unknown and stationary probability distribution, which can be physically stored and analyzed in multiple steps by a batch algorithm. Nonetheless, none of the latter assumptions are verifiable in the streaming scenario and the development of algorithms must take into account several constraints (BIFET, 2010; GAMA et al., 2014; NGUYEN; WOON; NG, 2014; SILVA et al., 2013). Firstly, instances continuously become available over time and there is no control over their arriving order nor how they should be processed. Additionally, streams are potentially unbounded, therefore, instances should be discarded right after their processing (or given available main memory space). Due to the inherent temporal aspect of data streams, their underlying data distribution is expected to dynamically change over time, implying in changes in the concept to be learned, a phenomenon named **concept drift** (see Section 2.2).

2.1.1 Assumptions and Constraints

Throughout this work a few assumptions and constraints of streaming environments are considered:

- **Single pass processing:** Classifiers must be able to process instances sequentially according to their arrival. A classifier must process instances as soon as they become available and discard them right after. Although there is no restriction against buffering instances for a limited amount of time, this must not put in jeopardy the memory space and processing time constraints.

- **Memory space:** Primary memory is finite and its usage must be optimized. Consequently, classifiers and their respective models must be bounded according to existing available hardware.

- **Processing time:** The processing time of each arriving instance must not surpass the ratio in which new instances become available. If the processing time of each instance scales up, arriving instances will be discarded or enqueued for processing until the system crashes. Also, if the algorithm is unable of processing instances in real-time, it will not be unable to adapt to concept drifts (see Section 2.2) rapidly neither.

- **Label availability:** In this thesis it is assumed that after the arrival of an instance $\vec{x}^t$, its respective label y^t becomes available for training before the arrival of the following instance $\vec{x}^{t+1}$. This is, by far, the most used framework for the development of data stream learners (GAMA et al., 2004; GAMA; KOSINA, 2011) and frameworks, e.g., Massive Online Analysis (MOA) (BIFET et al., 2010) and Scalable Advanced Massive Online Analysis (SAMOA) (MORALES; BIFET, 2015). Although other problem settings for data streams do exist, e.g., semi-supervised and unsupervised learning fashions, they are not yielded here, therefore, the reader is referred to specific works on other learning schemes, e.g., unsupervised (SILVA et al., 2013) and semi-supervised learning (LIU et al., 2013; MASUD et al., 2011).

2.2 Concept Drift

Due to the temporal and ephemeral characteristics of data streams, these are expected to undergo changes in their data distributions, thus giving rise to **concept drifts** (WEBB et al., 2018). Even though concept drifts are possible in both regression and clustering problems, the formalization provided aims exclusively at the classification task, which is the scope of the current work. For references on concept drift detection and adaptation in these two tasks, the reader is referred to the following works (GAMA, 2010; NGUYEN; WOON; NG, 2014).

Definition 5. Let Equation 2.2 denote a concept C, a set of prior probabilities of the classes and class-conditional probability density function (NGUYEN et al., 2012).

$$C = \bigcup_{y \in Y} \{(P[y], P[\vec{x}|y])\} \tag{2.2}$$

Given a stream $\mathcal{S}$, instances $(\vec{x}^t, y^t)$ will be generated by the current concept C^t. If during every instant t_i of $\mathcal{S}$ we have $C^{t_i} = C^{t_{i-1}}$, it occurs that the concept is stable. Otherwise, if between any two timestamps t_i and $t_j = t_i + \Delta$ occurs that $C^{t_i} \neq C^{t_j}$, we have a concept drift.

The cause of drifts cannot be determined nor predicted by conventional learning algorithms since it is not feasible to assume that these hold access to secondary data sources or its access cost is high enough to be considered prohibitive. Therefore, data stream classification algorithms must detect and adapt to these changes automatically and autonomously.

If the data generator process is non-stationary (as in most real applications), changes in the context impact the concept to be learned; therefore, detecting and adapting to concept changes is an obligation for new algorithms for data stream mining (GAMA et al., 2014). Besides detecting concept drifts, it is also expected that algorithms be capable of discerning between a concept drift and noisy data and outliers (WIDMER; KUBAT, 1996; TSYMBAL, 2004).

Concept drifts may occur in two fashions: abruptly or gradually. In order to determine whether a drift occurs abruptly or gradually, one must analyze the size of the drift window W_{drift}. Hypothetically, considering that a drift occurs after an instance $\vec{x}^t$ and that it becomes stable after an instance $\vec{x}^{t+W_{drift}}$, if $W_{drift} = 1$ holds, the drift is said abrupt, otherwise ($W_{drift} > 1$), gradual.

Inside a drift zone, the probability of an instance $\vec{x}^t$ of belonging to an old concept C_A or to the new one C_B is specific to each problem domain. However, many of these probabilities can be synthesized through well-known probability distribution functions (*pdfs*). Regardless of the function being adopted, we need to guarantee that $P[C_A] + P[C_B] = 1$, as $P[\vec{x}^t \in C_A] = 1 - P[\vec{x}^t \in C_B]$. Finally, an interesting point in these functions is the convergence of these probabilities, i.e., when $P[\vec{x}^t \in C_A] = P[\vec{x}^t \in C_B] = 0.5$, which is known as the "drift moment", hereafter denominated t_{drift}.

2.3 A Note on Drift Detectors

Learning with concept drifts is a challenging problem that raises several questions. Many proposals were introduced in the last years to tackle this issue, broadening the use of drift detectors to adaptation through ensemble-based learning. This section aims solely at drift detectors and does not claim for completeness, yet, focuses on recent proposals that presented better average results in recent empirical evaluations (BIFET et al., 2015; FRÍAS-BLANCO et al., 2015; GAMA et al., 2014; GONÇALVES et al., 2014; SIDHU, 2015). Furthermore, specific details about the functioning of each of the detectors mentioned in this section can be found in Appendix A.

It is important to highlight that all of the surveyed drift detectors assume as input the misclassification rates of a classifier. Therefore, a drift is signalled when the monitored misclassification rate deviates from its usual value past a certain detection threshold, computed based on a statistical upper bound or a significance technique. Nevertheless, their statistical procedures are generic and can be applied to any sequence of real numbers.

By far, the most widely used drift detector is the *Adaptive Sliding Window Algorithm* (ADWIN) (BIFET; GAVALDÀ, 2007). ADWIN keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window". The ADWIN change detector is parameter- and assumption-free in sense that it automatically detects and adapts to the current rate of change.

Next, the *Exponentially Weighted Moving Average Control Charts* (ECDD) weights the misclassification rates according to their position inside a sliding window using an exponential function (ROSS et al., 2012). The ECDD output rate fluctuates across three threshold levels: in-control, warning, and out-of-control. A drift will be flagged by ECDD whenever the misclassification rate of a learner reaches the out-of-control level.

Finally, the authors in (FRÍAS-BLANCO et al., 2015) proposed two variants of the Hoeffding Drift Detection Method (HDDM) detector: HDDM-A and HDDM-W. Both the former and the latter are similar to ECDD in the sense that they use moving averages to detect drifts, yet, only the latter uses an exponentially weighted procedure to provide higher importance to most recent data. In both cases, the moving averages are compared to flag concept drifts based on the misclassification rates of a classifier, where the Hoeffding Bound (see Section 5.1 for more details) is used to set an upper bound to the accepted level of difference between them.

2.4 Concluding Remarks

This chapter provided an overview of data stream mining, focusing on the classification task and its major challenges regarding processing time, memory usage, and concept drift identification. This chapter does not claim for completeness, therefore, the reader interested in more details regarding these topics is referred to surveys on data stream mining and concept drift adaptation (GAMA, 2010; GAMA et al., 2014; NGUYEN; WOON; NG, 2014; WANKHADE; HASAN; THOOL, 2013; WEBB et al., 2018).

Additionally, this chapter also reviewed the most used drift detectors. This review is important since change detectors presented here are part of existing works (see Chapter 5) and will be used during empirical analyses throughout this thesis (Chapters 7 through 9). The following chapter provides insights into features, broadening basic definitions, e.g., feature relevance, redundancy, and selection; that together enable the introduction of feature drifts, which is the main topic tackled in this work.

Chapter 3

Features: From Foundations to Drifts

Datasets and streams may contain hundreds, thousands, or even millions of features, many of which may be irrelevant or redundant to the mining task. Dealing with this massive amount of features is not only computationally expensive, but they are also likely to damage inductive learning algorithms. Observing too many features often leads to unnecessarily complex models and makes the problem too sparse, since the training set would cover a dwindling part of the attribute space. Even if we assume a large dataset with trillions of instances (examples) in a moderate attribute space of 100 binary dimensions, only about 10^{-18} of the feature space would be covered (DOMINGOS, 2012). Also, great dimensions can be a problem due to the *curse of dimensionality*, where learning algorithms based on distance computations are known to fail (AGGARWAL; HINNEBURG; KEIM, 2001). As a result, dimensionality reduction is of the utmost importance to overcome all of the aforementioned issues. Yet, if this reduction is poorly performed, and relevant attributes are left out or irrelevant ones are kept, the whole process would be damaged and result in patterns of poor quality (CHANDRASHEKAR; SAHIN, 2014).

This chapter is dedicated to features, broadening their foundations, the selection task, and finally, feature drifts. Firstly, Section 3.1 defines feature relevance, while Section 3.2 defines feature redundancy. Both are important definitions that will be extensively used throughout this work. Later, Section 3.3 formally defines the feature selection task and its paradigms, while Section 3.4 reviews the classic categorization of feature selection algorithms. Sections 3.5 and 3.6 provide additional information regarding the no free lunch theorem and the difference between feature selection and extraction, respectively. Section 3.7 is devoted to feature drifts, the main issue tackled in this thesis, while Section 3.8 details the differences between *dynamic feature selection* and *streaming feature selection*. Finally, Section 3.9 concludes this chapter by reviewing important topics that will be useful in the remainder of this work.

3.1 Feature Relevance

Up to this point, the term “relevance” was used without a formal definition. This section defines the concept of relevance in the feature selection task. As stated in (KOHAVI; JOHN, 1997; RUDNICKI; WRZESIEŃ; PAJA, 2015; CIOS et al., 2007), there do exist different definitions available in the literature, nevertheless, several may be contradictory and misleading. In this work, the definition provided in the seminal work of (KOHAVI; JOHN, 1997) is followed, as features are divided into irrelevant, strongly relevant, or weakly relevant.

Definition 6. Assuming $S_i = \mathcal{X} \setminus \{X_i\}$ and s_i to be a value-assignment to all variables in S_i, a feature X_i is **strongly relevant** *iff* there exists some q, y and s_i for which $P[X_i = q, S_i] > 0$ such that the following holds:

$$P[Y = y|X_i = q, S_i = s_i] \neq P[Y = y, S_i = s_i] \tag{3.1}$$

Definition 7. A feature X_i, will be considered as **weakly relevant** *iff* Definition 6 does not hold, and there exists a subset of features $S'_i \subset S_i$ for which exists some q, y and s'_i with $P[X_i = q, S'_i = s'_i] > 0$ such that the following holds:

$$P[Y = y, |X_i = q, S'_i = s'_i] \neq P[Y = y, S'_i = s'_i] \tag{3.2}$$

Otherwise, X_i is said to be **irrelevant**.

According to the previous definitions, if a feature that is statistically relevant is removed from a feature set, it will incur in changes in the prediction power, most likely reducing it. This definition encompasses two possibilities for a feature to be statistically significant: (i) it alone is strongly correlated with the class; or (ii) it forms a feature subset with other features and this subset is strongly correlated with the class (CHANDRASHEKAR; SAHIN, 2014; ZHAO et al., 2010).

3.2 Feature Redundancy

Besides relevance, another important property of features is redundancy. Redundancy notions are normally given in terms of feature correlation, since it is widely accepted that two features are redundant to a concept if their values are correlated. Redundant features are problematic and should be eliminated since they provide an extra computational cost for both storage and processing (YU; LIU, 2004) and make classifiers more prone to overfitting (YU; LIU, 2003). Therefore, there is also the need of determining and eliminating redundant features prior to learning.

Definition 8. Assuming $S_i = \mathcal{X} \setminus \{X_i\}$, a feature X_i is **redundant** *iff*

$$P[Y|S_i] \approx P[Y|X_i, S_i] \tag{3.3}$$

According to Definition 8, a feature becomes redundant due to the existence of another feature that provides similar prediction power. Several studies proposed the removal of redundant features as this might improve prediction accuracy since fewer features often lead to less overfitted models, while others noticed that the removal of this type of feature may cause the exclusion of potentially relevant features. Most of the existing works propose to find redundant features through correlations (HALL; SMITH, 1999; HALL, 2000; YU; LIU, 2003) or grouping similar patterns into feature clusters (OES et al., 2009; PARK, 2013).

3.3 Feature Selection and Paradigms

The number of possible definitions for the feature selection task is nearly as wide as the amount of proposed methods to perform it. This section introduces the three most used definitions and discuss their respective rationales. Regardless of the strategy being used, the goal of feature selection is to remove irrelevant and/or redundant features, while maintaining the probability distribution of the original data classes $P[Y]$. Learning from a dataset with smaller dimensionality has several benefits: (i) it results in a smaller amount of parameters in the patterns discovered, thus making the final classification model simpler and easier to understand, (ii) assuming that the feature selection was successful, we should obtain as good or better accuracy rates compared to the model trained with all the original features also while requiring less data (CARVALHO; COHEN, 2006; CHANDRASHEKAR; SAHIN, 2014), and (iii) smaller processing times and memory consumption rates, due to items mentioned above (NAIDU; DHENGE; WANKHADE, 2014).

3.3.1 Relevance and Redundancy Criteria-based Feature Selection

Based on the definitions of relevance and redundancy, the feature selection task can be formalized as a process that eliminates irrelevant and/or redundant features.

Definition 9. The feature selection task based on the definitions of relevance and redundancy can be posed as follows:

$$\exists \mathcal{X}_r, \mathcal{X}_t \subseteq \mathcal{X},\ \mathcal{X}^* = \mathcal{X} \setminus \{\mathcal{X}_r\} \setminus \{\mathcal{X}_t\} \tag{3.4}$$

where $\mathcal{X}_r$ is the subset of features that are deemed relevant (for which Definitions 6 and 7 hold) and $\mathcal{X}_t$ is the subset of redundant features in $\mathcal{X}$ (for which Definition 8 is true).

Computing redundancy is a task that is computationally expensive in the $\mathcal{O}(2^d)$ order since there is the need to verify the redundancy between each possible pair of features. Therefore, the most simplistic approaches (mostly filters – see Section 3.4) for feature selection often do not encompass redundancy verification by relaxing this constraint.

In order to determine if a feature is either relevant or redundant, proposals use both linear (e.g., Pearson correlation) and nonlinear approaches (e.g., Entropy, Information Gain and Symmetrical Uncertainty). This section refrains from providing a detailed description of such metrics since many of these are fundamental and will be explored throughout the proposed methods introduced in Chapters 6, 7, and 8.

3.3.2 Optimal Feature Selection

Optimal feature selection is a paradigm that relies on a predictor and its performance. More precisely, optimal feature selection depends on the predictor (usually a classifier), its performance evaluation, e.g., Accuracy, Area under the Receiver Operating Characteristic Curve (AUROC), and a dataset (CHANDRASHEKAR; SAHIN, 2014; WITTEN; FRANK, 2005). Even though this type of method claims for optimatily, their solution may not be unique since different subsets of features may incur in the same performance.

Definition 10. Optimal feature selection can be described an optimization problem $\mathcal{X}^*$ as follows:

$$\mathcal{X}^* = \underset{\mathcal{X}' \subseteq \mathcal{X}}{\operatorname{argmax}}\, Q(\mathcal{X}') \tag{3.5}$$

where $Q(\cdot)$ is the evaluation metric of a classifier in the possibly diminished dataset described by the features in $\mathcal{X}' \subseteq \mathcal{X}$.

Finding $\mathcal{X}^*$ is a difficult task that, assuming $d_{max} = d$, requires an exploratory search which, by definition, leads to a search with $\mathcal{O}(2^d)$ cost. Due to the exponential computational complexity, all algorithms do possess some kind of heuristic to guide the selection process, which may lead to different feature subsets. Therefore, designers of optimal feature selection algorithms must account for the bias/variance dilemma, which underlines the controversy of a learning algorithm and feature set complexity, namely the need to find the best model for a given dataset simultaneously to providing better generalization for future instances (CIOS et al., 2007; SAMMUT; WEBB, 2011). If the designer overfits the complex processing algorithm to given data in a large feature set, then the algorithm's ability to generalize for upcoming instances may deteriorate. By increasing the complexity of the processing algorithm and feature set, it is possible to reduce bias and increase variance. On the other hand, a processing algorithm with a small feature set may not be able to process a given dataset satisfactorily. Simplistic and inflexible learning algorithms (with a small number of parameters), when applied to small feature sets, may have too big a bias and too small variance. Therefore, robust learners must implement this tradeoff between its ability to maximize the gain into the training dataset while not overfitting to it, managing to augment its generalization capability for predicting future instances accurately.

3.3.3 Minimum Construction Paradigm Feature Selection

Performing optimal feature selection is problematic due to the number of feature subsets to be explored and evaluated. The minimum construction paradigm relaxes this problem of optimal feature selection by defining a maximum amount of features to be selected. This can be performed via heuristics like the *Occam's razor*[1], which hypothesizes that the simplest model obtained from observations of a phenomena is most likely the correct one; or analytical formalizations to it, as the Minimum Description Length (MDL) (BARRON; RISSANEN; YU, 1998).

Definition 11. Assuming the full set of features $\mathcal{X}$, the goal of feature selection is to select a subset $\mathcal{X}^*$ with a maximum length d_{max} that retains the relevant information in a dataset. Suppose that the goodness of a subset of features $\mathcal{X}' \subseteq \mathcal{X}$ is given by $Q(\cdot)$, then feature selection can be stated as in Equation 3.6, where d_{max} is the upper bound

[1] William of Ockham (c. 1287–1347) was an English Franciscan friar and philosopher that produced significant works on logic, physics and theology. The original claim behind its most notorious work, the Occam's razor, is that when confronted with a problem-solving situation, one should assume the hypothesis with least assumptions.

on the number of selected features.

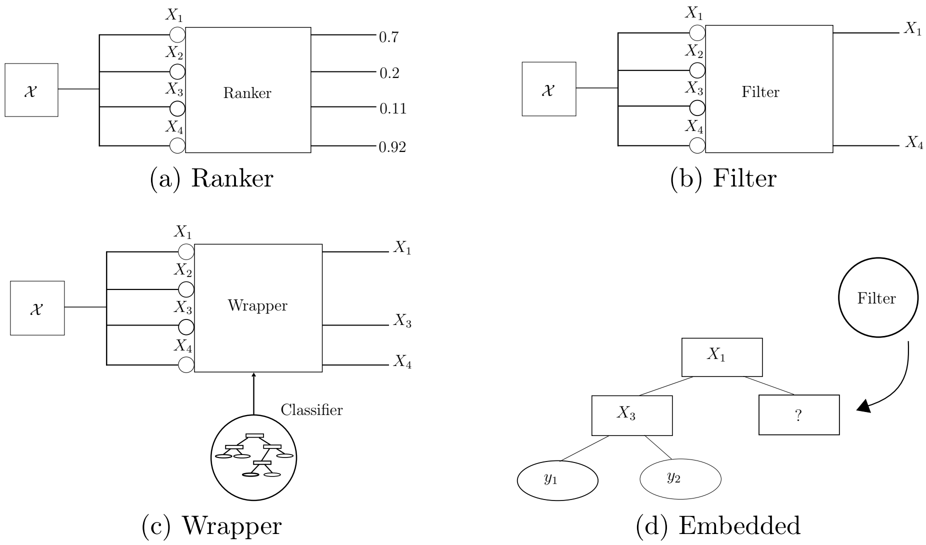
$$\mathcal{X}^* = \underset{\mathcal{X}' \subseteq \mathcal{X}}{\text{argmax}} \; Q(\mathcal{X}') \text{ subject to } |\mathcal{X}'| \leq d_{max} \tag{3.6}$$

This paradigm prunes the amount of subsets of features to be evaluated from 2^d to $2^{d_{max}}$, where $d_{max} \ll d$. Nonetheless, this decrease may still result in an unfeasible processing time for feature selection if the evaluation metric $Q(\cdot)$ depends on building and evaluating a classifier. Therefore, it is also common for minimum construction paradigm-based feature selection algorithms to assume heuristic evaluation metrics, e.g., Pearson Correlation, Entropy, Information Gain; to evaluate each possible subset of features. Although clearly non-optimal, algorithms designed according to this paradigm are important in applications where the classifier must act within limited memory space, therefore, its memory usage can be bounded according to d_{max}, which is often an user-given parameter.

3.4 Categorization of Feature Selection Methods

Beyond the paradigms provided in the last section, it is also important to categorize feature selection methods into the following classes: rankers, filters, wrappers and embedded methods (GUYON, 2003). Details on existing algorithms are not surveyed since they are neither incremental nor adaptive, and thus, not applicable in streaming scenarios. Nevertheless, this categorization is useful in introducing related works in Chapter 5.

Rankers assign a score to each feature based on a user-given criterion, e.g., mutual



(a) Ranker

(b) Filter

(c) Wrapper

(d) Embedded

Figure 3.1: Visual schematics for ranker, filter, wrapper and embedded feature selection approaches assuming a feature set $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$.

information, correlation, entropy. Rankers are often not referred as pure feature selection algorithms since they do not eliminate features, yet, provide a metric that determines to what extent each feature is relevant to the concept. Figure 3.1a presents a visual schematic of a ranker, where 4 features are inputted and the ranker outputs an importance metric to each one of them.

Filters are similar to rankers in the sense that they apply statistical measures to assign a score to each feature. Features are then ranked by scores and either selected to be kept or removed according to a threshold or a merit-guided selection. This threshold can be defined as the maximum amount of features to be maintained or a minimum score that deems a feature as relevant. These methods are usually univariate and consider each feature independently or regarding the dependent variable (class feature). Two important traits of filters are their independence from the learning algorithm adopted and low computational cost. Examples of filter methods include, but are not limited to, the χ^2 test, Information Gain, Entropy, Correlation Coefficient Scores (LIU; MOTODA, 2007), Las Vegas Filters, Relief, ReliefF (NAIDU; DHENGE; WANKHADE, 2014), Correlation-based Feature Selection (CFS) (HALL; SMITH, 1999; HALL, 2000) and the Fast Correlation-based Filter (FCBF) (YU; LIU, 2003). Figure 3.1b presents a visual representation of a filter, where only two of the original four features are selected.

Wrappers consider the selection of a subset of features as a search problem, where different combinations are prepared, evaluated and pairwisely compared, usually in bottom-up or top-down approaches. Also, a predictive model is used to evaluate each combination of features and assign a score based on each model accuracy, therefore, wrappers can be stated as optimization systems. Wrappers are sensitive to the learning algorithm's bias and recognize that certain algorithms may work better with different features (AGGARWAL, 2014). Nevertheless, an important drawback of wrappers is their computational cost, which is computationally prohibitive in high dimensional or in real-time scenarios. The most common search processes are best-first search, stochastic hill-climbing algorithms, forward and backward passes, beam search, and simulated annealing; to add or remove features from the current subset. Examples of wrapper feature selection methods are exhaustive search, recursive elimination algorithm (YUAN; HREBIEN; KAM, 2005), branch and bound (CHEN, 2003), stepwise forward search, greedy stepwise search (HALL et al., 2009) and backward feature selection (CIOS et al., 2007). Figure 3.1c depicts a wrapper, where a classifier (graphically represented as a decision tree) is inputted alongside the feature set $\mathcal{X}$.

Finally, embedded methods learn which features best contribute to the overall accuracy of the learning algorithm while the model is created. Decision trees and rules learning are examples of embedded methods since the construction of the model (tree or set of rules) and the selection of the features are interleaved, but the selection of features itself is done by filters. Embedded approaches interact directly with the learning algorithm and present better computational complexity than wrappers (CHANDRASHEKAR; SAHIN, 2014). Figure 3.1d exemplifies the construction of a decision tree, where a filter is used in order to decide which feature will be tested in every split.

3.5 The No Free Lunch Theorem and Feature Selection

The "No Free Lunch Theorem" is a result of optimization algorithms (DUDA; HART; STORK, 2001). It states that there is no algorithm that can outperform stochastic search on all problems. The same rationale can be applied to feature selection. For instance, empirical analysis conducted in (BACCIANELLA; ESULI; SEBASTIANI, 2010) showed that certain randomly picked pairs and triples of features may give better results than the best 10—20 single features ranked by Gini Impurity and Information Gain.

Another important consideration about feature selection algorithms is the effect of classification bias. In the case of filters, which are classifier-independent, it is possible that resulting selected features lead to significantly different results in terms of accuracy when these are inputted to different types of classifiers, e.g., decision tree and instance-based learning. Therefore, there is the need to evaluate the results obtained from a feature selection algorithm either (i) across multiple classifiers with different biases; or (ii) classifier-independently, if there is a known ground-truth set of non-redundant and relevant features to be used for comparison (SAMMUT; WEBB, 2011), e.g., Selection Accuracy, which is later discussed in Section 4.4.3.

3.6 Feature Selection versus Extraction

Feature selection and extraction are two different tasks that aim at performing dimensionality reduction of a dataset. The first selects a subset of features among the entire feature set from the original dataset, i.e., $\mathcal{X}' \subset \mathcal{X}$, while the latter generates new features based on the original dataset (CIOS et al., 2007). Feature extraction refers to the transformation of a original d-dimensional dataset into a d'-dimensional one, such that $d' \leq d$. The transformation and dimensionality reduction function can be considered as a non-linear transformation, which must be designed based on available knowledge about the domain and data statistics. The gains of data transformation through projection include: (i) the removal of irrelevant features, (ii) redundancy removal, (iii) dataset compression, and a (iv) low-dimensionality view of data which enables better clustering and visualization of other relationships in data. Even though the reasons for performing data transformation highly overlap with those provided by feature selection, the result of feature projection jeopardizes readability since transformed features may not evoke back to the original ones. The scope of this thesis does not encompass feature extraction, therefore, projection techniques, e.g., principal component analysis, random projections, semidefinite embedding and latent semantic analysis and Wavelets; are not part of this work. For details on feature extraction from data streams, the reader is referred to (KUNCHEVA; FAITHFULL, 2014; HUNTER; COLLEY, 2007; YAN et al., 2006).

3.7 Feature Drift

Most of existing algorithms for data streams tackle the infinite length and drifting concept characteristics. However, not much attention has been given to a specific kind of drift: feature drifts. Feature drifts occur whenever a subset of features becomes, or ceases to be, relevant to the concept to be learned. This enforces the learning algorithm to adapt its model to ignore the irrelevant attributes and account for the newly relevant ones (NGUYEN et al., 2012).

Definition 12. Given a feature space $\mathcal{X}$ at a timestamp t, we are able to select the ground-truth relevant subset $\mathcal{X}_t^* \subseteq \mathcal{X}$ such that $\forall X_i \in \mathcal{X}_t^*$ definitions 6 and 7 hold and $\forall X_j \in \mathcal{X} \setminus \mathcal{X}_t^*$ the same definitions do not. A feature drift occurs if, at any two time instants t_i and $t_j = t_i + \Delta$ with $\Delta > 0$, $\mathcal{X}_{t_i}^* \neq \mathcal{X}_{t_j}^*$ betides.

Definition 13. Let $r(X_i, t_j) \in \{0, 1\}$ denote a function which determines whether the disjunction between definitions 6 and 7 holds for a feature X_i in a timestamp t_j of the stream. A positive relevance ($r(X_i, t_j) = 1$) states that $X_i \in \mathcal{X}^*$ in a timestamp t_j and that X_i impacts the underlying probabilities $P[\vec{x}|y_i]$ of the concept C_t of $\mathcal{S}$. A feature drift occurs whenever the relevance of an attribute X_i changes in a timespan between t_j and t_k, as stated in Equation 3.7.

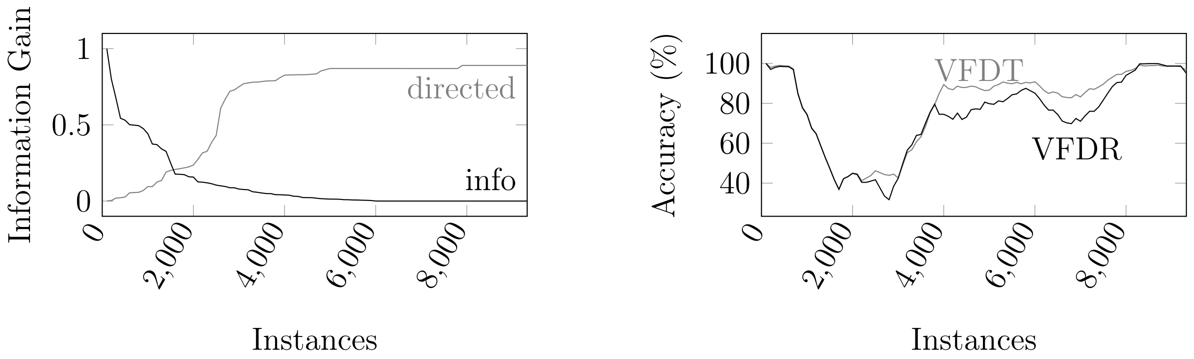
$$\exists t_j \exists t_k,\ t_j < t_k,\ r(X_i, t_j) \neq r(X_i, t_k) \tag{3.7}$$

Changes in $r(\cdot, \cdot)$ directly affect the ground-truth decision boundary to be learned by the learning algorithm. Therefore, feature drifts can be posed as a specific type of concept drift that may occur with or without changes in the data distribution $P[\vec{x}]$ (BARDDAL et al., 2017).

As in conventional concept drifts, changes in $r(\cdot, \cdot)$ may occur during the stream. This enforces learning algorithms to detect changes in $\mathcal{X}^*$, discerning between features that became irrelevant and the ones that are now relevant and vice-versa. Finally, it is necessary to either (i) discard and learn an entirely new classification model; or (ii) adapt the current model to relevance drifts (NGUYEN et al., 2012). It is important to emphasize that feature drifts are indeed targeted by the generic concept drift formalization, yet, most existing works on concept drift detection and adaptation assume that the relevant subset of features remains the same and that drifts occur if certain values (or ranges of values) of attributes have their class distribution re-skewed.

Although feature drifts may occur in a variety of environments, one of the most common is text mining. In order to exemplify a feature drift, one should consider the e-mail spam detection system presented in (KATAKIS; TSOUMAKAS; VLAHAVAS, 2006). This system is the result of a text mining process on an online news dissemination system. Essentially, this work intended on creating an incremental filtering of emails that classifies emails as spam or ham and, based on this classification, it decides whether an e-mail is relevant for dissemination among users. The dataset contains 9,324 instances and 39,917 features, such that each attribute represents the presence of a single word (feature) in an instance (e-mail). This dataset, namely Spam Corpus, is known for containing a feature drift which occurs gradually around the instance of number 1,500 (KATAKIS; TSOUMAKAS; VLAHAVAS, 2006) and that highly impacts the learner.

Figure 3.2a presents a plot of the information gain (a possible feature goodness measure for determining its discriminative power) (HALL et al., 2009) of two specific attributes presented in this problem, namely "directed" and "info", where one can see that the importance of these two features starts to gradually change around instance 1,500 (BARDDAL; GOMES; ENEMBRECK, 2015b; KATAKIS; TSOUMAKAS; VLAHAVAS, 2006). Detecting and discerning the 2 features out of nearly 40,000 that exchange relevances as the stream's progress is an important example of task that must be embedded within streaming learning algorithms, since changes greatly impact the accuracy of the model (Figure 3.2b) and learning with a subset of the original feature set is also computationally faster. A detailed description of these classifiers is omitted since the Very Fast Decision Tree (VFDT) and Very Fast Decision Rules (VFDR) are discussed in Sections 5.1 and 5.2, respectively.



(a) Information gain of features "directed" and "info" during the stream.

(b) Accuracy obtained for a decision tree (VFDT) and a decision rule learner (VFDR).

Figure 3.2: Analysis of information gain for two specific features and accuracy obtained on the Spam Corpus dataset. Adapted from (BARDDAL; GOMES; ENEMBRECK, 2015c; BARDDAL; GOMES; ENEMBRECK, 2015b).

3.8 Dynamic Feature Selection versus Streaming Feature Selection

At this point, it is important to emphasize the difference between Dynamic Feature Selection for data streams and Streaming Feature Selection (also commonly referred as Online Feature Selection (HOI et al., 2012; LI et al., 2013; YIN et al., 2015)). In this work, the dynamic feature selection is said to be the process that continuously keeps track of the relevance of features over time and that selects only a subset of them according to this heuristic. Conversely, streaming feature selection regards the possibility of finding the best subset of features in a very high-dimensional space (hundreds of thousands or millions of dimensions), which is a typical problem of big data (HOI et al., 2012). Although both tasks' objectives overlap in the sense that both tackle the issue of feature selection, streaming feature selection receives as input a stream of features (not instances), and their inclusion in the model is performed sequentially, without observing future features (ZHOU; HUANG; SCHÖLKOPF, 2005). Therefore, Streaming Feature Selection is used in batch learning, when the number of features is gigantic, scaling up to thousands or millions of attributes and the number of instances is static and does not vary with time (YIN et al., 2015).

3.9 Concluding Remarks

During this chapter, several definitions regarding features were presented and formalized. Feature relevance and redundancy are concepts that guide this work and will be exhaustively used during the remainder of this thesis. As discussed in this section, performing feature selection, even in static datasets, is a challenging task that requires heuristics in order to avoid exhaustive searches, and thus, efficient techniques that provide accurate feature subsets is a challenge. Finally, this chapter formalized feature drift, a nearly neglected issue of data streams that will be tackled throughout the next chapters. More specifically, the following chapter introduces the evaluation framework adopted to evaluate both existing algorithms for data streams and the proposed methods.

Chapter 4

Evaluation Framework

In order to assess the applicability of both existing and proposed algorithms, an analysis environment was constructed. First, Section 4.1 introduces the computational framework that allowed the implementation and evaluation of learning algorithms, data generators and evaluation metrics, the Massive Online Analysis (MOA) (BIFET et al., 2010). Next, Section 4.2 discusses synthetic data generators and the procedure adopted to introduce irrelevant and redundant data into streams, while Section 4.3 briefly describes the real-world datasets used. Section 4.4 presents the experimental protocol, broadening not only accuracy, processing time and memory usage but also feature selection specific metrics and the statistical testing procedure adopted. Finally, remarks on this analysis environment are discussed in Section 4.5.

4.1 Massive Online Analysis (MOA)

The Massive Online Analysis (MOA) is a popular open source framework for data stream mining that includes a collection of machine learning algorithms and tools for evaluation (BIFET et al., 2010). MOA was designed to deal with several problems of the streaming scenario, broadening synthetic data generation, the use of real datasets, the implementation of new algorithms and evaluation metrics. Due to the open source characteristic, MOA enables the quick test of hypotheses by data stream mining researchers.

One important aspect to be recalled here is that the classifiers embedded within the MOA framework are either incremental or adaptive. This is an important characteristic since no explicit windowing is required for classifiers to be retrained on. Otherwise, a conventional batch classifier, e.g., J48, Naive Bayes, Logistic Regression, would be trained every time a new batch of instances with a pre-defined window size becomes available. In practice, such behavior would require *a priori* knowledge about the data distribution and whether concepts drifts are expected to occur and when so that an appropriate window size is set before training.

Also, even though MOA has a strong aim towards the classification task, it also comprehends regression, clustering, outlier detection and recommending systems tasks, all in streaming fashion. MOA was written in Java and allows the quick addition or adaptation of existing code, therefore, is widely used in several works. MOA and several plug-ins are available for download from <http://moa.cms.waikato.ac.nz/>.

4.2 Synthetic Data Generators

In order to evaluate whether a learning algorithm is able to work in different scenarios, it is necessary to assess its performance across different datasets. In opposition to real-world data, synthetic data stream generators are important and widely used due to their flexibility since they allow a precise definition of drifts types and location during the streams. This section starts by presenting the general drift framework adopted for experiments. Later, several generators able to synthesize feature drifts are surveyed and proposed, thus enabling proper evaluations of learning algorithms in feature drifting scenarios. All of these generators were either available at MOA or will be made available in an upcoming release.

4.2.1 Drift Framework

Drifts in experiments are synthesized according to the framework proposed in (BIFET, 2010). This framework models a drift as the change between two pure distributions given by two different concepts C_A and C_B. Intuitively, at the beginning of a drift window there is a higher probability that instances belongs to the concept C_A. As we move towards its end, the probability that it belongs to concept C_B raises. The drift window ends when concept C_B becomes stable. To model the probability that every new instance i_t drawn from $\mathcal{S}$ belongs to concept C_A or C_B this framework adopts a sigmoid function stated in Equation 4.1, where $P[C_B]$ and $P[C_A] = 1 - P[C_B]$ are the probabilities of $(\vec{x}^t, y^t)$ belonging to C_A or C_B, W is the drift window size, t is the current timestamp and t_{drift} is the time of the drift, i.e., $P[C_A] = P[C_B] = 50\%$ occurs.

$$P[C_B] = f(t) = |1 - P[C_A]| = \frac{1}{\left(1 + e^{-W_{drift}(t - t_{drift})}\right)} \tag{4.1}$$

In (BIFET, 2010) authors observe that Equation 4.1 has a derivative at time t_{drift}

equal to $f'(t_{drift}) = {}^{s}\!/\!_{4}$ and that tan $\beta = f'(t_{drift})$, thus tan $\beta = {}^{s}\!/\!_{4}$. Also, tan $\beta = 1/W$ and as $W_{drift} = 4$ tan β then $\beta = {}^{4}\!/\!_{W}$, where β is an optional phase angle. In this sigmoid model there are only two parameters to be specified: t_{drift} and W_{drift}.

Therefore, with the following generators, feature drifts occur when the relevant subset of features $\mathcal{X}^*$ of C_A differs from the relevant subset of features $\mathcal{X}^*$ of the subsequent concept C_B.

4.2.2 AGRAWAL Generator (AGR)

The AGRAWAL generator (AGRAWAL; IMIELINSKI; SWAMI, 1993) produces data streams with the aim of determining whether a loan should (or not) be given to a bank customer. This generator is composed of 6 categorical and 3 numeric features, and possesses 10 functions for mapping instances to two possible classes, each of which relying on different subsets of features. A perturbation factor is used to add noise to data, thus, creating fuzzy decision borders. This factor changes the original value of a feature with the addition of a deviation value, which is defined by a uniform distribution.

4.2.3 Asset Negotiation Generator (AN)

The Asset Negotiation (AN) generator[1] was originally presented in (ENEMBRECK et al., 2007), where the aim was to simulate drifting bilateral multi-agent system negotiation of assets, each of which being described by 5 attributes. The task is to predict whether an opposing agent would be interested or not in an asset (binary classification problem). Drifts were synthesized by changing the interest of this agent by changing the concept through time according to five functions.

4.2.4 Binary Data with Feature Drift Generator (BG-FD)

The Binary Generator with Feature Drift (BG-FD) generates instances composed of boolean ($\{0, 1\}$) features. BG-FD has three functions: BG1-FD, BG2-FD, and BG3-FD, all inspired by the work of (HALL, 2000).

In BG1-FD, presented in Equation 4.2, from the entire set of features $\mathcal{X}$, only a random subset $\mathcal{X}^* \subset \mathcal{X}$ is relevant to the concept to be learned. Additionally, $|\mathcal{X}^*| = X_r$, where X_r is a user-given parameter. Conversely, in BG2-FD (Equation 4.3) and BG3-FD (Equation 4.4), the size of the relevant subset of features is fixed, where $\mathcal{X}^* =$

[1] The Asset Negotiation Generator has been contributed to the Massive Online Analysis (MOA) framework and will be made available in MOA 2016-10.

$\{X_\alpha, X_\beta, X_\epsilon\}$.

$$y = \begin{cases} 1, & \text{if } \bigwedge\limits_{X_i \in \mathcal{X}^*} X_i \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

$$y = \begin{cases} 1, & \text{if } (X_\alpha \wedge X_\beta) \vee (X_\alpha \wedge X_\epsilon) \vee (X_\beta \wedge X_\epsilon) \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

$$y = \begin{cases} 1, & \text{if } (X_\alpha \wedge X_\beta \wedge X_\epsilon) \vee (\neg X_\alpha \wedge \neg X_\beta \wedge \neg X_\epsilon) \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

In all cases, class labels $y_i \in Y$ are evenly likely to occur and instances have a 10% probability of being generated as noise.

4.2.5 Random Tree Generator with Feature Drift (RTG-FD)

The original Random Tree Generator (RTG) builds a decision tree by randomly performing splits on features and assigning a random class label to each leaf. Instances are created by generating a random valued $\vec{x}$ and traversing the tree for its corresponding label. This generator is extended in this work to allow that given a random $\mathcal{X}^* \subset \mathcal{X}$ is relevant, where $|\mathcal{X}^*| < |\mathcal{X}|$ is a user-given parameter. Therefore, the remaining attributes generated will be either irrelevant or redundant.

4.2.6 SEA-FD Generator

In (BARDDAL; GOMES; ENEMBRECK, 2015b), authors proposed a data stream generator that extends the SEA generator (STREET; KIM, 2001) namely SEA-FD. SEA-FD simulates streams with $d > 2$ uniformly distributed features given by the user, where $\forall X_i \in \mathcal{X}, X_i \in [0; 10]$ and only two randomly picked features are relevant to the concept to be learned: $\mathcal{X}^* = \{X_\alpha, X_\beta\}$. As in (STREET; KIM, 2001), the class value y is given according to Equation 4.5, where θ is a user-given threshold.

$$y = \begin{cases} 1, & \text{if } X_\alpha + X_\beta \leq \theta \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

4.2.7 Adding Irrelevant and Redundant Data into Generators

The generators introduced in the last section rely on few attributes to determine the class outcome of an instance. Originally, such generators do not allow the introduction of extra irrelevant features and thus, to harden the experiments, all were extended to allow their addition.

The proposed strategy to append irrelevant features is to simply increment the attribute set $\mathcal{X}$ of a data stream with numeric or categorical attributes. In the first case, values for a numeric attribute are sampled from a uniform distribution bounded in $[0;1]$, with no regard to the instance outcome. The procedure for categorical features is similar, where new irrelevant attributes possess m different values, such that m is a user-given value and the probability of each partition being used in an instance equals $^1/_m$. With the exception of BG-FD experiments that are composed of binary attributes, m was set to 10 for all other scenarios. This value was empirically set and its variation did not show major differences in the experiments results, yet, m can be easily customized for future experiments.

Another trait that can deceive feature selection algorithms is the presence of redundant data. In addition to irrelevant attributes, three proposals for embedding redundant data into existing generators are also introduced.

Copy with perturbation factor. Let X_i and X_j be features such that the Definition 8 (redundancy) holds. This strategy generates values for X_j by copying the value X_i and adding a perturbation factor $p \in [0;1]$. If X_i is a numeric attribute, X_j will be set to a value in the $\vec{x}[X_i] \pm p \times \Delta$ interval given that $\Delta = \max X_i - \min X_i$. On the other hand, if X_i is a categorical attribute, X_j is then set to the same value of X_i with $(1-p)$ probability, or otherwise set to one of the other possible values. Either way, this strategy leads to a linear correlation where the Pearson coefficient decays exponentially with the growth of p.

Radial Basis Function. The Radial Basis Function (RBF) strategy generates values by projecting an attribute into a Gaussian distribution. Working under the assumption that X_i's distribution is uniform (as it holds for all generators earlier described), its expectation is given by $E(X_i) = \frac{\max X_i - \min X_i}{2}$. Given the expectation, the value of X_i can then be projected using Equation 4.6. One of the drawbacks of this strategy is that it can only be applied to numeric data.

$$r_{\text{RBF}}(\vec{x}, X_i) = (\vec{x_i}[X_i] - E(X_i))^2 \tag{4.6}$$

Cosine. This strategy is similar to the latter one with regard to value projection.

Table 4.1: Synthetic data stream experiments.

Experiment Identifier	# of Features	# of Relevants	# of Irrelevants	# of Linear Redundants	# of RBF Redundants	# of Cosine Redundants	Drift Window Size (W_{drift})
AGR	20	3 ± 1	6 ± 1	11	–	–	1,000
AN	20	3.5 ± 1	1.5 ± 1	15	–	–	1,000
BG1	50	4 ± 1	31 ± 1	15	–	–	1,000
BG2	50	3	32	15	–	–	1,000
BG3	50	3	32	15	–	–	1,000
RTG	50	5	30	5	5	5	1,000
SEA	50	2	33	5	5	5	1,000

The rationale here is to use trigonometry cosine to project the value of an attribute X_i into a new range of values. Since cosine expects a degree as input, the value of X_i must be first converted in a degree as depicted in Equation 4.7. As with the RBF strategy, this scheme can only be applied to numeric attributes.

$$r_{\text{cosine}}(\vec{x}, X_i) = \cos\left(360 \frac{\vec{x}[X_i]}{\max X_i - \min X_i}\right) \tag{4.7}$$

4.2.8 Summary of Synthetic Experiments

Given the latter generators, several synthetic experiments were created encompassing different dimensions and both abrupt and gradual drifts. Table 4.1 presents an overview of these experiments, which will be repeatedly used in the upcoming chapters to evaluate the proposed methods. All experiments are binary problems, i.e., $|Y| = c = 2$, contain 100,000 instances, possess 9 drifts, each at every 10,000 instances (t_{drift}); and a noise rate of 10%. It is important to emphasize that even though certain experiments possess a static amount of relevant features, they do change over the stream as drifts occur. Furthermore, it is important to highlight that only a small fraction of the attributes in each experiment are deemed as relevant (from 4% to 10%). And finally, all experiments using synthetic data streams have been performed 30 times, each time with a different random seed for each of the inning concepts and the respective average results will be reported in the following chapters.

4.3 Real-world Data

In contrast to synthetic data, real-world problems present differentiated behavior: the distribution between classes is often unbalanced and there might exist temporal dependencies between instances, i.e., between two instances $(\vec{x}^t, y^t)$ and $(\vec{x}^{t+1}, y^{t+1})$, there is a high probability that $y^t = y^{t+1}$ (ŽLIOBAITĖ et al., 2014). On the other hand, it is nearly impossible to affirm if and where drifts occur, making evaluation harder.

This section presents real-world datasets used in evaluation and their goals. It is important to emphasize that the datasets used were chosen due to their appearance in other data stream mining works and reasonable number of features. For instance, other traditional datasets, e.g., Airlines (IKONOMOVSKA et al., 2011), Electricity (RODRIGUES; GAMA; PEDROSO, 2008), and Pokerhand (CATTRAL; OPPACHER; DEUGO, 2002), were not used since they possess a number of features that is already too small and feature selection would not be beneficial.

4.3.1 Internet Advertisements (IADS)

The Internet Advertisements (IADS) (KUSHMERICK, 1999) dataset targets the classification task of images on websites, either by labelling them as advertisements or not. A variety of features is available, including the geometry of the image (if available) as well as phrases occurring in the URL, the image's URL and alt text, the anchor text, and words occurring near the anchor text.

4.3.2 NOMAO

The NOMAO Challenge dataset (NOMAO) (CANDILLIER; LEMAIRE, 2012) was introduced during the ECMLPKDD'12 challenge was part of a deduplication task. This dataset targets the identification of whether two spots, represented by an instance, should have their data merged or not.

4.3.3 Spam Corpus

The Spam Corpus database was developed in (KATAKIS; TSOUMAKAS; VLAHAVAS, 2006) as a result of a text mining process on an online news dissemination system. Part of this work intended on creating an incremental filtering of emails classifying them as spam or not, and based on this classification, deciding whether this email was relevant for dissemination among users. This dataset has 9,324 instances and 39,917 boolean attributes, such that each attribute represents the presence of a single word (the attribute label) in the instance (e-mail). As discussed in Section 3.7, the Spam Corpus dataset possesses a drift around the instance 1,500 which impacts the learning accuracy of classification algorithms (KATAKIS; TSOUMAKAS; VLAHAVAS, 2006).

Table 4.2: Summary of real-world data used during experiments.

Dataset Identifier	Dataset	# of features	# of instances	Class distribution (%)
IADS	Internet Ads	1,559	3278	14–86
NOMAO	Nomao Challenge	118	34,465	29–71
SPAM	Spam Corpus	39,917	9,324	25–75

4.3.4 Summary of Real-world Experiments

Even though synthetic data stream generators allow higher flexibility, real-world data give researchers the possibility of evaluating their proposals in different scenarios with no *a priori* knowledge. It is also relevant to highlight the lack of real-world data streams that are made publicly available (GAMA et al., 2014), thus rendering the evaluation in such scenarios brief. Table 4.2 summarizes the main characteristics of evaluated datasets. All the presented datasets are binary classification problems and are sorted as a time series, thus, giving rise to one of the main traits of streaming scenarios.

4.4 Experimental Protocol

Evaluating learning algorithms is a challenge per se. Additionally, the evaluation of feature selection proposals is unclear and the procedures to be followed are not common sense in the machine learning community (GALELLI et al., 2014; KUNCHEVA, 2007). This section presents evaluation metrics that broadens accuracy, processing time, memory usage, selection accuracy and feature selection stability. Furthermore, it states the statistical testing procedure adopted for testing all the results obtained.

4.4.1 Accuracy Evaluation

Holdout error estimation is one of the most straight-forward evaluation methods for machine learning algorithms. The rationale behind holdout evaluation is to periodically sacrifice n instances to estimate classification accuracy.

Definition 14. Let $\mathcal{S}'$ be the set of n sacrificed instances and let $L(\cdot,\cdot)$ be the loss function (usually a 0-1 function). The holdout error H_e of a classifier in a chunk $\mathcal{S}'$ of instances is given by Equation 4.8, where y_i is the ground-truth class and $\hat{y}_i$ the the predicted class.

$$H_e = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) \tag{4.8}$$

Sacrificing chunks of data for evaluation may not faithfully represent a classifier's accuracy over a data stream. Adopting a test-then-train scheme, authors in (GAMA; RODRIGUES, 2009) proposed the Prequential procedure to monitor the evolution of a classifier's performance over time. Although the Prequential evaluation is known for being pessimistic, it converges to a periodic holdout estimative when estimated over a sliding window.

Definition 15. Given a sliding window of size w', the Prequential error P_e of a classifier at a timestamp t_i is given by Equation 4.9, where y_i is the ground-truth class and $\hat{y}_i$ the the predicted class.

$$P_e(w', t_i) = \frac{1}{w'} \sum_{k=i-w'+1}^{i} L(y_k, \hat{y}_k) \tag{4.9}$$

As stated in the original paper, the choice of the window size is critical and the outcome depends on the scale of change of the stream. Even though strategies to avoid this rather empirical definition of an evaluation window size exist (BIFET et al., 2015), they do not allow a precise comparison when evaluating different classifiers. Therefore, the original Prequential evaluation process is maintained where the window size is set to 10% of the stream since several pairwise evaluations will be performed throughout this work.

4.4.2 Processing Time and Memory Usage

Besides measuring accuracy, it is important that data stream mining algorithms perform both rapidly and within memory boundaries. In the following experiments, processing time is reported as the time that the algorithms spend in the processor (in seconds) namely CPU Time, while memory usage is presented in RAM-Hours, where 1 RAM-Hours equals 1 GB of RAM used per hour. All experiments' results reported in the following chapters were obtained on an Intel Xeon CPU E5649 @ 2.53GHz ×8 based computer running CentOS with 16GB of memory.

4.4.3 Selection Accuracy

The selection accuracy (SA) score expresses the degree to which a selected subset of features matches the true input subset of relevant features (GALELLI et al., 2014). SA is based on a simple similarity score that makes no distinction between irrelevant and redundant features and simply treats both as unnecessary and extraneous.

Definition 16. Assuming a feature set $\mathcal{X}$, its relevant subset of features $\mathcal{X}^*$ and the selected subset of features $\mathcal{X}'$, the selection accuracy SA is given by Equation 4.10, where

γ is a weighting factor that influences the penalty applied to the selection of extraneous features in relation to the gain obtained from each correctly selected feature.

$$SA(\mathcal{X},\mathcal{X}^*,\mathcal{X}') = \gamma \overbrace{\left(\frac{|\mathcal{X}^* \cap \mathcal{X}'|}{|\mathcal{X}^*|}\right)}^{\text{RSR}} + (1-\gamma) \underbrace{\left(1 - \frac{|(\mathcal{X} \setminus \mathcal{X}^*) \cap \mathcal{X}'|}{|\mathcal{X}| - |\mathcal{X}^*|}\right)}_{\text{CUCP}} \tag{4.10}$$

The SA score is bounded in the $[0;1]$ interval, where 1 corresponds to a perfectly specified feature selection, while 0 represents a selection with no relevant features and all extraneous ones selected. SA has as an advantage the fact that the information about the degree to which a model has been correctly or incorrectly specified is combined into a single value, which makes the comparison between several feature selection proposals clear.

One of the drawbacks of the SA score is that it requires the choice of an appropriate value of γ. This choice is subjective and depends on how much one wants to favor accuracy over parsimony or vice versa. Hereafter, the two components that compose the SA formula are referred as Relevant Selection Recall (RSR) and Complement of Unnecessary Complexity Penalty (CUCP). A suitable value for γ should reflect the fact that choosing an extraneous feature is usually better than missing a relevant one, something that can be achieved by selecting γ, such that $\frac{\gamma}{|\mathcal{X}^*|} > \frac{1-\gamma}{|\mathcal{X}|-|\mathcal{X}^*|}$ (MOLINA; BELANCHE; NEBOT, 2002). On the other hand, γ should not be large to the point that there is no substantial penalty to unnecessary model complexity. Authors in (GALELLI et al., 2014) provided an empirical evaluation of different values of γ and claimed that 0.7 is an interesting value since it satisfies the above condition while being sufficiently less than 1 to appropriately penalize unnecessary complexity.

The advantage of computing Selection Accuracy scores is that they express the degree to which a selection model over- or under-specifies. For instance, if a method is consistently over-specified over a range of datasets, this may indicate that the stopping criterion is inappropriately penalizing model complexity or that the significance level used to compute input relevance is inadequate. Yet, it also is worth to highlight that the SA score is impacted by the total number of features we have in the data stream being analyzed since the denominator in CUCP depends on the cardinality of the original input feature set $\mathcal{X}$.

4.4.4 Stability

Another important trait of feature selectors that deserves attention is stability. Stability measures the sensitivity of the feature selection solution given perturbations in input data. The goal is to provide evidence that the selected features are consistent across different input data samples. Therefore, stable feature selection algorithms are preferable when compared to those with highly volatile outputs. It is important to highlight that stability, however, does not relate to the performance of the selected features as it indicates how *unstable* a feature selection algorithm is w.r.t. perturbations in input data, and not on how *accurate* the selection is.

In batch learning, stability is often measured by repeatedly performing feature selection over n different bootstraps of disjoint folds of a static dataset, leading to a set of feature selection results. Let $\mathcal{X}'_i$ be the subset of features selected over the i^{th} samples of instances extracted from a static dataset. The stability of a feature selection algorithm can be computed by averaging the similarity coefficient ϕ for each of the k pairs of $(\mathcal{X}'_i, \mathcal{X}'_j)$ of selected features, as stated in Equation 4.11.

$$S = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \phi(\mathcal{X}'_i, \mathcal{X}'_j) \tag{4.11}$$

Although several similarity metrics (ϕ) for stability do exist, until recently, there has not been an agreement on which one to use (KUNCHEVA, 2007). Recently, the work of (NOGUEIRA; BROWN, 2016) has provided insights on the main properties a stability measure should possess. First, it should be **fully defined**, as a stability measure should be defined regardless of the selected feature sets and respective lengths. Also, it must have pre-defined **upper and lower bounds** to facilitate the comparison between selectors. The third trait is the relationship called **Deterministic Selection $\Leftrightarrow$ Maximum Stability**: if a selector always selects the same k features, then it should present maximum stability. The converse should also hold, i.e., the stability is maximum only if the selection is deterministic. Finally, it should have **chance correction**, so if the selector is random, its stability should be 0. Even though these traits are rather simple, the analysis conducted in (NOGUEIRA; BROWN, 2016) shows that most approaches do not fulfil these criteria. More importantly, in the same work, authors show that the Pearson coefficient overcomes this problem. This coefficient is given by Equation 4.12, where $d = |\mathcal{X}|$, $r_{i,j} = |\mathcal{X}'_i \cap \mathcal{X}'_j|$, and $v_i = \sqrt{\frac{k_i}{d}\left(1 - \frac{k_i}{d}\right)}$ with $k_i = |\mathcal{X}'_i|$.

$$\phi_{\text{Pearson}}(\mathcal{X}'_i, \mathcal{X}'_j) = \frac{r_{i,j} - \frac{k_i k_j}{d}}{d\, v_i v_j} \tag{4.12}$$

Finally, the last challenge to be tackled here regards how stability scores can be calculated in streaming scenarios. A naive proposition to select samples of a stream would be to adopt a landmark windowing scheme, where every m instances would be grouped and inputted to a feature selection algorithm. After performing feature selection over n batches, the stability could then be computed. The major drawbacks of such proposal are that it assumes that (i) the feature selection algorithm is not dynamic and that (ii) the underlying data distribution is static since the selected subset of features for each batch is expected to be the same. As discussed in the previous sections of this paper, none of the latter assumptions hold or are preferable, thus, evaluating with landmarks is not reasonable.

To overcome such limitations, the proposal is to adapt the Prequential Cross-Validation (Preq-CV) scheme presented in (BIFET et al., 2015) for stability computation. Following the original Preq-CV, three different k-fold approaches can be used to evaluate the stability of a feature selector: **cross-validation**, **split-validation** and **bootstrapping**. The first strategy updates $(k-1)$ folds, while the second updates only one of the k folds. Finally, the bootstrapping approach updates each of the k folds using a weight obtained with a Poisson distribution with a parameter $\lambda = 1$. In this scheme, x is the number of times in which an instance will be associated with a fold drawn from a Poisson distribution with λ mean. Therefore, the probability of an instance being used in each fold is approximately two thirds, as $P[x > 0] = 1 - P[x = 0] = 1 - \frac{e^{-1}}{x!} = 1 - \frac{e^{-1}}{0!} \approx 63\%$ and the same value depicts the intersection of instances used in each pair of folds.

Similarly to Selection Accuracy, calculating a Stability score is computationally intensive as it requires $\frac{k(k-1)}{2}$ pairwise similarity computations, and thus, these are only calculated according to an user-given evaluation window size. Also, even though this score is calculated only every n instances, it is important to notice that the actual feature selection process occurs incrementally, which causes the process to be different from performing batch feature selection and conventional stability computation.

4.4.5 Statistical Testing Procedure

The evaluation procedure resides in assessing algorithms' efficiency in accuracy, processing time, memory usage and feature selection accuracy. In order to provide statistical confidence to presented claims, Wilcoxon's test or a combination of Friedman's and Nemenyi's non-parametric hypothesis tests were adopted, depending on the number of evaluated hypotheses.

Wilcoxon's signed-rank test (WILCOXON, 1945) is used when there is the need to compare two paired samples $\vec{a}_1$ and $\vec{a}_2$ to assess whether their mean ranks differ significantly. Wilcoxon's test computes the absolute difference $|\vec{a}_2[j]-\vec{a}_1[j]|$ for each datum pair and ignores cases where differences are equal to zero, thus resulting in a new sample with size N_r. Later, it ranks (R_i) the absolute differences, from smallest to largest, employing tied ranks when needed. Signs $(sgn(\cdot) \in \{+1,-1\})$ are associated with ranks according to their original differences. Positive signs $(+1)$ occur if $\vec{a}_2[j] - \vec{a}_1[j] > 0$ and negative signs (-1) otherwise.

Under the null hypothesis, i.e., the samples possess the same population, it occurs that the $\mathcal{W}$ statistic (Equation 4.13) is below $\mathcal{W}_{\alpha,N_r}$, a value found in statistical tables.

$$\mathcal{W} = \sum_{j=1}^{N_r} sgn(\vec{a}_2[j] - \vec{a}_1[j]) \times R_i \tag{4.13}$$

Finally, if there is the need to compare more than two hypotheses at the same time, one should proceed with the Friedman test (FRIEDMAN, 1937). It ranks algorithms for each dataset separately, where the best performing algorithm receives the rank number 1, the second best rank number 2 and so forth. Let r_j^k be the rank of the k^{th} of m algorithms on the j^{th} of N datasets. Friedman's test compares the average ranks of algorithms $R_j = \frac{1}{N}\sum_j r_j^k$. Under the null hypothesis, all algorithms are equivalent if their ranks are equal according to the Friedman statistic (presented in Equation 4.14), which is distributed according to χ_F^2 with $(m-1)$ degrees of freedom (DEMSAR, 2006).

$$\chi_F^2 = \frac{12N}{m(m+1)}\left[\sum_j R_k^2 - \frac{m(m+1)^2}{4}\right] \tag{4.14}$$

If the null hypothesis is rejected, one can proceed with Nemenyi's posthoc test (NEMENYI, 1963), which compares all algorithms with each other. The performance of two algorithms are significantly different if the corresponding average ranks differ by at least one critical difference (CD) given by Equation 4.15, where q_α is a range statistic that relies on a required significance level α.

$$CD = q_\alpha\sqrt{\frac{m(m+1)}{6N}} \tag{4.15}$$

Finally, statistical differences are reported according to critical differences charts, where hypotheses connected by line segments do not present significant statistical differences.

4.5 Concluding Remarks

Evaluating data stream learning algorithms embeds the measuring and comparison of accuracy, processing time and memory usage metrics. Furthermore, a variety of feature drifting data generators were introduced, thus enabling a proper evaluation of algorithms in these scenarios. Also, one should always bear in mind that both synthetic and real-world data should be used, each one to provide insights into different aspects of the evaluation. Throughout this chapter, two specific metrics for the feature selection task, i.e., Selection Accuracy and Stability, have been introduced and considerations about their computations in streaming environments were discussed. Together, this set of metrics, data generators, datasets and statistical evaluation procedures, constitute a solid evaluation environment. This environment will be repeatedly used during the following chapters in order to evaluate the proposed algorithms.

The following chapter surveys related works on feature drift adaptation. Additionally, several of the algorithms presented are evaluated alongside drift detectors.

Chapter 5

Related Work

There are few works in the literature that perform feature selection or weighting during stream learning. There are even fewer that aim at explicitly detecting and adapting to feature drifts. This chapter summarizes existing algorithms that perform feature selection or weighting as the stream progresses, either assuming the existence or not of feature drifts. This chapter is an abbreviated version from the surveys presented in (BARDDAL; GOMES; ENEMBRECK, 2015c) and (BARDDAL et al., 2017). It is important to highlight that this chapter does not include a thorough study over ensembles that explore different subspaces in parallel. This is due to the fact that ensembles are computationally expensive compared to single classifier methods, and have an aggregate behavior that is hard to explain. For instance, it is hard to justify if an ensemble presents certain accuracy rates due to its resampling technique, drift detection scheme, or vote combination strategy, or if it is the mix of all these that yields such results. In practice, evaluating ensembles is puzzling since one must justify if an ensemble performs well because of the combination of all its internal procedures, or simply because some of these are very effective while others are useless or detrimental.

Furthermore, it must also be emphasized that this survey does not include feature selection proposals that assume that the stream's data generation is stationary. For instance, the work of (FONG; WONG; VASILAKOS, 2016) is not included since (i) streams are summarized into large data batches and a conventional batch Particle Swarm Optimization (PSO) feature selection algorithm is used to guide the classifier learning, and (ii) no drifting experiments were evaluated. The work of (YAN et al., 2006) is also omitted since the proposed dimensionality reduction techniques are incremental and assume that no drifts occur during the stream. More recently, feature selection from streams have received some attention, where the work of (TURKOV et al., 2016) is accentuated. Nevertheless, in this work no datasets with irrelevant features are tested,

Table 5.1: Summary of existing algorithms that perform feature selection during stream learning. Adapted from (BARDDAL et al., 2017).

Algorithm	Learning Approach	Feature Selection Algorithm	Feature Drift Adaptation Method	Explicit Dynamic Feature Selection	Reference
VFDT	Tree	Entropy Information Gain Gini Coefficient	–		(DOMINGOS; HULTEN, 2000)
Facil	Rules	Purity	–		(FERRER-TROYANO; AGUILAR-RUIZ; SANTOS, 2005)
VFDR	Rules	Entropy	–		(GAMA; KOSINA, 2011)
Streaming Random Forest	Ensemble (Trees)	–	Randomness/Combinatorics		(ABDULSALAM; SKILLICORN; MARTIN, 2007) (ABDULSALAM; SKILLICORN; MARTIN, 2011)
Random Rules	Ensemble (Rules)	–	Randomness/Combinatorics		(ALMEIDA; KOSINA; GAMA, 2013)
Streaming Stacking	–	Ensemble	Combinatorics		(BIFET EIBE FRANK; PFAHRINGER, 2010)
CVFDT	Tree	Entropy Information Gain Gini Coefficient	Windowing	✓	(HULTEN; SPENCER; DOMINGOS, 2001)
HEFT-Stream	Ensemble	FCBF	Windowing	✓	(NGUYEN et al., 2012)
HAT	Tree	Entropy Information Gain Gini Coefficient	Windowing	✓	(BIFET; GAVALDÀ, 2009)
HUWRS	Ensemble	–	Windowing		(HOENS; CHAWLA; POLIKAR, 2011)
ARF	Ensemble (Trees)	–	Randomness & Windowing		(GOMES et al., 2017)

and there are no experiments with feature drifts.

Table 5.1 categorizes the surveyed algorithms, which are layered given four characteristics: their learning approach, the feature selection algorithm used, feature drift adaptation method adopted; and whether explicit dynamic feature selection occurs or not. Sections 5.1 and 5.2 start this chapter by discussing two important and widely used approaches: decision trees and decision rules, respectively. Although most of the summarized algorithms presented in this section were not developed aiming at performing feature drift detection and adaptation, these are discussed and their capabilities to attack this problem are highlighted, either through randomness (Section 5.3), combinatorics (Section 5.4) or windowing (Section 5.5).

5.1 Decision Tree Learning

Learning with decision trees is a predictive approach used in statistics, data mining and machine learning. In its simplest implementations, each internal node contains a test on a feature $X_i \in \mathcal{X}$, each branch from a node corresponds to an outcome of the test and each leaf contains a possible prediction (class value from Y) (BIFET, 2010).

Predictions for instances $\vec{x}$ are obtained by traversing the tree with features' values, determining which branch should be followed until a leave is reached.

Decision trees are learned by recursion, replacing leaves by test nodes, starting at the root. The feature of each test node is chosen by comparing all the available attributes $X_i \in \mathcal{X}$ according to some heuristic measure.

By far, the most used decision tree for data streams is the Very Fast Decision Tree (VFDT) (DOMINGOS; HULTEN, 2000). The VFDT algorithm constructs decision trees by using constant time per instance. Trees are built by recursively replacing leaves with decision nodes, as data arrives. Different heuristic evaluation functions are used to determine whether a split should be performed or not, such as Entropy, Linear correlation, Information Gain and Gini Impurity (HAN; KAMBER; PEI, 2011).

Given the values for the chosen metric, to determine whether an split should be performed, VFDT assumes that the input data meets the Hoeffding bound (HOEFFDING, 1963).

Definition 17. The Hoeffding Inequality states that with probability $(1-\delta)$ the true mean of a variable is at least $\bar{r}-\epsilon$, where ϵ is given by Equation 5.1, δ is a user-given confidence bound, $r \in \mathbb{R}^+$ is a random variable with range R, n is the number of independent observations and $\bar{n}$ is the mean computed by the latter observations.

$$\epsilon = \sqrt{\frac{R^2 \ln \left(^1\!/_\delta\right)}{2n}} \tag{5.1}$$

The Hoeffding bound is able to give results regardless the probability distribution that generates data. However, the number of observations needed to reach certain values of δ and ϵ are different across different probability distributions (BIFET; GAVALDÀ, 2007), therefore, it must be seen as a pessimistic bound.

Generally, with probability $(1-\delta)$, one can say that one attribute is superior when compared to others when the observed difference of information gain (or any other metric that computes the importance of an attribute) is greater than the Hoeffding Bound ϵ (given by Equation 8.2). Although VFDT performs embedded feature selection in data streams, it assumes that the distribution generating data does not change over time, therefore, it does not detect nor adapt to a possible concept or feature drift. With this goal, adaptive versions such as Concept-adapting Very Fast Decision Tree and Hoeffding Adaptive Trees have been proposed and will be later discussed in Sections 5.5.1 and 5.5.3. It is also important to highlight that many works have been developed by extending Hoeffding Trees with different purposes, e.g., memory consumption (MANAPRAGADA; WEBB; SALEHI, 2018) and learning from imbalanced scenarios (LYON et al., 2014), but an exhaustive analysis of such proposals is not part of the current work.

5.2 Decision Rule Learning

Although decision trees account for readability, in some specific scenarios, where trees tend to grow largely, they become hard to understand since nodes appear in a specific context defined by tests at antecedent nodes (ALMEIDA; FERREIRA; GAMA, 2013). In contrast, classifiers based on rules have the advantage of both modularity and interpretability (RIVEST, 1987), where each rule is independent of the others and can also be interpreted isolated from others.

A decision rule is a logic predicate in the ***IF*** **antecedent** ***THEN*** **label** form, where the antecedent is a conjunction of conditions over features $X_i \in \mathcal{X}$ and the label is a possible class value that belongs to Y.

5.2.1 Facil

The first rule learner for data streams published was Facil (FERRER-TROYANO; AGUILAR-RUIZ; SANTOS, 2005). Facil creates rules according to the arrival of instances in an incremental fashion. In order to cope with concept drifts, Facil encompasses both explicit and implicit forgetting mechanisms. The explicit approach occurs when the examples are older than a user-given threshold W, adopting a sliding window approach to eliminate old rules. Conversely, implicit forgetting occurs when removing rules that are not relevant as they do not enforce any concept description boundary. This approach's rationale is that rules are inconsistent if they store both positive and negative instances that are near to one another at the decision boundary. Therefore, rules are removed if the impurity (ratio between positive instances it covers and its total number of cover examples) of a rule reaches a lower bound user-given threshold. Whenever the removal of a rule occurs, the subset originally covered by these rules are used to form two new rules that achieve satisfiable purity. One of the major restrictions of Facil is that input numeric data must be normalized in the $[0; 1]$ interval.

5.2.2 Very Fast Decision Rules

A more robust approach for learning rules from data streams, namely Very Fast Decision Rules (VFDR) is proposed by authors in (GAMA; KOSINA, 2011). The algorithm starts with an empty rule set and rules are grown and expanded as new instances become available according to the minimization of the entropy of class labels of instances covered by each rule and if the entropy values meet the Hoeffding bound (Equation 5.1).

VFDR considers two cases of rule learning: ordered and unordered sets of rules. In the former, all labeled instances update statistics of the first rule triggered by it, while in the latter labeled instances update statistics of all the rules that cover it. In both cases, if no rules cover an instance, the default rule is updated.

Finally, VFDR encompasses two classification strategies. The first uses only the information about class distribution and does not account for attribute's values. Since it uses a small part of the available information, it is a crude approximation of the instances. Conversely, in an informed strategy, instances are classified with the class that maximizes the posteriori probability assuming the independence of attributes given the class ($P[y_i|\vec{x}] \propto P[y_i] \prod P[\vec{x}_j|y_i]$).

5.3 Randomness

Diversity is a trait of a variety of recently proposed algorithms for learning from data streams (BIFET et al., 2009; BIFET, 2010; OZA, 2005). In these approaches, ensembles of experts are trained in parallel or cascade, and often each one receives different inputs for training (BREIMAN, 1996). The most well-known approach for inducing diversity in ensembles is Bagging (BREIMAN, 1996). Originally, a bagging ensemble is composed of m classifiers, which are trained with bootstraps (sampling with replacement) of the whole training set. However, sampling usually is not feasible in a data stream configuration, since that would require storing all instances before creating subsets. Therefore, authors in (OZA, 2005) observed that the probability of an instance $\vec{x}_i$ to be selected for a subset can be approximated by a Poisson distribution with $\lambda = 1$.

Although promoting diversity through instances is an interesting approach to boost the accuracy of learners, more recent approaches aims at promoting diversity through different feature subsets (ABDULSALAM; SKILLICORN; MARTIN, 2007; ABDULSALAM; SKILLICORN; MARTIN, 2011). By learning through ensembles with different subsets of features, experts learn partially (or completely) disjoint areas of the feature space, implying in a highly diverse ensemble. Although these algorithms do not focus explicitly on adapting to feature drifts, they possess implicit adaptation to this characteristic of data streams.

5.3.1 Streaming Random Forest (SRF)

The Streaming Random Forest (SRF) classifier is an adaptation of the ensemble-based Random Forest classifier (BREIMAN, 2001). Random forests are ensembles of decision trees. Assuming a dataset with n instances, each belonging to a feature set $\mathcal{X}$, random forests grow a set of trees, each from a bootstrap from the whole training set. Bootstrapping guarantees that about $^{n}/_{3}$ of the records are not included in the training set and are available for evaluation of each tree (ABDULSALAM; SKILLICORN; MARTIN, 2011; DENIL; MATHESON; FREITAS, 2013).

The construction of each tree follows a variant of typical decision tree building algorithm. In standard decision tree algorithms, the set of attributes considered at a node is the entire set $\mathcal{X}$. Conversely, in the random forest algorithm, the set of attributes considered at each test node of a tree in the ensemble is a randomly chosen subset $\mathcal{X}' \subset \mathcal{X}$, where $|\mathcal{X}'| \leq M$.

Since a random forest is an ensemble classifier, the classification of each new instance is the fusion of the votes of the containing trees. The random forest classification error depends on (i) the correlation among its component trees, since smaller correlations imply in higher variance canceling in voting and (ii) the strength of each individual tree, since the more accurate each subtree is, the better its individual vote and smaller is the error rate (ABDULSALAM; SKILLICORN; MARTIN, 2007).

Therefore, the value of M is a sensitive parameter of random forests and must be chosen carefully. Smaller values of M tend to increase the strength of each individual tree, while decreasing the correlation between them (ABDULSALAM; SKILLICORN; MARTIN, 2007).

More recently, another variant of Random Forests has been proposed in (GOMES et al., 2017), and it will be later discussed in Section 5.5.5 since it combines randomness and windowing, i.e., drift detectors, to overcome feature drifts.

5.3.2 Random Rules

In (ALMEIDA; KOSINA; GAMA, 2013), authors extend the VFDR algorithm by promoting randomness. This algorithm, namely Random Rules for Data Streams (RR), encompasses the following parameters: a number of rule sets (N_s) and the number of attributes M that must respect the $M < |\mathcal{X}|$ restriction.

Initially, each of the composing rule sets is empty and each of these is associated with a random subset $\mathcal{X}' \subset \mathcal{X}$ of size M. For each instance $(\vec{x}^t, y^t)$ retrieved from $\mathcal{S}$, RR generates a random number p between 0 and 1 for each rule set. If $p \geq T_{rnd}$, a user-given threshold, RR verifies whether each rule set contains a rule that covers $(\vec{x}^t, y^t)$, i.e., if all the literals of the rule are true for the given instance. If so, all covering rules are expanded using only the features adopted by the rule set. Otherwise, i.e., if no rules cover $(\vec{x}^t, y^t)$, the default rule is updated to cover it, again, respecting the features in $\mathcal{X}'$. If the sufficient statistics of the default rule reach certain thresholds discussed in (GAMA; KOSINA, 2011), this rule is added to the rule set and the default rule is reset.

Finally, authors presented two voting schemes. The first classifies $\vec{x}^t$ with the class y_i that maximizes $P[y_i]$, while the second assumes the class that maximizes the posteriori probability, i.e., $\mathrm{argmax}_{y_i \in Y} P[y_i|\vec{x}^t]$.

5.4 Combinatorics

By exploring combinatorics, both random forest, and random rules algorithms can be extended and posed as dynamic wrappers for dynamic feature selection for data streams. If one assumes a random forest or a random rule algorithm, where each of its containing experts is trained with a different subset of the entire feature set $\mathcal{X}$, and that the cardinality of each subset is at maximum M, the ensemble would contain $\sum_{i=1}^{M} \binom{M}{i}$ experts. Although training this high amount of experts is computationally expensive in terms of both processing time and memory space, it guarantees that a near optimal (or optimal, if $M \geq |\mathcal{X}^*|$) subset $\mathcal{X}'$ allocated to one of the experts will maximize its acuity metric (ABDULSALAM; SKILLICORN; MARTIN, 2011). Therefore, by applying dynamic weighted majority voting (KOLTER; MALOOF, 2007), feature drifts can be detected according to the increase of the weights of experts with the current most discriminative subsets of features, while those with subsets of irrelevant features will possess alleviated weights due to lower accuracy performance.

5.4.1 Streaming Stacking (SS)

In (BIFET EIBE FRANK; PFAHRINGER, 2010), authors produce a classification model based on an ensemble of decision trees, each of which is built from a random and distinct subset of $\mathcal{X}' \subset \mathcal{X}$. The overall model, namely Streaming Stacking (SS), is formed by combining the log-odds of the class probabilities of its containing trees using sigmoid perceptrons, with one perceptron per class. Contrarily to the conventional boosting approach, which forms an ensemble in a greedy fashion, each tree is built in sequence by assigning weights as a by-product. Trees are generated in parallel and their votes (individual predictions) are combined using stacking (WOLPERT, 1992) with perceptrons (FREUND; SCHAPIRE, 1999). Due to the streaming scenario, VFDTs are used as ensemble members since they are able to be trained incrementally. Additionally, the ensemble adopts the ADWIN change detector (see Section A.2) in order to detect and adapt to possible concept drifts. This approach is based on generating trees for all possible feature subsets of a given size M. Assuming a feature set $\mathcal{X}$ of size d, there are $\binom{d}{M}$ possible subsets. Clearly, only moderate values of M or values close to d are practical, since $\binom{d}{M} = \binom{d}{d-M}$. Authors claim that $M = 2$ is very practical for datasets with a moderate number of features, although certainly not feasible for high-dimensional data (e.g., Spam Corpus, discussed in Section 3.7).

5.5 Windowing

A common approach for both data management and dealing with drifting data is to maintain a predictive model consistent with a set of recent examples (SILVA et al., 2013). There are three major windowing techniques in the literature: sliding, damped and landmark; and in all cases, the difficulty is to select their appropriate size due to the plasticity-stability dilemma. While short windows reflect the current data distribution and ensure fast adaptation to drifts (plasticity), they usually worsen the performance of the system in stable areas. Conversely, larger windows give better performance in stable periods (stability), however, these imply in slower reaction to drifts (BARBARÁ, 2002; GAMA, 2010; GAMA et al., 2004; SILVA et al., 2013).

Sliding Window. Sliding windows store in memory a fixed or variable amount of recent examples. In the fixed approach, whenever a new instance arrives, it is enqueued in a FIFO (first in, first out) policy data structure, where the oldest one is discarded. In variable-sized windows, the number of instances in this data structure may change over time, usually according to the outputs of a change detector. A straightforward idea is to shrink the window when changes in data are detected so that the data stored in memory reflects the posterior concept, and maintain larger windows during stable areas of the stream.

Damping Window. In opposition to sliding windows, damping windows associate a weight to each datum, which decays with time (JIANG; GRUENWALD, 2006). Therefore, more recent instances receive a higher weight than older ones, and these weights decay with time according to a decaying function. This windowing technique is interesting because weights can be seen as indications of how important an instance is to the current concept, thus, may be accounted for during prediction.

Landmark Window. Finally, landmark windows require processing a stream by handling disjoint chunks of data separately by instances called “landmarks”. Landmarks can be defined in terms of time, in terms of the number of instances seen since the previous landmark or according to memory constraints (METWALLY; AGRAWAL; ABBADI, 2005). All instances belonging to the same landmark are stored or summarized into a common data structure, which is used for training. When a new landmark is reached, all data in the current window is discarded and further instances retrieved from the stream are kept until a new landmark is reached. Again, the issue in defining the gap between landmarks takes back to the plasticity-stability dilemma.

This section presents existing works that rely on windowing approaches to explicitly adapt to feature drifts.

5.5.1 Concept-adapting Very Fast Decision Tree (CVFDT)

The Concept-adapting Very Fast Decision Tree (CVFDT) algorithm is an extension to the VFDT to deal with concept drifts (HULTEN; SPENCER; DOMINGOS, 2001). It keeps a model consistent with respect to the current state of a sliding window from the data stream, thus creating and replacing alternate decision subtrees when it detects that the distribution of data is changing at a node. Whenever a new instance i^t arrives, CVFDT updates the statistics at its nodes by decrementing counters according to the oldest element in the window, which is about to be dequeued and "forgotten".

Therefore, CVFDT is a Hoeffding Tree which periodically verifies the statistics of nodes to determine if the Hoeffding criterion is still met. Given three user-given window sizes T_0, T_1 and T_2, CVFDT traverses the entire decision tree and checks at each node if the splitting feature is still the best when compared to others every T_0 instances. If there is an alternate better splitting attribute, the whole subtree is replaced by a new split node with this attribute. Later, during the next T_1 instances, all retrieved instances from $\mathcal{S}$ are used to build the new subtree, which is then tested with the following T_2 instances.

5.5.2 Heterogeneous Ensemble for Data Stream (HEFT-Stream)

The Heterogeneous Ensemble with Feature Drift for Data Streams (HEFT-Stream) is an algorithm that incorporates feature selection into a heterogeneous ensemble to adapt to different types of concept and feature drifts (NGUYEN et al., 2012). HEFT-Stream adopts a modification of the Fast Correlation-Based Filter (FCBF) algorithm so it dynamically updates the selected relevant feature subset of a data stream.

FCBF is a feature selection algorithm based on Information Theory where the class relevance and the pairwise dependency between features are accounted for. This metric will be further detailed and discussed in Chapter 6 since it is a part of the proposed methods.

HEFT-Stream adopts a landmark windowing approach. After the arrival of each data batch, a new FCBF feature selection is run, and if the resulting feature subset differs from the other computed in the previous batch, HEFT-Stream postulates that a feature drift has occurred and the worst classifier in the ensemble is substituted by a new classifier with the same learning model type, e.g., Naive Bayes, VFDT, as the best performing one in the ensemble. Additionally, in order to boost the ensemble overall accuracy, HEFT-Stream promotes diversity among member classifiers by encompassing a sampling technique inspired on Online Bagging (OZA, 2005).

5.5.3 Hoeffding Adaptive Tree (HAT)

Most of the decision tree-based algorithms for learning from data streams either assume that the underlying distribution is static, e.g., VFDT (see Section 5.1), or contain hardwired constants concerning the speed or frequency of change, e.g., CVFDT (see Section 5.5.1). These choices are inconclusive and often incorrect due to the plasticity-stability dilemma, but also since one cannot assume that all changes in a stream share the same frequencies and lengths.

In (BIFET; GAVALDÀ, 2009) authors proposed the adoption of an adaptive sliding window drift detector, named ADWIN (BIFET; GAVALDÀ, 2007), inside decision trees for data streams. Their proposal called Hoeffding Adaptive Tree (HAT), is an extension to CVFDT in which an ADWIN detector is used to monitor and flag changes in split nodes of the tree. Therefore, instead of relying on window parameters T_0, T_1 and T_2 for re-evaluating split nodes, HAT replaces split nodes when a significant error rate change occurs, given a significance level δ.

HATs are thus able to cope with both concept drifts and feature drifts since split nodes are re-evaluated. This allows split nodes to be consistent in terms of the feature adopted to perform the split and in which range/value of this feature the decision should be made.

5.5.4 Heuristic Updatable Weighted Random Subspaces (HUWRS)

The Heuristic Updatable Weighted Random Subspaces (HUWRS) is a random subspace ensemble for data streams (HOENS; CHAWLA; POLIKAR, 2011). HUWRS works under the hypothesis that when a feature drift occurs, there is no need to learn an entirely new predictive model so it builds its containing experts in different feature subspaces, while feature drifts are detected according to a landmark window. On each arriving batch, numeric features are discretized in equal-sized bins and the class distribution inside each bin of every feature is computed.

HUWRS postulates that a feature drift occurs in a feature X_i if the Hellinger weight between the class distribution of the current and prior landmarks differ at least by p, a user-given threshold. The Hellinger weight is given by Equation 5.2, which is a normalization to the Hellinger distance, given by Equation 5.3. In Equations 5.2 and 5.3, Y' and Y'' stand for the class distributions of the current and prior landmarks for an arbitrary feature X_i and q iterates over all the possible values of an attribute X_i.

$$w_H(Y', Y'') = \frac{\sqrt{2} - d_H(Y', Y'')}{\sqrt{2}} \tag{5.2}$$

$$d_H(Y', Y'') = \sqrt{\sum_{q \in X_i} \left(\sqrt{P[Y'|X_i = q]} - \sqrt{P[Y''|X_i = q]} \right)^2} \tag{5.3}$$

Since a low Hellinger distance[1] means a high agreement in the two distributions, a low Hellinger distance should correspond to a high weight.

Whenever a feature drift is flagged for a feature X_i, HUWRS resets only the experts associated with such feature, and thus, HUWRS is expected to adapt to feature drifts while performing less retraining when compared to full reset approaches.

5.5.5 Adaptive Random Forest (ARF)

The Adaptive Random Forest (ARF) is an ensemble-based classifier tailored for data stream classification proposed in (GOMES et al., 2017). In contrast to Streaming Random Forest (SRF) earlier reported in Section 5.3.1, ARF includes an effective resampling method and adaptive operators that can cope with different types of concept drifts without complex optimizations for different data sets. First, to induce diversity amongst the trees in the ensemble, each subtree is trained with different instances following a boostrapping process earlier proposed in Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010). Furthermore, each split decision is limited to observing a proportion m of the entire subset of features M, with $m < M$. Finally, each tree is associated with a drift detector. Instead of reseting a subtree whenever a drift is signalled, ARF also takes into account a "warning" level that starts the creation of a background tree. Each background tree is trained in parallel with the ensemble, yet, does not affect its predictions. Therefore, if a tree flags a drift, its respective background tree is used as a replacement to speed up the drift recovery process. Even though ARF is not bounded to a specific drift detector, authors in (GOMES et al., 2017) have adopted ADWIN (BIFET; GAVALDÀ, 2009) as the default detector.

[1] $\sqrt{2}$ is the maximum Hellinger distance between distributions for binary classification problems, thus, this value is used as a normalization factor so that the Hellinger weight is bounded in $[0; 1]$.

5.6 Benchmarking Related Work

This section assesses the performance of the surveyed algorithms and evaluates the use of conventional drift detectors presented in Section 2.3. The results stated in this section will serve as baselines for comparing the proposed methods to verify whether feature selection or feature weighting schemes are beneficial.

5.6.1 Drift Detectors

As discussed in Section 3.7, feature drifts are one specific kind of concept drift where the relevant subset of features changes. Intuitively, the first postulation regarding solutions for this problem is that conventional drift detectors will enable classifiers to quickly recover. This section evaluates ADWIN (BIFET; GAVALDÀ, 2007) and ECDD (ROSS et al., 2012) drift detectors and their combination with the Very Fast Decision Tree (VFDT), k-Nearest Neighbor (kNN), Naive Bayes (NB) and Hoeffding Adaptive Tree (HAT) classifiers. The remainder of the metrics are not judged since the focus of this analysis is strictly on classification rates. The parameter for ADWIN is the $\delta = 0.002$ for flagging changes, whereas ECDD was tested with $\lambda = 0.2$.

Table 5.2 presents the Prequential accuracy obtained in experiments, where HAT presents higher accuracy in most of the experiments. With the aid of hypothesis tests, it follows that {HAT, VFDT-ADWIN, VFDT-ECDD, VFDT, HAT-ADWIN, HAT-ECDD, NB-ECDD, NB-ADWIN} are statistically superior to others (Figure 5.1), thus highlighting the capability of HAT and tree-based learners combined with drift detectors to overcome feature drifts. The results obtained are expected since HAT performs evaluations of features used in test nodes of the tree according to drifts flagged by the ADWIN detector. This must be emphasized since HAT is the only algorithm capable of performing dynamic feature selection as the stream progresses, since bad performing splits are replaced on-the-fly given error rates.

Furthermore, HAT results also show that partial model resets are more interesting than full model resets (BARDDAL et al., 2017). In contrast to full model resets, partial resets are beneficial since classifiers still possess a partial model to classify upcoming instances, even after drifts.

5.6.2 Ensemble-based Classifiers

This section is devoted to the results obtained by the following algorithms: Random Rules (RR), Streaming Random Forest (SRF), HEFT-Stream (HEFT), Streaming

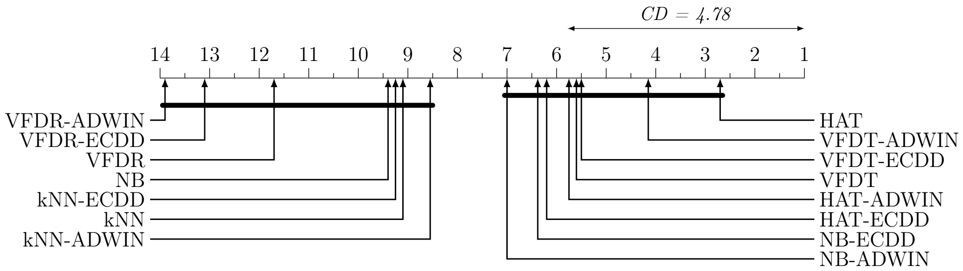


Figure 5.1: Critical differences chart for base learners and drift detectors.

Stacking (SS), and Adaptive Random Forest (ARF). All of these methods are ensembles, and thus, analyzing their results is difficult since these methods combine different building blocks, e.g., selection strategies, voting combinations and diversity induction schemes, since all impact the final outcome. The parameters for both algorithms are the ones stated in their original papers. HEFT-Stream has an ensemble size of 10 experts (NGUYEN et al., 2012), while Streaming Stacking covers all feature subsets with $M = 2$ (BIFET EIBE FRANK; PFAHRINGER, 2010). Both Random Rules and Streaming Random Forests explore all possible feature subsets in parallel with both $M = 1$ and $M = 2$ following the rationale behind Streaming Stacking by limiting the number of feature subset combinations. Finally, the Adaptive Random Forest (ARF) uses the ADWIN drift detector, randomly selects $(\sqrt{d} + 1)$ features for determining which feature to split on such that d is the cardinality of the original feature set $\mathcal{X}$, a grace period of 50 instances, and an ensemble size of 100 trees (GOMES et al., 2017).

Accuracy results obtained are presented in Table 5.3, where Streaming Stacking presents the best results in most experiments, followed by ARF, HEFT-Stream, and SRF ($M = 2$). In practice, Streaming Stacking is the best performing algorithm in all but two experiments, i.e., BG3 and SPAM, such that in the latter it failed due to lack of available memory. If one ignores these errors, Streaming Stacking and HEFT-Stream outperform other ensemble-based methods, as shown in Figure 5.2.

Table 5.2: Average Prequential Accuracy (%) obtained during experiments that benchmark base learners and drift detectors.

Experiment	No drift detector					ADWIN					ECDD				
	kNN	NB	VFDT	VFDR	HAT	kNN	NB	VFDT	VFDR	HAT	kNN	NB	VFDT	VFDR	HAT
AGR	54.71	59.13	57.21	56.16	**69.84**	53.95	63.04	63.48	37.74	63.35	53.95	64.27	62.89	54.49	63.24
AN	67.31	74.37	82.86	58.03	**84.74**	66.59	78.88	77.71	51.85	80.33	65.95	78.74	78.17	54.60	79.83
BG1	79.35	76.21	80.86	70.26	**91.76**	79.37	82.60	83.17	56.61	82.48	78.15	83.22	83.56	62.92	82.60
BG2	71.37	75.80	78.06	68.67	**78.92**	72.03	77.14	78.51	47.99	77.61	70.88	77.26	77.90	65.16	76.30
BG3	55.50	57.70	64.04	52.75	**64.45**	55.68	57.64	59.42	51.38	57.90	55.29	57.13	57.36	52.19	57.29
RTG	52.31	54.15	54.84	51.77	**56.64**	52.50	54.27	54.85	51.91	53.98	52.45	55.72	55.13	51.33	54.65
SEA	63.46	79.26	77.63	68.91	78.80	64.18	**79.27**	78.38	55.47	77.46	64.34	74.97	74.93	63.47	75.63
IADS	**100.00**	67.95	92.90	88.20	83.90	**100.00**	68.02	92.92	79.71	82.56	**100.00**	67.42	91.55	77.12	80.12
NOMAO	**95.16**	83.86	91.08	85.14	92.67	95.30	86.19	90.80	78.77	90.22	94.88	84.91	89.22	82.14	89.60
SPAM	83.72	71.13	78.07	74.81	84.48	84.15	79.16	85.50	81.42	85.50	84.98	66.82	**87.10**	83.30	86.79

Bold-faced results represent the best overall values obtained by all classifiers in given experiment. Underlined values stand for the best results obtained by each classifier in the given experiment. † represents an execution error (VFDR code failed).

Table 5.3: Average Prequential Accuracy obtained during ensemble-based experiments.

Experiment	Random Rules ($M = 1$)	Random Rules ($M = 2$)	Streaming Random Forest ($M = 1$)	Streaming Random Forest ($M = 2$)	HEFT-Stream	Streaming Stacking	Adaptive Random Forest
AGR	58.42	54.62	49.52	57.30	51.83	**69.65**	67.78
AN	58.24	63.15	58.59	65.32	75.06	**84.19**	82.18
BG1	68.13	69.54	64.99	70.22	72.88	**84.16**	81.22
BG2	51.85	48.91	52.78	68.84	73.25	**78.57**	75.99
BG3	50.52	52.03	49.35	49.80	57.17	64.18	**73.88**
RTG	52.80	53.64	**53.77**	52.81	53.09	53.52	52.51
SEA	53.82	53.34	50.09	59.21	76.45	**82.14**	72.90
IADS	58.33	58.88	56.18	67.08	74.15	**83.15**	79.75
NOMAO	56.25	56.46	54.16	60.50	65.68	**73.77**	77.26
SPAM	75.22	◇	74.17	◇	**82.65**	◇	80.07

Bold-faced results represent the best overall values obtained by all classifiers in given experiment. Underlined values stand for the best results obtained by each classifier in the given experiment. † represents an execution error while ◇ represents an insufficient memory error.

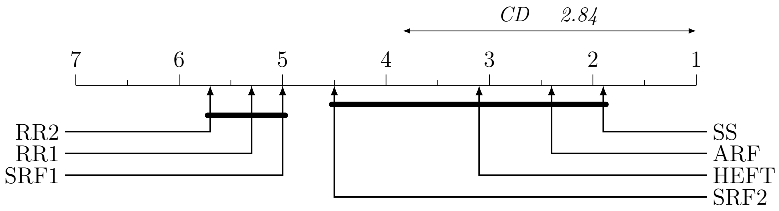


Figure 5.2: Critical differences chart for ensemble-based methods.

Given these results, it is important to compare the results obtained by Streaming Stacking against the Hoeffding Adaptive Tree. A comparison between these algorithms' accuracy metrics using Wilcoxon's paired test shows that these algorithms perform, on average, without significant differences. More importantly, in contrast to SS, HAT is a single decision tree, which allows not only better understanding, but also better efficiency in processing time and memory consumption. To corroborate with this claim, Figures 5.3 and 5.4 present comparisons between the processing time and memory consumption for both HAT and SS during experiments. Again, with the aid of Wilcoxon's paired test, it is possible to affirm that HAT is significantly faster and computationally cheaper than SS. Therefore, choosing HAT as a baseline for future comparisons is an intuitive choice since: (i) it provides as good as or better accuracy results, (ii) consumes less processing time and memory space, and (iii) clearer readability since understanding a single decision tree is much easier than understanding an ensemble that combines several decision trees with a perceptron.

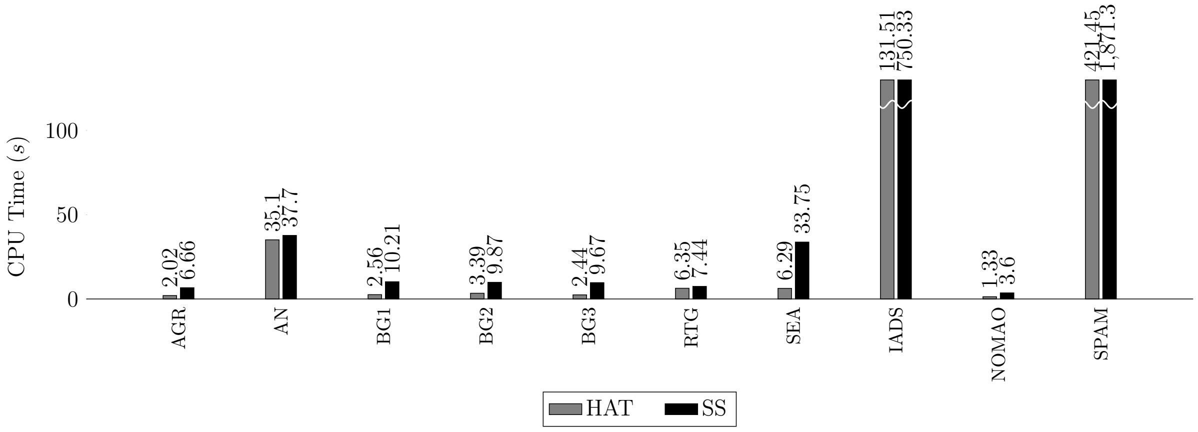


Figure 5.3: CPU Time (s) comparison between Hoeffding Adaptive Tree (HAT) and Streaming Stacking (SS) in experiments.

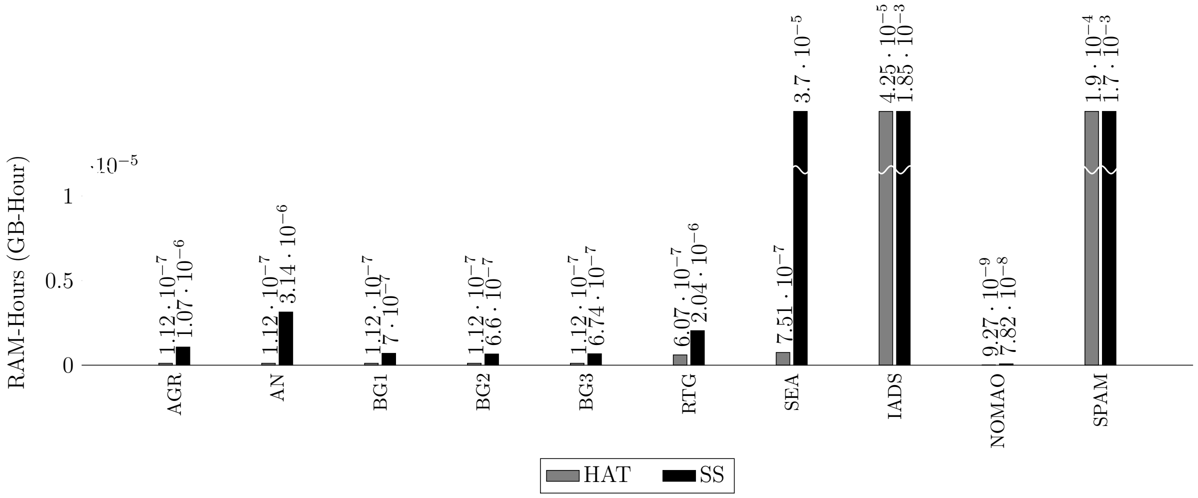


Figure 5.4: RAM-Hours (GB-Hour) comparison between Hoeffding Adaptive Tree (HAT) and Streaming Stacking (SS) in experiments.

5.7 Concluding Remarks

This chapter yielded a survey on existing works on feature drift adaptation. The brevity of surveyed works, especially compared to broader surveys on concept drift adaptation (GAMA et al., 2014; NGUYEN; WOON; NG, 2014; WANKHADE; HASAN; THOOL, 2013), shows the lack of efficient proposals for handling feature drifts. Thus, it is enlightened that feature drift is, so far, a nearly neglected problem in the data stream mining community. Besides serving as a mere survey on the topic, it is expected that the provided categorization aids in the positioning of new adaptive learning techniques and applications to which these apply.

The following chapter is devoted to the introduction of dynamic feature scoring operators. These operators will be used as weighting factors and as the core of a novel dynamic feature selection algorithm. In both cases, the best performing algorithm, i.e., the Hoeffding Adaptive Tree, will be used as a baseline for comparisons.

PART II

CONTRIBUTIONS

Chapter 6

Dynamic Feature Scoring

Tracking the relevance of features as a stream progresses is not straightforward. This chapter is devoted to the introduction of dynamic feature scoring operators based on the information theoretic Entropy (SHANNON, 1948) and Symmetrical Uncertainty (WITTEN; FRANK, 2005). These scoring operators are the core of the proposed methods, which are subsequently introduced in Chapters 7 and 8.

Section 6.1 introduces Entropy and Symmetrical Uncertainty metrics and how both can be dynamically computed along sliding windows with low complexity. Nevertheless, one of the drawbacks of entropy-based metrics in classification environments is that they require attributes to be discrete. In addition to that, discretizing features in streaming scenarios, where the maximum and minimum values of a variable are unknown, makes this task even more cumbersome. Therefore, Section 6.2 is dedicated to the attribute discretization scheme adopted.

As discussed in Section 5.5, any window-based algorithm suffers from the plasticity-stability dilemma, where deciding an appropriate window size is a trade-off without a clear solution, and thus, Section 6.3 is dedicated to an experimental evaluation of different window sizes that provides insights into reasonable window sizes for the upcoming proposals. Finally, Section 6.4 concludes this chapter with an overview of the proposed scoring operators, which will be used in the following chapters in a weighting scheme and in a dynamic feature selection algorithm.

6.1 Information Theoretic Scoring Operators

Shannon's Entropy[1] (SHANNON, 1948) measures how chaotic a system is and can be applied as a measure of uncertainty about a partition.

Definition 18. Given a discrete random variable A, its entropy can be computed as follows:

$$H(A) = -\sum_{a}^{A} P[A = a] \log_2 P[A = a] \tag{6.1}$$

where a iterates through all possible values of A.

Entropy is bounded in the $[0; \log_2 n]$ interval, where n is the number of possible values (partitions) the variable A has. A perfect distribution in terms of entropy is achieved when $H(A) = 0$, meaning that one partition is certain and the others are impossible while the extreme chaotic distribution occurs when $H(A) = \log_2 n$, meaning that all partitions are equally likely.

In order to introduce the computation of entropy over a sliding window, a slightly different notation is followed. All of the following provided definitions and proofs are based on the work of (SOVDAT, 2014).

Definition 19. Assuming $A = \{a^i\}_{i=1}^{n}$ to be a sample of real numbers and $S_n = \sum_{i=1}^{n} a^i$ be the sum of these elements, the sample entropy H_n is defined as follows:

$$H_n = -\sum_{i=1}^{n} \frac{a^i}{S_n} \log_2 \frac{a^i}{S_n} \tag{6.2}$$

Lemma 1. *Given a sample $A = \{a^i\}_{i=1}^{n}$, its sum $S_n = \sum_{i=1}^{n} a^i$ and H_n to be its entropy, for any positive real number $R > 0$, it occurs that:*

$$-\sum_{i=1}^{n} \left(\frac{a^i}{S_n + R} \log_2 \frac{a^i}{S_n + R} \right) = \frac{S_n}{S_n + R} \left(H_n - \log_2 \frac{S_n}{S_n + R} \right) \tag{6.3}$$

[1] Entropy is a broader concept originally introduced by Rudolf Clausius in 1854 to measure the unavailability of a system's thermal energy for conversion into mechanical work. It is also a landmark of the second law of thermodynamics, which states that *"The energy of the universe if constant. The entropy of the universe tends to a maximum."*

Proof. If one writes $\frac{a^i}{R+S_n} = 1 \cdot \frac{a^i}{R+S_n} = \frac{S_n}{S_n}\frac{a^i}{S_n+R} = \frac{a^i}{S_n}\frac{S_n}{S_n+R}$, it follows:

$$\begin{aligned}
-\sum_{i=1}^{n}\left(\frac{a^i}{S_n+R}\log_2\frac{a^i}{S_n+R}\right) &= -\sum_{i=1}^{n}\frac{a^i}{S_n}\frac{S_n}{S_n+R}\log_2\left(\frac{a^i}{S_n}\frac{S_n}{S_n+R}\right)\\
&= -\sum_{i=1}^{n}\frac{a^i}{S_n}\frac{S_n}{S_n+R}\left(\log_2\frac{a^i}{S_n}+\log_2\frac{S_n}{S_n+R}\right)\\
&= -\frac{S_n}{S_n+R}\left(\sum_{i=1}^{n}\frac{a^i}{S_n}\log_2\frac{a^i}{S_n}+\sum_{i=1}^{n}\frac{a^i}{S_n}\log_2\frac{S_n}{S_n+R}\right)\\
&= \frac{S_n}{S_n+R}\left(H_n-\log_2\frac{S_n}{S_n+R}\sum_{i=1}^{n}\frac{a^i}{S_n}\right)\\
&= \frac{S_n}{S_n+R}\left(H_n-\log_2\frac{S_n}{S_n+R}\right)
\end{aligned}$$

□

The sample entropy H_n can be updated when a new positive real number $a^{n+1} > 0$ enters it as follows.

Lemma 2. *Based on H_n and S_n, the entropy can be updated after the arrival of a new positive real number $a^{n+1} > 0$ according to:*

$$H_{n+1} = \frac{S_n}{S_{n+1}}\left(H_n - \log_2\frac{S_n}{S_{n+1}}\right) - \frac{a^{n+1}}{S_{n+1}}\log_2\frac{a^{n+1}}{S_{n+1}} \tag{6.4}$$

Proof. By definition, and by following Lemma 2, the updated entropy is given by:

$$\begin{aligned}
H_{n+1} &= -\sum_{i=1}^{n+1}\frac{a^i}{S_{n+1}}\log_2\frac{a^i}{S_{n+1}}\\
&= -\frac{a^{n+1}}{S_n} - \sum_{i=1}^{n}\frac{a^i}{S_{n+1}}\log_2\frac{a^i}{S_{n+1}}\\
&= \frac{S_n}{S_{n+1}}\left(H_n - \log_2\frac{S_n}{S_{n+1}}\right) - \frac{a^{n+1}}{S_{n+1}}\log_2\frac{a^{n+1}}{S_{n+1}}
\end{aligned}$$

□

The next lemma is a generalization to the last claim and provides formulas for the entropy of concatenated samples.

Lemma 3. *Let $A = \{a^i\}_{i=1}^{n}$ and $B = \{b^i\}_{i=1}^{m}$ be two data samples, $R_m = \sum_{i=1}^{n} a^i$ and $S_n = \sum_{i=1}^{m} b^m$ be their respective sums and G_m and H_n be their entropies. Defining a* *sample*

$$z_i = \begin{cases} a^i, \ \textit{if } 1 \leq i \leq n, \\ b^{i-n}, \ \textit{if } n+1 \leq i \leq m+n \end{cases}$$

and $Z_{m+n} = R_m + S_n = \sum_{j=1}^{n+m} z_j$*, it follows that*

$$E_{m+n} = \frac{R_m}{Z_{m+n}} \left(G_m - \log_2 \frac{R_m}{Z_{m+n}} \right) + \frac{S_n}{Z_{m+n}} \left(H_n - \log_2 \frac{S_n}{Z_{m+n}} \right) \tag{6.5}$$

Proof. Similarly as before, it occurs that

$$\begin{aligned} E_{n+m} &= -\sum_{i=1}^{n+m} \frac{z_i}{Z_{m+n}} \log_2 \frac{z_i}{Z_{m+n}} \\ &= -\sum_{i=1}^{m} \frac{a^i}{Z_{m+n}} \log_2 \frac{a^i}{Z_{m+n}} - \sum_{i=1}^{n} \frac{b^i}{Z_{m+n}} \log_2 \frac{b^i}{Z_{m+n}} \\ &= \frac{R_m}{Z_{m+n}} \left(G_m - \log_2 \frac{R_m}{Z_{m+n}} \right) + \frac{S_n}{Z_{m+n}} \left(H_n - \log_2 \frac{S_n}{Z_{m+n}} \right) \end{aligned}$$

where the last equality is produced after applying Lemma 1 twice, once per summation. □

Finally, Lemma 4 provides an entropy formula for when some elements a^i are increased by $r_i > 0$, where I is the index set and $r_i = 0$ for $i \notin I$.

Lemma 4. *Again, let* $A = \{a^i\}_{i=1}^n$*,* S_n *and* H_n *be a data sample, its sum and entropy, respectively. Suppose that* a^i *increases by* $r_i > 0$ *for* $i \in I$ *and* $r = \sum_{i=1}^n r_i$*, where* $r_i = 0$ *for* $i \notin I$*. In this case,* H_n *should be updated as follows:*

$$\frac{S_n}{S_n + r} \left(H_n - \log_2 \frac{S_n}{S_n + r} \right) - \sum_{i \in I} \left(\frac{a^i + r_i}{S_n + r} \log_2 \frac{a^i + r_i}{S_n + r} - \frac{a^i}{S_n + r} \log_2 \frac{a^i}{S_n + r} \right) \tag{6.6}$$

Proof. This is a direct application of Lemma 3. The rationale is to treat $\frac{a^i + r_i}{S_n + r}$ as new values to be added and subtract the replaced values $\frac{a^i}{S_n + r}$. It is important to highlight that in the subtraction, the denominator is $S_n + r$ since the subtraction is performed from the updated entropy, the one obtained after Lemma 3 was applied. □

Following the last lemma, it is straightforward to provide formulas for a sliding window computation of entropy. Algorithm 1 provides a pseudocode for the Entropy computation of a variable along a data stream. It is important to highlight that the

Algorithm 1: Sliding window entropy. Adapted from (SOVDAT, 2014).

```
input  : window size $w$, a data stream $\mathcal{S}$.
output : be ready to provide the entropy $h$ at any time.
Let $W \leftarrow \emptyset$ be the sliding window;
Let $h \leftarrow 0$ be the entropy;
Let $n \leftarrow 0$ be the number of instances in $W$;
Let $n_i \leftarrow 0$ be the number of instances that belong to the $i^{th}$ partition;
foreach $n_i \in \mathcal{S}$ do
    if $|W| = w$ then
        Dequeue oldest element from W from the $n_j$-th partition;
        $h \leftarrow DEC(h, n, n_j)$;
    $W \leftarrow W \cup \{n_i\}$;
    $h \leftarrow INC(h, n, n_i)$;
Function $INC(h, n, n_i)$
    Update $n \leftarrow n + 1$;
    Update $n_i \leftarrow n_i + 1$;
    return $\frac{n-1}{n}\left(h - \log_2 \frac{n-1}{n}\right) - \frac{n_i}{n}\log_2 \frac{n_i}{n} + \frac{n_i-1}{n}\log_2 \frac{n_i-1}{n}$
Function $DEC(h, n, n_i)$
    Update $n \leftarrow n - 1$;
    Update $n_i \leftarrow n_i - 1$;
    return $\frac{n+1}{n}\left(h + \frac{n_i+1}{n+1}\log_2 \frac{n_i+1}{n+1} - \frac{n_i}{n+1}\log_2 \frac{n_i}{n+1}\right) + \log_2 \frac{n}{n+1}$
```

algorithm provided is very efficient, since the entropy computation is performed in $\mathcal{O}(1)$ for both enqueueing and dequeueing of a value.

Clearly, one of the major drawbacks of picking entropy-based metrics as a goodness measure is that they are unable to work with numeric features, unless they are discretized. Discretizing numeric variables in which their minimum and maximum values are *a priori* unknown, such as data streams, is not trivial and thus, specific techniques shall be used for this purpose. The proposed solution for discretizing numeric attributes in data streams is discussed in Section 6.2.

6.1.1 Conditional Entropy Scoring Operator

The first scoring operator is conditional entropy. Conditional entropy quantifies the amount of information needed to describe the outcome of the class Y given that the value of another a feature X_i is known. This conditional entropy is stated in Equation 6.7, where $H(Y|X_i) = 0$ occurs if Y is completely determined by X_i and $H(Y|X_i) = H(Y)$ if

X_i and Y are independent random variables.

$$\begin{aligned} H(X_i|Y) &= -\sum_{y_j}^{Y} P[Y = y_j] \times H(X_i \mid Y = y_j) \\ &= -\sum_{y_j}^{Y} P[Y = y_j] \sum_{q}^{X} P[X_i = q \mid Y = y_j] \log_2 P[X_i = q \mid Y = y_j] \end{aligned} \tag{6.7}$$

Taking this definition to the classification scenario, the conditional entropy scoring operator for an attribute X_i w.r.t. its class Y discrimination would be computed as $H(Y|X_i)$. As stated in Equation 6.7, this entropy can be break down into several $H(Y|X_i = q)$ conditional entropies, such that the conditional entropy for the q^{th} value of X_i can be computed given Algorithm 1. Furthermore, the final conditional also requires the probabilities of each class, a computation that is trivial and depends on a single counter per class c_j and the window size w, since $P[Y = y_j] = {}^{c_j}/_{w}$.

6.1.2 Symmetrical Uncertainty Scoring Operator

One of the major problems with entropy is that it is biased towards attributes with more values. For instance, let us assume a dataset that contains an unique identifier attribute. In this case, each identifier would be associated with a single class value, and thus, the entropy would be optimal, i.e., zero. This is an extreme case, where the identifier would not be an interesting attribute since it does provide any insights for predicting the final class since it is massively overfitted to the training data.

On the other hand, entropy is the basis for more refined metrics. For instance, entropy can be used to compute Information Gain (IG), a popular choice for learning decision trees. Information Gain is the amount by which the Entropy of a variable Y decreases reflecting additional information about Y provided by X_i, and is given by:

$$IG(Y, X_i) = H(Y) - H(Y|X_i) \tag{6.8}$$

An important trait of Information Gain is that it is symmetrical, i.e., $IG(X_i, Y) = IG(Y, X_i)$. To prove it, one needs to verify that $H(X_i) - H(X_i|Y) = H(Y) - H(Y|X_i)$ and this can be derived from $H(X_i, Y) = H(X_i) + H(Y|X_i) = H(Y) + H(X_i|Y)$.

However, as Entropy, Information Gain is biased towards features with more values. Therefore, different metrics that compensate for this bias are preferred. For instance, one could proceed with Gain Ratio (GR), given by Equation 6.9, which is a normalization that aims at compensating the latter bias by computing the ratio between the Information Gain and the entropy of the non-class variable.

$$GR(Y, X_i) = \frac{IG(Y, X_i)}{H(X_i)} \tag{6.9}$$

Similarly to Gain Ratio, another possibility would be Symmetrical Uncertainty (SU). As an attribute goodness measure, Symmetrical Uncertainty is also known for atoning the bias provided by attributes with more distinct values and has been successfully used as the core of many batch feature selection algorithms (YU; LIU, 2003), and thus, was chosen as the second scoring operator. The Symmetrical Uncertainty between two discrete random variables X and Y is given by:

$$\begin{aligned} SU(X_i, Y) &= 2\left[\frac{IG(X_i, Y)}{H(X_i) + H(Y)}\right] \\ &= 2\left[\frac{H(Y) - H(Y|X_i)}{H(X_i) + H(Y)}\right] \end{aligned} \tag{6.10}$$

The range of possible values for SU is the $[0; 1]$ interval, where 1 indicates that the value of a variable completely predicts the other, while 0 indicates that X_i and Y are completely independent.

In order to compute SU along a sliding window, one must keep track of $H(X_i)$, $H(Y)$ and $H(Y|X_i)$ entropies. Both $H(X_i)$ and $H(Y)$ can be incremented and decremented in $\mathcal{O}(1)$ according to Algorithm 1, while the Conditional Entropy $H(Y|X_i)$ can be computed with separate $H(Y|X_i = q)$ entropies (see Equation 6.7), also given by Algorithm 1. If one assumes that $q \in X_i$ and $|X_i| = m$, then SU can be computed with low computational complexity in the $\mathcal{O}(m)$ order for a single feature and $\mathcal{O}(dm)$ for all features in a d-dimensional data stream.

Memory-wise, the cost of tracking $H(Y)$ is $\mathcal{O}(|Y|)$, while the cost for $H(X_i)$ is $\mathcal{O}(m)$, thus, the total complexity is $\mathcal{O}(md)$ for a d-dimensional stream. Finally, $H(Y|X_i = q)$ incurs a cost of $\mathcal{O}(|Y|)$, therefore the total cost is $\mathcal{O}(md \times |Y|)$, when considering all features $X_i \in \mathcal{X}$. In practice, this cost can be reduced since each of these entropies is independent from one another, and thus are allowed to be computed in parallel. The implementation of both scoring operators introduced in this work adopts the Java 8 parallel stream processing and functional programming frameworks, which together use all of the available cores and threads to update the required entropies.

6.2 Discretization of Numeric Features on Data Streams

Discretization is an important task in machine learning used either in classification or feature selection methods. As discussed earlier, one of the drawbacks of adopting entropy-based metrics for quantifying the relevance of a feature is that it is only capable of dealing with discrete variables. Discretizing numeric variables in which their minimum and maximum values are unknown is not trivial, and thus, specific techniques shall be used for this purpose.

In this work, a two-layer histogram strategy is proposed as an extension to Partition Incremental Discretization (PiD) (GAMA; PINTO, 2006). In the first layer, PiD constructs and maintains equal-size bins that summarize the values provided by a stream. Given the partitions computed in the first layer, periodically, or whenever a partition is accentuated compared to others, PiD's second layer of partitions is reconstructed, following an equal frequency strategy. One of the pitfalls of PiD is that it is unable to "forget" older data, and thus, this section introduces Partition Adaptive Discretization (PaD), in which the histograms in both layers are updated to reflect the distribution of the data in a sliding window. Another drawback of PiD is that partitions are stored in a linear search structure, in which the corresponding bin for a certain value incurs in a $\mathcal{O}(n)$ complexity since no binary search process was used, where n is the amount of bins stored.

Algorithm 2 depicts the pseudocode for PaD. As in PiD, PaD's first layer summarizes data, while the second layer constructs the final histogram. In contrast to PiD, partitions in PaD are layered in a red-black tree structure[2], where both insertions and searches occur in $\mathcal{O}(\log_2 n)$ for the first layer and $\mathcal{O}(\log_2 F)$ for the second, where F is the number of partitions requested by the user. The first layer retrieves and stores the first w values in a buffer V to create a histogram with equal-width bins, such that each bin has a length equal to $\frac{1}{\max(V)-\min(V)}$. After the initial computation of the first layer (lines 30-41), incoming values are stored in a sliding window (line 6), where enqueued values v_i are used to increment existing partitions (or to allocate new ones if none contains v_i) and dequeued values decrement their corresponding partitions (lines 8-12). Therefore, the process of updating the first layer is linear as it performs a linear scan over arriving data.

The second layer is reconstructed (i) periodically or (ii) whenever a "significant" change is detected in the first layer of histograms. In opposition to the first strategy, which reconstructs the second layer regardless of the distribution of the data into bins, the histogram change detection scheme provided by PiD is here extended. Given two

[2] In practice, a common sorted array or any other tree-like structure could be used, as long as a binary search process is adopted.

Algorithm 2: Partition Adaptive Discretization (PaD) pseudocode, which is inspired by Gama's PiD (GAMA; PINTO, 2006).

```
input   : a stream of numeric values $\mathcal{S}$ that correspond to a single feature $X_i$, the amount of partitions $F$ and
          the sliding window size $w$.
output  : be able to provide at any moment $F$ equal-width bins: $sLayer$.
/* $lb$ = lower bound, $up$ = upper bound, $c$ = counter                                              */
[1] Let $fLayer \leftarrow \emptyset$ be a red-black tree structure to store the first layer bins $b = (lb, up, c)$;
[2] Let $sLayer \leftarrow \emptyset$ be a red-black tree structure to store the second layer bins $b = (lb, up, c)$;
[3] Let $V \leftarrow \emptyset$ be a sliding window;
[4] foreach $x^t \in \mathcal{S}$ do
[5]     $mustReconstruct \leftarrow FALSE$ ;                          /* flag for second layer reconstruction */
[6]     $V \leftarrow V \cup \{x^t\}$;
[7]     if $|V| > w$ then
[8]         $incrementLayer(x^t, fLayer)$;
[9]         Dequeue oldest element $x^{t-w}$ from $V$;
[10]        $decrementLayer(x^{t-w}, fLayer)$;
[11]        $mustReconstruct \leftarrow decrementLayer(x^{t-w}, sLayer)$;
[12]        $mustReconstruct \leftarrow mustReconstruct$ or $incrementLayer(x^t, sLayer)$;
[13]    else if $|V| = w$ then
            /* first window is complete, thus, both layers can be constructed                          */
[14]        $buildFirstLayer()$;
[15]        $mustReconstruct \leftarrow TRUE$;
[16]    if $mustReconstruct$ then $buildSecondLayer()$;
[17] Function $incrementLayer(x^t, layer)$
[18]    Traverse $layer$ to find the node $(lb, ub, c)$ containing $x^t$;
[19]    if $If\ no\ such\ partition\ exists$ then
[20]        Create a new bin with $binLength$ size at the head or tail of the list that contains $x^t$;
[21]    else
[22]        Increment the counter $c$;
[23]    if $layer = sLayer$ and $c > (1 + \alpha) \times T_{max}$ then return $TRUE$ ;
[24]    return $FALSE$
[25] Function $decrementLayer(x^t, layer)$
[26]    Traverse $layer$ to find the node $(lb, ub, c)$ containing $x^t$;
[27]    Decrement the counter $c$;
[28]    if $layer = sLayer$ and $c < (1 - \alpha) \times T_{min}$ then return $TRUE$;
[29]    return $FALSE$
[30] Function $buildFirstLayer()$
[31]    $fLayer \leftarrow \emptyset$;
[32]    $binLenght \leftarrow 1/(\max V - \min V)$;
[33]    $init \leftarrow \min V$;
[34]    $end \leftarrow \min V + binLenght$;
[35]    while $end \neq F \times binLength$ do
[36]        $fLayer \leftarrow fLayer \cup \{(init, end, c = 0)\}$;
[37]        $init \leftarrow end$;
[38]        $end \leftarrow init + binLength$;
[39]    foreach $x^t \in \mathcal{S}$ do
[40]        Traverse $fLayer$ to find the node $(lb, ub, c)$ containing $x^t$;
[41]        Increment the counter c;
[42] Function $buildSecondLayer()$
[43]    $sLayer \leftarrow \emptyset$;
[44]    $np \leftarrow w/F$;
[45]    $init \leftarrow end \leftarrow counter \leftarrow 0$;
[46]    foreach $(lb, ub, c) \in fLayer$ do
[47]        if $init = 0$ then $init \leftarrow lb$;
[48]        $end \leftarrow ub$;
[49]        $counter \leftarrow counter + c$;
[50]        if $counter \geq np$ then
[51]            $sLayer \leftarrow sLayer \cup \{(init, end, counter)\}$;
[52]            $init \leftarrow end \leftarrow counter \leftarrow 0$;
```

thresholds T_{min} and T_{max}, which are the minimum and maximum frequencies of all existing partitions, the second layer of PaD is reconstructed (lines 42-52) whenever a partition with frequency above $(1 + \alpha) \times T_{max}$ (line 23) or below $(1 - \alpha) \times T_{min}$ (line 28) is observed. As noticed by authors in (GAMA; PINTO, 2006), the choice of α is not trivial, however, $\alpha = 1\%$ seemed reasonable in PiD's original experiments and the same is adopted here.

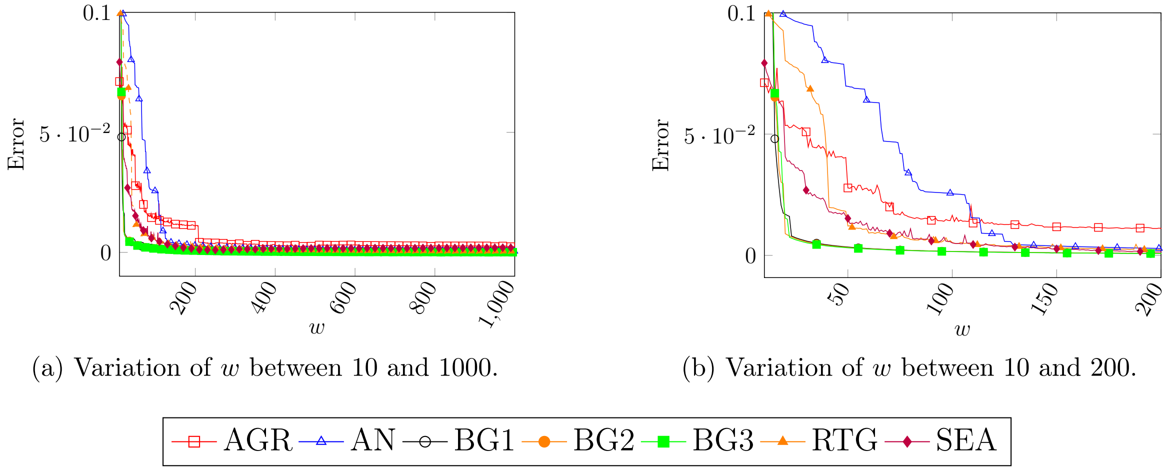
The construction of the second layer is trivial. Given a number of partitions F and the sliding window size w, each partition in the second layer will possess approximately $^{w}/_{F}$ frequency. The second layer is constructed linearly by traversing and aggregating the first layer bins until a $^{w}/_{F}$ frequency is reached (lines 42-52).

In the following experiments, the classic rule of thumb that numeric features are discretized into $F = 10$ equal-frequency partitions is followed, however, different number of partitions can be easily configured and assessed in further studies.

6.3 The Plasticity-Stability Trade-off and Symmetrical Uncertainty Computation over Sliding Windows

As any other window-based approach for learning from data streams, the proposed Symmetrical Uncertainty scoring operator also requires the definition of a proper window size. As discussed in Section 5.5, the size of a window should be as small as possible to allow quick drift recognition and adaptation, but at the same time, large enough so it correctly reflects the distribution of stable regions of a stream. Following this trend, an experimental window size evaluation strategy was set. First, several synthetic stationary data streams were generated, each with $100,000$ instances. Given each stream, all of its features had their attributes' Symmetrical Uncertainty to the class computed in batch mode to serve as a gold standard. Later, different sliding window sizes $w \in [10; 1000]$ were evaluated by computing the deviation between the obtained Symmetrical Uncertainty values and the gold standard for each feature. The final results reported here are averages obtained for all features in each experiment. Standard deviations were omitted since they are barely visible. With these results, it is possible to verify the smallest window size value that also satisfies a feasible Symmetrical Uncertainty deviation compared to the overall sample distribution.

Results are presented in Figure 6.1a, where is depicted that the error rates between the obtained SU values over sliding windows quickly decay with the increase of w. In practice, very small values, e.g., 100 to 150, already allow a fair SU computation regardless of the experiment domain during stationary experiments. Therefore, slightly higher values around 200 or 300 (see Figure 6.1b) are expected to provide a fair trade-off between drift



(a) Variation of w between 10 and 1000.

(b) Variation of w between 10 and 200.

Figure 6.1: Average error between the SU computed over the whole dataset and the one provided by the sliding window version.

adaptation and correct SU rendering during stable regions.

Moreover, this experiment also provides evidence that the Symmetrical Uncertainty provided by the proposed scoring operator is correct, with errors below 10^{-3} when $w \geq 250$.

6.4 Concluding Remarks

This chapter introduced a time and memory-bounded solution for the dynamic relevance scoring of features based on the information theoretic concepts of Entropy and Symmetrical Uncertainty. Both of the latter metrics are tracked over sliding windows. As other existing techniques for data streams, the definition of the window size is a trade-off without a solution. Albeit the plasticity-stability dilemma, empirical experiments have shown that even small window sizes (around 200), are enough to correctly represent the Symmetrical Uncertainty of a stationary stream. These results validate the first hypothesis of this thesis, as the proposed scoring schemes are able to highlight the importance of features compared to the traditional batch implementation of the same metrics.

In the following chapters, the use of the proposed scoring operators is investigated. Chapter 7 explores both Entropy and Symmetrical Uncertainty operators as a feature weighting procedure, while Chapter 8 investigates the Symmetrical Uncertainty operator as the core of a feature selection algorithm for data streams.

Chapter 7

Dynamic Feature Weighting

Feature weighting is a technique used to approximate the optimal degree of influence of features in data. When successful, relevant features are attributed with high weights, whereas irrelevant features are associated with weight values close to zero. Feature weighting is broadly used in batch learning (AGGARWAL, 2014; CHEN; WANG, 2012), while very few feature weighting techniques to streaming environments have been proposed so far (ALIPPI; BORACCHI; ROVERI, 2009). The proposed technique significantly differs from the above cited work since there, the weights are given by temporal functions developed for smooth changes and do not rely on information theory aspects.

In opposition to static learning scenarios, the relevance of features may increase or decrease during the processing of a data stream, thus, techniques for tracking and quantifying the proportions of such changes in weights are needed and expected to alleviate the impact of feature drifts. This chapter introduces the use of Entropy and Symmetrical Uncertainty scoring operators as weights for the Naive Bayes and k-Nearest Neighbors (kNN) classifiers (BARDDAL et al., 2016). The main hypothesis behind this proposal is that features can be dynamically weighted given the current data distribution of the stream and the updating of these weights will incur in improvements of prediction accuracy in feature drifting streams.

The proposed weighted versions of Naive Bayes and *kNN* are evaluated and later investigated as custom leaves of the Hoeffding Adaptive Tree as presented in (BARDDAL et al., 2016). Instead of computing weights globally, the use of weighted classifiers at the leaves of decision trees will incur in specialized weights for different samples of data determined by different branches of the tree.

7.1 k-Nearest Neighbor with Feature Weighting for Data Streams (kNN-FW)

k-Nearest Neighbors (kNN) is one of the most fundamental, simple and widely used classification methods, which is able to learn complex (non-linear) functions (AHA; KIBLER, 1991). kNN is a lazy learner since it does not require building a model before actual use. It classifies unlabeled instances according to "closest" previously seen labeled ones stored in a buffer. The definition of "close" means that a distance measure is used to determine how similar/dissimilar two instances are. There are several approaches to compute distances between instances, nevertheless, the most used one is the Euclidian distance, given by Equation 7.1, where $\vec{x}_i$ and $\vec{x}_j$ are two arbitrary instances, and the summation occurs over all features $X_k \in \mathcal{X}$.

$$d_{\text{Euclidian}}(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{X_k \in \mathcal{X}} (\vec{x}_i[X_k] - \vec{x}_j[X_k])^2} \tag{7.1}$$

kNN classifies unlabeled instances according to the label of the majority of the k closest instances, and thus, picking an appropriate value of k for each application is also significant. If k is too small, kNN becomes more prone to overfitting and tends to misclassify instances in easy situations. Conversely, bigger values of k may mislead classification in cases when an instance is surrounded by several instances of an opposite label in fuzzy decision borders.

Another important trait of kNN refers to the dimensionality of the data, either in static or streaming scenarios. As discussed in a variety of works (CAO et al., 2006; SILVA et al., 2013), Euclidian distances fail on representing in effective fashion the distance between points (instances) in a high-dimensional space, a phenomenon named *curse of dimensionality*. Although the *curse of dimensionality* is commonly tackled in static learning scenarios, very few works aim at providing techniques for dealing with it in streaming scenarios, either through feature selection or dimensionality reduction.

Performing kNN classification in data streams requires an additional important trait: dealing with time and memory limitations. Continuously buffering instances as they arrive is unfeasible since the stream is potentially unbounded, therefore, an incremental version of kNN must "forget" older instances as the stream progresses. A naive approach for "forgetting" is storing instances in a queue with size W. Again, defining a value for W is nontrivial and it must be set according to available memory space and processing time since the computational time for classifying each new instance is $\mathcal{O}(Wd)$.

The success of kNN relies on which instances are deemed close, a concept defined by its distance function. A Euclidian distance allows irrelevant and noisy features to have as much effect on distances as relevant ones. The hypothesis behind this proposal is that an incremental version of *kNN* can be extended to overcome irrelevant features and feature drifts through the incremental track of their discriminative power.

k-Nearest Neighbor with Feature Weighting (*kNN-FW*) is an extension presented in (BARDDAL et al., 2016a) to the original *kNN* algorithm that performs dynamic feature weighting to overcome both irrelevant features and feature drifts. Hence, *kNN-FW* is associated with the Entropy and Symmetrical Uncertainty scoring operators to emphasize relevant features and disdain irrelevant ones. The computation of distances in *kNN-FW* follows Equation 7.2, where $w(X_i)$ is the weight associated to a feature X_i and depends on the chosen scoring operator.

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{X_k \in \mathcal{X}} w(X_k) \times (\vec{x}_i[X_k] - \vec{x}_j[X_k])^2} \tag{7.2}$$

If entropy is chosen, the weight $w(\cdot)$ will be given according to Equation 7.3 since smaller values of entropy highlight discriminative features.

$$w(X_i) = \log_2 |Y| - H(Y|X_i) \tag{7.3}$$

On the other hand, if Symmetrical Uncertainty is used, then $w(\cdot)$ follows Equation 7.4, which depicts the Symmetrical Uncertainty between X_i and the class Y.

$$w(X_i) = SU(X_i, Y) \tag{7.4}$$

Due to the dynamic computation of the scoring operators, it is expected *kNN-FW* to be able to dynamically assign weights to features according to their current discriminative power. During non-feature-drifting scenarios, it is expected that irrelevant features are unaccounted for during voting, while discriminant features are emphasized. In feature-drifting cases, features that become, or cease to be, relevant to the learning task will be promptly detected by changes in their entropies, implying in changes in features' weights.

7.2 Updatable Naive Bayes with Feature Weighting (NB-FW)

Naive Bayes (NB) is a probabilistic classifier based on Bayes theorem that works under the naive independence assumption between features. These predictors are easy to build, can easily be incremented, and have no complicated parameter estimation, making them useful for large datasets and streams. The labelling of instances in this learning scheme is given by Equation 7.5, that is, the class is chosen according to the label y_i that maximizes the conditional probability of features given classes.

$$y = \operatorname*{argmax}_{y_i \in Y} P[y_i] \prod_{j=1}^{d} P[\vec{x}[X_j] \mid y_i] \tag{7.5}$$

Although Naïve Bayes is commonly referred as an appropriate solution for high dimensionality problems (CHEN; WANG, 2012), it has been shown to be prone to feature drifts (BARDDAL; GOMES; ENEMBRECK, 2015b).

Analogously to *kNN-FW*, the adoption of scoring operators in Naive Bayes prediction is simple. Naive Bayes with Feature Weight (NB-FW) labels instances according to Equation 7.6, where $w(\cdot)$ is the weighting function that follows either Equation 7.3 or 7.4 depending on the chosen scoring operator and ξ is a small padding factor, set to 0.0001, used to avoid zero weights which would nullify the probabilities of all possible class outcome values.

$$y = \operatorname*{argmax}_{y_i \in Y} P[y_i] \prod_{j=1}^{d} [w(X_i) + \xi] \times P\left[\vec{x}[X_j] \mid y_i\right] \tag{7.6}$$

7.3 Evaluating Feature Weighted Base Learners

Table 7.1 presents the prequential accuracy results obtained by the proposed feature weighted classifiers during the experiments. In all cases, the use of dynamic feature weighted classifiers was beneficial, providing an average increase of 5.36% for kNN and 3.64% for Naive Bayes using SU, and 2.67% for kNN and 3.14% for Naive Bayes if entropy is followed.

To provide statistical significance to the aforementioned claim, Figure 7.1 presents the Nemenyi critical differences, which shows that HAT is still the best performing algorithm followed by kNN-SU and NB-SU. These results highlight that Symmetrical Uncertainty provides better description of data when compared to Entropy and provides compelling accuracy gains when compared to their original versions.

In contrast to the enhancements obtained in accuracy, the proposed weighting scheme comes at the expense of both processing time and memory space. Figures 7.2 and 7.3 present the ratio between the weighted versions of *kNN* and NB classifiers against the original ones for processing time and memory consumption rates. The increases in

Table 7.1: Prequential accuracy (%) obtained by weighted classifiers.

Experiment	*k*NN	*k*NN-ENTROPY	*k*NN-SU	NB	NB-ENTROPY	NB-SU	HAT
AGR	54.71	58.54	64.20	59.13	67.60	67.71	**69.84**
AN	67.31	72.04	79.81	74.37	76.36	76.93	**84.74**
BG1	79.35	80.62	82.98	76.21	75.34	76.55	**91.76**
BG2	71.37	73.67	78.16	75.8	76.84	76.88	**78.92**
BG3	55.50	60.97	63.00	57.70	**64.65**	64.65	64.45
RTG	52.31	53.78	54.11	54.15	57.67	**57.68**	56.64
SEA	63.46	70.46	74.11	**79.26**	78.65	78.72	78.80
IADS	**100.00**	**100.00**	**100.00**	67.95	76.54	76.64	83.90
NOMAO	95.16	**95.66**	**95.66**	83.86	83.88	84.99	92.67
SPAM	83.72	83.87	**84.52**	71.13	73.47	75.22	84.48

Bold-faced results represent the best overall values obtained by all classifiers in given experiment. Underlined values stand for the best results obtained by each classifier in the given experiment.

processing time are expected, since the weight computation for all features is an overhead that is added to the classifier. As discussed in Section 6.1, the overhead for maintaining the scoring operators relies on the dimensionality of the stream and results in the following increases of 4.78% for kNN-Entropy, 9.36% for kNN-SU, 12.29% for NB-Entropy and 13.98% for NB-SU. Statistically, as shown in Figure 7.4, these increases are significant once the weighting schemes are worse than the original classifier with a 95% confidence level.

Similarly to the results obtained in processing time, the memory consumption rates obtained by the weighted classifiers also increase, as depicted in Figure 7.3. Due to the cost of maintaining entropy calculators and discretization structures for numeric features, such increases are also expected and result in an average ratio increment of 6.62% for kNN-Entropy, 6.78% for kNN-SU, 12.10% for NB-Entropy and 15.50% for NB-SU. Furthermore, the increasing rates are consistent across all scenarios, and Friedman and Nemenyi tests' results (presented in Figure 7.5) show that the original *kNN* and NB classifiers outperform the weighted versions in memory consumption.

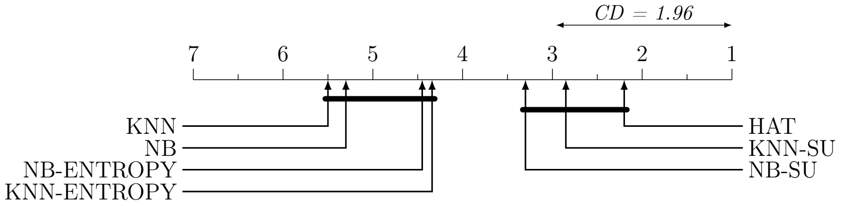


Figure 7.1: Critical differences chart for the accuracy obtained by weighted classifiers.

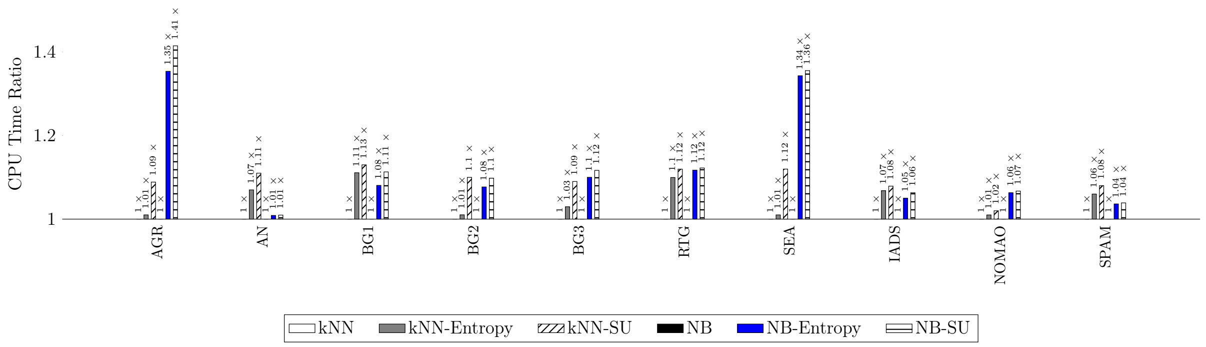


Figure 7.2: CPU Time ratio comparison between base learners and their Entropy and Symmetrical Uncertainty weighted versions.

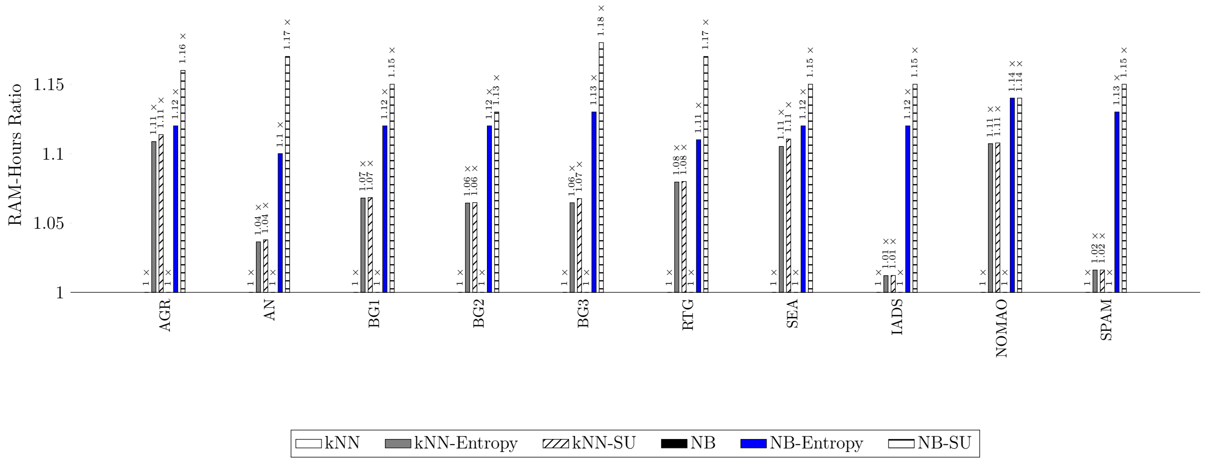
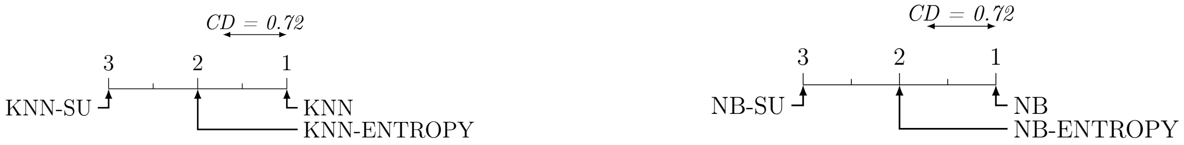


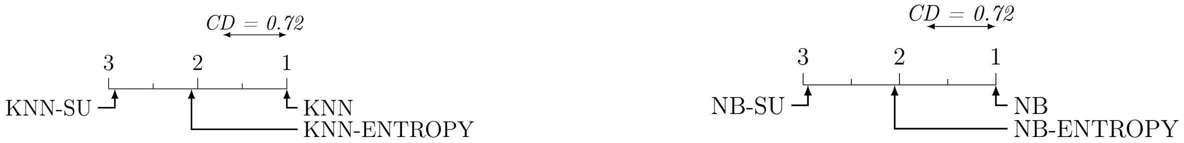
Figure 7.3: RAM-Hours ratio comparison between base learners and their Entropy and Symmetrical Uncertainty weighted versions.



(a) *kNN*

(b) NB

Figure 7.4: Critical differences chart for the processing time (CPU Time) obtained by weighted classifiers.



(a) *kNN*

(b) NB

Figure 7.5: Critical differences chart for the memory consumption (RAM-Hours) obtained by weighted classifiers.

7.4 Improving Hoeffding Adaptive Tree with Custom Feature Weighted Leaves

The results obtained by the feature weighted versions Naive Bayes and *kNN* showed compelling accuracy gains when compared to their original versions. Nevertheless, both classifiers are still outperformed by a single Hoeffding Adaptive Tree (HAT). As presented in Section 5.5.3, HATs are adaptive trees that perform model prunes whenever a drift is detected in one of its internal nodes. At the leaves, a Naive Bayes Adaptive (NBAdaptive) prediction occurs, which is given by the best performing strategy: a simple majority class voting or a Naive Bayes classifier.

This section investigates the adoption of the proposed feature weighted classifiers at the leaves of HATs in replacement of the NBAdaptive strategy. Results obtained are presented in Table 7.2, where it is clear that adopting a weighting scheme is beneficial, since it provides better accuracy rates in all of the 10 experiments.

A comparison between the best performing leaf-weighted HAT to its original version presents a 0.02% to 12.06% gain and an average improvement of 2.91%, showing that even when working with few data, i.e., only instances that were traversed in the tree to each leaf, the proposed weighting schemes are beneficial in terms of accuracy. Figure 7.6 presents the Nemenyi test results for a comparison of different leaf prediction strategies, where all of the proposed schemes are superior to the original HAT.

Similarly to the results obtained by the weighted classifiers presented in the previous section, the proposed weighting scheme also impacts the processing time and memory usage of the custom HATs. Figures 7.7 and 7.9 depict comparisons between the original HAT and its leaf-weighted variants in CPU Time and RAM-Hours, respectively. In terms

Table 7.2: Prequential Accuracy (%) for different leaf prediction strategies in HAT.

Experiment	HAT	HAT *kNN* - Entropy	HAT *kNN* - SU	HAT NB - Entropy	HAT NB - SU
AGR	69.84	70.54	**81.90**	70.54	71.23
AN	84.74	86.43	88.24	87.28	**88.97**
BG1	91.76	92.37	94.42	**94.51**	**94.51**
BG2	78.92	82.87	**86.04**	79.71	81.29
BG3	64.45	68.53	**72.47**	65.74	65.10
RTG	56.64	57.47	58.16	57.21	**58.34**
SEA	78.80	79.59	79.62	81.16	**82.74**
IADS	83.90	94.55	**94.72**	84.72	85.08
NOMAO	92.67	94.81	**94.94**	92.69	93.24
SPAM	84.48	86.91	**92.62**	84.89	85.25

Bold-faced results represent the best overall values obtained.

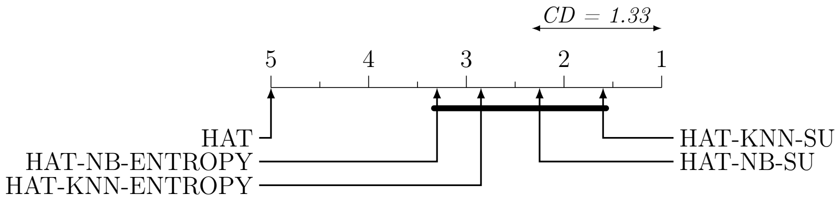


Figure 7.6: Critical differences chart for the accuracy rates of different leaf prediction strategies for HAT.

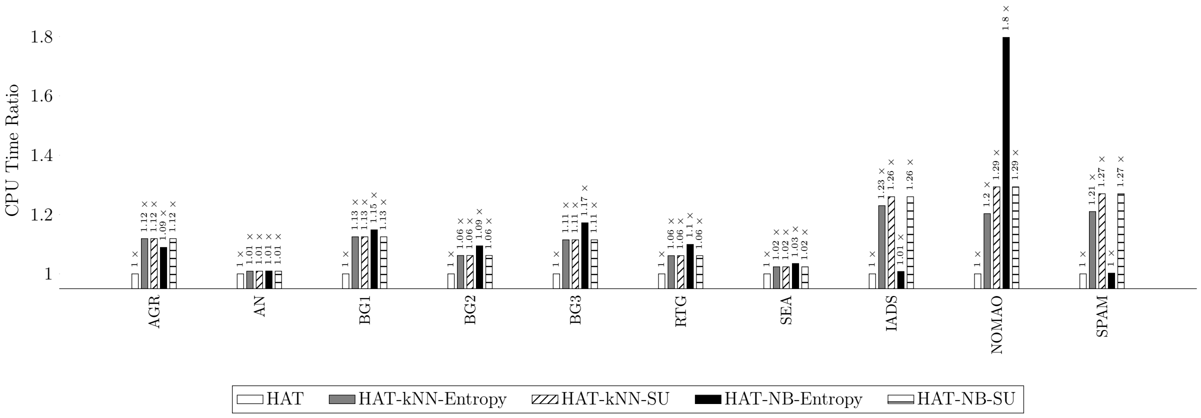


Figure 7.7: CPU Time (s) comparison between HAT and its variations using feature weighted custom leaves.

of average processing time, exchanging the original NBAdaptive leaf prediction strategy by a kNN-Entropy incurs in an increase of 11.57% and kNN-SU by 13.38%, while the weighted NB strategies results in the following increases: NB-Entropy of 14.55% and NB-SU of 12.44%. According to Friedman and Nemenyi tests, such increases are statistically significant, as shown in Figure 7.8. However, as discussed in Section 6.1, the computation of both scoring operators is bounded to the dimensionality, and since all experiments possess a fairly great dimensionality these increases can be considered acceptable in most scenarios given that the implementation is easily parallelized as the one here evaluated.

Memory-wise, the increases shown in Figure 7.9 obtained by adopting dynamically weighted leafs in HATs have the following average rates: 2.59% for NB-Entropy, 4.30% for NB-SU, 6.80% for kNN-Entropy and 8.90% for kNN-SU. Despite such increases, the critical differences chart presented in Figure 7.10 show that {HAT, HAT-NB-ENTROPY} ≻ {HAT-NB-SU} ≻ {HAT-KNN-ENTROPY, HAT-KNN-SU}. At this point, it becomes important to highlight that the higher memory usage expressed by *kNN* leafs is expected since, in contrast to the original NBAdaptive strategy, *kNN* is a lazy classifier that maintains a buffer of instances, instead of conditional probabilities.

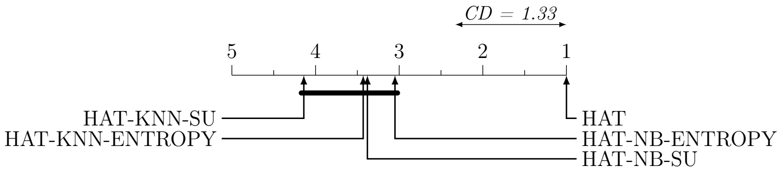


Figure 7.8: Critical differences chart for the processing time obtained by different leaf prediction strategies for HAT.

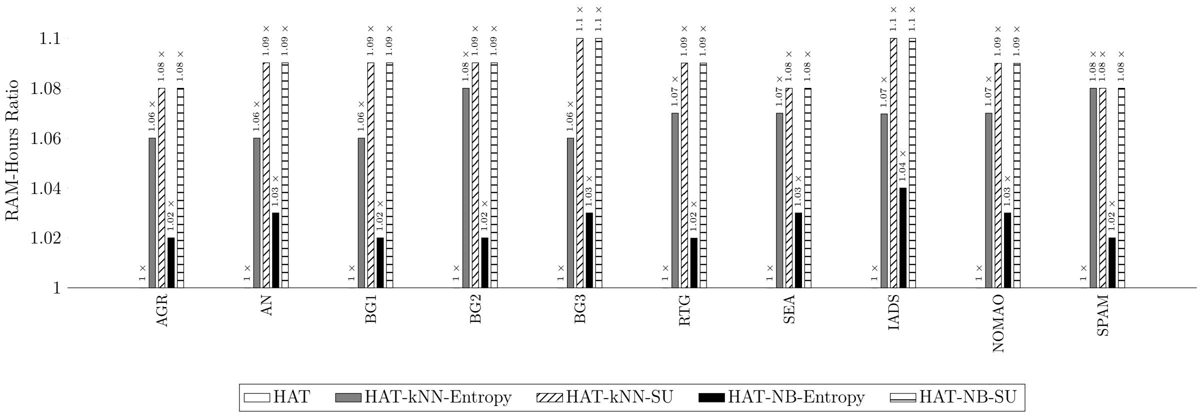


Figure 7.9: RAM-Hours (GB-Hour) comparison between HAT and its variations using feature weighted custom leaves.

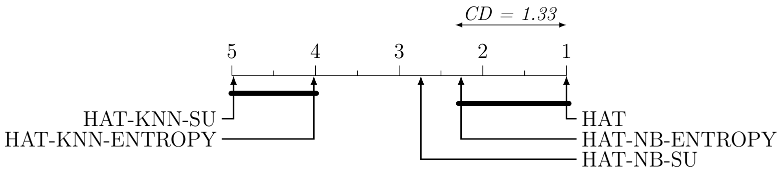


Figure 7.10: Critical differences chart for the memory consumption rates of different leaf prediction strategies for HAT.

7.5 Concluding Remarks

This chapter introduced dynamically weighted versions of the Naive Bayes and *kNN* classifiers where weights are given by Entropy and Symmetrical Uncertainty scoring operators. Validating the third hypothesis of this thesis, empirical evidence shows that the weighted versions present interesting prediction accuracy improvements when compared to the original classifiers in both synthetic and real-world datasets. Furthermore, these classifiers were investigated as leaves of Hoeffding Adaptive Trees and lead to interesting accuracy improvements to the currently best-performing decision tree for data streams. In spite of the accuracy gains, the weighting scheme comes at the expense of extra processing time and memory space, which are statistically significant, yet, still considered feasible in most scenarios.

The results presented in this chapter shed light on the importance of tracking the relevance of features throughout the processing of the stream. The next chapter introduces a dynamic filter for feature selection over data streams that has as its core the Symmetrical Uncertainty scoring operator.

Chapter 8

Merit-guided Dynamic Feature Selection

In Chapter 6, two dynamic scoring operators (Conditional Entropy and Symmetrical Uncertainty) based on Information Theory were introduced and later used in Chapter 7 as weighting factors during predictions. As a result, both *kNN* and Naive Bayes classifiers were extended and showed compelling accuracy improvements in several streams since weights were promptly updated given changes in the underlying concept of each scenario. Furthermore, the use of these dynamically weighted classifiers at the leaves of the Hoeffding Adaptive Tree was investigated, and this strategy also lead to sound results.

This chapter investigates the Symmetrical Uncertainty scoring operator as the core of a dynamic filter for feature selection over data streams. This operator was chosen over Entropy since it reduces the bias towards features with more distinct values and has shown better accuracy results in the feature weighting scope. First, Section 8.1 introduces the proposed filter, namely DynamIc SymmetriCal Uncertainty Selection for Streams (DISCUSS) (BARDDAL et al., 2019). The rationale behind DISCUSS is to (i) keep track of the relevance of each feature w.r.t. the class, (ii) eliminate irrelevant features, and (iii) discard redundant attributes. The proposed filter can be customized according to two different selection strategies, i.e., n Best and Thresholding, which are later introduced in Section 8.2. One of the major bottlenecks of DISCUSS is that redundancy computations must be performed in batch mode. To manage this bottleneck, Section 8.2.4 introduces a heuristic to decrease the complexity of the proposed method by adopting the Hoeffding Bound as a criterion for redundancy computation.

DISCUSS and its selection strategies are then evaluated in Section 8.3 to analyze the impact of their parameters and to verify their applicability comparing the results obtained with the ones from the original classifiers and the Hoeffding Adaptive Tree.

Despite the promising results obtained by DISCUSS, it still fails in some specific scenarios. To explain the limitations of DISCUSS, Section 8.5 describes when and why

DISCUSS fails in probability terms.

Finally, Section 8.6 concludes this chapter by discussing the main results obtained.

8.1 DynamIc SymmetriCal Uncertainty Selection for Streams (DISCUSS)

This section introduces a novel dynamic filter for feature selection from data streams. This filter has as its core the Symmetrical Uncertainty scoring operator, which is used to quantify how important a feature is w.r.t. class prediction and whether it is redundant to another peer attribute.

DISCUSS consists primarily of (i) updating the necessary entropies for Symmetrical Uncertainty computation to the class, (ii) a baseline comparison, (iii) a selection strategy, and (iv) a redundancy check. The central claim on proposing DISCUSS is that it will dynamically check the importance of features w.r.t. class prediction, while also identifying redundant ones. Given that, DISCUSS will continuously try to maximize a merit function that aims the maximization of feature relevance while decreasing feature redundancy. As a result, classification models should be learned within smaller dimensionalities, also prospectively resulting in smaller processing times and memory consumption rates.

Algorithm 3[1] presents the pseudocode for the DynamIc SymmetriCal Uncertainty Selection for Streams algorithm (DISCUSS). The proposed filter receives as input a data stream $\mathcal{S}$ and the buffer size w. Given $\mathcal{S}$, DISCUSS retrieves its first w instances (lines 25-30) that are used to increment the necessary entropies for Symmetrical Uncertainty computation and initial selection, which is performed in line 30, and provides a selected subset of features $\mathcal{X}'$ and a baseline attribute $X_{baseline}$ that will be used for comparisons. In practice, a new feature selection will occur if a feature that had a Symmetrical Uncertainty w.r.t the class below the baseline has its metric now above it, or vice-versa. After the burnout window, DISCUSS operates over its buffer W as a sliding window (lines 5-24). Given arriving instances and the oldest ones which are being dropped as $\mathcal{S}$ progresses, its corresponding entropies and Symmetrical Uncertainty values are updated (lines 6-10).

After SU computation, a comparison of these updated values is performed to verify whether a feature that was irrelevant has now surpassed the baseline feature $X_{baseline}$ or a feature that was relevant had its discriminant power decreased and is now below $X_{baseline}$

[1] DISCUSS is based on a baseline to verify whether a new feature selection is required. This algorithm presents a feature as the baseline, yet, the baseline can also be a threshold value, as later introduced in Section 8.2.2.

Algorithm 3: Pseudocode of the proposed filter.

```
     input : a data stream $\mathcal{S}$ that provides instances $(\vec{x}^t, y^t)$ and a buffer size $w$.
 [1] Let $W \leftarrow \emptyset$ be a queue that maintains a sliding window with length $w$;
 [2] Let $expert$ be a classifier;
 [3] $\mathcal{X}' \leftarrow \emptyset$ ;                              /* Subset of selected features */
 [4] $X_{baseline} \leftarrow NULL$ ;                                   /* Baseline feature */
 [5] foreach $(\vec{x}^t, y^t) \in \mathcal{S}$ do
 [6]     if $|W| = w$ then
 [7]         $W \leftarrow W \cup \{(\vec{x}^t, y^t)\}$;
 [8]         Increment entropies $H(Y)$, $H(X_i)$ and $H(Y|X_i)$ with $(\vec{x}^t, y^t)$;
 [9]         Dequeue the oldest instance $(\vec{x}^{t-w-1}, y^{t-w-1})$ from $W$;
[10]         Decrement entropies $H(Y)$, $H(X_i)$ and $H(Y|X_i)$ given $(\vec{x}^{t-w-1}, y^{t-w-1})$;
[11]         $flag \leftarrow FALSE$;
[12]         foreach $X_i \in \mathcal{X} \setminus \{X_{baseline}\}$ do
[13]             if $(SU(X_i, Y) > SU(X_{baseline}, Y)$ and $X_i \notin \mathcal{X}')$ or
                   $(SU(X_i, Y) < SU(X_{baseline}, Y)$ and $X_i \in \mathcal{X}')$ then
                     /* This condition is satisfied when a feature either (i) becomes
                        relevant and surpasses $X_{baseline}$, or (ii) turns irrelevant and
                        has its SU w.r.t. the class now below $X_{baseline}$'s          */
[14]                 $flag \leftarrow TRUE$;
[15]                 break;
[16]         if $flag$ then
[17]             $(\mathcal{X}', X_{baseline}) \leftarrow selectFeatures(\mathcal{X})$ ; /* Selects new features and defines a
                   new baseline feature based on one of the selection strategies
                   described in Section 8.2 */
[18]             expert.reset() ;                                       /* Resets the learner */
                 /* Starts the learning of a new model with the instances in buffer
                    given the newly selected attributes                               */
[19]             foreach $i' \in W$ do
[20]                 $i' = extract(i', \mathcal{X}')$;
[21]                 $expert.train(i')$;
[22]         else
[23]             $i' \leftarrow extract(i', \mathcal{X}')$;
[24]             $expert.train(i')$;
[25]     else
             /* Condition met during the first $w$ instances obtained from $\mathcal{S}$.        */
[26]         $W \leftarrow W \cup \{(\vec{x}^t, y^t)\}$;
[27]         Update entropies $H(X_i)$, $H(Y)$ and $H(Y|X_i)$ given $(\vec{x}^t, y^t)$;
[28]         $expert.train((\vec{x}^t, y^t))$;
[29]         if $|W| = w$ then
                 /* Selects the first subset of relevant features given the instances
                    stored in $W$ and also sets a baseline $\mathcal{X}'$ that will be used during
                    the main loop.                                                    */
[30]             $(\mathcal{X}', X_{baseline}) \leftarrow selectFeatures(\mathcal{X})$;
```

(lines 12-15). If such condition holds, a new feature selection is triggered (lines 17-21), or otherwise, the classification model is updated using the arriving instance (lines 23-24). This selection is configurable, and such strategies are next discussed in Section 8.2. In addition to the selection of new features, the underlying classification system is reset and re-trained with the instances stored in the buffer using only the selected attributes (lines

19-21).

8.2 Selection Strategies

This section introduces two different selection strategies for DISCUSS. Both are simple and aim at selecting relevant features while ignoring both redundant and irrelevant ones. The first is an iterative approach that selects the n best-ranked attributes with the guarantee that they are not redundant to one another. The second strategy is similar, where features are selected if they possess a Symmetrical Uncertainty w.r.t. the class above a threshold θ and if they are not redundant to one another. One of the pitfalls of this second approach is that picking a proper θ is not trivial since it is domain dependent. With this limitation in mind, an automatic thresholding technique based on the Hoeffding bound is offered and evaluated.

8.2.1 n Best

The first proposed selection strategy, detailed in Algorithm 4, iteratively selects the n best-ranked features with the guarantee that none of these are redundant to one another. This strategy receives as input a single parameter, which is the number n of attributes to be selected and as an outcome, it outputs both the selected subset of features $\mathcal{X}'$ and the baseline attribute, which is used for comparisons.

The rationale behind the comparison against the baseline attribute is that feature drifts occur if an attribute that was in a lower position in the ranking has now surpassed the baseline or the contrary, i.e., when an attribute that was above the baseline is not anymore, and in these cases, a new selection is deemed necessary.

First, the list of features is sorted in descending order given their Symmetrical Uncertainty w.r.t the class. In practice, during the first time the features will be sorted, QuickSort (HOARE, 1961) is used since it has $\mathcal{O}(d \log_2 d)$ cost, yet, during posterior executions, Insertion Sort is preferred as the list of features is mostly sorted already, resulting in a $\mathcal{O}(dk)$, where each input element is no more than k positions from its sorted position. Next, the selected subset of attributes $\mathcal{X}'$ is initialized as an empty set, which will be iteratively incremented given the feature ranking until the n best-ranked attributes are selected or there are not any candidate features left in $\mathcal{X}$. The loop described by lines 3–13 retrieves at each iteration the best-ranked attribute, which is then evaluated against the attributes already selected and stored in $\mathcal{X}'$ for a redundancy check following the concepts of Predominant Correlation and Predominant Feature.

Definition 20. (Predominant Correlation) The correlation between $X_i \in \mathcal{X}$ and the class Y is **predominant** *iff* $SU(X_i, Y) \geq \theta$ and $\forall X_j \in \mathcal{X}, (i \neq j)$ there exists no X_j such that $SU(X_i, X_j) \geq SU(X_i, Y)$ (YU; LIU, 2003).

Definition 21. (Predominant Feature) A feature X_i is predominant to the class Y *iff* its correlation is predominant or it *can become after removing its redundant peers* (YU; LIU, 2003).

Given the previous definitions, a feature is good if it is predominant in predicting the class, and thus, a feature selection algorithm should identify all predominant features and ignore the rest. Also following the previous definitions, a redundancy is flagged between two attributes X_i and X_j if $SU(X_i, X_j) > SU(X_i, Y)$ (see line 9). In other words, a feature is deemed as redundant if it is correlated to an already selected attribute to a bigger extent compared to how it is to the class.

Clearly, performing redundancy checks is the main drawback of this proposal, since computing $SU(X_i, X_j)$ requires, by definition, $H(X_i)$, $H(X_j)$ and $IG(X_i, X_j)$, where the last component must be computed in batch mode.

Finally, this selection strategy returns both the selected subset of features and the baseline attribute, which is used for comparisons in Algorithm 3. The rationale behind the comparison against the baseline attribute is that feature drifts occur if an attribute that was in a lower position in the ranking has now surpassed the baseline or the contrary,

Algorithm 4: Selection scheme for the n best features.

```
input  : the feature set $\mathcal{X}$ and the amount of features to be selected $n$
output : the selected features $\mathcal{X}$' and the baseline feature $X_{baseline}$
[1] Sort $\mathcal{X}$ in descending order of $SU(\cdot, Y)$;
[2] Let $\mathcal{X}' \leftarrow \emptyset$ be the set of selected features;
[3] while $|\mathcal{X}'| < n$ and $\mathcal{X} \neq \emptyset$ do
[4]     $candidate \leftarrow head(\mathcal{X})$;        /* Removes the first item from $\mathcal{X}$ */
[5]     $redundant \leftarrow FALSE$;
[6]     $\gamma \leftarrow SU(candidate, Y)$;
[7]     foreach $X_i \in \mathcal{X}'$ do
[8]         $\xi \leftarrow$ Compute $SU(X_i, candidate)$ /* Batch computation */
[9]         if $\xi > \gamma$ then
                /* This holds if the candidate feature is so correlated with another
                   selected feature that this surpasses its correlation to the class,
                   i.e., it is redundant. */
[10]            $redundant \leftarrow TRUE$;
[11]            break;
[12]    if not $redundant$ then
[13]        $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{candidate\}$;
[14] $X_{baseline} \leftarrow peekLast(\mathcal{X}')$ ;        /* Peeks the last item in $\mathcal{X}'$. */
[15] return $(\mathcal{X}', X_{baseline})$;
```

i.e., when an attribute that was above the baseline is not anymore, and in these cases, a new selection is deemed necessary.

At a first sight, this selection strategy seems to be computationally prohibitive since it requires d^2 redundancy computations, however, it is important to emphasize and clarify that in practice $|\mathcal{X}'|$ is bounded in $[0; n]$. Since the goal is to provide a fair dimensionality reduction, only values $n \ll d$ should be used, and thus, this proposal shall be considered feasible. Therefore, the complexity of this selection strategy is $\mathcal{O}(nd)$.

8.2.2 Thresholding

The thresholding strategy is quite similar to the previous one. It assumes the same heuristics to select attributes with the difference that there is no baseline feature since the baseline is now a relevance threshold θ. Algorithm 5 presents the pseudocode for this selection strategy. It receives as input the relevance threshold θ. Similarly to the previous scheme, the attribute set $\mathcal{X}$ is sorted in descending Symmetrical Uncertainty order. In this scheme, sorting is necessary[2] to avoid comparisons with attributes that are guaranteed to possess SU values below θ. While candidate features X_i drawn from $\mathcal{X}$ have their Symmetrical Uncertainty values $SU(X_i, Y)$ above θ, they are evaluated for redundancy against all the previously selected attributes in $\mathcal{X}'$. The redundancy verification procedure is the same as the one used in the n Best strategy, and thus, will not be detailed here.

8.2.3 Automatic Thresholding via Hoeffding Bound

The major drawback of the thresholding selection strategy is picking an appropriate threshold value θ. Intuitively, the optimal threshold value for each data domain is different, and thus, there is no evidence that there is a one-fits-all standard value to be followed. Still, a last investigation is necessary and aims at verifying how well DISCUSS behaves when working with a fixed threshold θ_{auto}. The principle behind this scheme is simple and assumes that a feature X_i is relevant if its Symmetrical Uncertainty w.r.t. the class $SU(X_i, Y)$ is significantly different from 0, which is the minimum possible value that depicts complete uncorrelation. To perform such verification, the Hoeffding bound (Definition 17) is once again recollected and used similarly as in Hoeffding Trees.

Definition 22. (Automatic Thresholding via Hoeffding Bound) A feature X_i will be

[2] In practice, the main loop depicted by lines 4–14 of the algorithm could evaluate all features with a $\mathcal{O}(d)$ cost, however, sorting using Quicksort ($\mathcal{O}(d \log_2 d)$) and then drawing attributes is faster since it requires fewer redundancy checks.

Algorithm 5: Selection scheme for the the thresholding scheme.

```
input  : the feature set $\mathcal{X}$ and the relevance threshold $\theta$
output : the selected features $\mathcal{X}'$
Sort $\mathcal{X}$ in descending order of $SU(\cdot, Y)$;
Let $\mathcal{X}' \leftarrow \emptyset$ be the set of selected features;
$candidate \leftarrow head(\mathcal{X})$;
while $SU(candidate, Y) > \theta$ do
    $redundant \leftarrow FALSE$;
    $\gamma \leftarrow SU(candidate, Y)$;
    foreach $X_i \in \mathcal{X}'$ do
        $\xi \leftarrow$ Compute $SU(X_i, candidate)$ /* Batch computation */
        if $\xi > \gamma$ then
            /* This holds if the candidate feature is so correlated with another
               selected feature that this surpasses its correlation to the class,
               i.e., it is redundant. */
            $redundant \leftarrow TRUE$;
            break;
    if not $redundant$ then
        $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{candidate\}$;
    $candidate \leftarrow head(\mathcal{X})$;
return $(\mathcal{X}')$;
```

deemed relevant if Equation 8.1 holds,

$$\begin{aligned} SU(X_i, Y) &> \theta_{auto} \\ SU(X_i, Y) &> \epsilon \\ SU(X_i, Y) &> \sqrt{\frac{R^2 \ln\left(^1/_\delta\right)}{2n}} \end{aligned} \tag{8.1}$$

where $\theta_{auto} = \epsilon$ is the Hoeffding Bound earlier described in Definition 17, δ is the significance level, $R = 1$ is the range of Symmetrical Uncertainty and n is the window size.

Following the latter definition, if one uses $\theta = \theta_{auto} = \epsilon$, it will be guaranteed that all selected features X_i have a correlation to the class that is statistically significant and non-zero. Table 8.1 depicts different values of Hoeffding Bound obtained by varying the parameters δ and n. One of the drawbacks of using these threshold values is that they are fixed over time. This will fatally result in good behaviors for some scenarios and bad for others.

8.2.4 A Heuristic to Decrease the Number of Redundancy Computations

The proposed redundancy verification presented in Algorithm 3 verifies if each candidate feature is redundant to any of the attributes already selected. Computing redundancy values is one of the bottlenecks of the proposed method since part of its entropies that compose the Information Gain must be computed in batch mode when

Table 8.1: Hoeffding bound values (ϵ) obtained by varying n and δ parameters.

n	$(1-\delta)$	$\theta_{auto} = \epsilon$
100	0.95	0.0106
	0.99	0.0047
200	0.95	0.0075
	0.99	0.0033
300	0.95	0.0061
	0.99	0.0027
400	0.95	0.0053
	0.99	0.0023
500	0.95	0.0047
	0.99	0.0021

Algorithm 6: Pseudocode for the heuristic redundancy check in DISCUSS.

```
foreach $X_i \in \mathcal{X}'$ do
    $\psi \leftarrow SU(candidate, Y)$;
    if $|\gamma - \psi| < \epsilon$ then
        $\xi \leftarrow$ Compute $SU(X_i, candidate)$ /* Batch computation */
        if $\xi > \gamma$ then
            $redundant \leftarrow TRUE$;
            break;
```

necessary. In this section, a heuristic to reduce the number of redundancy computations is proposed. This heuristic is straightforward and also adopts the Hoeffding bound to verify whether a redundancy computation must be performed between two attributes.

The following heuristic works under the assumption that if two attributes are redundant to one another, they possess similar Symmetrical Uncertainty values w.r.t. the class.

Definition 23. Given two attributes $X_i, X_j \in \mathcal{X}$, a redundancy computation between such features must occur *iff* Equation 8.2 holds, where ϵ is the Hoeffding bound.

$$|SU(X_i, Y) - SU(X_j, Y)| < \epsilon \tag{8.2}$$

In other words, Definition 23 states that a redundancy verification between two attributes must occur when the difference between the Symmetrical Uncertainties of these attributes w.r.t. the class is less than the Hoeffding Bound, meaning that they are not statistically different.

In practice, to adopt this heuristic in DISCUSS, the procedure depicted in Algorithm 6 should replace lines 7–11 in both Algorithms 4 and 5.

8.3 Evaluation

This section evaluates the two proposed selection strategies of DISCUSS by varying their main parameters. The results are obtained by applying DISCUSS to kNN, Naive Bayes and Very Fast Decision Tree (VFDT) classifiers. The results obtained are then benchmarked against the original learners and against the Hoeffding Adaptive Tree, which has been used so far as a baseline in this project. Accuracy results obtained for the heuristic redundancy verification are omitted since they are, on average, 0.5% lower than the ones obtained here, whereas processing time and memory usage metrics are further discussed in Section 8.3.4.

8.3.1 n Best

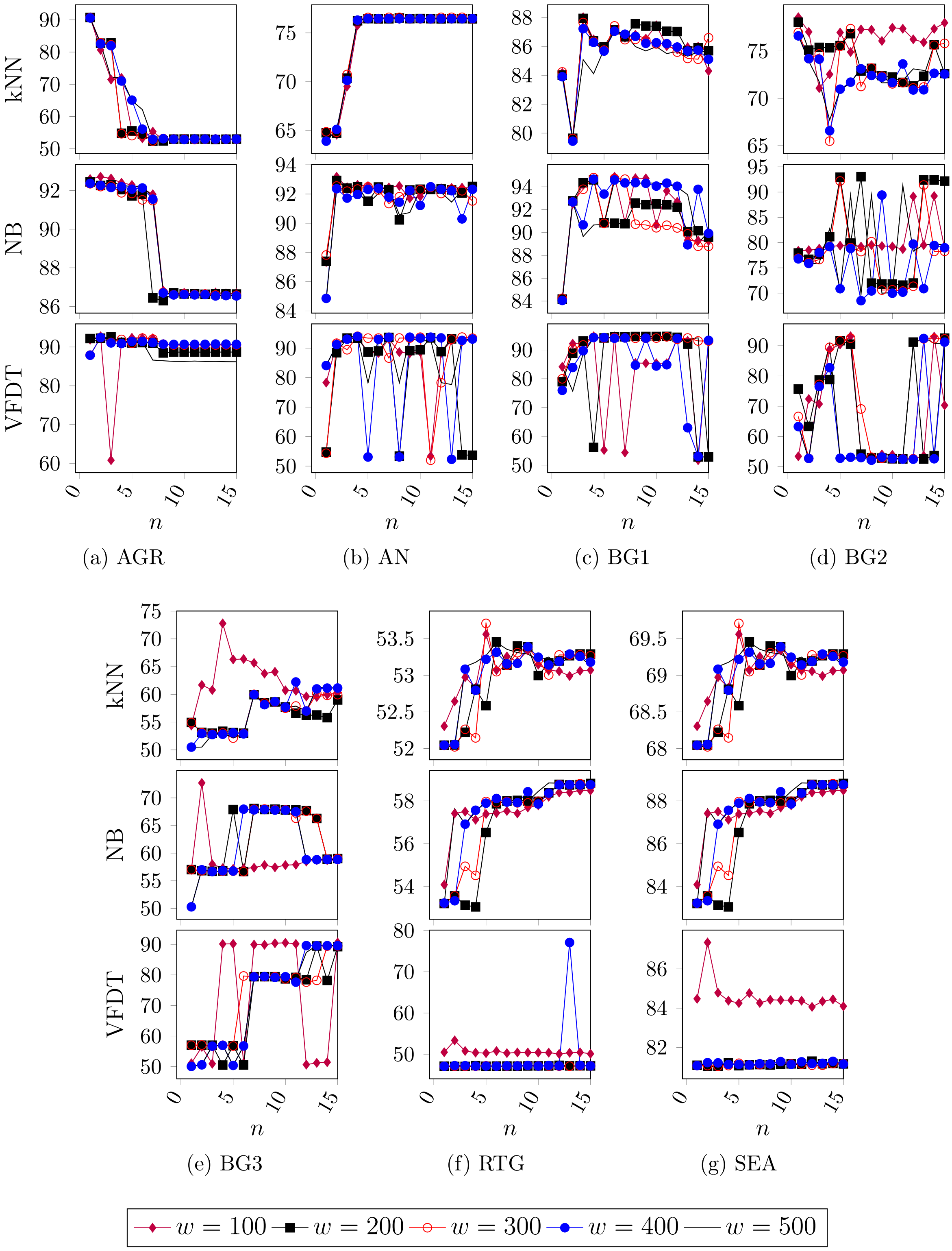
The only parameter for the n Best selection strategy is the number of attributes to be selected n. This section evaluates the impact of n given the complex guidelines provided in Chapter 4, that is, including accuracy, processing time, memory usage and selection accuracy. Results for real-world datasets were omitted from this section since they do not allow the computation of classifier independent metrics, whereas results for the gradual drifted experiments were also omitted since they present the same behavior as the abrupt ones. Results for real-world datasets will be later evaluated in Section 8.3.3.

Figure 8.1 presents the average prequential accuracy obtained during synthetic experiments using different classifiers, window sizes, and values of n. In almost half of the experiments, i.e., AN, BG1, BG3, it follows that bigger values of n lead to increases in accuracy. Corroborating with the claims presented in Section 3.5, the results presented here show that indeed different classifiers act differently given the same set of attributes. An example of that is the AN experiment, where the kNN classifier obtains accuracies up to 75%, while Naive Bayes obtains 90%+ using the same attributes. Also, there are some experiments which are interestingly odd, such as the RTG, where a single experiment presents an accuracy increase of approximately 20% compared to the rest (Figure 8.1g) or the AGR experiment, where VFDT has it's accuracy fall in 30% (Figure 8.1a). These results are expected since our filter selects features independently from the others and does not evaluate subsets of complementary features, causing classifiers to underfit.

Figures 8.2, 8.3 and 8.4 depict the Selection Accuracy (SA), Relevant Selection Recall (RSR) and Complement of Unnecessary Complexity Penalty (CUCP) results, respectively. Again, the results presented in Figure 8.2 are inconclusive, showing that with the increase of n, the overall selection accuracy can both increase or decrease, depending on the stream. In order to facilitate the understanding of these results, selection accuracy is split into RSR and CUCP. The RSR results (Figure 8.3) clearly enlighten that with the rapid increase of n, RSR also increases and swiftly reaches a plateau, meaning that small values of n allow the selection of the desired relevant attributes. This is expected since there are few relevant attributes and they figure amongst the best ones for each stream, and thus, were correctly selected. Conversely, results in Figure 8.4 show that with the increase of n, CUCP drops, meaning that more irrelevant attributes were selected incorrectly. Combining the results obtained in terms of accuracy, SA, RSR and CUCP, there does not seem to be a direct correlation showing that maximizing a single metric, e.g., SA; overall accuracy would be improved.

Furthermore, it is important to highlight that different window sizes barely impact the SA, RSR and CUCP results obtained, showing that the Symmetrical Uncertainty computation is robust, as earlier stated in Section 6.3.

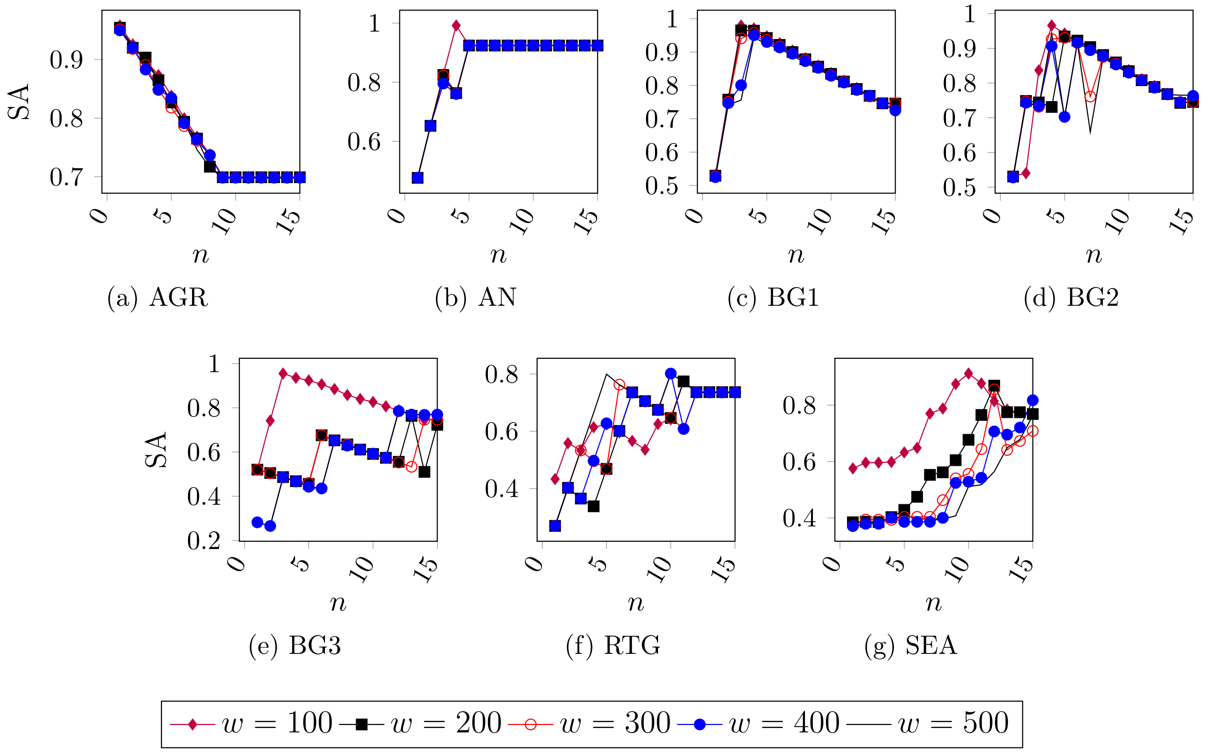
Figures 8.5 and 8.6 present the results obtained for processing time and memory usage. In both cases, these metrics slightly increase with the increment of n, models become more complex and more computationally intensive. Nevertheless, the greatest increases come from the use of different window sizes, which lead to important differences among the executions of DISCUSS. Given these results, it is possible to state that smaller values of w are preferred, e.g., $w = 100$ or $w = 200$, since they perform just as well as bigger windows, whilst requiring less computational resources.



(a) AGR (b) AN (c) BG1 (d) BG2

(e) BG3 (f) RTG (g) SEA

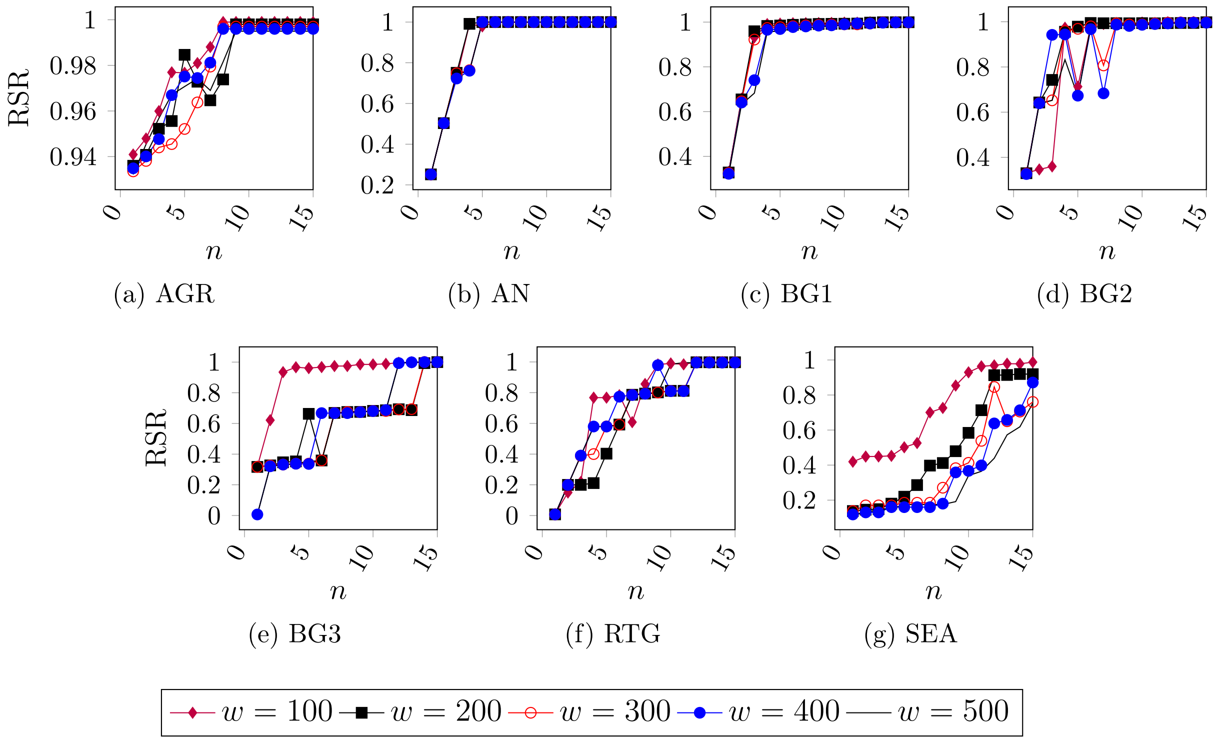
Figure 8.1: Prequential Accuracy (%) obtained by varying the n parameter.



(a) AGR (b) AN (c) BG1 (d) BG2

(e) BG3 (f) RTG (g) SEA

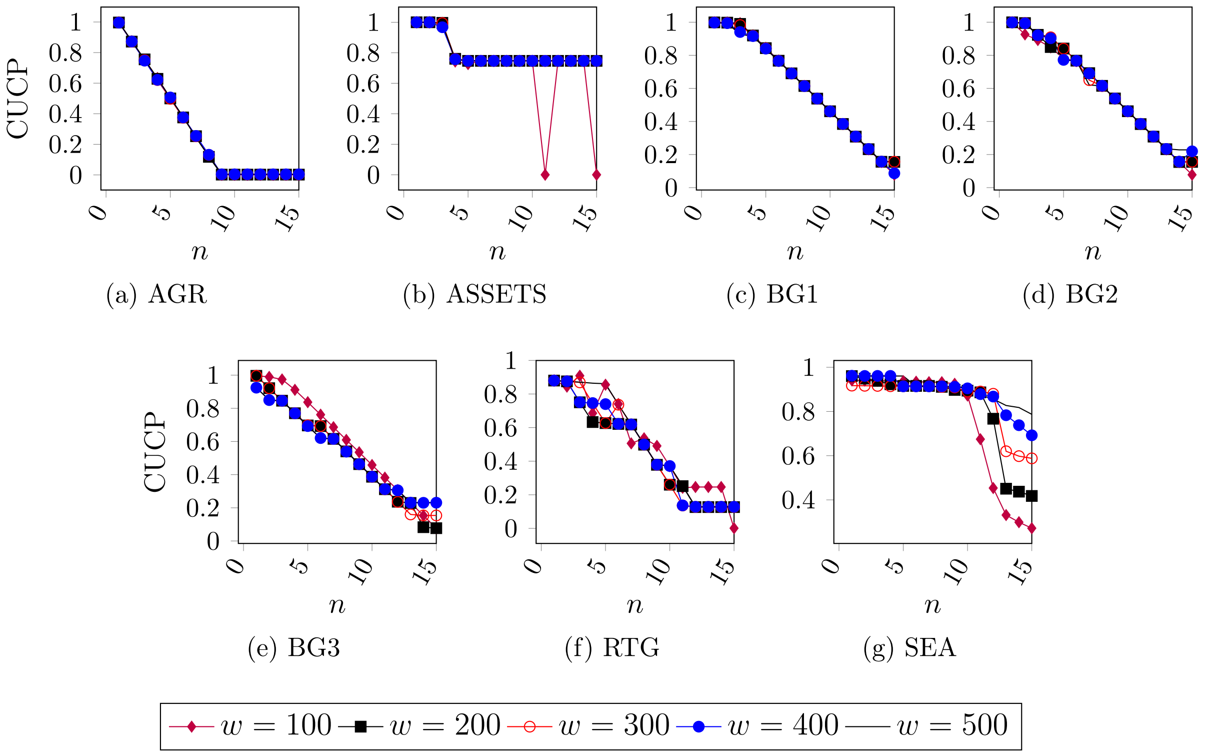
Figure 8.2: Selection Accuracy obtained by varying the n parameter.



(a) AGR (b) AN (c) BG1 (d) BG2

(e) BG3 (f) RTG (g) SEA

Figure 8.3: Relevant Selection Recall (RSR) obtained by varying the n parameter.



(a) AGR

(b) ASSETS

(c) BG1

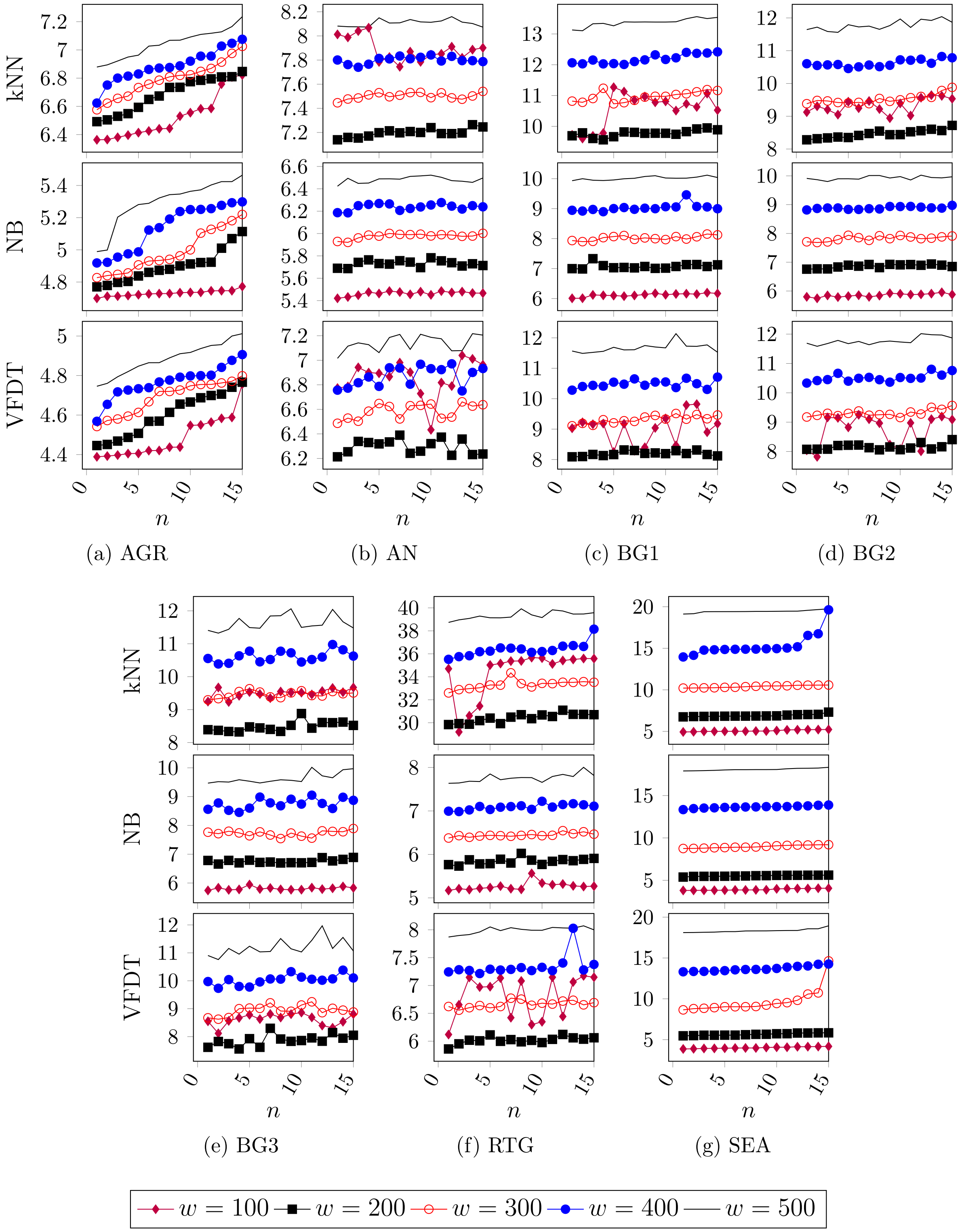
(d) BG2

(e) BG3

(f) RTG

(g) SEA

Figure 8.4: Complement of Unnecessary Complexity Penalty (CUCP) obtained by varying the n parameter.



(a) AGR

(b) AN

(c) BG1

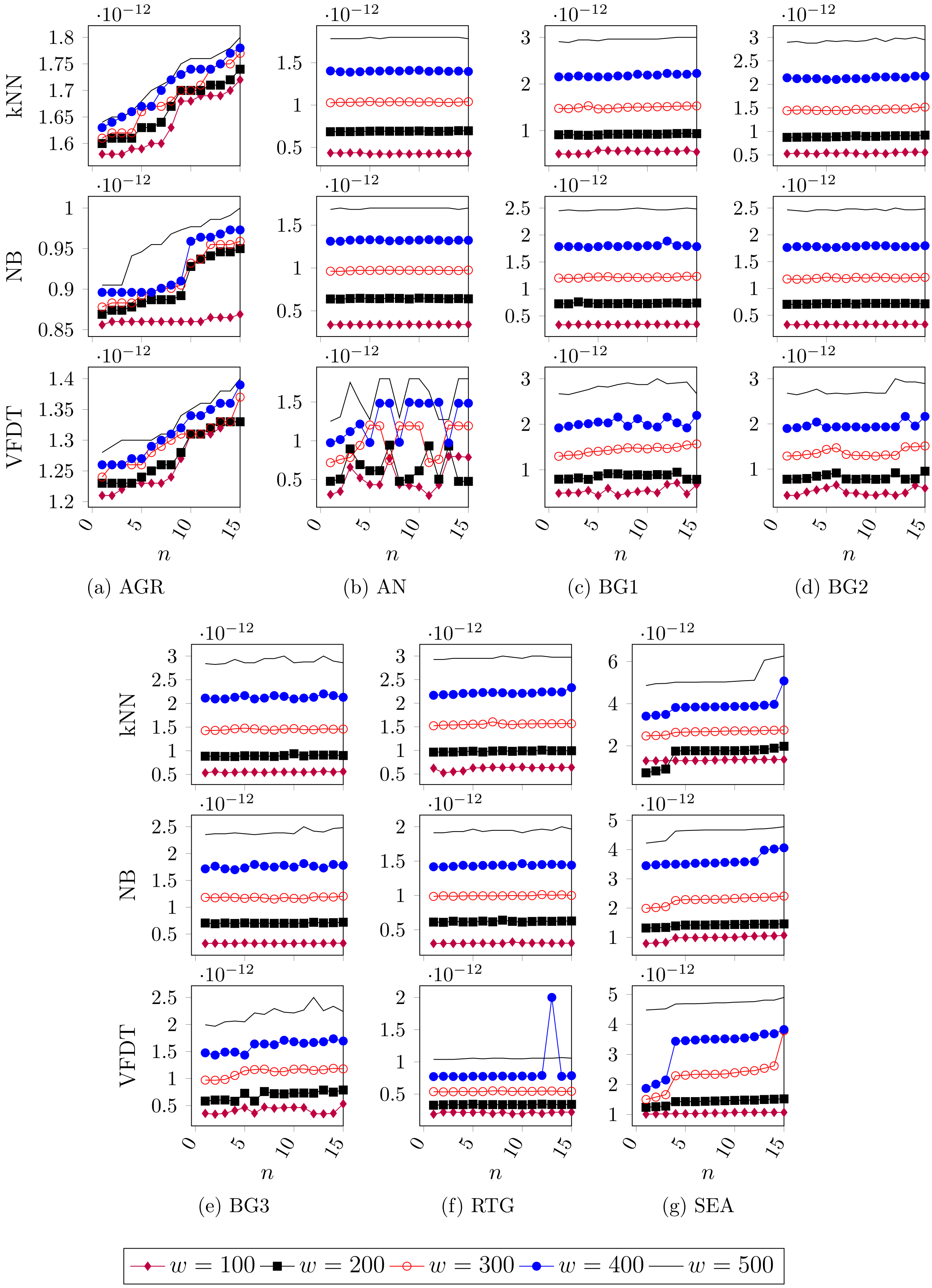
(d) BG2

(e) BG3

(f) RTG

(g) SEA

Figure 8.5: CPU Time (s) obtained by varying the n parameter.



(a) AGR

(b) AN

(c) BG1

(d) BG2

(e) BG3

(f) RTG

(g) SEA

Figure 8.6: RAM-Hours (GB-Hour) obtained by varying the n parameter.

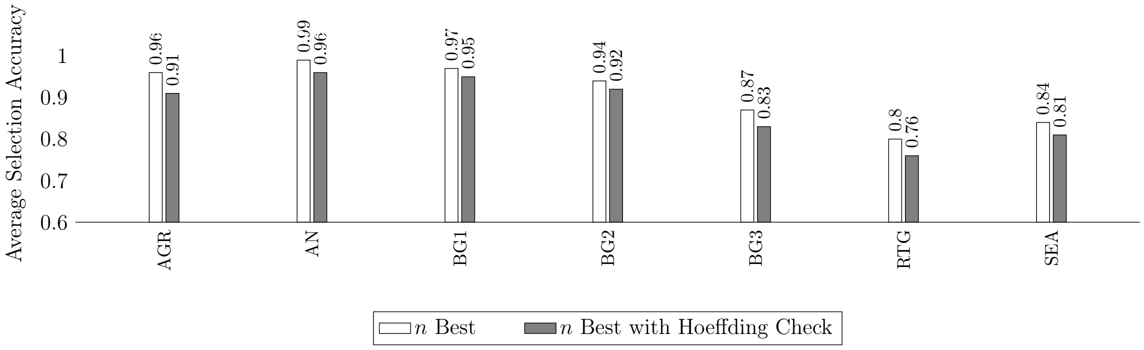


Figure 8.7: Comparison of the impact on Selection Accuracy by using the Hoeffding Bound to trigger redundancy checks.

Finally, Figure 8.7 presents a comparison between the average Selection Accuracy (SA) obtained by the original n Best strategy and the same strategy using the Hoeffding bound-based heuristic for triggering redundancy checks. Results show that the Hoeffding versions obtained slightly lower results in all synthetic experiments. In practice, these results show that the Hoeffding heuristic allows redundant features to be selected, thus leading to lower SA results. The impact of this Hoeffding heuristic in processing time and memory space is later discussed in Section 8.3.4.

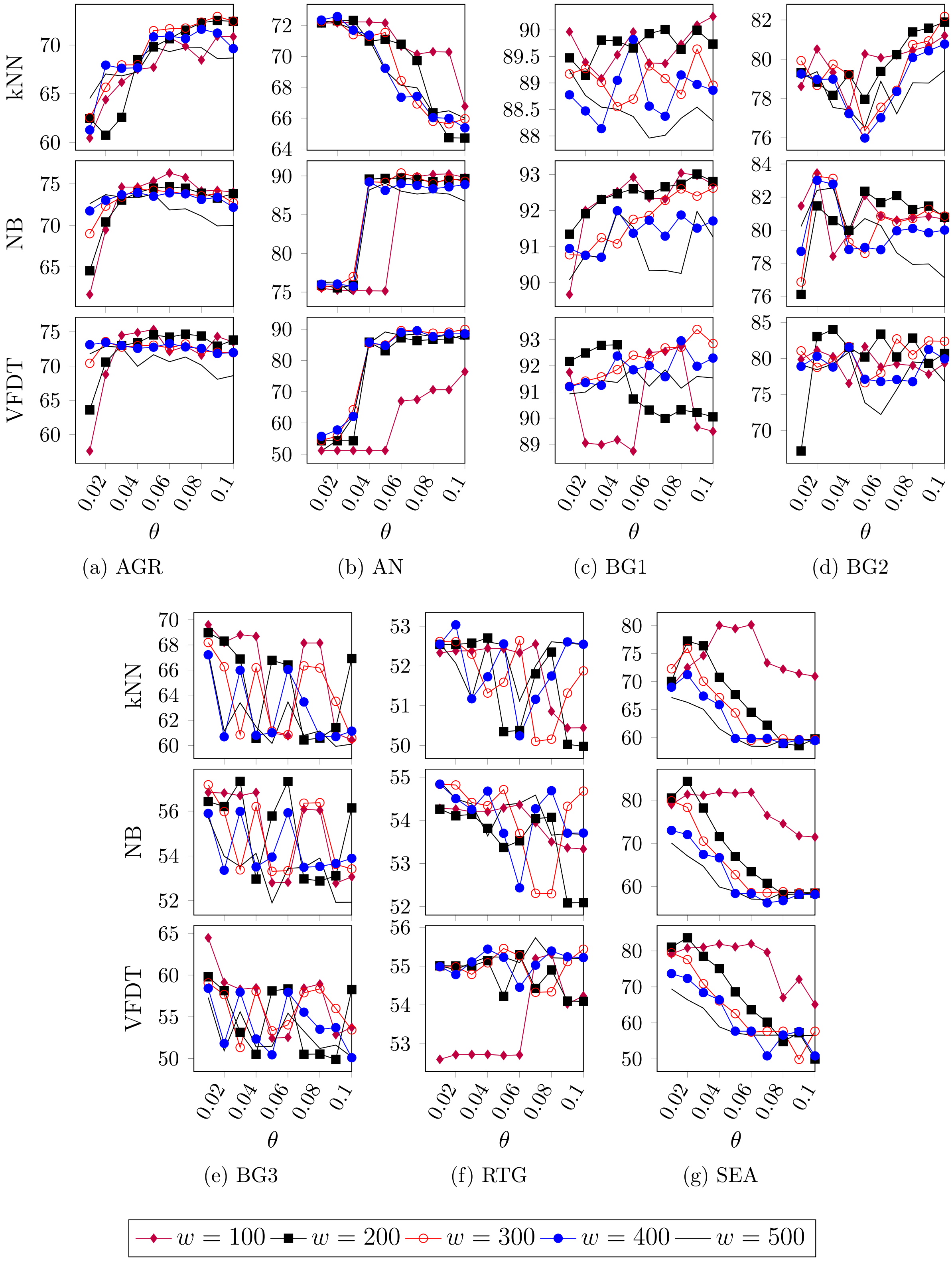
8.3.2 Thresholding

Analogously to the n best strategy, the thresholding strategy also relies on a single parameter: the relevance threshold θ. As the analysis performed in the previous section, the evaluation of the thresholding results should also encompass accuracy, processing time, memory usage and selection accuracy results.

Figure 8.8 presents the average prequential accuracy results obtained by varying base learners, window sizes and the relevance threshold θ. At a first glance, it is possible to notice that there is no deterministic behavior such as "increase θ to obtain greater accuracy rates", since in some experiments, as for instance AGR, the overall accuracy increases, while in others, such as the SEA, the accuracy decreases with the increment of θ. Following the same trend presented in the previous section, it becomes once again clear that different classifiers obtain different accuracy rates using the same attributes, such as the AN experiment, where the kNN classifier obtained approximately 72% of accuracy while Naive Bayes achieves results above 90%. Also, the relevance threshold can impact different classifiers in dissimilar fashions. An example of that is the AN experiment (Figure 8.8b), where with the increase of θ both Naive Bayes and Very Fast Decision Tree classifiers are improved, while kNN is jeopardized.

Figures 8.9, 8.10 and 8.11 depict the Selection Accuracy (SA), Relevant Selection Recall (RSR) and Complement of Unnecessary Complexity Penalty (CUCP) results obtained by varying the same parameters. With the exception of the BG1 experiment (see kNN results in Figure 8.8c), there seems to exist a decreasing tendency of SA with higher values of θ. Focusing on the RSR results, it becomes clear that with the increase of the relevance threshold, the RSR results decrease dramatically. These results are expected since higher values of threshold difficult the selection procedure, where attributes, even if relevant, are not selected if θ is inappropriately picked. A similar rationale can be applied when analyzing the CUCP results, depicted in Figure 8.11. In this case, the increase of the relevance threshold θ leads to less irrelevant attributes being selected. Again, these results are predicted since smaller values of θ allow lowly discriminative attributes to be selected, whereas with θ's increase, such attributes would be ignored.

Finally, Figures 8.12 and 8.13 show the processing time and memory usage results obtained. The results are the opposite compared to the ones obtained in the n Best selection strategy, where both CPU Time and RAM-Hours decrease with the increment of θ. As described earlier, the augment of θ leads to fewer attributes being selected, which makes model simpler (memory-wise) and faster to compute. In addition to that, the same characteristic that bigger sliding windows lead to higher results is still valid, meaning that buffering more instances is more costly in both memory space and processing time.



(a) AGR

(b) AN

(c) BG1

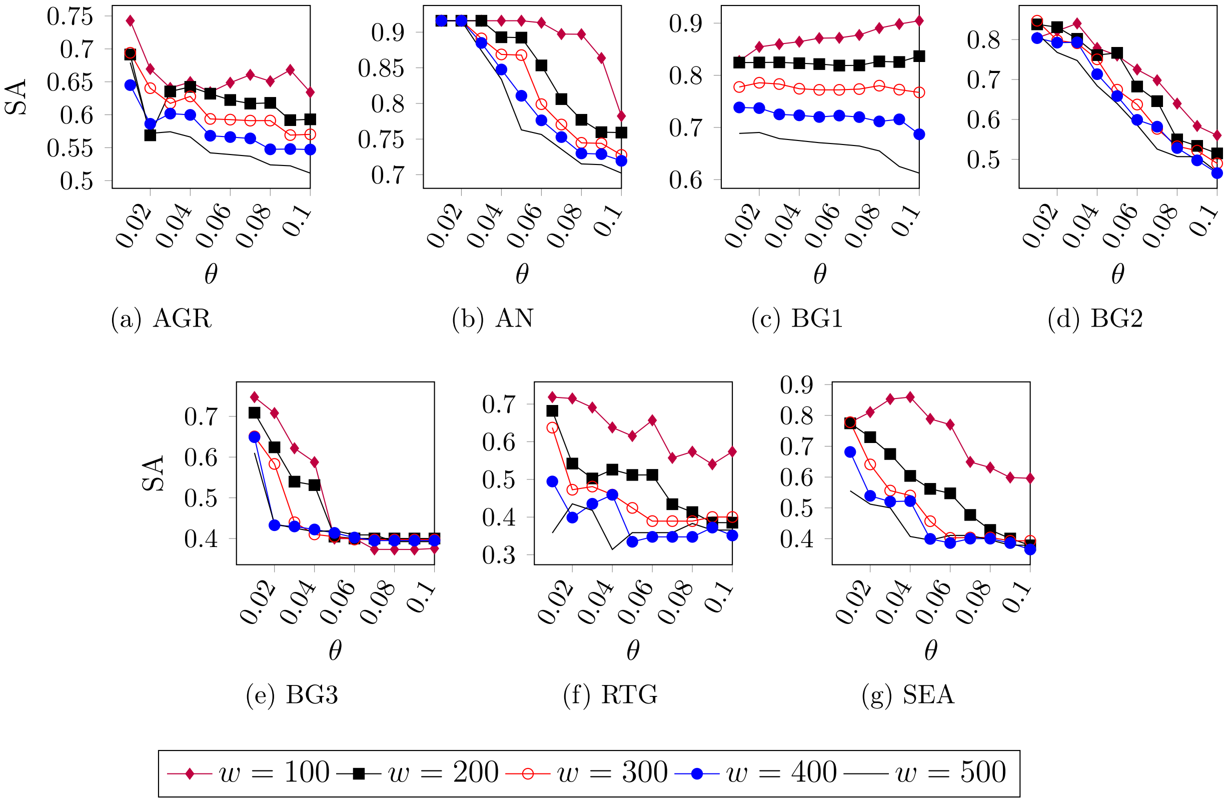
(d) BG2

(e) BG3

(f) RTG

(g) SEA

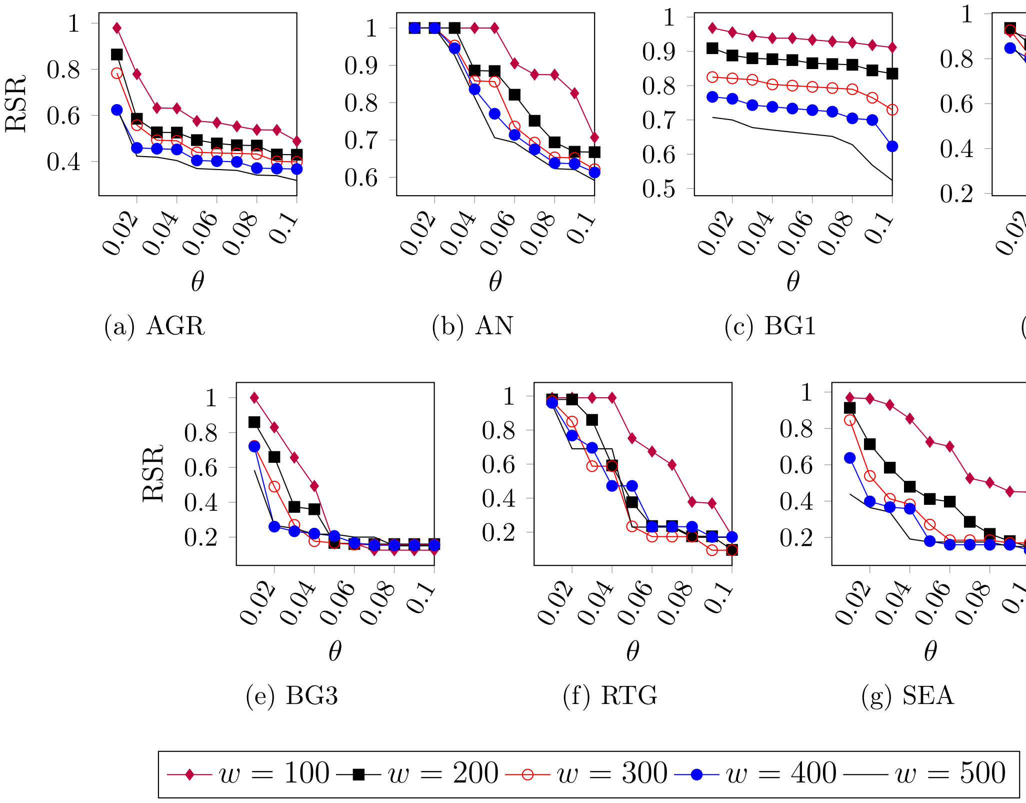
Figure 8.8: Average Prequential Accuracy (%) obtained by varying the threshold θ.



(a) AGR (b) AN (c) BG1 (d) BG2

(e) BG3 (f) RTG (g) SEA

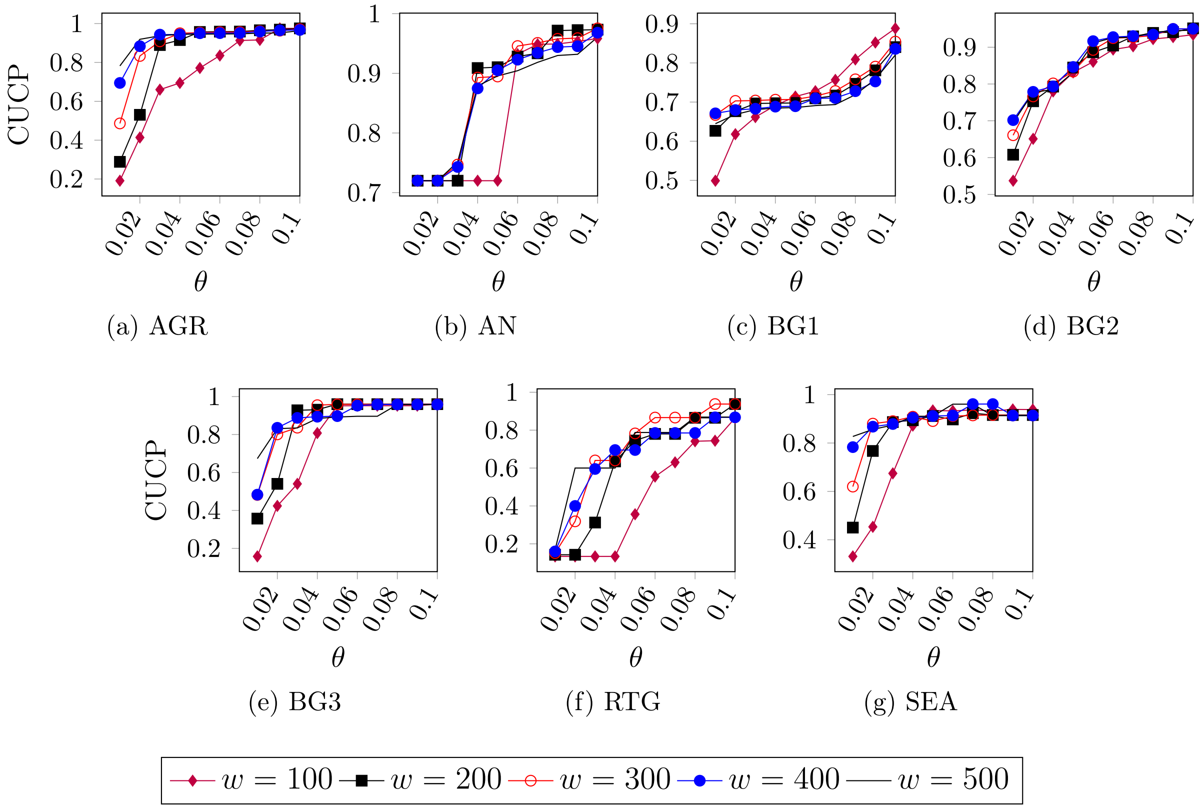
Figure 8.9: Selection Accuracy obtained by varying the threshold θ.



(a) AGR (b) AN (c) BG1 (d) BG2

(e) BG3 (f) RTG (g) SEA

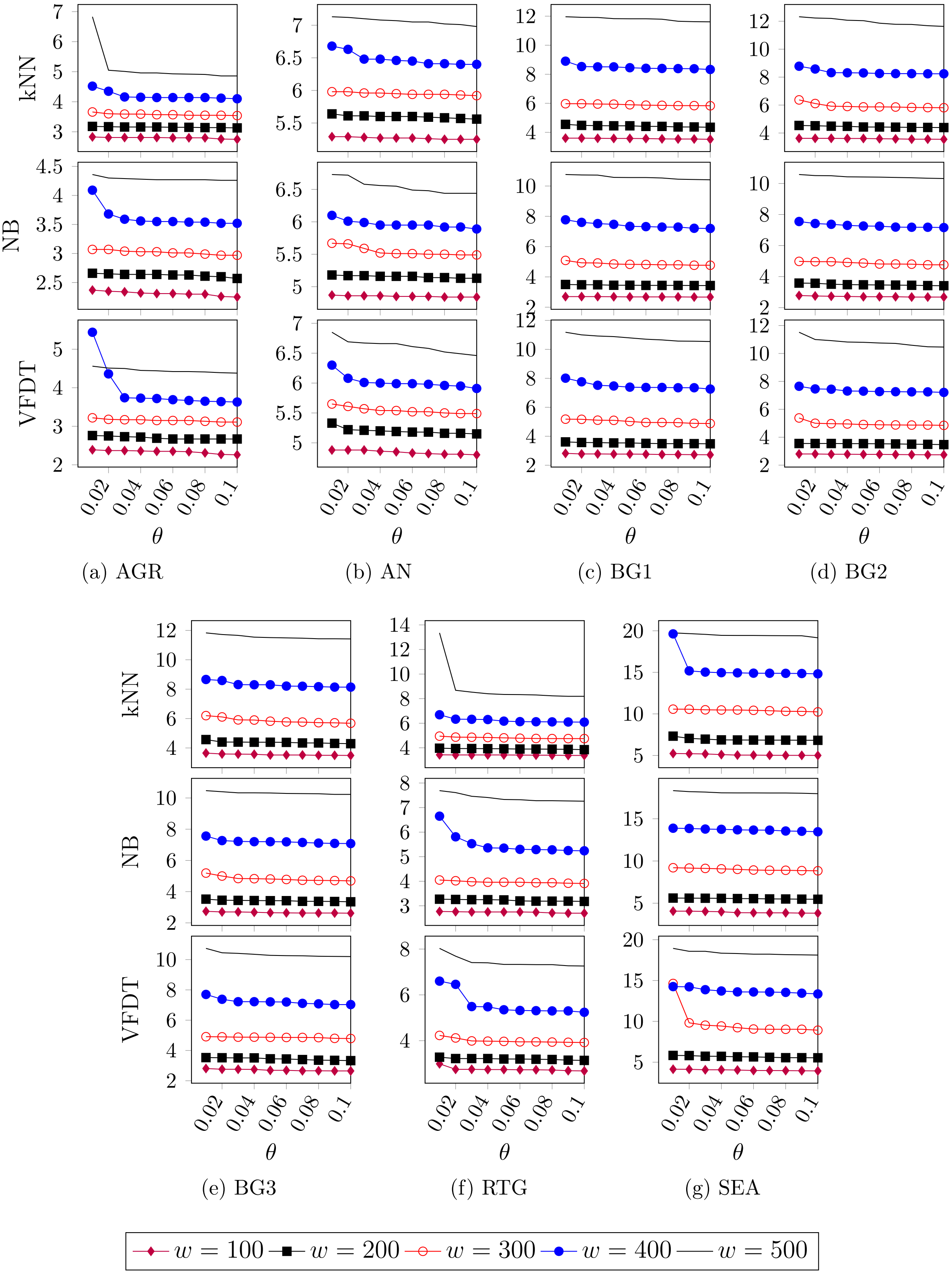
Figure 8.10: Relevant Selection Recall (RSR) obtained by varying the threshold θ.



(a) AGR (b) AN (c) BG1 (d) BG2

(e) BG3 (f) RTG (g) SEA

Figure 8.11: Complement of Unnecessary Complexity Penalty (CUCP) obtained by varying the threshold θ.



(a) AGR

(b) AN

(c) BG1

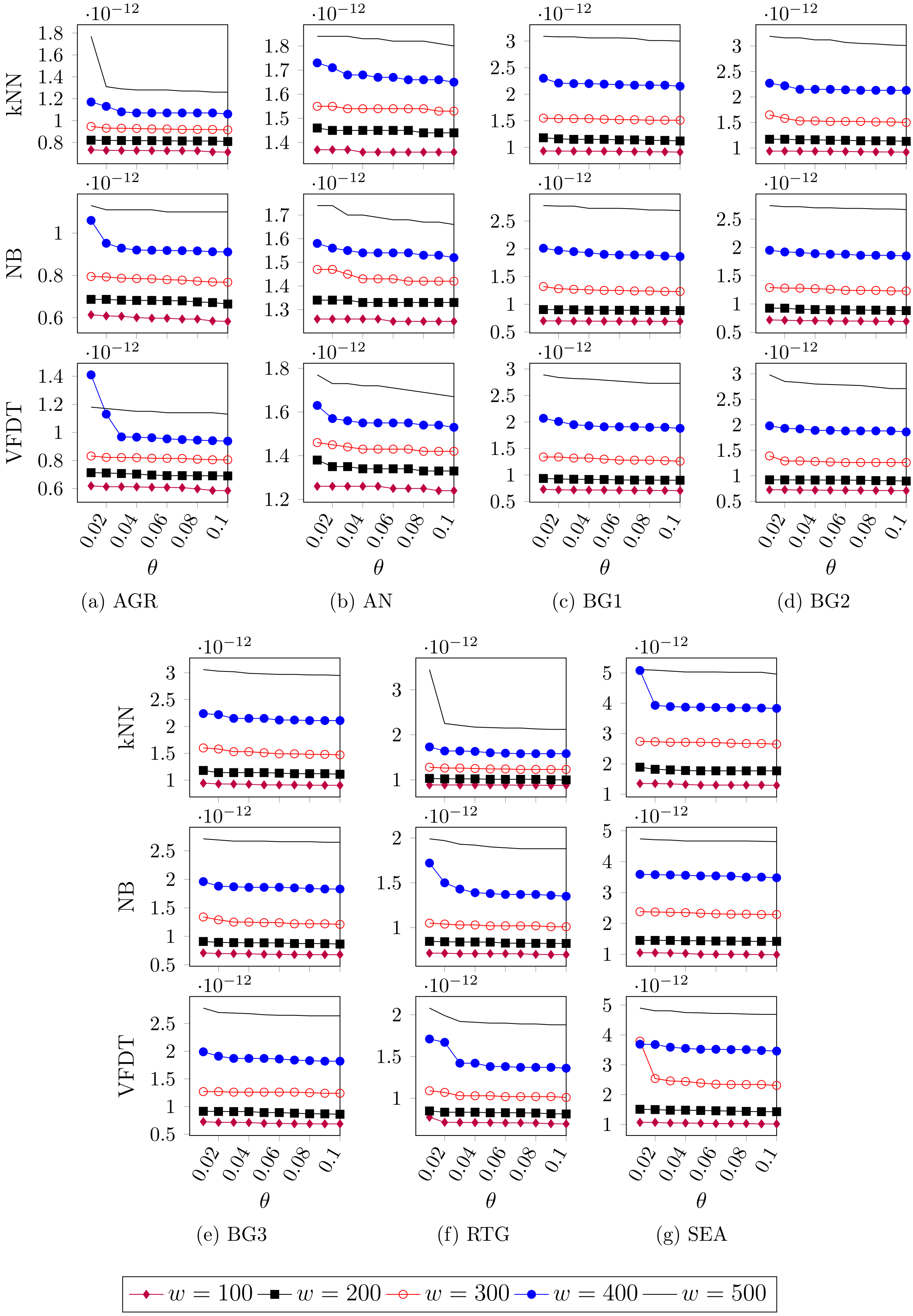
(d) BG2

(e) BG3

(f) RTG

(g) SEA

Figure 8.12: CPU Time (s) obtained by varying the threshold θ.



(a) AGR

(b) AN

(c) BG1

(d) BG2

(e) BG3

(f) RTG

(g) SEA

Figure 8.13: RAM-Hours (GB-Hour) obtained by varying the threshold θ.

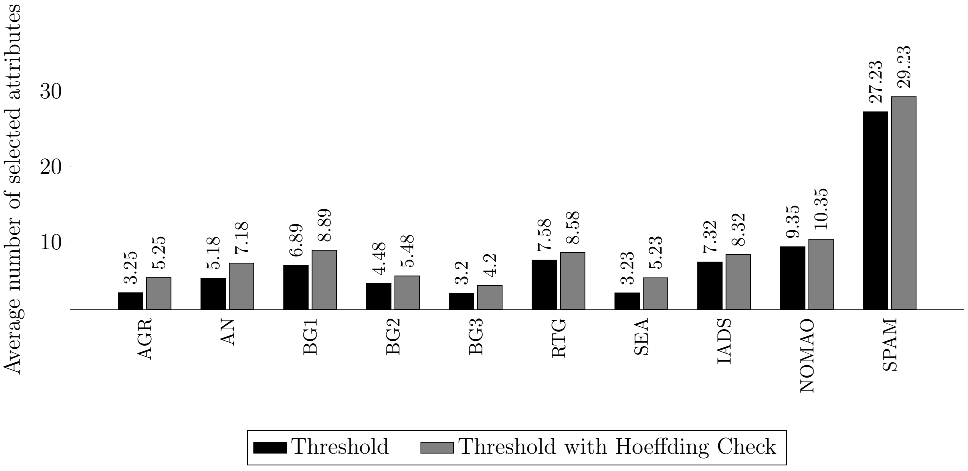


Figure 8.14: Average number of features selected using the threshold selection strategy.

At this point, it is important to provide default parameters to DISCUSS, hereafter represented by $(\bar{n}, \bar{w})$ and $(\bar{\theta}, \bar{w})$. Both parameters were determined following the best average rankings amongst all parameter variations across all different classifiers. The justification to this procedure is to facilitate the comparison of the final results and to allow DISCUSS to be easily parametrized regardless of which classifier will be used. As a result, the parameters chosen as default ones are the following: $\bar{n} = 5$ with a window size $w = 300$ for the n Best strategy and $\bar{\theta} = 0.03$ with a window size $w = 200$ for the thresholding strategy.

Another open topic regarding the thresholding strategy is the number of attributes selected with and without the heuristic for redundancy checks. Figure 8.14 presents the average number of features selected using or not the Hoeffding redundancy check heuristic. Clearly, adopting the Hoeffding redundancy check strategy allows more features to be selected, meaning that redundant peers are ignored and wrongly selected.

Similarly as before, Figure 8.15 presents the average Selection Accuracy (SA) ob-

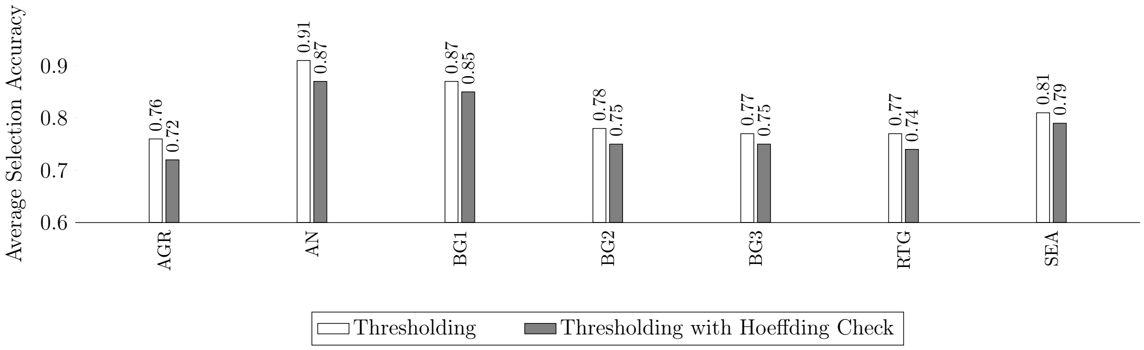


Figure 8.15: Comparison of the impact on Selection Accuracy by using the Hoeffding Bound to trigger redundancy checks.

tained by the conventional thresholding strategy and its variation using the Hoeffding-based heuristic to trigger redundancy checks. Once more, the adoption of the heuristic for redundancy checks decreases the overall SA scores obtained, meaning that more redundant attributes are inappropriately selected. As a reminder, the impact of adopting the Hoeffding heuristic for redundancy tests in processing time and memory usage is later discussed in Section 8.3.4.

8.3.3 Comparison against Original Learners and the Hoeffding Adaptive Tree

Both of DISCUSS' strategies were evaluated focusing on variations in their main parameters and window sizes. Additionally, it is critical to verify whether DISCUSS enables classifiers to overcome feature drifts accurately and how these classifiers behave when compared to the chosen baseline: the Hoeffding Adaptive Tree. Tables 8.2 and 8.3 present the results obtained by the n Best and thresholding selection strategies, respectively. In the following reports, (n^*, w^*) and (θ^*, w^*) represent the n and θ values that maximize the average prequential accuracy in each experiment, while $(\theta^*_{auto}, w^*_{auto})$ is the automatic threshold that also optimizes the accuracy obtained given different window sizes. As mentioned in the previous section, the parameters used for this comparison are as follows: $\bar{n} = 5$ with a window size $w = 300$ for the n Best strategy and $\bar{\theta} = 0.03$ with a window size $w = 200$ for the thresholding strategy.

Results for the application of DISCUSS to HAT were omitted since they jeopardize the final accuracy results. This is due the fact that performing full model resets, in a classifier that is able to perform local resets, is often unnecessary and makes the tree to perform less accurately (BARDDAL; GOMES; ENEMBRECK, 2015b).

Starting with the n Best strategy, depicted in Table 8.2, it is clear that all classifiers benefit from the selection provided by DISCUSS in almost all scenarios. Experiments, where classifiers had their accuracies decreased include mainly real-world datasets, where the distribution is unknown and there is no clear justification of whether they contain or not feature drifts, and thus, resetting the classifier is possibly prejudicial. On average, the accuracy increases are promising as they average 5.47% for kNN, 10.74% for NB and 11.00% for VFDT. To test the hypothesis that DISCUSS is able to significantly improve the classifiers' accuracy, several pairwise Wilcoxon tests were performed, and as a result, DISCUSS is able to significantly improve kNN's, NB's and VFDT's accuracy with a 99% confidence. Finally, an execution of Friedman and Nemenyi tests was performed in order to compare all combinations against the HAT classifier. Figure 8.16a presents the resulting critical differences chart, where {VFDT-n^*, NB-n^*, VFDT-$\bar{n}$, kNN-n^*, HAT} $\succ$ {NB-$\bar{n}$,

Table 8.2: DISCUSS results comparison for the n Best strategy against base learners and Hoeffding Adaptive Tree.

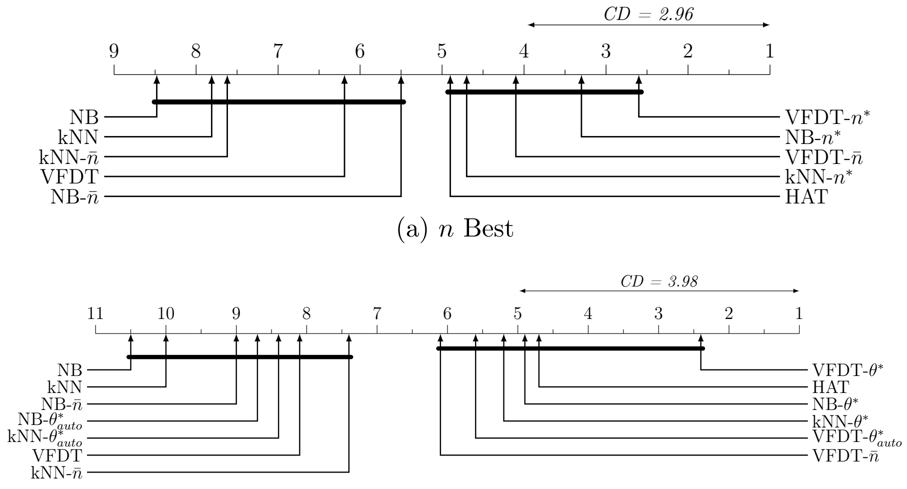
Experiment	kNN	kNN $(\bar{n}, \bar{w})$	kNN (n^*, w^*)	NB	NB $(\bar{n}, \bar{w})$	NB (n^*, w^*)	VFDT	VFDT $(\bar{n}, \bar{w})$	VFDT (n^*, w^*)	HAT
AGR	54.71	62.28	89.88	59.13	91.3	**91.60**	57.21	91.28	91.56	69.84
AN	67.31	75.86	76.85	74.37	89.99	91.80	82.86	92.61	**93.31**	84.74
BG1	79.35	85.39	87.13	76.21	89.99	93.64	80.86	93.52	**93.76**	91.76
BG2	71.37	72.71	77.99	75.8	78.01	92.05	78.06	89.89	**92.84**	78.92
BG3	55.50	52.27	70.60	57.70	56.74	67.81	64.04	70.44	**90.28**	64.45
RTG	52.31	54.41	54.50	54.15	59.62	60.78	54.84	63.83	**83.21**	56.64
SEA	63.46	64.73	69.36	79.26	83.22	**87.40**	77.63	80.57	81.51	78.80
IADS	100.00	100.00	100.00	67.95	68.02	78.28	92.90	92.09	93.87	83.90
NOMAO	95.16	95.66	96.77	83.86	83.94	84.78	91.08	91.09	92.34	92.67
SPAM	83.72	84.32	84.56	71.13	78.95	**86.00**	78.07	78.30	78.85	84.48

Underlined values stand for the best results obtained by a classifier, while bold-faced ones represent the best overall results.

Table 8.3: DISCUSS results comparison for the thresholding strategy against base learners and Hoeffding Adaptive Tree.

Experiment	kNN	kNN $(\bar{\theta}, \bar{w})$	kNN (θ^*, w^*)	kNN (θ^*_{auto}, w^*)	NB	NB $(\bar{\theta}, \bar{w})$	NB (θ^*, w^*)	NB (θ^*_{auto}, w^*)	VFDT	VFDT $(\bar{\theta}, \bar{w})$	VFDT (θ^*, w^*)	VFDT (θ^*_{auto}, w^*)	HAT
AGR	54.71	57.79	64.95	65.15	59.13	62.92	65.94	62.13	57.21	62.73	65.73	62.48	**69.84**
AN	67.31	72.05	72.27	67.11	74.37	75.54	77.91	77.59	82.86	83.55	84.35	84.67	**84.74**
BG1	79.35	82.21	84.49	77.88	76.21	81.15	81.16	80.63	80.86	81.61	82.03	81.63	**91.76**
BG2	71.37	77.10	79.92	73.44	75.80	77.57	80.35	75.05	78.06	82.52	**84.22**	77.88	78.92
BG3	55.50	61.48	64.59	53.22	57.70	54.07	57.60	56.82	64.04	53.32	**65.51**	64.83	64.45
RTG	52.31	52.63	53.78	54.21	54.15	54.77	55.09	55.03	54.84	56.86	**57.34**	54.90	56.64
SEA	63.46	71.15	74.56	67.98	79.26	80.14	**84.02**	81.98	77.63	80.98	81.21	80.29	78.80
IADS	100.00	100.00	100.00	100.00	67.95	70.77	85.12	82.91	92.90	98.58	100.00	99.19	83.90
NOMAO	95.16	95.81	96.69	96.22	83.86	87.22	92.83	89.80	91.08	92.20	95.44	94.22	92.67
SPAM	83.72	84.79	86.16	85.39	71.13	73.98	86.56	71.84	78.07	81.19	**86.97**	79.63	84.48

Underlined values stand for the best results obtained by a classifier, while bold-faced ones represent the best overall results.



(a) n Best

(b) Thresholding

Figure 8.16: Critical differences chart DISCUSS variations against base learners and Hoeffding Adaptive Tree.

VFDT, kNN-$\bar{n}$, kNN, NB} is observed with a 95% confidence.

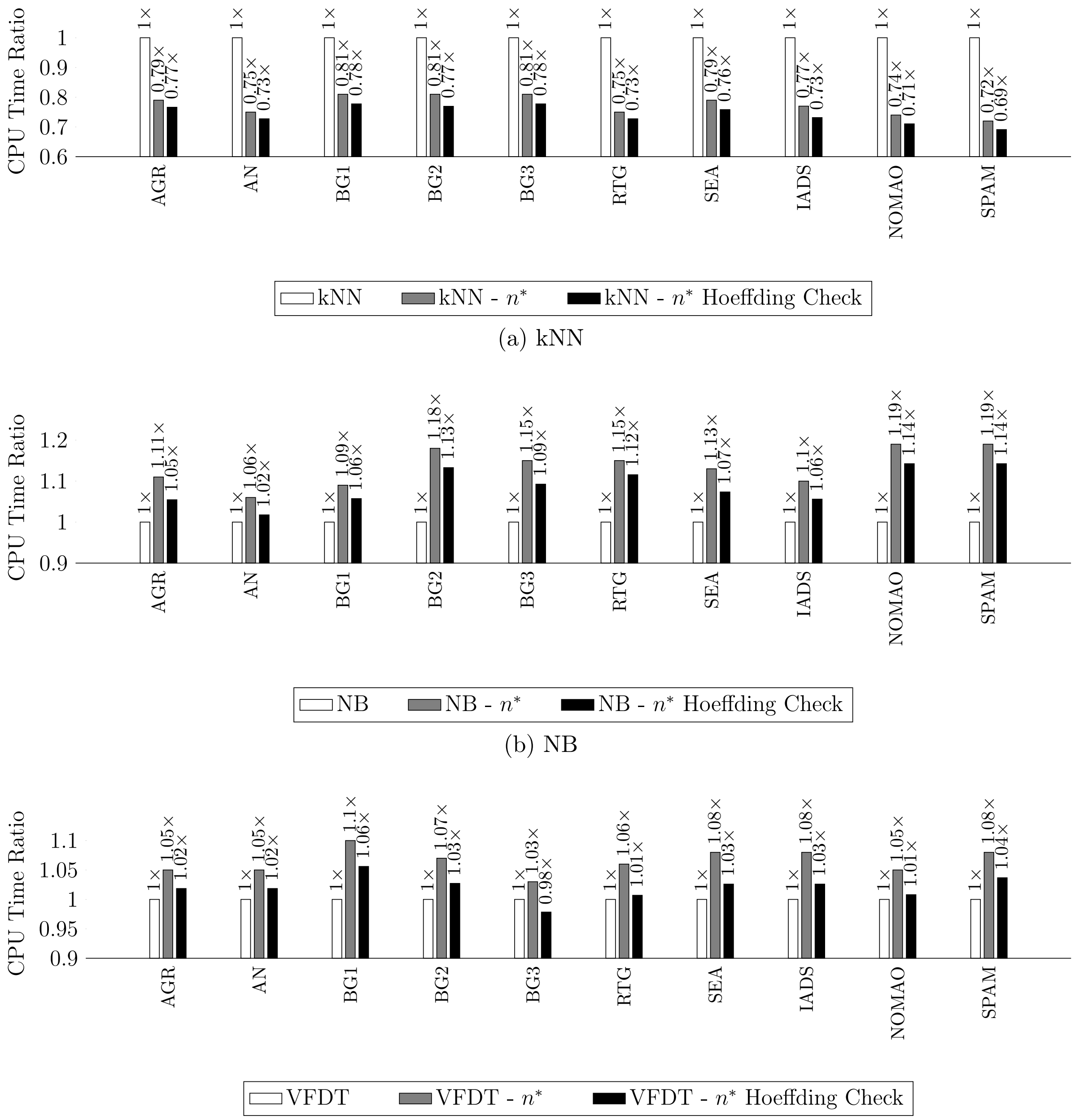
Table 8.3 presents the results obtained for the thresholding selection of discuss, which enable an analogous analysis. Similarly to the results obtained for the previous selection strategy, DISCUSS using thresholding is also capable of boosting classifiers' accuracy in interesting rates. For instance, kNN, NB and VFDT classifiers see their average accuracies increase in 5.01%, 4.57% and 3.32%, respectively. Pairwise Wilcoxon tests were once again performed and as a result, DISCUSS is able to enhance all classifiers' accuracy with a 99% confidence level.

Another required analysis is to verify whether automatic thresholding is beneficial compared to the results obtained by base learners alone. The increases provided by automatic thresholding are not as appealing, however, they exist and are as follows: 3.48% for kNN, 4.00% for NB and 0.74% for VFDT. With the aid of Wilcoxon's test, it follows that DISCUSS using automatic thresholding is also statistically superior to the original base learners with 99% confidence level. A multiple comparison amongst all classifiers and their combinations with thresholding strategies of DISCUSS was performed, and the critical differences chart reporting the results is depicted in Figure 8.16b, where {VFDT-θ^*, HAT, NB-θ^*, kNN-θ^*, VFDT-θ^*_{auto}, VFDT-$\bar{\theta}$} $\succ$ { kNN-$\bar{\theta}$, VFDT, kNN-θ^*_{auto}, NB-θ^*_{auto}, NB-$\bar{\theta}$, kNN, NB} with 95% confidence.

8.3.4 Processing Time and Memory Usage of DISCUSS Variations

The evaluation of DISCUSS should also encompass processing time and memory usage. Traditional feature selection is known for decreasing both metrics in batch learning, however, there is no guarantee that the same would hold in streaming scenarios. To righteously evaluate the efficiency of a feature selection algorithm in streaming scenarios, one must not only accumulate its processing time, but also the classifier's training and prediction times. By doing so, it is possible to verify if the overhead of computing the scoring operators and selecting attributes can be justified by sufficiently decreasing the complexity of the classifier. This section compares the processing time and memory usage of DISCUSS' selection strategies n^* and θ^* when applied to kNN, NB and VFDT classifiers. To facilitate visualization, both CPU Time and memory usage as presented as ratios of DISCUSS to the base learner.

Figures 8.17 and 8.18 present the processing time ratio results obtained for the n Best and thresholding strategies, respectively. In addition to that, these figures also report the ratios obtained by the Hoeffding-based heuristic for redundancy testing. In both figures, it becomes clear that DISCUSS produces an overhead that produces an increment to the running times of both NB and VFDT, while kNN has it's running time decreased. These results are interesting since DISCUSS, by selecting features, decreases the complexity of kNN sufficiently to justify the overhead of computing the Symmetrical Uncertainty of each attribute w.r.t the class over a sliding window and selecting features given feature drifts. An important example of that is the Spam Corpus (SPAM) experiment, where kNN becomes approximately 30% faster compared to its original version, while being more accurate. The combination of these results shed light on the fact that DISCUSS is able to improve kNN not only in accuracy, but also in processing time.

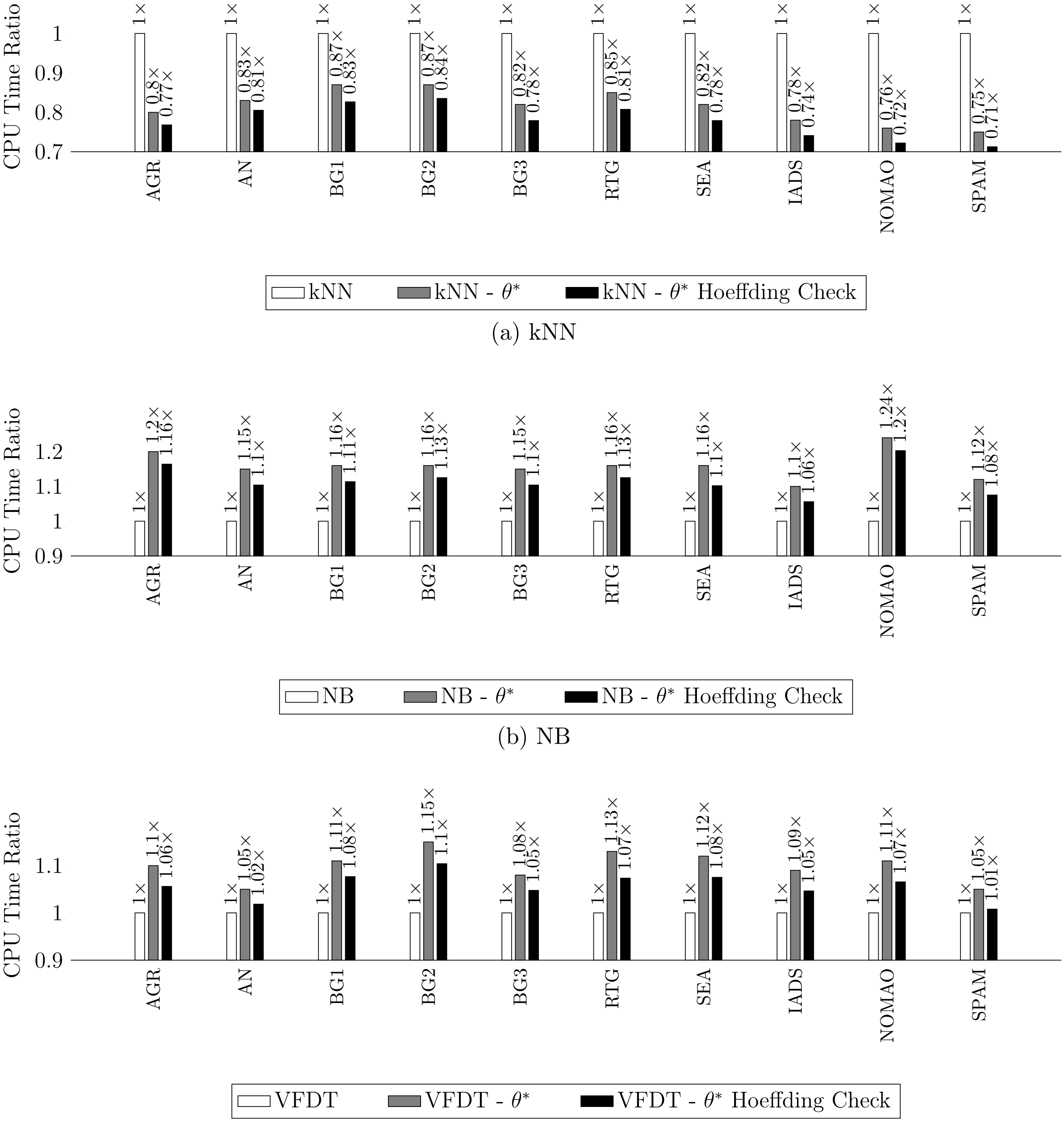


(a) kNN

(b) NB

(c) VFDT

Figure 8.17: CPU Time (s) comparison between base learners and DISCUSS using n Best strategy.



(a) kNN

(b) NB

(c) VFDT

Figure 8.18: CPU Time (s) comparison between base learners and DISCUSS using thresholding strategy.

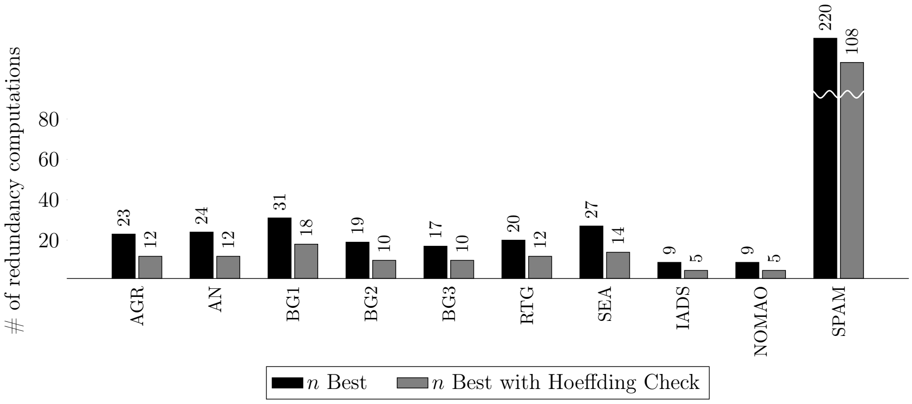


Figure 8.19: Number of redundancy computations using n best selection strategy.

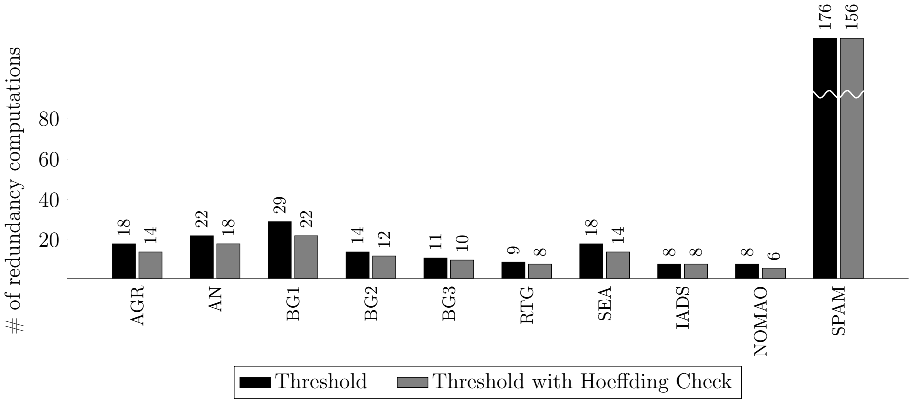
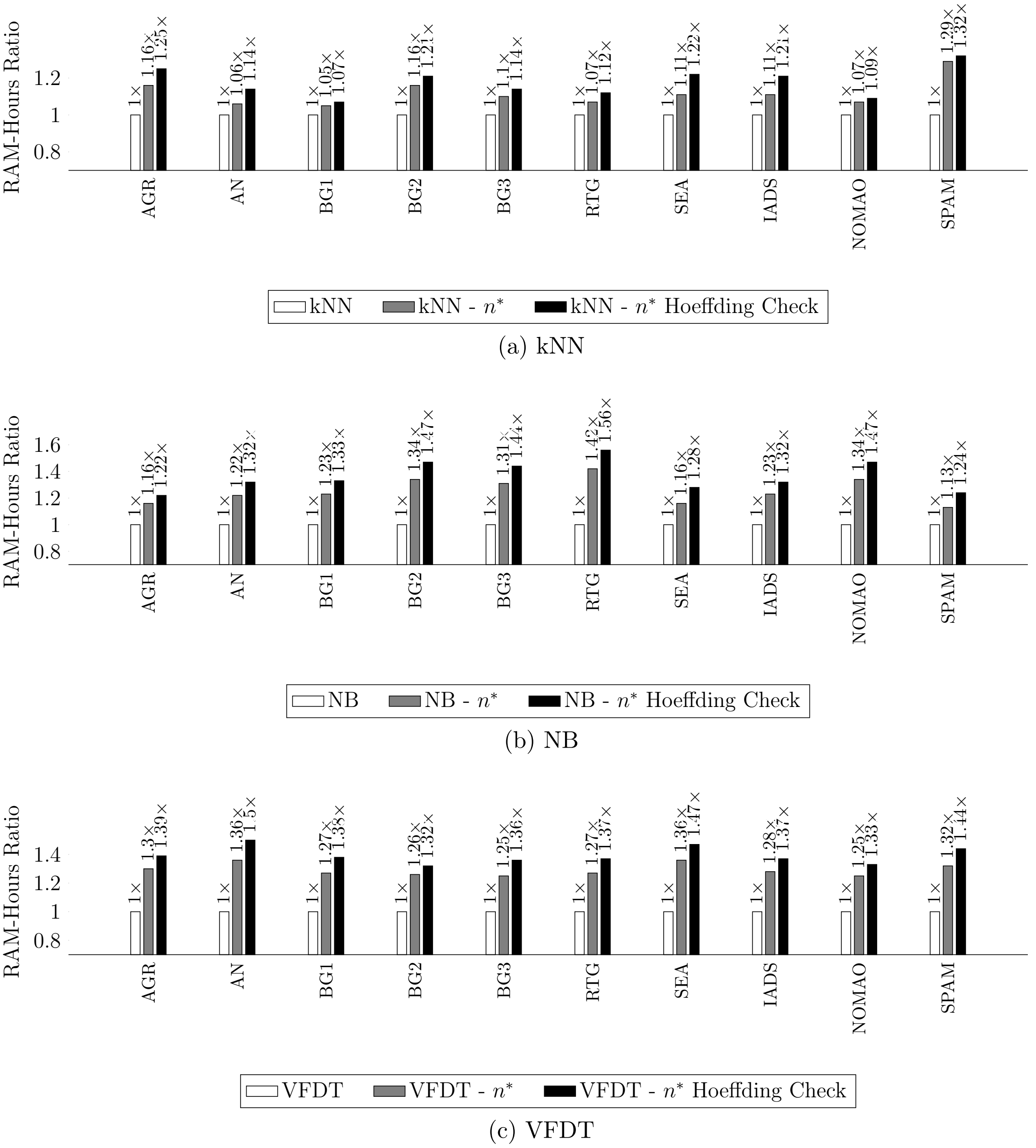


Figure 8.20: Number of redundancy computations using the threshold selection strategy.

The processing time results are easily explained if one compares the number of redundancy computations made in each strategy using or not the Hoeffding-based heuristic for redundancy verifications. Figures 8.19 and 8.20 depict the number of redundancy computations performed in each of the experiments in n Best and thresholding variations of DISCUSS. In all experiments, it is verified that adopting the Hoeffding bound to trigger redundancy verifications leads to a smaller number of redundancy computations, which is known to be the major bottleneck of the proposed method (see Section 8.2.4). The decreases in the number of redundancy computations is interesting especially in real-world datasets, where the number of attributes is usually bigger than in synthetic data.

The results for memory consumption are presented in Figures 8.21 and 8.22 for the n Best and Thresholding strategies, respectively. The results are clear in showing that all classifiers, i.e., kNN, NB and VFDT, suffer from increases in memory consumption when combined to DISCUSS. Such increases are expected, since DISCUSS requires storing (i) a sliding window of instances, (ii) structures for computing the necessary entropies, and (iii) discretization structures if the stream contains numeric data. Discretion is advised when analyzing memory consumption results, since such increases may not be prohibitive in most scenarios, mainly if the base learner is memory-efficient, e.g., NB or VFDT classifier, while smaller increases may cause kNN-based systems to fail due to lack of memory.

Finally, it is also important to point out that adopting the Hoeffding-based heuristic for redundancy tests leads to increased memory consumption, meaning that the classification model is built upon more attributes (the redundant ones that were inappropriately selected).

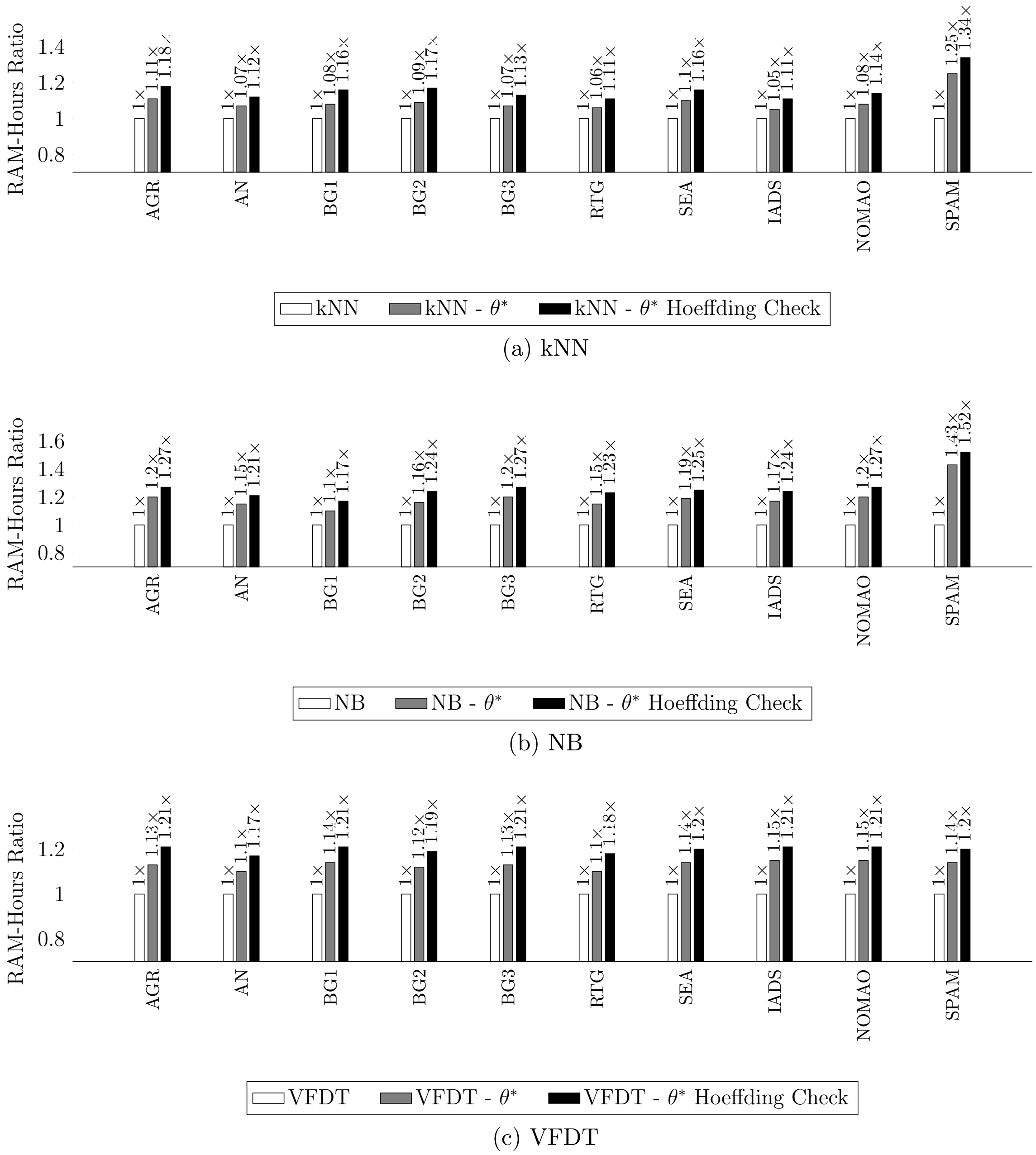


(a) kNN

(b) NB

(c) VFDT

Figure 8.21: RAM-Hours (GB-Hour) comparison between base learners and DISCUSS using n Best strategy.



(a) kNN

(b) NB

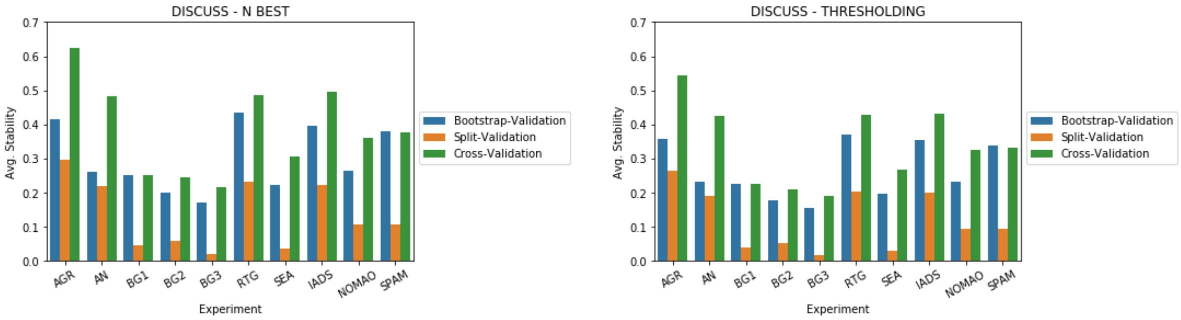
(c) VFDT

Figure 8.22: RAM-Hours (GB-Hour) comparison between base learners and DISCUSS using thresholding strategy.

8.4 Stability

In addition to verifying how accurate the feature selection of DISCUSS is, this section is devoted to quantifying how stable its selection is when the algorithm is fed with different inputs according to the method proposed in Section 4.4.4.

In Figure 8.23 the results obtained during the experiments are reported, following bootstrap-, split-, and cross-validation schemes in a 10-fold validation environment (see Section 4.4.4). At first, it is important to highlight that the stability rates achieved by DISCUSS vary according to the experiment conducted, but more importantly, according to the validation process adopted. Naturally, the highest stability rates are achieved using the cross-validation scheme, as 9 out of the 10 folds are updated with the arrival of each instance, thus making the selection process much more uniform across the folds. The same rationale can be applied to explain the rates obtained by the bootstrap-based validation experiments, as each instance is used to update the feature selection process allocated in each fold approximately 66% of the times. The results obtained with the split-validation process are the lowest, as only 1 out of the 10 folds are updated with the arrival of each instance. Also, it is worthy to highlight that it is hard to tell how ‘stable’ DISCUSS is due to the lack of competing techniques, and as a result, the results reported here may serve as baselines for future works on the area. Finally, the results obtained across the n-Best and Thresholding strategies, depicted in Figures 8.23a and 8.23b, respectively; show similar results, where the latter is, in average, 12% less stable.



(a) n Best

(b) Theresholding

Figure 8.23: Average Stability results obtained by DISCUSS on synthetic experiments.

8.5 When and Why DISCUSS Fails

In the previous sections, DISCUSS was evaluated using different classifiers and in a variety of experiments. Focusing on accuracy, it has been observed that the scores obtained for BG3 and RTG experiments were reasonably lower when compared to the remaining experiments. Now, these experiments are used as examples to explain when and why DISCUSS fails.

BG3. In this concept, earlier introduced in Equation 4.4, we see that three dimensions X_α, X_β and X_ϵ are relevant to the class in a manner that resembles an XOR problem. During this section, it is assumed that we have a stream that also encompasses an irrelevant attribute X_ξ. In this case, if we generate instances following this concept, we would have the following 16 features combinations and the following probabilities:

X_α	X_β	X_ϵ	X_ξ	Y
1	1	1	1	1
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	1	0
0	1	1	0	0
0	1	0	1	0
0	0	1	1	0
1	1	1	0	1
1	1	0	1	0
1	0	1	1	0
0	1	1	1	0
0	0	0	0	1

and since all features combinations are equally likely to occur, it follows that:

Features Probabilities

$P[X_\alpha = 1] = {}^{1}/_{2}$

$P[X_\beta = 1] = {}^{1}/_{2}$

$P[X_\epsilon = 1] = {}^{1}/_{2}$

$P[X_\xi = 1] = {}^{1}/_{2}$

Class Probabilities

$P[Y = 1] = {}^{4}/_{16} = {}^{1}/_{4}$

$P[Y = 0] = {}^{12}/_{16} = {}^{3}/_{4}$

This analysis is started by comparing a relevant and an irrelevant attribute: X_α and X_ξ. For X_α, if we compute the joint probability $P[X_\alpha = 1, Y = 1]$, it follows that we have only 2 cases out of 16 (${}^{1}/_{8}$) that match these criteria, which occur when $X_\alpha = X_\beta = X_\epsilon = 1$ regardless of the value of X_ξ, i.e., $X_\xi = 1$ or $X_\xi = 0$. In this case, we could verify if X_α and Y are independent from each other, by checking if $P[X_\alpha = 1, Y = 1] = P[X_\alpha = 1] \times P[Y = 1]$. The last assertion is true, since $P[X_\alpha = 1] \times P[Y = 1] = {}^{1}/_{2} \times {}^{4}/_{16} = {}^{1}/_{8}$, which is the same probability presented above.

We could then repeat the same process for X_ξ. First, the joint probability $P[X_\xi = 1, Y = 1]$ would end up with the same value ${}^{1}/_{8}$. Next, by comparing if X_ξ and Y are independent, it follows that $P[X_\xi = 1, Y = 1] = P[X_\xi = 1] \times P[Y = 1]$ holds, because $P[X_\xi = 1] \times P[Y = 1] = {}^{1}/_{2} \times {}^{4}/_{16} = {}^{1}/_{8}$. Now, we can infer the following: (i) both X_α and X_ξ are equally biased towards the class, and thus, probability-based measures would score them equally, and that (ii) probability-wise, both attributes are claimed to be independent w.r.t. the class, which should not have been held true for X_α, since the class determination depends on the value of X_α.

The rationale here is simple: by evaluating each feature solely, any "single feature metric" such as Symmetrical Uncertainty, would fail at depicting the discriminative power

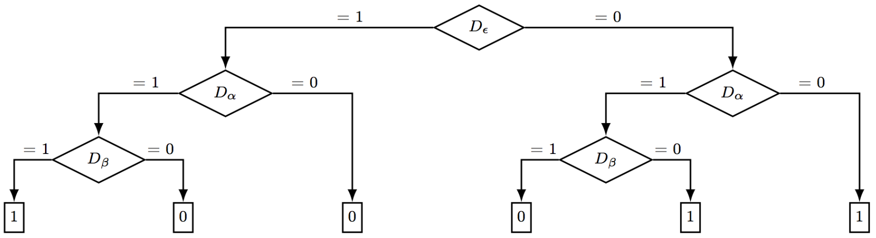


Figure 8.24: Example of RTG concept.

of attributes in cases where the class seems independent from attributes. In such cases, more complex metrics that evaluate subsets of attributes, such as a joint Symmetrical Uncertainty $SU([X_\alpha, X_\beta, X_\epsilon], Y)$ would be sufficient at depicting these complex relationships between features, however, it's computation is not trivial nor efficient to be performed on data streams.

RTG. Similarly to the BG3 discussion, let us now work under the assumption that our RTG concept relies on X_α, X_β and X_ϵ to determine the class Y and that we also have an irrelevant feature X_ξ. Given that all of these features are binary, a possible RTG concept would be the one depicted in Figure 8.24, which would result in features-class combinations and probabilities presented below.

X_α	X_β	X_ϵ	X_ξ	Y
1	1	1	1	1
1	0	0	0	1
0	1	0	0	1
0	0	1	0	0
0	0	0	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	1	1
0	1	1	0	0
0	1	0	1	1
0	0	1	1	0
1	1	1	0	1
1	1	0	1	0
1	0	1	1	0
0	1	1	1	0
0	0	0	0	1

and since all features combinations are equally likely to occur, it follows that:

Features Probabilities

$P[X_\alpha = 1] = {}^1/_2$

$P[X_\beta = 1] = {}^1/_2$

$P[X_\epsilon = 1] = {}^1/_2$

$P[X_\xi = 1] = {}^1/_2$

Class Probabilities

$P[Y = 1] = {}^8/_{16} = {}^1/_2$

$P[Y = 0] = {}^8/_{16} = {}^1/_2$

Using the same procedure as before, let us now compare two relevant features (X_α and X_ϵ) and an irrelevant one (X_ξ). Computing the joint probabilities between each feature and the class, we obtain: $P[X_\alpha = 1, Y = 1] = {}^1/_4$, $P[X_\epsilon = 1, Y = 1] = {}^1/_8$ and $P[X_\xi = 1, Y = 1] = {}^1/_4$.

Next, we could verify if these attributes are independent w.r.t the class as follows: (i) X_α is independent since $P[X_\alpha = 1, Y = 1] = P[\alpha = 1] \times P[Y = 1]$ holds, (ii)

X_ϵ is not dependent since $P[X_\epsilon = 1, Y = 1] \neq P[\epsilon = 1] \times P[Y = 1]$; and (iii) X_ξ is independent since $P[X_\xi = 1, Y = 1] = P[\xi = 1] \times P[Y = 1]$ also holds. As a result, DISCUSS would compute the following Symmetrical Uncertainty values: $SU(X_\alpha, Y) = 0.0000$, $SU(X_\epsilon, Y) = 0.1887$ and $SU(X_\xi, Y) = 0.0000$. These results show that DISCUSS is unable to depict correlations between relevant attributes and the class since it performs a "flat" evaluation of the features, while on a RTG concept it would be necessary to evaluate different partitions of data, such as the ones that are traversed by the first node of the tree. Similarly as before, these complex relationships amongst subsets of variables would require the computation of joint Symmetrical Uncertainties.

8.6 Concluding Remarks

This section introduced DISCUSS, a novel filter for feature selection from data streams. DISCUSS has as its core the Symmetrical Uncertainty operator, which is able to portrait the discriminative power of an attribute given a window of most recent data. Furthermore, Symmetrical Uncertainty is also used to verify whether two features are redundant to one another, however, in batch mode.

DISCUSS is generic in the sense that allows different selection strategies to be used and is classifier-independent. Until this point, DISCUSS has two selection strategies: the selection of the n best-ranked attributes and the selection given a relevance threshold θ. Even though the results obtained showed that single classifiers have their accuracies boosted in feature drifting scenarios, while corroborating the fourth hypothesis of this thesis, it is rather unclear if there is still room for accuracy improvements. The major drawback of DISCUSS is that both selection strategies verify the individual relevance of each attribute w.r.t. the class but not how discriminative a subset of features is together. As earlier discussed in Section 3.3.2, doing such verification is demanding as the problem space grows exponentially with the number of dimensions in the $\mathcal{O}(2^d)$ order.

In the next chapter, a different approach for feature selection based on adaptive boosting is proposed. This chapter will introduce the basic aspects of adaptive boosting for data streams and how it can be adapted for the feature selection task.

Chapter 9

Boosting-based Dynamic Feature Selection

In the previous chapter, a novel dynamic feature selection method based on merit was introduced. This method continuously kept track of the importance of features as the stream progressed, and upon significant changes on these values, redundancy amongst them was computed so that a subset of features with high relevance and low redundancy was selected. In this chapter, a different method for dynamically selection features during the processing of data streams called Adaptive Boosting for Feature Selection (ABFS). ABFS chains decision stumps and drift detectors, and as a result, identifies which features are relevant to the learning task as the stream progresses with success. Internally, ABFS takes advantage of the 'mistakes' made by a set of boosting units to highlight *hard-to-classify* instances. By intuition, these instances are either (i) located at the decision boundaries of classes, or (ii) noise. If one works under the assumption that the labels of incoming instances are trustworthy (not noisy), decision stumps will be able to select the most important features according to these hard-to-classify instances as they naturally account for these weights during the feature selection process. In ABFS, each decision stump will be responsible for finding the feature that maximizes a gain function, while observing features that have been selected previously.

This chapter is divided as follows. Section 9.1 introduces preliminary content on Boosting and its adaptive variants for streaming scenarios. These concepts are relevant as they are the basis for the proposed method. Section 9.2 introduces ABFS, detailing how decision stumps, drift detectors work both individually and together, thus allowing dynamically feature selection to occur in streaming scenarios. Next, Section 9.3 assesses ABFS following the experimental protocol adopted in the previous chapters (see Chapter 4). Finally, Section 9.5 concludes this chapter.

9.1 Preliminaries on Boosting

In machine learning, Boosting is a family of meta-learning methods that target the construction of a strong learner by combining multiple weak learners that, by definition, are slightly better than random guessing.

The most widely used and well-known implementation of Boosting is AdaBoost (SCHAPIRE, 1990), and multiple variants of it were proposed throughout the years (APPEL; PERONA, 2017; MIAO et al., 2016). In AdaBoost, a set of weak learners H is trained over a series of rounds $t = 1, \dots, T$. During each iteration, a new weak learner $h_t : \vec{x} \rightarrow y$ is trained over the dataset $(\vec{x}^1, y^1), \dots, (\vec{x}^n, y^n)$ taking into account a distribution of weights D_t for these instances. In the first round, it is assumed that all instances have the same weight, i.e., $D_1(i) = \frac{1}{n}$. The error of a weak learner in the t-th round is the sum of the weights of misclassified instances, as shown in Equation 9.1.

$$\epsilon_t = Pr_{i \sim D_t} \left[h_t(\vec{x}^i) \neq y^i \right] = \sum_{\forall i, h_t(\vec{x}^i) \neq y^i} D_t(i) \tag{9.1}$$

In each of the following rounds, the weights $D_t(i)$ are updated according to a parameter α_t, which is calculated according to Equation 9.2.

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{9.2}$$

In practice, α_t quantifies how "important" h_t is, as $\alpha_t \geq 0$ if $\epsilon_t \leq 1/2$ and that α_t increases with the decrease of ϵ_t. According to α_t, the weight distribution can be updated following Equation 9.3, where $Z_t = \sum_i^n D_t(i)$ is a normalization factor to guarantee that D_t is a distribution.

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\vec{x}^i) = y^i \\ e^{\alpha_t} & \text{if } h_t(\vec{x}^i) \neq y^i \end{cases} \tag{9.3}$$

With this update, the weights of correctly classified instances will decrease, while misclassified instances will increase. As a result, this process highlights *hard-to-classify* instances for future rounds. And finally, predictions[1] can be extracted from the final

[1] The original prediction scheme presented in (SCHAPIRE, 1999) focuses only on binary classification tasks and has been extended here to account for multi-class problems. Details about the use of AdaBoost in binary classification tasks and proofs on its error bounds can also be found in the same paper.

"strong" learner as depicted in Equation 9.4.

$$H(\vec{x}) = \operatorname*{argmax}_{y_i \in Y} \left(\sum_{t=1}^{T} \begin{cases} \alpha_t h_t(\vec{x}) & \text{if } h_t(\vec{x}) = y_i \\ 0 & \text{if } h_t(\vec{x}) \neq y_i \end{cases} \right) \tag{9.4}$$

Even though AdaBoost is iterative, it works under the assumption that all instances of the dataset are available at all times so that re-weighting occurs. Naturally, this is an assumption that does not hold in streaming scenarios, as each instance should be processed and discarded right after. Targeting the development of boosting techniques for data streams, different approaches for classification (OZA, 2005; PELOSSOF et al., 2009; WANG; PINEAU, 2016) and regression (HU et al., 2017) tasks have been developed over the years.

In this work, we follow a similar framework proposed in (OZA, 2005), called OzaBoost. OzaBoost was tailored to be an approximation of AdaBoost for data streams. In opposition to AdaBoost, where the number of rounds T determines the ensemble size, OzaBoost has a predefined number of weak learners M and counters for correctly $(\lambda_i^c, 1 < i < M)$ and incorrectly $(\lambda_i^e, 1 < i < M)$ classified instances which are updated as new instances are processed. For each instance $(\vec{x}^t, y^t)$ drawn from the data stream S, a weight $\lambda = 1$ is set. The instance is then traversed along the weak learners $h_1, \ldots, h_M$ sequentially. For each h_i, the instance is tested to check whether if it is correctly classified or not, i.e., $h_i(\vec{x}^t) = y^t$ or not; and as a result, the counters λ_i^c and λ_i^e are incremented with λ (Equations 9.5 and 9.6, respectively). Next, the value of λ is incremented or decremented following Equation 9.7. This is the same procedure adopted by AdaBoost in Equations 9.3 and 9.4, except the normalization factor Z_t, which cannot be used in data streams as past instances have been discarded.

$$\lambda_i^c \leftarrow \lambda_i^c + \lambda; \tag{9.5}$$

$$\lambda_i^e \leftarrow \lambda_i^e + \lambda; \tag{9.6}$$

$$\lambda \leftarrow \lambda \times \begin{cases} \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^c} & \text{if } h_i(\vec{x}) = y^t \\ \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^e} & \text{if } h_i(\vec{x}) \neq y^t \end{cases} \tag{9.7}$$

9.2 Adaptive Boosting for Feature Selection in Data Streams

The properties of boosting have been investigated to improve classification rates, but also as a proxy for feature selection in batch scenarios. For instance, the work of (XU et al., 2014) uses gradient boosted regression trees to select features. Also, related to our approach, authors in both (DAS, 2001) and (MIAO et al., 2015) proposed different boosting techniques that use decision stumps to select features in batch scenarios.

We now propose a novel method based on Boosting to dynamically select features in streaming scenarios hereafter referred to as *Adaptive Boosting for Feature Selection* (ABFS). At this point, it is important to disclaim that the term *Adaptive* used here stands for the fact that the proposed method incrementally selects features as the stream is processed, but it is also able to detect feature drifts and adapt to them on the fly.

ABFS combines decision stumps and drift detectors to perform dynamic feature selection. Decision stumps are light-weighted, incremental, easy to implement and understand, but more importantly, an elegant approach to identify which feature maximizes a purity criterion and selects a feature accordingly. Next, each of these components are described individually, followed by how they are chained together to allow dynamic feature selection in data streams.

9.2.1 Decision stumps

The decision stump implementation used here is the core unit of incremental decision trees, e.g., Hoeffding Trees (DOMINGOS; HULTEN, 2000), and receive as input three parameters: a selection threshold θ, a grace period gp and a purity metric $\Omega(\cdot)$ that one wishes to maximize, e.g., Information Gain and Gini Index. By definition, a decision stump ds gathers statistics on the arriving data until the grace period gp is reached. After that, all features $f_i \in \mathcal{F}$ are evaluated according to a criterion $\Omega(\cdot)$. Let f_α and f_β be the two best-ranked features according to Ω. As proposed in (DOMINGOS; HULTEN, 2000), a decision stump will split on f_α if $\Omega(f_\alpha) - \Omega(f_\beta) > \epsilon$, where ϵ is the Hoeffding bound (HOEFFDING, 1963), previously discussed in Section 5.1.

As in (DOMINGOS; HULTEN, 2000), the Hoeffding bound is used here to approximate how many samples are required to achieve the optimal selection of a feature that would occur if the entire data stream was observed. As a result, with probability $(1 - \delta)$, it is statistically valid that f_α is the best feature to be selected (DOMINGOS; HULTEN, 2000).

In the proposed method, the decision stump is extended in two aspects. First, a decision stump will select the most appropriate feature f_α from a subset of features that have not been previously selected by other decision stumps. The idea on using boosting with decision stumps is that by traversing each instance across all the boosting units, instances that are hard to classify will be highlighted and will force the decision stump that is about to split to select a feature that better separates such samples. And second, the best-ranked feature f_α will only be selected if $\Omega(f_\alpha) > \theta$, which is a user-given threshold. Naturally, the definition of a selection threshold θ depends on the data domain being worked on, and different values are evaluated in Section 9.3.

9.2.2 Drift detectors

A drift detector is a statistical method that observes a data sequence and upon on its distribution, flags the occurrence of significant changes. In data streams, most of the drift detectors are used to monitor the error rates of a classifier. In this work, ψ denote a drift detector that receives as input a value of 1 if $h(\vec{x}^t) \neq y^t$, or 0 otherwise. Evidently, different realizations of ψ exist (see Section 2.3 and further details in Appendix A), e.g., ADWIN (BIFET; GAVALDÀ, 2009), HDDM-A and HDDM-W (FRÍAS-BLANCO et al., 2015); and the impact of different techniques are also assessed in Section 9.3.

9.2.3 Chaining decision stumps and drift detectors in a boosting scheme

The rationale behind ABFS is that boosting gives more weight to instances that are hard to classify. By intuition, these instances are either (i) located at the decision boundaries between classes, or (ii) are noise. If we work under the assumption that the labels of incoming instances are trustworthy (not noisy), decision stumps will be able to select the most important features according to these hard-to-classify instances as they naturally account for these weights during the feature selection process. In ABFS, each decision stump will be responsible for finding the feature that maximizes the gain function Ω without observing features that have been selected previously.

Since ABFS was tailored for feature selection in classification scenarios and not for actually training classifiers, a slightly different notation from the boosting schemes presented earlier is adopted. ABFS is composed of a dynamic set of boosting units U such that each unit $u_i \in U$ is a 4-tuple in the $(ds_i, \lambda_i^c, \lambda_i^e, \psi_i)$ form, where ds_i is a decision stump, λ_i^c and λ_i^e are counters for correctly and incorrectly classified instances by ds_i and ψ_i is a drift detector. The functioning of ABFS is detailed in Algorithm 7 by lines 1 to 4, where ABFS is initialized (INITIALIZE), and then updated with labeled instances

(UPDATE), and periodically used to retrieve the selected features during predictions (SELECT).

In the initialization step (lines 5-8 of Algorithm 7), ABFS instantiates both the set of boosting units U and the subset of selected features $\mathcal{F}'$ as empty lists, a candidate decision stump $ds_{candidate}$ that will gather statistics about incoming data to determine which feature to split on and select.

During the update step (lines 9-36 of Algorithm 7), ABFS updates its internal structures according to the arrival of an instance $(\vec{x}^t, y^t)$. First, the instance weight λ and an index to store the first layer that detects a drift i_{drift} are initialized. Next, the arriving instance is sequentially traversed along all of the boosting units in U. In each boosting unit $u_i = (ds_i, \lambda_i^c, \lambda_i^e, \psi_i)$, it is verified if the decision stump is able to correctly predict the class label ($ds_i(\vec{x}^t) = y^t$), or not ($ds_i(\vec{x}) \neq y^t$). Here, AdaBoost's weighting strategy is followed (Equations 9.5, 9.6 and 9.7), where higher values of λ will be associated with instances that are hard to classify. It is also important to highlight that after the classification of the instance in a unit u_i, the selected feature used in its decision stump ds_i is removed from $\vec{x}$ (line 25) so that the candidate decision stump $ds_{candidate}$ is enforced to select a feature that has not been selected already by the decision stumps in U.

In addition to the definition of λ, the drift detector is fed with the classification result (1 represents an error, while 0 represents a correct classification). Therefore, each drift detector is used to keep track of the error distribution of each decision stump. The rationale here is that changes in these distributions work as a proxy to identify when the importance of a feature changes, and thus, upon the flagging of a drift, it becomes necessary to re-start the feature selection process. In practice, if a drift is flagged by ψ_i, its index i will be stored in i_{drift} so that this and the following units are removed, and that the feature selection process can self-adjust upon the new data distribution. Naturally, depending on the drift, it would be possible that multiple units flag drifts, and thus, only the first layer that detects such changes is stored.

If no changes are detected (lines 26-31), the candidate decision stump $ds_{candidate}$ is trained[2] with $(\vec{x}^t, y^t)$ assuming a weight λ. With the arrival of multiple instances, the candidate decision stump will reach the grace period gp, and as a result, it will eventually select a new feature f_α according to the process described earlier. When this condition holds, a new boosting unit is instantiated with this decision stump, and it is added to U. A new candidate decision stump is then created to select the next best feature, and the selected subset of features is incremented with f_α.

[2] In decision stumps, whenever an instance $(\vec{x}^t, y^t)$ is used for training with a weight λ, it means that the same instance has been observed λ times

Algorithm 7: ABFS pseudocode. $\mathcal{S}$ is the data stream, h is a pointer to the classifier, $ds_{candidate}$ a candidate decision stump, $\mathcal{F}'$ the currently selected subset of features, θ a selection threshold used in decision stumps, and U the set of boosting units such that the u_i is the i-th unit and it is composed of a decision stump ds_i, a set of counters for correctly (λ_i^c) and misclassified (λ_i^e) instances, and a drift detector ψ_i.

```
Function ABFS($h, \mathcal{F}, \theta, \mathcal{S}$)
    INITIALIZE($h, \mathcal{F}, \theta$);
    for ($\vec{x}^t, y^t$) $\in \mathcal{S}$ do
        TRAIN($\vec{x}^t, y^t, h, \theta$);

Function INITIALIZE($h, \mathcal{F}, \theta$)
    $U \leftarrow \emptyset$;
    $ds_{candidate} \leftarrow$ new DecisionStump($\theta$);
    $\mathcal{F}' \leftarrow \emptyset$;

Function UPDATE($\vec{x}^t, y^t, h, \theta$)
    $\lambda \leftarrow 1$;
    $i_{drift} \leftarrow -1$;
    if $|U| > 0$ then
        for $i \leftarrow 1$ to $|U|$ do
            if $ds_i(\vec{x}^t) = y^t$ then
                $\lambda_i^c \leftarrow \lambda_i^c + \lambda$;
                $\lambda \leftarrow \lambda \times \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^c}$;
                Update $\psi_i$ with 0;
            else
                $\lambda_i^e \leftarrow \lambda_i^e + \lambda$;
                $\lambda \leftarrow \lambda \times \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^e}$;
                Update $\psi_i$ with 1;
            if $\psi_i$ flagged a drift and $i_{drift} = -1$ then
                $i_{drift} \leftarrow i$;
                break;
            Remove from $\vec{x}$ the feature selected at $ds_i$;
    if $i_{drift} = -1$ then
        Train $ds_{candidate}$ with ($\vec{x}^t, y^t$) assuming a weight $\lambda$;
        if $ds_{candidate}$ has selected a feature $f_\alpha \in \mathcal{F}$ then
            $U \leftarrow U \cup \{$new BoostingUnit($ds_{candidate}$)$\}$;
            $ds_{candidate} \leftarrow$ new DecisionStump($\theta$);
            $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{f_\alpha\}$;
    else
        while $|U| > i_{drift}$ do
            Remove from $\mathcal{F}'$ the feature selected in U.last();
            $U \leftarrow U \setminus \{$U.last()$\}$;
        Reset the learner $h$;

Function SELECT($\vec{x}$)
    return $\vec{x}$ after selecting the features selected in $\mathcal{F}'$ and dropping the
      remainder of the features;
```

On the other hand, it is, a feature drift is detected, all boosting units from the index that detected the change until the end of the list are removed (loop described by lines 33-35), as a boosting unit u_i affected the creation of its following units $\forall u_j, j \geq i$. Next, and the classifier h is reset to allow faster adaptation to the new concept.

Finally, the last part is the testing step (lines 37-38 of Algorithm 7), ABFS filters the arriving instance $\vec{x}$ so that only the features in $\mathcal{F}'$ are selected. This instance can then be passed to the classifier with a reduced dimensionality equals to $|\mathcal{F}'|$.

9.2.4 Complexity Analysis

The initialization step of ABFS is trivial, as it simply instantiates the required structures, which result in $\mathcal{O}(1)$. Naturally, the most computationally intensive part of ABFS is training step. In practice, the upper bound cost of ABFS is given by the loop described by lines 8 to 20 of Algorithm 7, which has a cost of $\mathcal{O}(|U|) = \mathcal{O}(|\mathcal{F}'|)$ which is the number of features selected. Another important part of the computational cost resides in the decision stump training step (line 22), which has to iterate over all features that have not been selected previously, i.e., $\mathcal{O}(|\mathcal{F} \setminus \mathcal{F}'|)$, yet, this procedure occurs only every gp instances, i.e., $\mathcal{O}\left(\frac{|\mathcal{F} \setminus \mathcal{F}'|}{gp}\right)$. As a result, in the training step, the overall complexity in the worst case is of $\mathcal{O}\left(|\mathcal{F}'| + \frac{|\mathcal{F} \setminus \mathcal{F}'|}{gp}\right)$. Finally, the selection step is also simple, as it builds a new instance by iterating over the selected subset of features $\mathcal{F}'$, and thus, with a cost of $\mathcal{O}(|\mathcal{F}'|)$.

9.3 Evaluation

This section analyzes the proposed method in light of the evaluation framework defined in Chapter 4. First, synthetic data is used to assess the impact of different parametrizations of ABFS, followed by a comparison against base learners. Next, ABFS is applied to real-world data, also including a discussion about computational resources and characteristics of the models learned.

Furthermore, due to the lack of techniques that dynamically select features during the processing of data streams, ABFS is compared to a theoretical upper bound hereafter referred to as the "oracle"[3], which always selects the relevant features and ignores the irrelevant ones resulting in $SA = 1$. As a result, every time a change in the relevant subset of features is detected, the classifier is reset. The term ORACLE is used to label this strategy in the following experiments.

[3] The term *oracle* is borrowed from dynamic selection methods in ensemble learning.

9.3.1 Synthetic Data

In this section, the results obtained by different classifiers with and without ABFS in synthetic experiments are reported. In contrast to real-world datasets, synthetic experiments allow greater flexibility. As depicted in the previous section, the target here is the dimensionality aspect of data streams, where 100, 200, and 500 features are appended to each of the experiments. The rationale behind this process is to verify how each learner and ABFS behave when noisy features are added to a data stream regarding accuracy, processing time, and memory consumption.

This section starts by investigating how different values for each of the main parameters of ABFS impact final classification accuracy and selection accuracy rates. Our investigation targets the parameters and values detailed below, whereas each one will be analyzed individually w.r.t. classification accuracy and selection accuracy metrics, and finally, the best parametrization will be chosen as the default one. The parameters analyzed are as follows:

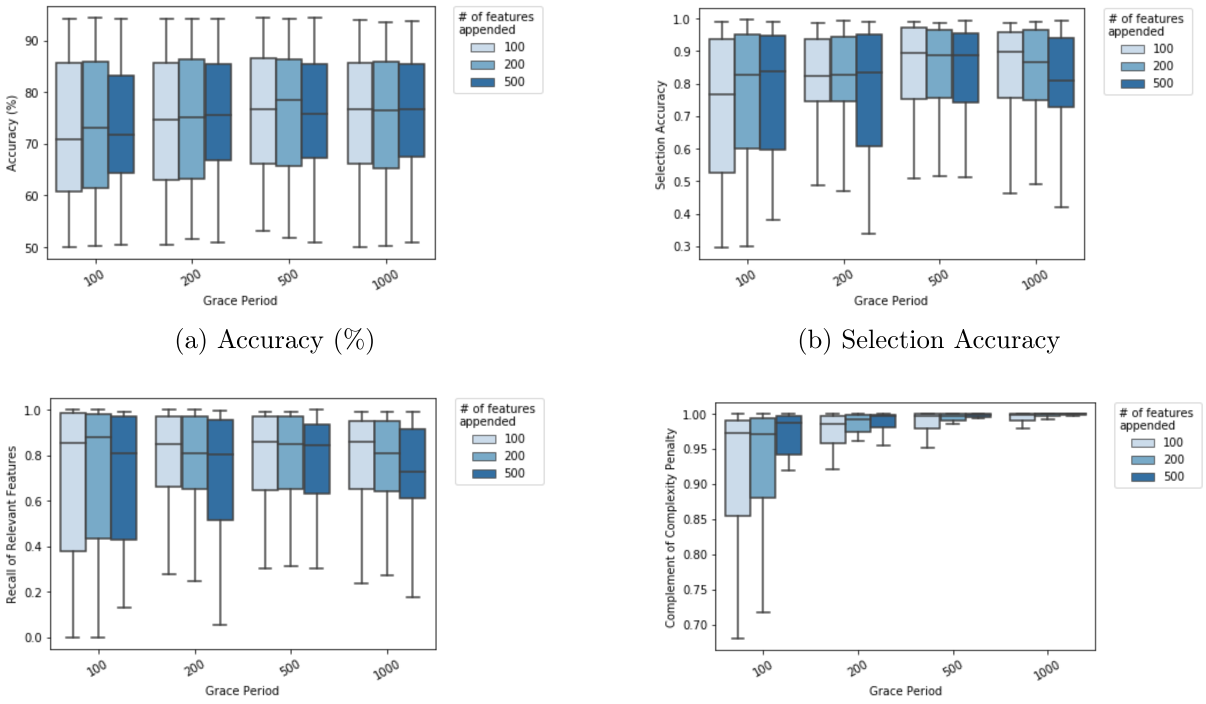
- **Grace period** (gp): This parameter controls how "fast" the candidate decision stump will attempt to select a feature. Smaller values of gp allow the decision stump to branch quicker, yet, the sample distribution obtained during this grace period is expected to be less precise compared to the samples obtained with greater grace periods. The values of 100, 200, 500 and 1,000 were tested for this parameter.

- **Selection threshold** (θ): This parameter determines the minimum value of Ω so that a feature is selected. In practice, if the candidate decision stump determines that f_α is the most appropriate feature to be selected, it will only select it if $\Omega(f_\alpha) \geq \theta$. Three different values were tested for this parameter: 0.01, 0.05 and 0.1.

- **Drift detector** (ψ): this parameter determines which type of drift detector is used in each boosting unit. Three different competitive methods have been tested here, namely ADWIN (BIFET; GAVALDÀ, 2009), HDDM-A and HDDM-W (FRÍAS-BLANCO et al., 2015).

As a result, 36 different configurations for ABFS were tested in association with Naive Bayes, KNN, Hoeffding Tree and Hoeffding Adaptive Tree classifiers, culminating in a total of 144 executions per stream. Hereafter, box-plots are used to report the results obtained across different classifiers, streams, and parameter values.

In Figure 9.1 the results obtained by different grace period values across experiments grouped by the number of irrelevant features appended are reported. Even though



(a) Accuracy (%)

(b) Selection Accuracy

(c) Recall of Relevant Features

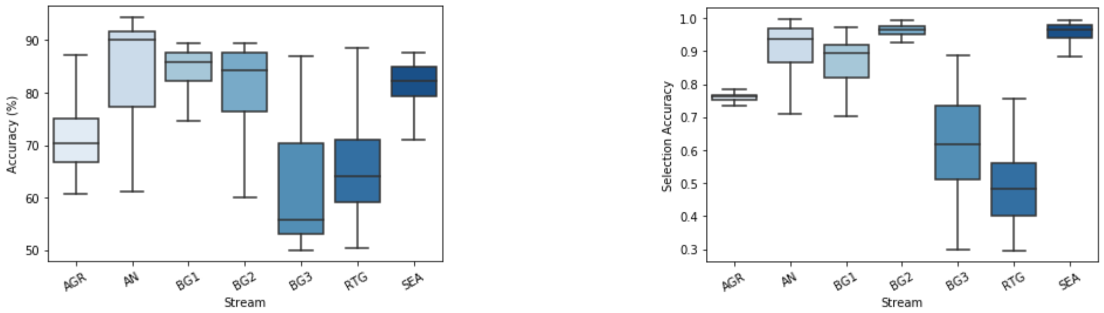
(d) Complement of Complexity Penalty

Figure 9.1: Results obtained across different grace period values.

no clear difference is observable across different grace period concerning accuracy (Figure 9.1a), the highest results are obtained when the grace period is set to either 500 or 1000, showing that higher grace periods are preferable. Nevertheless, the results observed in Figures 9.1b, 9.1c and 9.1d show the results for Selection Accuracy and its components, where $gp = 500$ is the most stable and preferred value regardless of the experiment dimensionality in terms of Selection Accuracy and Recall of Relevant Features.

Naturally, an important aspect here is the high variance observed in the results, as the rates go from 50% up to 90% or more. This high variance occurs mainly because of the BG3 and RTG experiments. If we analyze the classification and selection accuracy rates, depicted in Figures 9.2a and 9.2b, respectively; we can observe that these experiments result in rates that are much lower than the rest. The explanation is that these concepts are much more complex than the others, as BG3 is a XOR-like classification problem (HALL, 2000), and RTG has complex interactions between the features (BARDDAL et al., 2017).

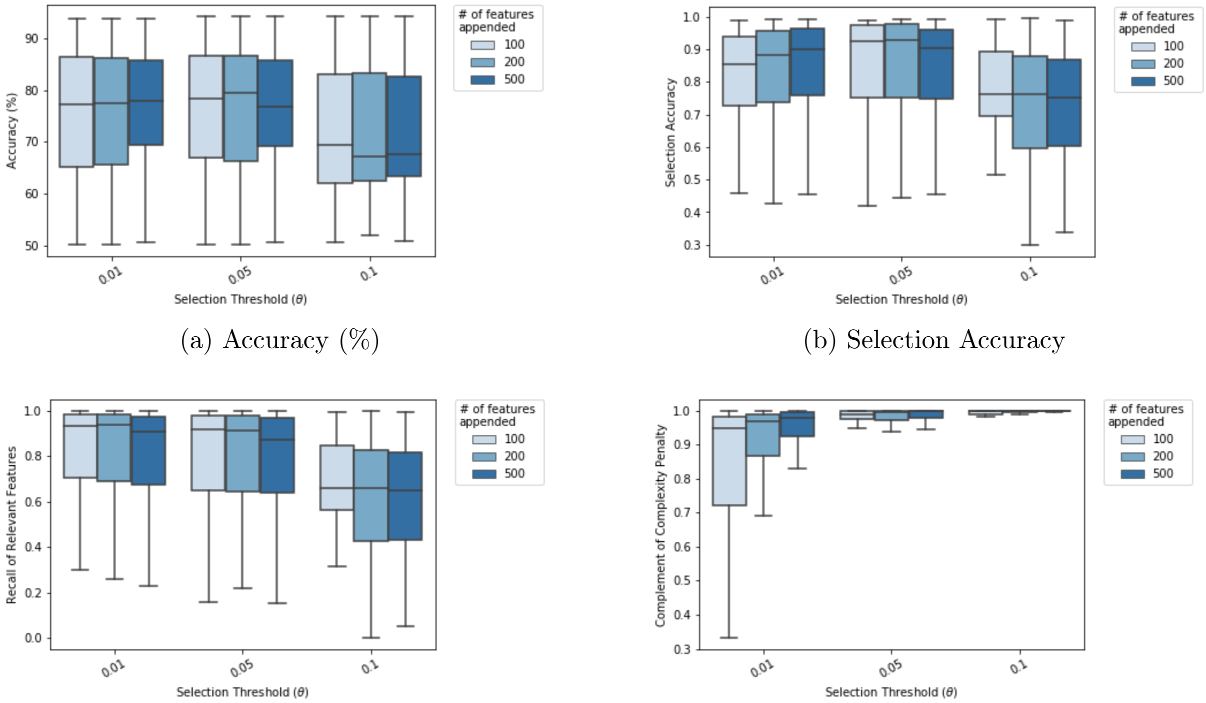
In Figure 9.3 a similar analysis for the selection threshold (θ) parameter is conducted. From the accuracy results shown in Figure 9.3a, the three threshold values behave similarly in terms of variance, yet smaller values, i.e., 0.01 and 0.05, show higher accuracy rates. When analyzing Selection Accuracy rates (Figure 9.3b) and its components (Fig-



(a) Accuracy (%)

(b) Selection Accuracy (%)

Figure 9.2: Classification and selection accuracy rates obtained per experiment.



(a) Accuracy (%)

(b) Selection Accuracy

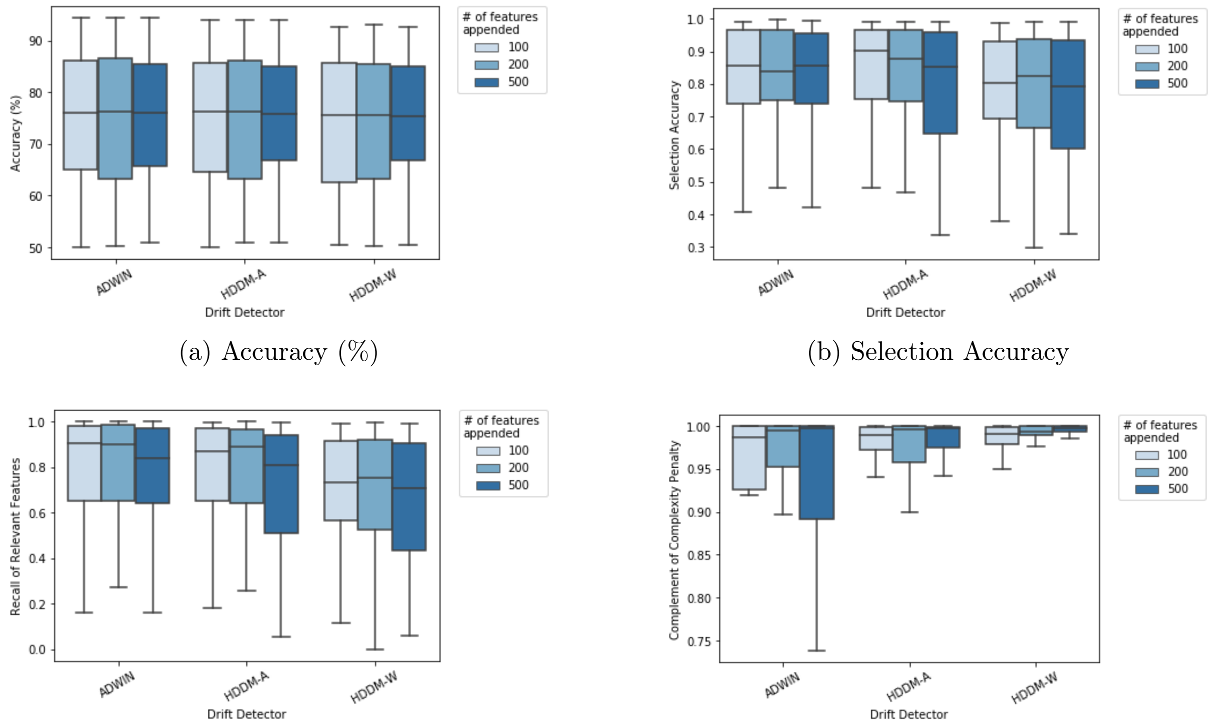
(c) Recall of Relevant Features

(d) Complement of Complexity Penalty

Figure 9.3: Results obtained across different selection threshold (θ) values.

ures 9.3c and 9.3d), we observe a trade-off between θ and the accuracy of the selection process. In practice, higher threshold values are more 'selective' as less irrelevant features are selected (higher Complement of Complexity Penalty rates), while it misses the relevant ones (lower Recall of Relevant Features values). Overall, both $\theta = 0.01$ and $\theta = 0.05$ seem reasonable as they are able to correctly identify relevant features in all the tested dimensionalities (Figure 9.3c), while reasonably ignoring the irrelevant ones (Figure 9.3d).

Finally, the results for different drift detectors are reported in Figure 9.4. Regarding classification accuracy, depicted in Figure 9.4a, the use of different drift detectors barely impact the overall results regardless of the dimensionality of the experiments.



(a) Accuracy (%)

(b) Selection Accuracy

(c) Recall of Relevant Features

(d) Complement of Complexity Penalty

Figure 9.4: Results obtained across different drift detectors.

Yet, when analyzing the results for Selection Accuracy and its components (Figures 9.4b through 9.4d), we observe that HDDM-A is slightly better in overall Selection Accuracy rates, while experiments with ADWIN are better at retaining the relevant features, and HDDM-W is the best performing in terms of ignoring the irrelevant features. It is important to note that even though these drift detectors are not part of the feature selection process, they indirectly impact the entire process, as they may flag drifts at different moments, which cause the feature selection process adapt itself at different regions of the stream. As a result, ABFS becomes more or less precise according to each of the metrics mentioned above depending on the drift detector being used.

Naturally, since the goal of classification is to achieve the highest classification rates possible, Figure 9.5 shows the 10 best-ranked configurations of ABFS. In this figure, we can corroborate the values identified in the previous analyses, as the best performing parametrization, in average, for ABFS in synthetic experiments was ($gp = 500, \theta = 0.01, \psi =$ ADWIN), and this configuration is assumed for comparisons against the base learners and the ORACLE feature selector. It is relevant to highlight, however, that this configuration is not the optimal one for each of the experiments conducted, and thus, these parameters' values must not be assumed to be the result of a tuning process.

Accuracy rates. The accuracy rates obtained by the classifiers without feature

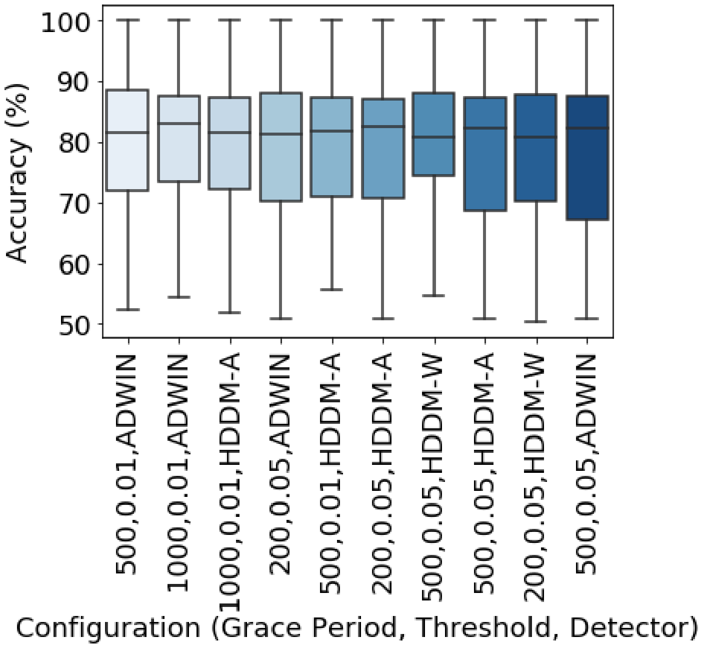


Figure 9.5: Accuracy rates (%) obtained across the 10-best ranked ABFS configurations in synthetic experiments.

selection, with the ORACLE selector and ABFS are reported in Tables 9.1, 9.2 and 9.3. Focusing on the accuracy rates obtained in experiments with 100 irrelevant features, we observe that ABFS can improve the classification rates of the NB, KNN and HT classifiers in all scenarios. In average, the improvements for NB, KNN and HT classifiers are of 7.67%, 11.95%, and 4.64%, respectively. On the other hand, the combination of ABFS with the HAT classifier results in accuracy decreases in most scenarios with an average of -5.76%, which shows that combining two adaptive approaches that concomitantly select features jeopardizes the learning process. It is also important to highlight that ABFS is even able to surpass the ORACLE selector in several experiments. The main exception is the KNN classifier, in which the ABFS selector never surpasses the ORACLE results. Despite counter-intuitive, the ORACLE selector only guarantees that the selection accuracy will be maximum, and not that the classification rates will be maximized. In practice, the relationship between achieving higher selection accuracy and classification rates depicted in Figure 9.6 is not as clear as one would expect. These results show that different learners benefit differently when fed with the same subset of features. For instance, despite irrelevant features being created using uniform distributions across all classes, when analyzed in conjunction with other features, they might still present some predictive power. Here, it is also important to highlight that the fact that most of the results obtained by ABFS in Figure 9.6 are located in regions of high Selection Accuracy and classification accuracy rates, thus showing the efficacy of the proposed method.

Table 9.1: Accuracy (%) obtained by different classifiers and feature selection methods in experiments with 100 irrelevant features. Results in bold highlight the best accuracy rates per classifier and underlined results are the best across learners and selectors.

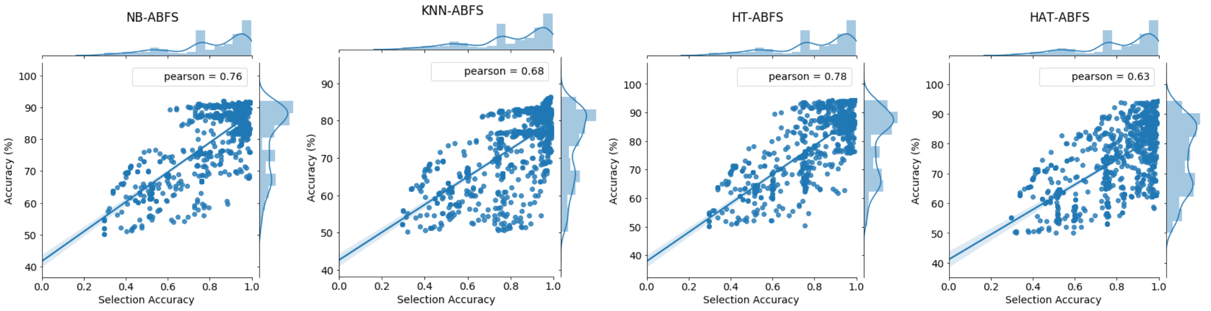
Experiment	NB	NB-ORACLE	NB-ABFS	KNN	KNN-ORACLE	KNN-ABFS	HT	HT-ORACLE	HT-ABFS	HAT	HAT-ORACLE	HAT-ABFS
AGR	67.27	74.72	**76.10**	50.62	**84.38**	73.30	77.38	**86.96**	85.90	**91.15**	86.95	81.38
AN	81.52	88.82	**91.20**	64.57	**74.59**	70.05	92.92	90.86	**93.63**	**94.33**	82.37	93.61
BG1	80.17	**87.06**	86.96	70.94	**80.48**	76.92	86.41	87.09	**88.79**	89.09	74.77	**89.13**
BG2	74.11	86.41	**88.56**	57.63	**82.18**	75.98	79.91	86.34	**88.55**	**88.06**	67.96	84.76
BG3	55.84	56.39	**60.94**	53.11	**74.26**	65.46	70.46	**80.90**	72.17	**85.61**	68.78	67.31
RTG	59.22	65.41	**65.93**	54.57	**67.07**	55.64	66.09	**84.71**	74.48	**88.56**	86.12	80.01
SEA	79.15	**83.49**	81.33	59.14	**81.52**	76.90	84.05	84.71	**86.22**	86.41	84.78	**86.66**

Table 9.2: Accuracy (%) obtained by different classifiers and feature selection methods in experiments with 200 irrelevant features. Results in bold highlight the best accuracy rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ORACLE	NB-ABFS	KNN	KNN-ORACLE	KNN-ABFS	HT	HT-ORACLE	HT-ABFS	HAT	HAT-ORACLE	HAT-ABFS
AGR	67.19	74.72	**75.67**	50.55	**84.38**	61.62	77.53	**86.96**	86.06	**91.07**	86.95	81.39
AN	81.58	88.85	**91.21**	61.15	**74.75**	69.67	92.53	90.99	**93.85**	**94.36**	82.48	93.12
BG1	79.72	**86.89**	86.81	65.87	**80.56**	75.38	85.87	86.98	**88.76**	89.11	74.47	**89.22**
BG2	74.11	86.48	**89.09**	55.41	**82.00**	69.01	79.01	86.41	**89.11**	88.02	68.24	**89.12**
BG3	55.47	55.88	**60.52**	51.90	**74.32**	65.29	68.56	**80.85**	71.11	**86.22**	69.32	78.70
RTG	59.22	**67.02**	66.18	55.78	**65.47**	57.66	63.25	**85.51**	69.98	**91.01**	88.15	84.07
SEA	79.04	83.57	**84.63**	56.79	**81.51**	76.58	82.52	84.55	**86.18**	84.69	84.61	**86.56**

The results obtained in experiments with 200 and 500 irrelevant features, reported in Tables 9.1 and 9.2, follow the same behavior as noticed in Table 9.1, where NB, KNN, and HT classifiers benefit from ABFS, while HAT has its accuracy rates decreased. In quantitative terms, accuracy changes of 8.25% and 7.99% for NB, 11.11% and 9.24% for KNN, 5.11% and 5.21% for HT, -3.19% and -2.35% for HAT, are observed in experiments with 200 and 500 irrelevant features, respectively.

Computational resources. In addition to the comparisons conducted in terms of classification accuracy and selection accuracy, it is also important to verify if the introduction of ABFS in the data stream classification process is not computationally prohibitive, or in the best case scenario, improves the processing time and memory consumption rates



(a) Naive Bayes (b) K-Nearest Neighbors (c) Hoeffding Tree (d) Hoeffding Adaptive Tree

Figure 9.6: Relationship between Selection Accuracy and classification accuracy rates across different classifiers with ABFS. The results plotted in this figure report the rates obtained with different ABFS configurations and stream dimensionalities (100, 200, and 500).

Table 9.3: Accuracy (%) obtained by different classifiers and feature selection methods in experiments with 500 irrelevant features. Results in bold highlight the best accuracy rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ORACLE	NB-ABFS	KNN	KNN-ORACLE	KNN-ABFS	HT	HT-ORACLE	HT-ABFS	HAT	HAT-ORACLE	HAT-ABFS
AGR	66.97	74.72	**75.78**	50.41	**84.38**	52.49	75.60	**86.96**	85.02	**90.80**	86.95	81.44
AN	81.56	88.85	**91.31**	56.99	**74.75**	70.12	92.74	90.99	**93.58**	**94.39**	82.48	93.00
BG1	79.49	86.44	**86.81**	60.18	**80.43**	73.53	85.84	87.02	**88.71**	89.04	74.61	**89.25**
BG2	73.84	86.55	**88.19**	53.28	**82.18**	63.93	77.57	**86.49**	85.79	87.50	67.86	**88.48**
BG3	55.85	56.14	**59.72**	50.85	**74.20**	54.61	67.51	**80.73**	71.91	**85.46**	69.09	83.22
RTG	68.08	**76.92**	76.59	58.84	**73.99**	62.50	70.17	**87.94**	77.81	**89.93**	88.88	82.14
SEA	79.05	**83.54**	82.39	53.90	**81.43**	71.99	82.28	84.80	**85.34**	81.81	84.64	**84.98**

of learners. For the sake of brevity, only the computational resources required for the biggest experiments are compared, i.e., those with 500 irrelevant features, as they are the most computationally intensive. In Table 9.4, the processing times obtained by classifiers both with and without ABFS are reported. From these results, we observe that the introduction of ABFS impacts different learners differently. For instance, NB has its processing times significantly increased in all scenarios, while KNN has the opposite behavior. Regarding KNN, such processing time decreases are expected as the complexity of computing distances between instances with reduced dimensionality are faster than computing distances with the entire set of features. It is also worthy to highlight that even decision trees have their processing times decreased in a handful of scenarios.

Similarly, the results obtained for memory consumption are reported in Table 9.5. Regarding the NB and HAT classifiers, the introduction of ABFS introduces significant overheads in memory consumption rates, while KNN highly benefits from it, as the buffered instances are stored in reduced dimensionality. Next, the results for the HT classifier show that in most cases ABFS does introduce a relatively small overhead, yet, some improvements are also observed.

Finally, it is important to highlight that when analyzing the computational resource metrics mentioned above, the technology in which the method is implemented on is important. As the implementation of ABFS evaluated here has been performed on the Massive Online Analysis framework, it is important to highlight that when a classifier is fed with an instance for training, it still loops over all the original feature set $\mathcal{F}$ and not only over the selected subset $\mathcal{F}'$. As a result, the overall processing times are expected to be incremented, but this behavior may change if the base learners allow sparse data representations. Similarly, the NB, HT and HAT classifiers still instantiate data structures for each of the original features in $\mathcal{F}$ and not only for $\mathcal{F}'$, and as a result, the introduction of ABFS negatively impacts the memory consumption rates of these learners. In practice, this a gap that should be pursued in future works, as the assessment conducted here relies on the MOA implementation, whereas a theoretical evaluation based on the computational cost in terms of asymptotic notation could be done.

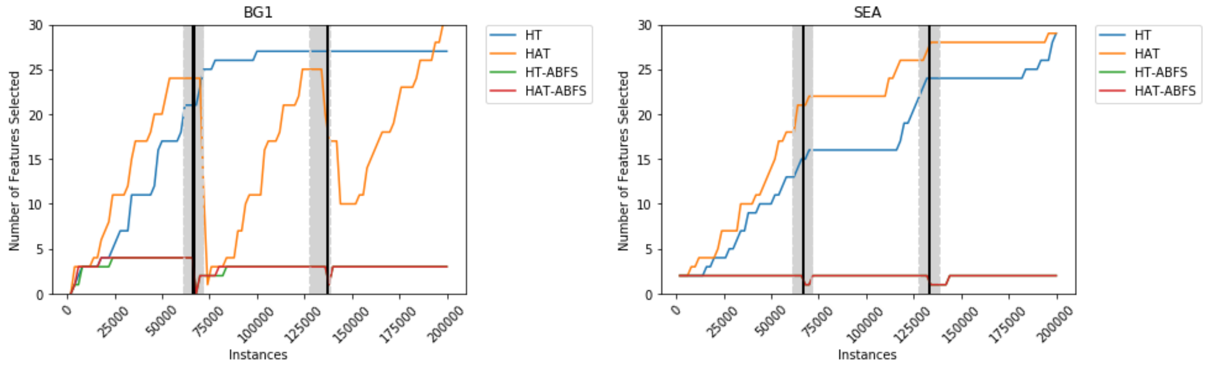
Table 9.4: Processing time (s) obtained by different classifiers and feature selection methods in experiments with 500 irrelevant features. Results in bold highlight the smallest times per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ABFS	KNN	KNN-ABFS	HT	HT-ABFS	HAT	HAT-ABFS
AGR	**32.03**	58.81	818.80	**525.66**	178.27	**98.85**	**165.19**	215.49
AN	**69.53**	117.76	1827.45	**1023.79**	392.34	**267.95**	395.43	**367.15**
BG1	**13.08**	51.10	612.49	**570.14**	**44.27**	75.90	**45.93**	80.55
BG2	**11.55**	30.86	606.38	**387.84**	**45.24**	52.23	**51.92**	59.97
BG3	**12.78**	38.11	596.88	**437.59**	**43.52**	45.00	**50.33**	70.55
RTG	**52.23**	71.95	698.87	**422.32**	187.92	**148.03**	145.52	**117.78**
SEA	**27.01**	28.86	656.02	**259.64**	85.78	**53.86**	195.08	**79.92**

Table 9.5: RAM-Hours (GB-Hour) obtained by different classifiers and feature selection methods in experiments with 500 irrelevant features. Results in bold highlight the smallest memory consumption rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ABFS	KNN	KNN-ABFS	HT	HT-ABFS	HAT	HAT-ABFS
AGR	$\mathbf{1.76 \times 10^{-6}}$	7.82×10^{-4}	6.63×10^{-3}	$\mathbf{1.23 \times 10^{-4}}$	$\mathbf{1.21 \times 10^{-3}}$	1.45×10^{-3}	$\mathbf{4.05 \times 10^{-4}}$	3.68×10^{-3}
AN	$\mathbf{7.87 \times 10^{-6}}$	8.33×10^{-4}	7.04×10^{-3}	$\mathbf{5.32 \times 10^{-4}}$	4.73×10^{-3}	$\mathbf{2.60 \times 10^{-3}}$	$\mathbf{2.65 \times 10^{-3}}$	4.53×10^{-3}
BG1	$\mathbf{6.03 \times 10^{-7}}$	2.40×10^{-4}	2.65×10^{-3}	$\mathbf{9.01 \times 10^{-5}}$	$\mathbf{7.84 \times 10^{-5}}$	4.21×10^{-4}	$\mathbf{3.69 \times 10^{-5}}$	4.04×10^{-4}
BG2	$\mathbf{5.32 \times 10^{-7}}$	8.40×10^{-5}	1.04×10^{-3}	$\mathbf{8.92 \times 10^{-5}}$	$\mathbf{8.53 \times 10^{-5}}$	1.71×10^{-4}	$\mathbf{5.14 \times 10^{-5}}$	1.81×10^{-4}
BG3	$\mathbf{5.90 \times 10^{-7}}$	3.36×10^{-4}	3.77×10^{-3}	$\mathbf{8.80 \times 10^{-5}}$	$\mathbf{7.46 \times 10^{-5}}$	4.21×10^{-4}	$\mathbf{4.47 \times 10^{-5}}$	6.88×10^{-4}
RTG	$\mathbf{2.97 \times 10^{-6}}$	2.69×10^{-4}	1.62×10^{-3}	$\mathbf{1.03 \times 10^{-4}}$	2.24×10^{-3}	$\mathbf{1.32 \times 10^{-3}}$	$\mathbf{4.15 \times 10^{-4}}$	6.58×10^{-4}
SEA	$\mathbf{1.61 \times 10^{-6}}$	5.89×10^{-4}	5.16×10^{-3}	$\mathbf{9.53 \times 10^{-5}}$	$\mathbf{3.13 \times 10^{-4}}$	1.18×10^{-3}	$\mathbf{1.03 \times 10^{-3}}$	1.95×10^{-3}

Number of selected features. To finalize the discussion on synthetic experiments, two examples on the number of selected features over the processing of streams are highlighted. In Figure 9.7 the number of features that were selected by HT and HAT classifiers with and without ABFS in BG1 and SEA experiments are shown. These experiments are targetted as these are cases where the overall accuracy of tree-based learners has improved with ABFS. In Figure 9.7a we can observe that the number of features used by the Hoeffding Tree (HT) classifier continuously increases, while the Hoeffding Adaptive Tree (HAT) can discard features when drifts occur, which are the areas highlighted in the plot. It is important to remember that in this experiment, only 3 features are relevant, and thus, both HT and HAT are rapidly growing and selecting features as new instances become available. In contrast to this behavior, we observe that the same classifiers with ABFS selects up to 4 features and quickly flags and adapts to drifts, which are marked as a vertical line in the plot. A different behavior is observed in Figure 9.7b, where HAT has the same behavior of a conventional incremental HT, as the number of selected features continuously increases, showing the HAT is unable to discard features that become irrelevant after drifts. Again, ABFS shows a limited number of selected features, which result in much smaller decision trees, thus improving their readability and understandability.



(a) BG1

(b) SEA

Figure 9.7: Number of features selected and used by decision tree models with and without ABFS in experiments with 500 irrelevant features. Grayed areas are drifting regions and vertical black lines depict the moments where drifts have been flagged by ABFS. The drift moments for HT-ABFS and HAT-ABFS as ABFS is classifier independent.

9.3.2 Real-world Data

As conducted in the synthetic experiments, the different configurations of ABFS were ranked across all the real-world experiments according to the accuracy rates obtained. The 10 best configurations among the 36 tested are reported in Figure 9.8 with the accuracy results. In contrast to what was observed for synthetic experiments, smaller grace periods combined with the ADWIN drift detector dominate the top positions, and as a result, $gp = 100$, $\theta = 0.05$ and ADWIN are selected as the default configuration for real-world experiments.

In Table 9.6 the accuracy rates obtained by different classifiers with and without ABFS are compared. Here, we note that the NB and HT classifiers benefit from ABFS in all three experiments. The observed increases are relevant as they broaden 9.77% to 32.05% for NB and 3.34% to 5.18% for HT. Regarding KNN, the accuracy in IADS is maintained, while in NOMAO decreases by 0.73%, while increases in 9.60% for SPAM. Similarly, the results for HAT show no difference for IADS, while a decrease of 4.19% is observed in NOMAO and an increase of 1.90% for SPAM. Following the outcome of the Wilcoxon test, both NB and HT classifiers are significantly improved regarding accuracy in these scenarios.

Table 9.6: Accuracy rates (%) obtained by different classifiers and ABFS in real-world experiments. Results in bold are the highest accuracy rates obtained per classifier type.

Experiment	NB	ABFS-NB	KNN	ABFS-KNN	HT	ABFS-HT	HAT	ABFS-HAT
IADS	67.95	**100.00**	**100.00**	**100.00**	92.90	**100.00**	**83.90**	**83.90**
NOMAO	83.86	**93.63**	**95.16**	94.43	91.08	**94.42**	**92.67**	88.48
SPAM	76.86	**88.90**	85.04	**94.64**	83.58	**88.76**	83.60	**85.50**

Table 9.7: Processing times (s) obtained by different classifiers and ABFS in real-world experiments. Results in bold are the smaller rates obtained per classifier type.

Experiment	NB	ABFS-NB	KNN	ABFS-KNN	HT	ABFS-HT	HAT	ABFS-HAT
IADS	**4.07**	9.78	48.86	**28.08**	**5.93**	10.93	**6.81**	12.55
NOMAO	**3.40**	7.83	33.52	**22.98**	**4.97**	10.04	**7.16**	11.17
SPAM	**563.77**	586.37	8074.58	**3062.64**	**686.36**	716.81	**739.13**	1277.11

In Table 9.7 the processing times of the classifiers with and without ABFS are compared. Here, we observe similar behavior to what has been observed for synthetic data, where the processing time rates of all classifiers have increased, except for KNN. Here, the Wilcoxon test showed that ABFS significantly improves the KNN running times, while NB is worsened. The results obtained for the remainder of the classifiers are inconclusive. The memory consumption results, depicted in Table 9.8 show that ABFS also introduces overheads to all classifiers. One exception worthy to mention is that memory consumption of HAT in the IADS experiment, which has significantly decreased, while the accuracy rate was maintained. Again, the Wilcoxon test was used, and its outcomes show that both NB and HT are significantly penalized when combined with ABFS, while the remainder of the classifiers is not.

Finally, it is relevant to highlight the improvements observed in the SPAM experiment, which are important as it is the experiment with the highest dimensionality. In this experiment, all classifiers have their accuracy rates significantly improved (Table 9.6), while their processing time and memory consumption rates decreased (Tables 9.7

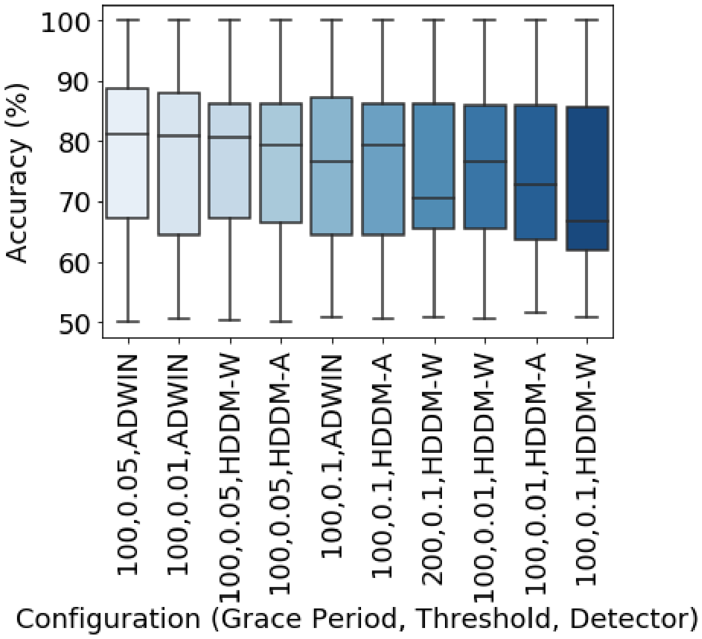


Figure 9.8: Accuracy rates (%) obtained across the 10-best ranked ABFS configurations in real-world experiments.

Table 9.8: RAM-Hours (GB-Hour) obtained by different classifiers and ABFS in real-world experiments. Results in bold are the smaller rates obtained per classifier type.

Experiment	NB	ABFS-NB	KNN	ABFS-KNN	HT	ABFS-HT	HAT	ABFS-HAT
IADS	$\mathbf{7.78 \times 10^{-7}}$	7.25×10^{-4}	$\mathbf{1.48 \times 10^{-4}}$	1.73×10^{-3}	$\mathbf{1.87 \times 10^{-6}}$	7.95×10^{-4}	2.62×10^{-6}	$\mathbf{6.16 \times 10^{-9}}$
NOMAO	$\mathbf{5.59 \times 10^{-8}}$	2.80×10^{-5}	$\mathbf{6.38 \times 10^{-6}}$	5.64×10^{-5}	$\mathbf{7.32 \times 10^{-7}}$	3.02×10^{-5}	$\mathbf{5.70 \times 10^{-7}}$	3.43×10^{-5}
SPAM	$\mathbf{4.09 \times 10^{-3}}$	9.15×10^{-2}	6.32×10^{-1}	$\mathbf{4.82 \times 10^{-1}}$	$\mathbf{8.51 \times 10^{-3}}$	1.26×10^{-1}	$\mathbf{1.28 \times 10^{-2}}$	2.22×10^{-1}

and 9.8). To understand the impact of ABFS in the SPAM experiment, Figure 9.9 shows the number of features selected by ABFS and used by decision tree-based classifiers. Here, we observe that out of the nearly 40 thousand features, and only 16 were used by the HAT alone, while the maximum number of features used by the same classifier with ABFS was 5. A similar behavior can be observed for the HT classifier, which used 10 features, while its version with ABFS used only 5. These results are particularly interesting as it shows that despite the fact that decision trees select a small subset of features to build its predictive model, they can still be further simplified so that their models are smaller and achieve higher generalization rates.

9.4 Stability

The stability scores of ABFS are reported in Figure 9.10. As performed for DISCUSS, the results for bootstrap-, split-, and cross-validation schemes were obtained in a 10-fold validation environment. The results observed here show similar behavior to the one earlier observed in Section 8.4, as the stability results obtained mainly vary according to the validation process adopted. Compared to the results obtained by DISCUSS, the stability rates achieved here are, in average, 1.80% higher for bootstrap-validation, 2.48% higher for cross-validation, and 1.00% higher for split-validation. As stated previously, the results obtained for ABFS are also inconclusive due to the lack of existing techniques for comparison. Therefore, these rates shall be used as a baseline for future comparisons.

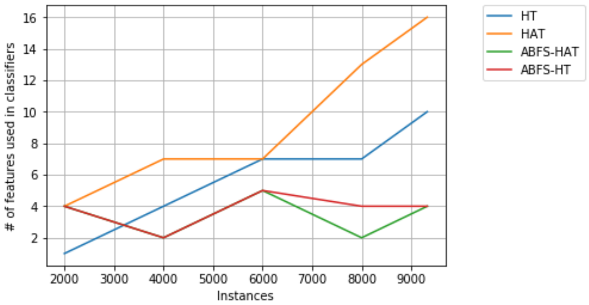


Figure 9.9: Number of features selected and used during the SPAM experiment.

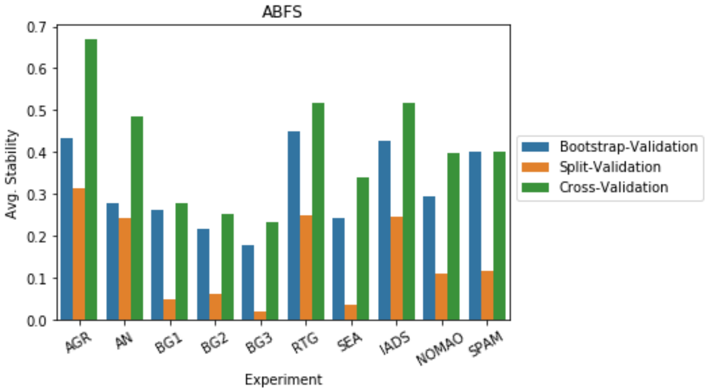


Figure 9.10: Stability results obtained across different experiments.

9.5 Concluding Remarks

In this chapter ABFS was introduced. ABFS is a novel adaptive boosting-based feature selection algorithm for data streams. ABFS includes strategies to select features over data streams and to detect and adapt to concept drifts. The proposed method is classifier-independent, and results show that ABFS can improve all types of classifiers in different scenarios, thus, corroborating the fifth hypothesis of this thesis. Despite performing interesting cuts to the dimensionality of data streams, ABFS still increases the processing time and memory consumption of Bayesian and decision tree-based types of learners. An important exception is the KNN classifier, in which the results show that both processing times and memory consumption rates are improved.

In addition to the proposed method, is is expected that the contributions on feature selection evaluation and the framework added to the Massive Online Analysis software to help in the assessment and comparison of future works in the area. Feature selection-specific metrics, such as Selection Accuracy and Stability have been introduced to streaming scenarios, and the results reported here can be assumed as baselines in future works of the area.

PART III

CONCLUSION AND FUTURE WORK

Chapter 10

Conclusion

Feature drifts is a relevant issue of data streams that should be accounted for by novel learning techniques. This thesis contributes to the machine learning as it proposes data generators (Chapter 4), evaluation and validation schemes (also on Chapter 4), and mainly techniques for handling feature drifting scenarios (Chapters 7, 8, and 9). Regarding techniques for handling feature drifts, it is important to recall the dynamic feature scoring operators proposed in Chapter 6 (Conditional Entropy and Symmetrical Uncertainty), which allow the tracking of features' importance throughout the processing of data streams. Both scoring operators were successfully used as the core of a weighting scheme for the *kNN* and Naive Bayes classifiers, which were later used as custom leaves predictors to boost the already powerful Hoeffding Adaptive Tree (Chapter 7). Next, the Symmetrical Uncertainty scoring operator was used as the core of a dynamic filter for feature selection over data streams (Chapter 8). This filter encompasses two different selection strategies, which were evaluated and uncovered promising results. Between these strategies, thresholding has showed the most promising results, as it allowed simple classifiers to achieve accuracy results that are as high as Hoeffding Adaptive Tree's. As a downside to the aforementioned proposed techniques, results highlighted that our feature scoring operator based on Symmetrical Uncertainty is incapable of correctly depicting the discriminative power of subsets of features since this operator is only capable of depicting the correlation between an attribute individually w.r.t. the class. To overcome this important drawback, an adaptive boosting-based feature selection method called *Adaptive Boosting for Feature Selection* (ABFS) is proposed in Chapter 9. ABFS chains decision stumps and drift detectors, and as a result, it is able to identify which features are relevant throughout the processing of a data stream with reasonable success.

10.1 Future Work

Finally, it is important to highlight that even though this thesis shed light on the feature drift topic and introduced novel methods for tackling it, most of the contributions obtained should be considered as the seed for upcoming research on the field. Below, future works that are planned on data stream mining and feature drifts are layered:

1. Hoeffding trees are an elegant, efficient and robust approach that learns decision trees using constant time per instance and has theoretical guarantees that the convergence between the decision trees learned in streaming and batch fashions basically depend on the sample size. Despite the aforementioned positive characteristics, the original Hoeffding trees have two important drawbacks: (i) they assume that the data distribution is stationary, and (ii) they continuously grow in terms number of selected features and of nodes as new data becomes available, regardless of (i). As observed in several experiments, the Hoeffding Adaptive Tree (BIFET; GAVALDÀ, 2009) is an example of method tailored for tackling issue (i), yet, does not confront issue (ii). This is relevant since Hoeffding trees overfit to data, and they lose the important characteristic of being "white-boxes" in the sense that they become too complex and are no longer human-understandable. As a result, a proposed future work would be to bring forward a regularization scheme for Hoeffding trees with the goal of preventing them from splitting - and consequently growing - unnecessarily as new data becomes available. Regularization is a process that discourages learning algorithms from unnecessary complexity. In decision tree learning, regularization may have many flavors, such as: (i) limiting the maximum depth of the tree, (ii) bagging more than a single tree, or even (iii) setting a stricter stopping criterion (such as a minimum gain function value) to avoid unnecessary splits. A proposal being developed brings regularization to Hoeffding trees by taking into account: (i) a penalty factor that controls the gain obtained by creating a new split using a feature that has not been selected thus far (DENG; RUNGER, 2012), and (ii) the information retained in previous splits to determine whether the gain observed in a leaf is indeed sufficient to justify a new split. Finally, the hypothesis is that these smaller trees will contain a small number of split nodes, and their conjunction will contain the most relevant features from a concept thus helping in the feature selection process for data streams.

2. Random Forests is a family of meta-learners that is recurrently used in several real-world batch machine learning applications. Random Forests are often preferred since

they display high learning performance and nearly zero demand in terms of input preparation and hyper-parameter tuning. Even though Streaming Random Forests exist and were surveyed in this thesis project, the obtained accuracies were not desirable. This can be attributed mainly to the lack of adaptiveness of the ensemble, where nearly no drift detection is performed such as necessary adjustments. In a recent collaboration, a novel Adaptive Random Forest algorithm (ARF) was introduced (GOMES et al., 2017). It combines drift detection, ensemble adjustments, and background learning to improve accuracy rates over concept drifting data streams. As future work, ARF could be investigated in feature drifting scenarios, such as new adaptive subspace selection for branching trees could also be embedded to improve ARF and make it even more robust. Furthermore, ARF could be the cornerstone for the computation of feature scores such as Mean Decrease Impurity (MDI) or Mean Decrease in Accuracy (MDA) (STROBL et al., 2008).

3. One important drawback of the experimental setting adopted in this thesis regards the class distribution of the synthetic experiments. As observed in Chapter 4, all of the synthetic experiments are (nearly) class-balanced, meaning that the probability of observing an instance from each of the classes during the stream is the same. Yet, this is a strong assumption that is unlikely to hold in real-world scenarios, and thus, it is important to verify how the proposed methods behave when applied to scenarios with class imbalance and whether they are able to improve the accuracy rates of classifiers. Furthermore, it is also worthy to verify how the proposed feature selection and weighting schemes behave in scenarios where the ground-truth labels of instances are delayed.

4. Recently, the work of (WEBB et al., 2018) has proposed the use of covariances to identify, track, and most importantly, quantify concept drifts according to their magnitude per feature or globally, i.e., across all features. Even though the computation of covariances is too computationally intensive to performed incrementally, the framework provided by the authors could be used to identify whether concept drifts occur in real-world data, and thus, shedding light on whether the datasets contain feature drifts or not.

5. Finally, the scope of the research towards dimensionality reduction could be expanded. For instance, Principal Components Analysis (GHASHAMI; PERRY; PHILLIPS, 2015; YANG; XU, 2015), Random Projections (PHAM et al., 2017), and Hashing Tricks (LANGFORD; LI; STREHL, 2007) are techniques that are gain-

ing traction in streaming scenarios and that should be analyzed in feature drifting scenarios.

Bibliography

ABDULSALAM, H.; SKILLICORN, D. B.; MARTIN, P. Streaming random forests. In: *Proceedings of the 11th International Database Engineering and Applications Symposium.* Washington, DC, USA: IEEE Computer Society, 2007. (IDEAS '07), p. 225–232. ISBN 0-7695-2947-X.

ABDULSALAM, H.; SKILLICORN, D. B.; MARTIN, P. Classification using streaming random forests. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 23, n. 1, p. 22–36, jan. 2011. ISSN 1041-4347.

AGGARWAL, C. C. *Data Streams: Models and Algorithms (Advances in Database Systems).* Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387287590.

AGGARWAL, C. C. (Ed.). *Data Classification: Algorithms and Applications.* 1st. ed. Boca Raton, Florida: CRC Press, 2014. v. 1. ISBN 978-1-4665-8674-1.

AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. A framework for clustering evolving data streams. In: *Proceedings of the 29th International Conference on Very Large Data Bases.* [S.l.]: VLDB Endowment, 2003. (VLDB '03, v. 29), p. 81–92. ISBN 0-12-722442-4.

AGGARWAL, C. C.; HINNEBURG, A.; KEIM, D. A. On the surprising behavior of distance metrics in high dimensional space. In: BUSSCHE, J. Van den; VIANU, V. (Ed.). *Database Theory ICDT 2001.* [S.l.]: Springer Berlin Heidelberg, 2001, (Lecture Notes in Computer Science, v. 1973). p. 420–434. ISBN 978-3-540-41456-8.

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Database mining: a performance perspective. *Knowledge and Data Engineering, IEEE Transactions on*, v. 5, n. 6, p. 914–925, Dec 1993. ISSN 1041-4347.

AHA, D.; KIBLER, D. Instance-based learning algorithms. *Machine Learning*, v. 6, p. 37–66, 1991.

ALIPPI, C.; BORACCHI, G.; ROVERI, M. Just in time classifiers: Managing the slow drift case. In: *International Joint Conference on Neural Networks, 2009. IJCNN 2009.* [S.l.: s.n.], 2009. p. 114–120. ISSN 1098-7576.

ALMEIDA, E.; FERREIRA, C. A.; GAMA, J. a. Adaptive model rules from data streams. In: *ECML/PKDD (1)*. [S.l.]: Springer, 2013. (Lecture Notes in Computer Science, v. 8188), p. 480–492. ISBN 978-3-642-40987-5.

ALMEIDA, E.; KOSINA, P.; GAMA, J. Random rules from data streams. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*. [S.l.: s.n.], 2013. p. 813–814.

APPEL, R.; PERONA, P. A simple multi-class boosting framework with theoretical guarantees and empirical proficiency. In: PRECUP, D.; TEH, Y. W. (Ed.). *Proceedings of the 34th International Conference on Machine Learning*. International Convention Centre, Sydney, Australia: PMLR, 2017. (Proceedings of Machine Learning Research, v. 70), p. 186–194.

BACCIANELLA, S.; ESULI, A.; SEBASTIANI, F. Feature selection for ordinal regression. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010. (SAC '10), p. 1748–1754. ISBN 978-1-60558-639-7.

BARBARÁ, D. Requirements for clustering data streams. *SIGKDD Explor. Newsl.*, ACM, New York, NY, USA, v. 3, n. 2, p. 23–27, jan. 2002. ISSN 1931-0145.

BARDDAL, J. P.; ENEMBRECK, F.; GOMES, H. M.; BIFET, A.; PFAHRINGER, B. Merit-guided dynamic feature selection filter for data streams. *Expert Systems with Applications*, v. 116, p. 227 – 242, 2019. ISSN 0957-4174.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F. Advances on concept drift detection detection in regression tasks using social networks theory. *International Journal on Natural Computing Research*, p. 1–14, 2015.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F. Analyzing the impact of feature drifts in streaming learning. In: *Neural Information Processing*. Cham: Springer International Publishing, 2015. p. 21–28. ISBN 978-3-319-26532-2.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F. A survey on feature drift adaptation. In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2015. p. 1053–1060. ISSN 1082-3409.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F.; PFAHRINGER, B.; BIFET, A. On dynamic feature weighting for feature drifting data streams. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II*. [S.l.: s.n.], 2016. p. 129–144.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F.; PFAHRINGER, B. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, v. 127, p. 278 – 294, 2017. ISSN 0164-1212.

BARDDAL, J. P.; GOMES, H. M.; GRANATYR, J.; JR., A. de S. B.; ENEMBRECK, F. Overcoming feature drifts via dynamic feature weighted k-nearest neighbor learning. In: *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*. [S.l.: s.n.], 2016. p. 2186–2191.

BARDDAL, J. P.; GOMES, H. M.; JR., A. de S. B.; ENEMBRECK, F. A benchmark of classifiers on feature drifting data streams. In: *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*. [S.l.: s.n.], 2016. p. 2180–2185.

BARRON, A. R.; RISSANEN, J.; YU, B. The Minimum Description Length Principle in Coding and Modeling. *IEEE Trans. Inf. Theory*, v. 44, n. 6, p. 2743–2760, 1998.

BIFET, A. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. [S.l.]: IOS Press, 2010. v. 207. 1-212 p. (Frontiers in Artificial Intelligence and Applications, v. 207). ISBN 978-1-60750-090-2.

BIFET, A.; GAVALDÀ, R. Learning from time-changing data with adaptive windowing. In: *In SIAM International Conference on Data Mining*. [S.l.: s.n.], 2007.

BIFET, A.; GAVALDÀ, R. Adaptive learning from evolving data streams. In: ______. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 249–260. ISBN 978-3-642-03915-7.

BIFET, A.; HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. MOA: Massive online analysis. *The Journal of Machine Learning Research*, v. 11, p. 1601–1604, 2010.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: BALCáZAR, J. L.; BONCHI, F.; GIONIS, A.; SEBAG, M. (Ed.). *Machine Learning and Knowledge Discovery in Databases*. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6321). p. 135–150. ISBN 978-3-642-15879-7.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; GAVALDÀ, R. Improving adaptive bagging methods for evolving data streams. In: *Proceedings of the 1st Asian Conference on Machine Learning*. [S.l.]: Springer, 2009. (Nanjing, China).

BIFET, A.; MORALES, G. de F.; READ, J.; HOLMES, G.; PFAHRINGER, B. Efficient online evaluation of big data stream classifiers. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2015. (KDD '15), p. 59–68. ISBN 978-1-4503-3664-2.

BIFET, A.; READ, J.; ZLIOBAITE, I.; PFAHRINGER, B.; HOLMES, G. Pitfalls in benchmarking data stream classification and how to avoid them. In: *ECML/PKDD (1)*. [S.l.: s.n.], 2013. p. 465–479.

BIFET EIBE FRANK, G. H. A.; PFAHRINGER, B. (Ed.). *Accurate Ensembles for Data Streams: Combining Restricted Hoeffding Trees using Stacking*, v. 13 de *JMLR Proceedings*, (JMLR Proceedings, v. 13). [S.l.]: JMLR.org, 2010.

BRAVO-MARQUEZ, F.; FRANK, E.; PFAHRINGER, B. From unlabelled tweets to twitter-specific opinion words. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 2015. (SIGIR '15), p. 743–746. ISBN 978-1-4503-3621-5.

BREIMAN, L. Bagging predictors. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 24, n. 2, p. 123–140, ago. 1996. ISSN 0885-6125.

BREIMAN, L. Random forests. *Machine Learning*, Kluwer Academic Publishers, v. 45, n. 1, p. 5–32, 2001. ISSN 0885-6125.

CANDILLIER, L.; LEMAIRE, V. Design and analysis of the nomao challenge active learning in the real-world. In: *Proceedings of the ALRA: Active Learning in Real-world Applications, Workshop ECML-PKDD*. [S.l.: s.n.], 2012.

CAO, F.; ESTER, M.; QIAN, W.; ZHOU, A. Density-based clustering over an evolving data stream with noise. In: *SDM*. [S.l.: s.n.], 2006. p. 328–339.

CARVALHO, V. R.; COHEN, W. W. Single-pass online learning: Performance, voting schemes and online feature selection. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2006. (KDD '06), p. 548–553. ISBN 1-59593-339-5.

CATTRAL, R.; OPPACHER, F.; DEUGO, D. Evolutionary data mining with automatic rule generalization. In: *Recent Advances in Computers, Computing and Communications*. [S.l.: s.n.], 2002. p. 296–300.

CHANDRASHEKAR, G.; SAHIN, F. A survey on feature selection methods. *Computers & Electrical Engineering*, v. 40, n. 1, p. 16 – 28, 2014. ISSN 0045-7906. 40th-year commemorative issue.

CHEN, L.; WANG, S. Automated feature weighting in naive bayes for high-dimensional data classification. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 2012. (CIKM '12), p. 1243–1252. ISBN 978-1-4503-1156-4.

CHEN, X. An improved branch and bound algorithm for feature selection. *Pattern Recognition Letters*, v. 24, n. 12, p. 1925 – 1933, 2003. ISSN 0167-8655.

CIOS, K.; PEDRYCZ, W.; SWINIARSKI, R.; KURGAN, L. *Data Mining: A Knowledge Discovery Approach*. [S.l.]: Springer US, 2007. ISBN 9780387367958.

DAS, S. Filters, wrappers and a boosting-based hybrid for feature selection. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. (ICML '01), p. 74–81. ISBN 1-55860-778-1.

DEMSAR, J. Statistical comparisons of classifiers over multiple datasets. *J. Mach. Learn. Res.*, JMLR.org, v. 7, p. 1–30, dez. 2006. ISSN 1532-4435.

DENG, H.; RUNGER, G. Feature selection via regularized trees. In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2012. p. 1–8. ISSN 2161-4393.

DENIL, M.; MATHESON, D.; FREITAS, N. de. Consistency of online random forests. In: *ICML (3)*. [S.l.]: JMLR.org, 2013. (JMLR Proceedings, v. 28), p. 1256–1264.

DOMINGOS, P. A few useful things to know about machine learning. *Commun. ACM*, ACM, New York, NY, USA, v. 55, n. 10, p. 78–87, out. 2012. ISSN 0001-0782.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2000. (KDD '00), p. 71–80. ISBN 1-58113-233-6.

DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification*. [S.l.]: Wiley-Interscience Publication, 2001.

ENEMBRECK, F.; ÁVILA, B. C.; SCALABRIN, E. E.; BARTHÈS, J.-P. A. Learning drifting negotiations. *Applied Artificial Intelligence*, v. 21, n. 9, p. 861–881, 2007.

FERRER-TROYANO, F. J.; AGUILAR-RUIZ, J. S.; SANTOS, J. C. R. Incremental rule learning and border examples selection from numerical data streams. *J. UCS*, v. 11, n. 8, p. 1426–1439, 2005.

FONG, S.; WONG, R.; VASILAKOS, A. V. Accelerated pso swarm search feature selection for data stream mining big data. *IEEE Transactions on Services Computing*, v. 9, n. 1, p. 33–45, Jan 2016. ISSN 1939-1374.

FREUND, Y.; SCHAPIRE, R. E. Large margin classification using the perceptron algorithm. *Machine Learning*, v. 37, n. 3, p. 277–296, 1999. ISSN 1573-0565.

FRÍAS-BLANCO, I.; CAMPO-ÁVILA, J. D.; RAMOS-JIMENEZ, G.; MORALES-BUENO, R.; ORTIZ-DIAZ, A.; CABALLERO-MOTA, Y. Online and non-parametric drift detection methods based on hoeffding bounds. *IEEE Transactions on Knowledge and Data Engineering*, v. 27, n. 3, p. 810–823, March 2015. ISSN 1041-4347.

FRIEDMAN, M. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, American Statistical Association, v. 32, n. 200, p. 675–701, dez. 1937. ISSN 01621459.

GALELLI, S.; HUMPHREY, G. B.; MAIER, H. R.; CASTELLETTI, A.; DANDY, G. C.; GIBBS, M. S. An evaluation framework for input variable selection algorithms for environmental data-driven models. *Environmental Modelling & Software*, v. 62, p. 33 – 51, 2014. ISSN 1364-8152.

GAMA, J. *Knowledge Discovery from Data Streams*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2010. ISBN 1439826110, 9781439826119.

GAMA, J.; KOSINA, P. Learning decision rules from data streams. In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. [S.l.: s.n.], 2011. p. 1255–1260.

GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. Learning with drift detection. In: BAZZAN, A.; LABIDI, S. (Ed.). *Advances in Artificial Intelligence – SBIA 2004*. [S.l.]: Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 3171). p. 286–295. ISBN 978-3-540-23237-7.

GAMA, J.; PINTO, C. Discretization from data streams: Applications to histograms and data mining. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2006. (SAC '06), p. 662–667. ISBN 1-59593-108-2.

GAMA, J.; RODRIGUES, P. Issues in evaluation of stream learning algorithms. In: ACM SIGKDD. *Proc. of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.* [S.l.], 2009. p. 329–338.

GAMA, J.; ZLIOBAITE, I.; BIFET, A.; PECHENIZKIY, M.; BOUCHACHIA, A. A survey on concept drift adaptation. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 46, n. 4, p. 44:1–44:37, mar. 2014. ISSN 0360-0300.

GHASHAMI, M.; PERRY, D. J.; PHILLIPS, J. M. Streaming kernel principal component analysis. *CoRR*, abs/1512.05059, 2015.

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

GONÇALVES, P. M.; SANTOS, S. G. de C.; BARROS, R. S.; VIEIRA, D. C. A comparative study on concept drift detectors. *Expert Systems with Applications*, v. 41, n. 18, p. 8144 – 8156, 2014. ISSN 0957-4174.

GUHA, S. Tight results for clustering and summarizing data streams. In: *Proceedings of the 12th International Conference on Database Theory.* New York, NY, USA: ACM, 2009. (ICDT '09), p. 268–275. ISBN 978-1-60558-423-2.

GUYON, I. An introduction to variable and feature selection. *Journal of Machine Learning Research*, v. 3, p. 1157–1182, 2003.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, ACM, New York, NY, USA, v. 11, n. 1, p. 10–18, nov. 2009. ISSN 1931-0145.

HALL, M. A. Correlation-based feature selection for discrete and numeric class machine learning. In: *Proceedings of the Seventeenth International Conference on Machine Learning.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (ICML '00), p. 359–366. ISBN 1-55860-707-2.

HALL, M. A.; SMITH, L. A. *Feature Selection for Machine Learning: Comparing a Correlation-based Filter Approach to the Wrapper.* 1999.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques.* 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123814790, 9780123814791.

HOARE, C. A. R. Algorithm 64: Quicksort. *Commun. ACM*, ACM, New York, NY, USA, v. 4, n. 7, p. 321–, jul. 1961. ISSN 0001-0782.

HOEFFDING, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, v. 58, n. 301, p. 13–30, March 1963.

HOENS, T. R.; CHAWLA, N. V.; POLIKAR, R. Heuristic updatable weighted random subspaces for non-stationary environments. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. [S.l.: s.n.], 2011. p. 241–250. ISSN 1550-4786.

HOI, S. C. H.; WANG, J.; ZHAO, P.; JIN, R. Online feature selection for mining big data. In: *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. New York, NY, USA: ACM, 2012. (BigMine '12), p. 93–100. ISBN 978-1-4503-1547-0.

HU, H.; SUN, W.; VENKATRAMAN, A.; HEBERT, M.; BAGNELL, J. A. Gradient boosting on stochastic data streams. *CoRR*, abs/1703.00377, 2017.

HUANG, D. T. J.; KOH, Y. S.; DOBBIE, G.; BIFET, A. Drift detection using stream volatility. In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I*. [S.l.]: Springer International Publishing, 2015. p. 417–432. ISBN 978-3-319-23528-8.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2001. (KDD '01), p. 97–106. ISBN 1-58113-391-X.

HUNTER, J.; COLLEY, M. Feature extraction from sensor data streams for real-time human behaviour recognition. In: KOK, J.; KORONACKI, J.; MANTARAS, R. Lopez de; MATWIN, S.; MLADENIč, D.; SKOWRON, A. (Ed.). *Knowledge Discovery in Databases: PKDD 2007*. [S.l.]: Springer Berlin Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4702). p. 115–126. ISBN 978-3-540-74975-2.

IKONOMOVSKA, E.; GAMA, J.; ZENKO, B.; DZEROSKI, S. Speeding-up hoeffding-based regression trees with options. In: *ICML*. [S.l.: s.n.], 2011. p. 537–544.

JIANG, N.; GRUENWALD, L. Cfi-stream: Mining closed frequent itemsets in data streams. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2006. (KDD '06), p. 592–597. ISBN 1-59593-339-5.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Dynamic feature space and incremental feature selection for the classification of textual data streams. In: *in ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams. 2006*. [S.l.]: Springer Verlag, 2006. p. 107.

KOHAVI, R.; JOHN, G. H. Wrappers for feature subset selection. *Artificial Intelligence*, v. 97, n. 1–2, p. 273 – 324, 1997. ISSN 0004-3702. Relevance.

KOLTER, J. Z.; MALOOF, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.*, JMLR.org, v. 8, p. 2755–2790, dez. 2007. ISSN 1532-4435.

KOSINA, P.; GAMA, J. Very fast decision rules for multi-class problems. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2012. (SAC '12), p. 795–800. ISBN 978-1-4503-0857-1.

KRANEN, P.; ASSENT, I.; BALDAUF, C.; SEIDL, T. The clustree: Indexing micro-clusters for anytime stream mining. *Knowl. Inf. Syst.*, Springer-Verlag New York, Inc., New York, NY, USA, v. 29, n. 2, p. 249–272, nov. 2011. ISSN 0219-1377.

KUNCHEVA, L. I. A stability index for feature selection. In: *Proceedings of the 25th Conference on Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications*. Anaheim, CA, USA: ACTA Press, 2007. (AIAP'07), p. 390–395.

KUNCHEVA, L. I.; FAITHFULL, W. J. PCA feature extraction for change detection in multidimensional unlabelled data. *IEEE Transactions on Neural Networks and Learning Systems*, v. 25, n. 1, p. 69–80, 2014.

KUSHMERICK, N. Learning to remove internet advertisements. In: ACM. *Proceedings of the third annual conference on Autonomous Agents*. [S.l.], 1999. p. 175–181.

LANGFORD, J.; LI, L.; STREHL, A. *Vowpal Wabbit*. 2007.

LI, H.-G.; WU, X.; LI, Z.; DING, W. Online group feature selection from feature streams. In: DESJARDINS, M.; LITTMAN, M. L. (Ed.). *AAAI*. [S.l.]: AAAI Press, 2013. ISBN 978-1-57735-615-8.

LIU, H.; MOTODA, H. *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. [S.l.]: Chapman & Hall/CRC, 2007. ISBN 1584888784.

LIU, J.; XU, G.-S.; XIAO, D.; GU, L.; NIU, X. A semi-supervised ensemble approach for mining data streams. *JCP*, v. 8, n. 11, p. 2873–2879, 2013.

LYON, R. J.; BROOKE, J. M.; KNOWLES, J. D.; STAPPERS, B. W. Hellinger distance trees for imbalanced streams. In: *Proceedings of the 2014 22Nd International Conference on Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2014. (ICPR '14), p. 1969–1974. ISBN 978-1-4799-5209-0.

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, KDD 2018, London, UK, August 19-23, 2018*. [S.l.: s.n.], 2018. p. 1953–1962.

MASUD, M. M.; GAO, J.; KHAN, L.; HAN, J.; THURAISINGHAM, B. M. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Trans. Knowl. Data Eng.*, v. 23, n. 6, p. 859–874, 2011.

METWALLY, A.; AGRAWAL, D.; ABBADI, A. E. Duplicate detection in click streams. In: *Proceedings of the 14th International Conference on World Wide Web*. New York, NY, USA: ACM, 2005. (WWW '05), p. 12–21. ISBN 1-59593-046-9.

MIAO, Q.; CAO, Y.; SONG, J.; LIU, J.; QUAN, Y. Boostfs: A boosting-based irrelevant feature selection algorithm. *IJPRAI*, v. 29, n. 7, 2015.

MIAO, Q.; CAO, Y.; XIA, G.; GONG, M.; LIU, J.; SONG, J. Rboost: Label noise-robust boosting algorithm based on a nonconvex loss function and the numerically stable base learners. *IEEE Transactions on Neural Networks and Learning Systems*, v. 27, n. 11, p. 2216–2228, Nov 2016. ISSN 2162-237X.

MOLINA, L.; BELANCHE, L.; NEBOT, A. Feature selection algorithms: a survey and experimental evaluation. In: *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. [S.l.: s.n.], 2002. p. 306–313.

MORALES, G. D. F.; BIFET, A. SAMOA: Scalable advanced massive online analysis. *Journal of Machine Learning Research*, v. 16, p. 149–153, 2015.

NAIDU, K.; DHENGE, A.; WANKHADE, K. Feature selection algorithm for improving the performance of classification: A survey. In: *Proceedings of the 2014 Fourth International Conference on Communication Systems and Network Technologies*. Washington, DC, USA: IEEE Computer Society, 2014. (CSNT '14), p. 468–471. ISBN 978-1-4799-3070-8.

NEMENYI, P. *Distribution-free multiple comparisons.* Tese (Doutorado), New Jersey, USA, 1963.

NGUYEN, H.-L.; WOON, Y.-K.; NG, W.-K. A survey on data stream clustering and classification. *Knowledge and Information Systems*, Springer London, p. 1–35, 2014. ISSN 0219-1377.

NGUYEN, H.-L.; WOON, Y.-K.; NG, W.-K.; WAN, L. Heterogeneous ensemble for feature drifts in data streams. In: TAN, P.-N.; CHAWLA, S.; HO, C.; BAILEY, J. (Ed.). *Advances in Knowledge Discovery and Data Mining.* [S.l.]: Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7302). p. 1–12. ISBN 978-3-642-30219-0.

NOGUEIRA, S.; BROWN, G. Measuring the stability of feature selection. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II.* [S.l.: s.n.], 2016. p. 442–457.

OES, T. F. C.; HRUSCHKA, E. R.; CASTRO, L. N. de; SANTOS, A. M. A cluster-based feature selection approach. In: *Hybrid Artificial Intelligence Systems.* [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5572). p. 169–176. ISBN 978-3-642-02318-7.

OZA, N. Online bagging and boosting. In: *Systems, Man and Cybernetics, 2005 IEEE International Conference on.* [S.l.: s.n.], 2005. v. 3, p. 2340–2345 Vol. 3.

PARK, C. H. A feature selection method using hierarchical clustering. In: *Mining Intelligence and Knowledge Exploration.* [S.l.]: Springer International Publishing, 2013, (Lecture Notes in Computer Science, v. 8284). p. 1–6. ISBN 978-3-319-03843-8.

PELOSSOF, R.; JONES, M.; VOVSHA, I.; RUDIN, C. Online coordinate boosting. In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops.* [S.l.: s.n.], 2009. p. 1354–1361.

PHAM, X. C.; DANG, M. T.; DINH, S. V.; HOANG, S.; NGUYEN, T. T.; LIEW, A. W. C. Learning from data stream based on random projection and hoeffding tree classifier. In: *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA).* [S.l.: s.n.], 2017. p. 1–8.

RIVEST, R. L. Learning decision lists. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 2, n. 3, p. 229–246, nov. 1987. ISSN 0885-6125.

RODRIGUES, P.; GAMA, J.; PEDROSO, J. Hierarchical clustering of time-series data streams. *Knowledge and Data Engineering, IEEE Transactions on*, v. 20, n. 5, p. 615–627, May 2008. ISSN 1041-4347.

ROSS, G. J.; ADAMS, N. M.; Tasoulis, D. K.; HAND, D. J. Exponentially Weighted Moving Average Charts for Detecting Concept Drift. *ArXiv e-prints*, dez. 2012.

RUDNICKI, W. R.; WRZESIEŃ, M.; PAJA, W. All relevant feature selection methods and applications. In: *Feature Selection for Data and Pattern Recognition*. [S.l.]: Springer Berlin Heidelberg, 2015, (Studies in Computational Intelligence, v. 584). p. 11–28. ISBN 978-3-662-45619-4.

SAMMUT, C.; WEBB, G. I. *Encyclopedia of Machine Learning*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. ISBN 0387307680, 9780387307688.

SCHAPIRE, R. E. The strength of weak learnability. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 5, n. 2, p. 197–227, jul. 1990. ISSN 0885-6125.

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

SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, v. 27, n. 3, p. 379–423, July 1948. ISSN 0005-8580.

SIDHU, M. B. P. Empirical support for concept drifting approaches: Results based on new performance metrics. *International Journal of Intelligent Systems and Applications (IJISA)*, v. 7, n. 6, p. 1–20, 2015.

SILVA, J. A.; FARIA, E. R.; BARROS, R. C.; HRUSCHKA, E. R.; CARVALHO, A. C. P. L. F. d.; GAMA, J. Data stream clustering: A survey. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 46, n. 1, p. 13:1–13:31, jul. 2013. ISSN 0360-0300.

SOVDAT, B. Updating formulas and algorithms for computing entropy and gini index on time-changing data streams. *CoRR*, abs/1403.6348, 2014.

STREET, W. N.; KIM, Y. A streaming ensemble algorithm (sea) for large-classification. In: ACM SIGKDD. *Proc. of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2001. p. 377–382.

STROBL, C.; BOULESTEIX, A.-L.; KNEIB, T.; AUGUSTIN, T.; ZEILEIS, A. Conditional variable importance for random forests. *BMC Bioinformatics*, v. 9, n. 1, p. 307, Jul 2008. ISSN 1471-2105.

TSYMBAL, A. *The problem of concept drift: definitions and related work*. Dublin, Ireland, 2004.

TURKOV, P.; KRASOTKINA, O.; MOTTL, V.; SYCHUGOV, A. Feature selection for handling concept drift in the data stream classification. In: _____. *Machine Learning and Data Mining in Pattern Recognition: 12th International Conference, MLDM 2016, New York, NY, USA, July 16-21, 2016, Proceedings*. [S.l.]: Springer International Publishing, 2016. p. 614–629. ISBN 978-3-319-41920-6.

WANG, B.; PINEAU, J. Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering*, v. 28, n. 12, p. 3353–3366, Dec 2016. ISSN 1041-4347.

WANKHADE, K.; HASAN, T.; THOOL, R. A survey: Approaches for handling evolving data streams. In: *Communication Systems and Network Technologies (CSNT), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 621–625.

WEBB, G. I.; LEE, L. K.; GOETHALS, B.; PETITJEAN, F. Analyzing concept drift and shift from sample data. *Data Mining and Knowledge Discovery*, 2018.

WIDMER, G.; KUBAT, M. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 23, n. 1, p. 69–101, abr. 1996. ISSN 0885-6125.

WILCOXON, F. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, International Biometric Society, v. 1, n. 6, p. 80–83, dez. 1945. ISSN 00994987.

WIRTH, N. A plea for lean software. *Computer*, v. 28, n. 2, p. 64–68, Feb 1995. ISSN 0018-9162.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. ISBN 0120884070.

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

XU, Z.; HUANG, G.; WEINBERGER, K. Q.; ZHENG, A. X. Gradient boosted feature selection. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2014. (KDD '14), p. 522–531. ISBN 978-1-4503-2956-9.

YAN, J.; ZHANG, B.; LIU, N.; YAN, S.; CHENG, Q.; FAN, W.; YANG, Q.; XI, W.; CHEN, Z. Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing. *IEEE Transactions on Knowledge and Data Engineering*, v. 18, n. 3, p. 320–333, March 2006. ISSN 1041-4347.

YANG, W.; XU, H. Streaming sparse principal component analysis. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. [S.l.]: JMLR.org, 2015. (ICML'15), p. 494–503.

YIN, C.; FENG, L.; MA, L.; WANG, J.; YIN, Z.; KIM, J. U. A feature selection algorithm of dynamic data-stream based on hoeffding inequality. In: *2015 4th International Conference on Advanced Information Technology and Sensor Application (AITS)*. [S.l.: s.n.], 2015. p. 92–95.

YU, L.; LIU, H. Feature selection for high-dimensional data: A fast correlation-based filter solution. In: *Proceedings of the Twentieth International Conference on Machine Learning*. [S.l.]: AAAI Press, 2003. p. 856–863.

YU, L.; LIU, H. Efficient feature selection via analysis of relevance and redundancy. *J. Mach. Learn. Res.*, v. 5, p. 1205–1224, dez. 2004. ISSN 1532-4435.

YUAN, Y.; HREBIEN, L.; KAM, M. Speed up svm-rfe procedure using margin distribution. In: *Machine Learning for Signal Processing, 2005 IEEE Workshop on*. [S.l.: s.n.], 2005. p. 297 –302.

ZHAO, Z.; MORSTATTER, F.; SHARMA, S.; ALELYANI, S.; ANAND, A.; LIU, H. Advancing feature selection research. *ASU feature selection repository*, p. 1–28, 2010.

ZHOU, D.; HUANG, J.; SCHÖLKOPF, B. Learning from labeled and unlabeled data on a directed graph. In: *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*. [S.l.: s.n.], 2005. p. 1036–1043.

ŽLIOBAITĖ, I.; BIFET, A.; READ, J.; PFAHRINGER, B.; HOLMES, G. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, v. 98, n. 3, p. 455–482, 2014. ISSN 1573-0565.

Appendix A

Drift Detectors

This appendix describes the functioning of the drift detectors used throughout this thesis, namely the: (i) Exponentially Weighted Moving Average Control Charts (ECDD), (ii) Adaptive Sliding Window Algorithm (ADWIN), and the Hoeffding-based Drift Detection Methods (HDDM-A and HDDM-W).

A.1 Exponentially Weighted Moving Average Control Charts (ECDD)

The Exponentially Weighted Moving Average (ECDD) was proposed in (ROSS et al., 2012), where misclassification rates are exponentially weighted according to their position inside a sliding window. ECDD maintains three threshold levels: in-control, warning, and out-of-control. Given the overall misclassification rates inside the sliding window, the current system state fluctuates in between these three thresholds. ECDD postulates that a concept drift occurs whenever the misclassification rate achieves the out-of-control level.

Let $X = x^1, x^2, \ldots, x^t$ be a random variable, where t is the arrival timestamp, which has a common mean μ_0 before a drift and μ_1 after it. To note that this notation is generic, it is adopted μ_t to be the mean at an arbitrary timestamp t. Even though the mean μ_0 and the standard deviation σ_x are *a priori* unknown, authors work under the assumption that they are known. The ECDD estimator of μ_t is defined following Equation A.1.

$$Z_t = \begin{cases} \mu_0, & \text{if } t = 0 \\ (1-\lambda)Z_{t-1} + \lambda x^t, & \text{otherwise} \end{cases} \tag{A.1}$$

This estimator forms a “recent” estimate of μ_t, where older data is progressively down-weighted. In practice, λ is the parameter that controls how much weight is given to recent data compared to older data.

Additionally, authors in (ROSS et al., 2012) show that independently of the underlying distribution of x^t, the mean and standard deviation of Z_t are given by Equations A.2 and A.3, respectively.

$$\mu_{Z_t} = \mu_t \tag{A.2}$$

$$\sigma_{Z_t} = \sigma_{x^t}\sqrt{\frac{\lambda}{1-\lambda}\left(1-(1-\lambda)^{2t}\right)} \tag{A.3}$$

Before the drift point, $\mu_t = \mu_0$ holds, and Z_t fluctuates around this value. When a change occurs, μ_t will change to μ_1, and Z_t will react accordingly to this trend by diverging away from μ_0 and towards μ_1. Intuitively, this can be used to flag drifts by verifying whether Equation A.4 holds.

$$Z_t > \mu_0 + L_{\sigma_{Z_t}} \tag{A.4}$$

The parameter L is a “control limit” that determines how far Z_t must diverge from μ_0 before a drift is flagged. L should be defined correctly to ensure that the detector achieves a predefined performance level. The proposed performance measure is the expected time between false positive detections, the Average Run Length, here denoted as ARL_0. The choice of ARL_0 and the computation of L are not trivial, and thus, the reader is referred to (ROSS et al., 2012) for details.

Finally, ECDD postulates that its input sequence $X = x^1, x^2, \ldots, x^t$ is a stream of errors, which can be seen as a Bernoulli sequence with a parameter p_t representing the probability of misclassifying an instance in a time t. Drift detection is then posed as a problem of detecting an increase in the parameter p_t, while p_t is assumed to have only two possible values: p_0 before the drift point and p_1 after.

Following the previous definitions, it becomes clear that they require modifications before they can be used for actual drift detection. The main problem is that p_0 is assumed to be known, while this is not the case in real-world scenarios as it must be estimated from the stream along with σ_0. Therefore, a new estimator $p_{0,t}$ is proposed to substitute Z_t:

$$\hat{p}_{0,t} = \frac{1}{t}\sum_{i=1}^{t} x^i = \frac{t-1}{t}\hat{p}_{0,t-1} + \frac{1}{t}x^t \tag{A.5}$$

Unlike Z_t, this estimator does not assign bigger weights to recent observations. This implies that Z_t is more sensitive to changes in p_0 and should give an estimate close to its current value. On the other hand, $\hat{p}_{0,t}$ is less sensitive to changes in p_0 and it is intended to be an estimate of the pre-change value. Virtually, Z_t is expected to react quicker and converge towards the new value, while $\hat{p}_{0,t}$ should converge towards the same value, however slower. Along these lines, Algorithm 8 presents the pseudocode for ECDD, where drifts and warnings are flagged following Equations A.6 and A.7, the pre-change standard deviation is estimated by Equation A.8 and the standard deviation follows Equation A.9.

$$Z_t > \hat{p}_{0,t} + L_{\sigma_{Z_t}} \tag{A.6}$$

$$Z_t > \hat{p}_{0,t} + {}^{1}\!/_{2} L_{\sigma_{Z_t}} \tag{A.7}$$

$$\hat{\sigma}_{0,t} = \hat{p}_{0,t}(1 - \hat{p}_{0,t}) \tag{A.8}$$

$$\sigma_{Z_t} = \sqrt{\hat{p}_{0,t} \times (1 - \hat{p}_{0,t}) \times \frac{\lambda}{2 - \lambda} \times (1 - (1 - \lambda)^{2t})} \tag{A.9}$$

Finally, it is necessary to correctly determine λ. The optimal value of λ will depend on the pre- and post-change values of p_t. Since these are not known in advance, it is followed the rule of thumb provided in (ROSS et al., 2012), where λ is set in the $[0.1; 0.3]$ interval, being $\lambda = 0.2$ the default in several papers and data stream frameworks.

A.2 Adaptive Sliding Window Algorithm (ADWIN)

The Adaptive Sliding Window (ADWIN) change detector keeps a variable length window of recently seen items consistent with the hypothesis "there has been no change in the average output value inside the window according to a confidence bound δ" (BIFET; GAVALDÀ, 2007).

More formally, ADWIN receives as input a significance level δ and a sequence of values $x^t \in \mathbb{R}$, where t is the arrival timestamp. Let μ_t denote the expected value of the inputted distribution, which is unknown since the ground-truth generator distribution is also uncertain.

ADWIN uses a sliding window W with the most recently read x^t. Let $\hat{\mu}_W$ be the known average of the elements in W and μ_W be the unknown average of μ_t for $t \in W$. ADWIN's pseudocode is depicted in Algorithm 9. The rationale behind it is simple:

Algorithm 8: The ECDD drift detection algorithm. Adapted from (ROSS et al., 2012).

```
input  : a sequence of values $x^t$ defined as $x^t = 1$ if the classifier correctly predicted an
         instance and $x^t = 0$ otherwise, a fading factor $\lambda$ and the average run length
         $ARL_0$.
output : a flag for drift and warning detection $flag$.
$Z_0 \leftarrow 0$;
$\hat{p}_{0,0} \leftarrow 0$;
foreach $t > 0$ do
    $\hat{p}_{0,t} \leftarrow \frac{t}{t+1}\hat{p}_{0,t-1} + \frac{1}{t+1}x^t$;
    $\hat{\sigma}_{x^t} \leftarrow \hat{p}_{0,t}(1 - \hat{p}_{0,t})$;
    $\hat{\sigma}_{Z_t} \leftarrow \hat{\sigma}_{x^t}\sqrt{\frac{\lambda}{2-\lambda}\left(1-(1-\lambda)^{2t}\right)}$;
    Compute $L_t$ given the current value of $\hat{p}_{0,t}$ using functions presented in (ROSS et
     al., 2012);
    $Z_t \leftarrow (1-\lambda)Z_{t-1} + \lambda x^t$;
    if $Z_t > \hat{p}_{0,t} + L_t\hat{\sigma}_{Z_t}$ then
        $flag \leftarrow DRIFT$;
    else if $Z_t > \hat{p}_{0,t} + {}^1\!/_2 L_t\hat{\sigma}_{Z_t}$ then
        $flag \leftarrow WARNING$;
    else
        $flag \leftarrow NONE$;
    return $flag$
```

Algorithm 9: The ADWIN algorithm. Adapted from (BIFET; GAVALDÀ, 2007).

```
input  : a sequence of values $x^t$ and a confidence level $(1-\delta)$.
output : a flag for drift detection $flag$.
$W \leftarrow \emptyset$;
foreach $t > 0$ do
    $W \leftarrow W \cup \{x^t\}$;
    $flag \leftarrow FALSE$;
    while $|\hat{\mu}_0 - \hat{\mu}_1| < \epsilon_{cut}$ holds for every split of $W$ into $W_0$ and $W_1$ do
        Drop an element from the tail of $W$;
        $flag \leftarrow TRUE$;
return $flag$
```

whenever two “large enough” sub-windows of W exhibit “distinct enough” averages, one can conclude that their corresponding expected values are different, and thus, originated from distinct generators, which then forces ADWIN to drop the older portion of the window and flag a drift. Evidently, “large enough” and “distinct enough” should be made precise by using the Hoeffding bound (HOEFFDING, 1963), which has been previously presented in Section 5.1.

ADWIN’s test comes down to verifying whether the average of two sub-windows is larger then ϵ_{cut}, which is given by Equation A.10.

$$\epsilon_{cut} = \sqrt{\frac{1/|W_0| + 1/|W_1|}{4} \times \ln \frac{4|W|}{\delta}} \tag{A.10}$$

The algebra behind ϵ_{cut} is presented in (BIFET; GAVALDÀ, 2007), just as the proofs for ADWIN's bound on false positives and false negatives. ADWIN is parameter- and assumption-free since it automatically detects and adapts to the current rate of change. One of the major drawbacks of this method is that ADWIN is known for triggering too many false positives (HUANG et al., 2015), i.e., it flags changes when they do not really occur.

A.3 Hoeffding-based Drift Detection Methods (HDDM-A and HDDM-W)

The Hoeffding-based Drift Detection Methods were proposed in (FRÍAS-BLANCO et al., 2015), hereafter referred to as HDDM-A and HDDM-W. Here, the suffixes A and W stand for *Averages* and *Weighted averages*, respectively, as these concepts are at the core of the statistical tests being adopted. In practice, both HDDM-A and HDDM-W follow the same algorithm, depicted in Algorithm 10, yet, their differences reside in the way statistics about the data stream is maintained and statistically tested.

A simple method for monitoring the occurrence of changes in data streams is to use interval estimators. Two different intervals, warning and drift, are defined based on a probabilistic distribution, e.g., the gaussian distribution. Thus, the method computes an statistic p from the previously observed is statistically far from the expected one μ with a standard deviation σ. For instance, if one sets the warning level at 95% and the drift level at 99%, a warning would be raised whenever p lays outside the $[\mu - 2\sigma; \mu + 2\sigma]$ interval, while a drift would be flagged if p falls outside $[\mu - 3\sigma; \mu + 3\sigma]$. What HDDM does is to relax the normality assumption, and substitutes the estimation of the standard deviation σ with another that fixes the desired significance level α and estimates the confidence interval based on ε_α, or $\hat{\varepsilon}_\alpha$, depending on whether the A or W variant is being adopted. Also, two different confidence levels are set. The first, α_W represents the warning level, while α_D is used for the drift level, such that $\alpha_W < \alpha_D$.

Algorithm 10 receives as input a sequence of values $x^1, x^2, \ldots, x^{\text{cut}}, x^{\text{cut}+1}, \ldots, x^n$ which are the error rates of a classifier and the problem is to detect a significant increase in the mean of this sequence. Therefore, the first goal is to estimate a relevant cut point

Algorithm 10: The HDDM algorithm. Adapted from (FRÍAS-BLANCO et al., 2015).

```
input  : a sequence of values $x^t$, a confidence level $\alpha_W$ for the warning level, and a
         confidence level $\alpha_D$ for the drift level.
output : a flag for warning and drift detection $flag$.
Let $\hat{X}_{\text{cut}}$ be the statistic computed until the cut point $x^1, \ldots, x^{\text{cut}}$;
Let $\hat{Y}_{n-\text{cut}}$ be the statistic computed from after the cut point until $t$: $x^{\text{cut}+1}, \ldots, x^t$;
Let $\hat{Z}_n$ be the statistic computed from the entire sequence $x^1, \ldots, x^t$;
$W \leftarrow \emptyset$;
foreach $t > 0$ do
    Update $\hat{Y}_{n-\text{cut}}$, $\hat{Z}_i$, and $\varepsilon_{\hat{Z}_i}$ using $x^t$;
    if $\hat{Z}_i + \varepsilon_{\hat{Z}_i} \leq \hat{X}_i + \varepsilon_{\hat{X}_i}$ then
        $\hat{X}_{\text{cut}} \leftarrow \hat{Z}_i$;
        $\varepsilon_{\hat{X}_{\text{cut}}} \leftarrow \varepsilon_{\hat{Z}_i}$;
        Reset $\hat{Y}_{n-\text{cut}}$ and $\varepsilon_{\hat{Y}_{n-\text{cut}}}$;
    if $H_0 : E[\hat{X}_{cut}] \geq E[\hat{Y}_{n-cut}]$ is rejected with $\alpha_D$ then
        $flag \leftarrow DRIFT$;
    else if $H_0 : E[\hat{X}_{cut}] \geq E[\hat{Y}_{n-cut}]$ is rejected with $\alpha_W$ then
        $flag \leftarrow WARNING$;
    else
        $flag \leftarrow NONE$;
return $flag$
```

referred to as cut in this sequence, to later carry out either the A or W tests by comparing the subsamples until and after the cut point, i.e., $x^1, x^2, \ldots, x^{\text{cut}}$ and $x^{\text{cut}+1}, \ldots, x^n$.

Assuming $\hat{X}_i + \varepsilon_{\hat{X}_i}$ $(1 \leq i \leq n)$, where $\hat{X}_i$ can either be a moving average or a weighted moving average, and $\varepsilon_{\hat{X}_i}$ is its corresponding error bound, calculated either using Equations A.11, or A.12 depending on whether the A-test or W-test is being followed, respectively. The variables involved in these equations are as follows: $[a; b]$ is the range in which the arriving data values x^t are bounded to, n is the length of the subsample until the cut point, m is the length of the subsample after the cut point, and $\mathcal{D}_{n,m} = (b-a)^2 \left[\sum_{i=1}^{\text{cut}} v_i + \sum_{i=\text{cut}+1}^{n} v_i'\right]$ where v_i and v_i' are the weights of instances generated by a time-decay function.

$$\varepsilon_\alpha = (b-a)\sqrt{\frac{n^{-1}+m^{-1}}{2} \ln \frac{1}{\alpha}} \tag{A.11}$$

$$\hat{\varepsilon}_\alpha = \sqrt{\frac{\mathcal{D}_{n,m}}{2} \ln \frac{1}{\alpha}} \tag{A.12}$$

In practice, both of the equations listed above are direct applications of the Hoeffding Bound earlier discussed in Section 5.1. Yet, for the sake of brevity, the reader is referred to the work of (FRÍAS-BLANCO et al., 2015) for the mathematical formalization behind these formulas and their respective bounds for error types I and II.

Given the aforementioned processes, Algorithm 10 works as follows. With the arrival of an input value x^t, the statistics $\hat{X}_{\text{cut}}$, $\hat{Y}_{n-\text{cut}}$, and $\hat{Z}_n$ are updated. The former ($\hat{X}_{\text{cut}}$) is the average, or weighted moving average, computed from the beginning of the stream x^1 until the cut point x^{cut}, while the second ($\hat{Y}_{n-\text{cut}}$) is the same statistic, yet, computed from the cut point $x^{\text{cut}+1}$ until the last observed value x^t, and finally, the last value $\hat{Z}_n$ is the statistic given all observed values, from x^1 until x^t (line 6).

Next, the algorithm verifies if the cut point should be updated depending on the error bounds (line 7) given by Equations A.11 or A.12, depending on the test adopted, and if the condition holds, all the respective statistics are updated accordingly (lines 8–10). Given the updated statistics, HDDM then checks the current state of the stream by computing the expected values of the statistics before and after the cut point $\hat{X}_{\text{cut}}$ and $\hat{Y}_{n-\text{cut}}$ against the user-given confidence levels α_W and α_D (lines 11 and 13), and warnings and drifts are flagged accordingly (lines 12 and 14).