HEITOR MURILO GOMES

ADVANCES IN NETWORK-BASED ENSEMBLE OF CLASSIFIERS FOR DATA STREAMS

Doctoral examination presented to the Post-graduate Program in Informatics (PPGIa) of the Pontifical Catholic University of Paraná.

Curitiba
2017

HEITOR MURILO GOMES

ADVANCES IN NETWORK-BASED ENSEMBLE OF CLASSIFIERS FOR DATA STREAMS

Doctoral examination presented to the Post-graduate Program in Informatics (PPGIa) of the Pontifical Catholic University of Paraná.

Concentration area: Computer Science

Advisor: Fabrício Enembreck

Curitiba
2017

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

G633a
2017

Gomes, Heitor Murilo
Advances in network-based ensemble of classifiers for data streams / Heitor Murilo Gomes ; advisor, Fabricio Enembreck. – 2017.
xiii, 149 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2017
Bibliografia: f. 119-133

1. Mineração de dados (Computação). 2. Aprendizado por computador. 3. Redes sociais. 4. Classificação. 5. Informática. I. Enembreck, Fabrício. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 20. ed. – 004

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

PUCPR
GRUPO MARISTA

ATA DE SESSÃO PÙBLICA

DEFESA DE DISSERTAÇÃO DE DOUTORADO Nº 43/2017

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA – PPGIa
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ - PUCPR

Em sessão pública realizada às 14h00 de 15 de Março de 2017, no Auditório Guglielmo Marconi, ocorreu a defesa da tese de doutorado intitulada **"Advances in Network-Based Ensemble Classifiers For Data Streams"** elaborada pelo aluno **Heitor Murilo Gomes**, como requisito parcial para a obtenção do título de **Doutor em Informática**, na área de concentração **Ciência da Computação,** perante a banca examinadora composta pelos seguintes membros:

Prof. Dr. Fabrício Enembreck (orientador) - PPGIA/PUCPR

Prof. Dr. Alceu de Souza Britto Junior – PPGIA/PUCPR

Prof. Dr. Luiz Eduardo Soares de Oliveira - UFPR

Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho – USP São Carlos

Prof. Dr. Albert Bifet - Université Paris - Saclay

Após a apresentação da tese pelo aluno e correspondente arguição, a banca examinadora emitiu o seguinte parecer sobre a tese:

Membro	**Parecer**	
Prof. Dr. Fabrício Enembreck	(X) Aprovada	( ) Reprovada
Prof. Dr. Alceu de Souza Britto Junior	(X) Aprovada	( ) Reprovada
Prof. Dr. Luiz Eduardo Soares de Oliveira	(X) Aprovada	( ) Reprovada
Prof. Dr. André Carlos P. de Leon Ferreira de Carvalho	(X) Aprovada	( ) Reprovada
Prof. Dr. Albert Bifet - Université Paris	(X) Aprovada	( ) Reprovada

Portanto, conforme as normas regimentais do PPGIa e da PUCPR, a tese foi considerada:

(X) **APROVADA**
(aprovação condicionada ao atendimento integral das correções e melhorias recomendadas pela banca examinadora, conforme anexo, dentro do prazo regimental)

( ) **REPROVADA**

E, para constar, lavrou-se a presente ata que vai assinada por todos os membros da banca examinadora. Curitiba, 15 de Março de 2017.

Prof. Dr. Fabrício Enembreck
(orientador)

Prof. Dr. Alceu de Souza Britto Junior

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

Prof. Dr. Luiz Eduardo Soares de Oliveira

Prof. Dr. André Carlos P. de L. Ferreira de Carvalho

Prof. Dr. Albert Bifet

Rua Imaculada Conceição, 1155 Prado Velho CEP 80215 901 Curitiba Paraná Brasil
Tel.: (41) 3271 1669 Fax.: (41) 3271 2121 www.ppgia.pucpr.br

To my beloved mother, Evelyn.

Contents

List of Figures

List of Tables

Abstract

This work encompasses the proposal of a new kind of ensemble classifier for data stream classification, namely the network-based ensemble. This kind of ensemble explicitly defines relations between component classifiers that are used in various forms, for example, to combine classifiers' decisions. Effectively, this work is based on the hypothesis that an ensemble classifier can obtain accurate predictions with the aid of structural analysis of a weighted network of its component classifiers. A limited set of works already use relational data, or even networks, to abstract the ensemble, however there is still a lack of thorough analysis to justify its use for data stream classification. The overall objective, in this work, is to define, analyze and propose implementations of network-based ensembles for data stream classification. We present a formal description of a network-based ensemble, a thorough analysis of the literature (including a taxonomy of existing methods), a preliminary implementation (Social Adaptive Ensemble 2 - SAE2), two methods for relation definition - Pairwise Accuracy (PA) and Pairwise Patterns (PP) -, and a final enhanced implementation (Complex Network Ensemble - CNE). Results show that SAE2 and CNE are able to achieve results, w.r.t. accuracy, memory and processing time, that are competitive with state-of-the-art ensembles, and that many interesting voting patterns can be obtained through PA and PP, which sometimes lead to accuracy improvements.

Keywords: Data Stream Mining; Ensemble Learning; Concept Drift; Machine Learning; Network-based Ensembles.

Chapter 1

Introduction

Data stream mining has grown in importance in recent years due to the tremendous amount of real-time data generated by computer networks, smartphones and all sorts of sensors available. The analysis of these data streams can be very useful to companies and to their customers, since it can be used to leverage products and services. For example, an e-commerce website can log users' shopping behavior and devise a model capable of recommending products for new users based on the products they have clicked. Other examples include: network intrusion detection, spam identification, social media analysis, real time sensor based system, and many others. These examples have in common many important characteristics of data stream mining, the most prominent being: vast amounts of streaming data, which relevance may decay over time. In this work, we focus on the problem of data stream classification. We argue that this problem is even more challenging than traditional classification. Our claim is based on the fact that data stream classification is not only subject to virtually all problems that affect traditional classification (e.g., absent values, outliers, noise, unbalanced classes, and others), but also to specific issues that arise in a data stream configuration. These issues include: concept drifts, great to infinite amount of instances, limited resources (memory and processing time) and temporal dependencies.

In recent years, new classifiers have been proposed to cope with different dimensions of the data stream classification problem. A lot of interest has been shown for methods that are based on ensembles of classifiers (OZA, 2005; BIFET et al., 2009; BIFET; HOLMES; PFAHRINGER, 2010; KOLTER; MALOOF, 2007; GOMES; ENEMBRECK, 2013; BARDDAL; GOMES; ENEMBRECK, 2014). The reason for that can be attributed to ensembles, frequently, being able to achieve higher accuracy when compared to single classifiers. Also, ensembles can deal with concept drift in a less drastic way than a single classifier. For example, a single classifier may discard its hypothesis completely

when faced with a concept drift, while an ensemble may only replace (or reset) a few of its component classifiers. Some of the first works on ensemble classifiers for data stream were focused on adapting existing algorithms to a data stream context. That was the case for Online Bagging and Online Boosting (OZA, 2005). Although many ensembles were specifically designed to work with data streams, some of which were able to deal with concept drift either explicitly (BIFET et al., 2009; BIFET; HOLMES; PFAHRINGER, 2010)) or implicitly (KOLTER; MALOOF, 2007; GOMES; ENEMBRECK, 2013; BARDDAL; GOMES; ENEMBRECK, 2014). Most ensemble classifiers are designed following the intuition that its component classifiers must be diverse in order to allow their combination to achieve higher accuracy than a single classifier. That is true for many successful ensemble methods not limited to data stream mining, namely Bagging (BREIMAN, 1996), AdaBoost (FREUND; SCHAPIRE et al., 1996) and Random Forest (BREIMAN, 2001). The subjective notion of diversity has been explored thoroughly (KUNCHEVA et al., 2003; KUNCHEVA; WHITAKER, 2003) and yet there is not a "one size fits it all" metric for measuring diversity between ensemble members, or a theoretical proof that correlates a given diversity measure and its impact on accuracy. Even though it is difficult to measure, or formalize, the contribution of "diversity" to the ensemble overall prediction accuracy, intuitively it is easy to rationalize why an homogeneous set of classifiers when combined cannot achieve accuracy any better (or worse) than any of its members would individually.

In this work, we explore a new area of ensemble classifiers for data stream mining, namely the network-based ensembles. This class of ensemble explicitly defines relations between its members that are used in various ways, for example, to combine similar classifiers decisions. More specifically, we define the fundamental characteristics of a network-based ensemble, present a preliminary algorithm (SAE2), experiment with novel relations (PA and PP), and present a final implementation named CNE. Experiments include an empirical evaluation of SAE2, experiments with optional relation definitions, an analysis of the combination method used in SAE2 and thorough analysis of CNE.

1.1 Problem

Currently there are many data stream classifiers, some of which are based on ensemble of classifiers. A very restricted subset of these ensembles (GOMES; ENEMBRECK, 2013; BARDDAL; GOMES; ENEMBRECK, 2014) uses a network of classifiers to arrange the component classifiers. These network-based ensembles do not use the network abstraction to its maximum extend. For example, SAE (GOMES; ENEMBRECK,

2013) uses a network of classifiers to discover subgroups of very similar classifiers, but it does it by simplifying the weighted connections between classifiers, that measure their similarity, to dichotomous connections according to a fixed threshold. This simplification may cause a loss of useful information. Also, current works do not provide detailed analysis that justify the network abstraction. For example, in (GOMES; ENEMBRECK, 2013) and (BARDDAL; GOMES; ENEMBRECK, 2014) it is not possible to conclude how the network abstraction actually contributes to the ensemble decisions.

1.2 Objectives

In this work, the overall objective is to define, analyze and propose implementations of network-based ensembles for data stream classification. By arranging classifiers in a network structure that obey to a given relation R it is possible to not only identify similarities and dissimilarities among component classifiers, but also use this structure to obtain accurate predictions. We are concerned only about scalable methods, since data stream classifiers are supposed to deal with huge amounts of instances, therefore any method that is not able to scale to this scenario might not be useful in practice. While proposing a new data stream classification method it would be temerarious to not consider the benefits of well-established techniques, such as drift detection (BAENA-GARCÍA et al., 2006; BIFET; GAVALDÀ, 2007). Therefore, this work will also include the analysis of how network-based ensembles and traditional data stream classification techniques can be combined. In order to meet the general objective of this work, we outline the following specific objectives:

1. Define a taxonomy for data stream ensemble classifiers, which includes aspects of network-based ensembles;

2. Formalize and analyze the proposed network-based ensemble;

3. Investigate alternative ways to define relation R and define methods to explore these in benefit of the ensemble;

4. Define a flexible adaption method that does not depend on predefined parameters and can effectively exploit network-based ensemble properties.

Objective 1 allows us to situate our new proposal in relation to existing ensembles. Objective 2 concerns formalization of a theoretical network-based ensemble and the instantiation of a network-based ensemble method. The instantiation of a network-based

ensemble is useful to illustrate advantages and shortcomings, while providing a testable algorithm. Objective 3 deepen our analysis in one of the most important aspects of the proposed network-based ensembles, which are the relations that define connections between classifiers. Objective 4 is related to the adaptation and use of existing data stream learning methods for network-based ensembles.

1.3 Hypothesis

Hypothesis. *An ensemble classifier can obtain accurate predictions with the aid of structural analysis of a weighted network of its component classifiers.*

This hypothesis depends on the existence of an ensemble that is able to extract relational information from its component classifiers, and interpret this data in benefit of its overall decisions. In simple terms, we hypothesize that it is possible to use relational data that correlates component classifiers, such as their similarity with respect to predictions, in benefit of the ensemble overall decisions. The weighted network of classifiers referred in the hypothesis is simply a set of classifiers that obeys to relation R. The relation R is a measure between pairs (or sets) of classifiers, which must explicitly measure useful information about them, e.g., their similarities or, conversely, their differences. In previous works (GOMES; ENEMBRECK, 2013), R was defined as the "similarity between classifiers", i.e., the fraction of all "recent" predictions in which both classifiers predicted the same class label. Notice that relation R is instantiate in the form of connections between pairs of classifiers and that these connections form a network of interconnected classifiers.

Another fundamental aspect of our hypothesis is the structural analysis. This portion comprises the extraction of knowledge from the network of classifiers. In this analysis, not only individual statistics, such as component classifier's accuracy, are taken into account, but also information that can be extracted from pairs, triples or subsets of component classifiers. The insights obtained from the structure can be used to enhance predictions, update component classifiers, or even to guide training for diversity induction.

1.4 Expected outcomes and contribution

Through this work we expect to advance the current understanding of ensemble classifiers for data streams, specifically those based on networks of classifiers, as well as developing novel methods to tackle data streams classification. Concretely, we provide

empirical evidence to illustrate the advantages and disadvantages of network-based ensembles in the context of data stream classification. These empirical evidences include statistical tests comparing state-of-the-art ensemble classifiers and the methods presented on this work, and experiments developed to validate the contributions of structural analysis to the ensemble.

Chapter 2

Data Stream Mining

Recently, the amount of data generated by smart phones, social networks and all kinds of sensors has grown tremendously. Companies, governments and individuals are flooded with data about their costumers, citizens and even their own body through wearables. All this data is only useful if it can be efficiently processed, so that individuals can make timely decisions based on them. A lot of progress has been made towards obtaining useful models out of massive amounts of data under the research area of data mining. Even though classical data mining is able to address many issues related to data processing, most of its algorithms were designed to deal with static data, i.e., while the algorithm is running the data distribution does not change. This assumption does not hold for many practical problems. Consider the problem of classifying customer's opinions in a social network as either positive or negative related to a given topic. In this problem, despite the need to perform a preprocessing step on the user comments, there are two other issues that must be considered: data is generated continuously and users' opinions may change over time. A continuous flow (stream) of data requires that the learning algorithm update its model incrementally or through batches of instances. Also, if users' opinions change, then the algorithm must update its model as fast as possible to continue providing accurate predictions. A data stream can be defined as "A sequence of digitally encoded signals used to represent information in transmission."[1].

Data streams pose several challenges for learning algorithms, including, but not limited to: concept drifts, temporal dependences, great to infinity amount of instances, restrictive time and memory requirements. On top of that, data stream mining includes most existing problems in traditional data mining, such as absent values, noise, irrelevant features, class imbalance, and others.

In this work, we focus on the task of classification for data streams. Succinctly,

[1]Institute for Telecomunication Sciences: <http://www.its.bldrdoc.gov/fs-1037/dir-010/_1451.htm>

classification refers to the problem of learning a function that maps instances into one of two or more predefined classes. It is possible to use traditional (batch) classifiers in a data stream configuration as long as instances are grouped into batches before training. This requirement may be restrictive for some streams due to demanding time and memory constraints, and the presence of concept drifts. Thus, incremental classifiers are usually preferred over their batch counterparts. To date, many incremental algorithms have been proposed for data stream classification, some of which are adaptations of existing batch algorithms, for example, Online Bagging (OZA, 2005) is an adaption of the Bagging algorithm (BREIMAN, 1996).

In the following sections the main concepts and challenges related to data stream classification are presented. Section 2.1 presents a detailed description of the stream classification problem that we use as well as its challenges. Section 2.2 presents the concept drift problem and its different kinds. In Section 2.3, the existing methods for evaluating stream classifiers are presented. Section 2.4 presents datasets and synthetic stream generators commonly used for data stream classifiers evaluation.

2.1 Context and Assumptions

Data stream classification is a variation of the traditional supervised machine learning task of classification. Both tasks are concerned with the problem of predicting a nominal value of an unknown instance represented by a vector of characteristics. The main difference between these tasks is that in the context of data stream classification instances are not readily available to the classifier as being part of a large static dataset, instead, instances are provided one by one by a continuous data stream. Therefore, a data stream classifier must be ready to deal with a great amount of instances, possible infinity, such that each instance can only be inspected once or stored for only a short period of time. In order to properly work in a data stream environment, a classifier must attend to the following constraints, presented in (BIFET et al., 2010):

1. Process instances one at a time and inspects them only once. Each instance must be processed, or ignored, in the order at which it is presented to the algorithm and deleted right after. There is no restriction against storing instances for a short period of time as long as characteristic 2 is preserved.

2. Use a limited amount of memory. One of the major motivations to use data stream classifiers is that these allow processing large amounts of data with a limited memory. This is possible as long as the algorithm is able to respect the maximum space of memory that can be used. To allow more memory usage, the algorithm can store data

in a secondary storage, although this must take into account characteristic 3.

3. Use a limited amount of processing time. The time taken to process one instance must not surpass the time t, which is the ratio at which new instances are presented to the algorithm. If the algorithm happens to increase the time it needs to process each instance over time, i.e., it does not scale linearly, then the algorithm will either start ignoring instances or lose its capability to process instances as soon as they arrive. If the algorithm is not able to process instances in real time, then it will not be able to adapt itself to concept drifts in a timely fashion.

4. Be ready to predict at any time. An ideal algorithm must be ready to use its best possible model at any time, independently of the number of instances that it has already processed. This implies that the construction model must be efficient (characteristic 3). To achieve this the algorithm must be able to update its model incrementally as instances arrive, i.e., without the need to reconstruct the whole model every time a new instance is presented to it.

In this work, we assume that instances from a data stream S appear sequentially as a sequence, in intervals of u time units, of unlabeled instances x^t, where x^t represents a vector of attribute values. Also, it is assumed that the true class label y^t of a given instance x^t is available before the next instance x^{t+1}, thus the classifier can use it for training immediately after it has been used for testing. Specifically, we do not consider other forms of learning, such as semi-supervised classification where labels are not available for all incoming instances. Also, it is expected that the underlying concept is unstable, i.e., the data distribution is expected to change through different types of concept drifts.

More recently, in (ŽLIOBAITĖ et al., 2014) it was advised to consider temporal dependences in evaluating and designing data stream classifiers. A temporal dependence occurs whenever the current instance, x^t label is not independent of previous instances $x^{t-1}, x^{t-2}, \ldots$ labels. To date only a few classifiers have been designed to take into account temporal dependences and an evaluation metric has been proposed, both presented in (ŽLIOBAITĖ et al., 2014). The problem setting described, with exception of the temporal dependence part, is the same used in other existing works, such as (OZA, 2005; BIFET; HOLMES; PFAHRINGER, 2010; BIFET et al., 2009; GOMES; ENEMBRECK, 2013; BARDDAL; GOMES; ENEMBRECK, 2014; BRZEZINSKI; STEFANOWSKI, 2014) and on data stream analysis frameworks, e.g. MOA (BIFET et al., 2011). We use an extended version of the traditional problem setting considering temporal dependences in the evaluation metric when necessary.

2.2 Concept Drift

Real world problems tend to be very dynamic. For example, a consumer behavior may change as he/she ages, a group of people can change their opinion about a product or a political party, the attacks a network receives may change as new barriers are created, and so on. Learning from data which distribution may change over time is a challenging task as conventional machine learning algorithms assumes the data distribution is static. Conventionally, streams that contain drifts are identified as evolving streams, while streams in which data distribution is stationary are named stationary streams.

There are many aspects to consider concerning a drift, including its cause, rate and where the data distribution has changed.

The cause of a drift can be related to the hidden variables (hidden context) (TSYMBAL, 2004) of the learning problem. The attributes available to a learner do not contains all possible variables that affects its output, simply because these variables are either sporadic or very complex. For example, considering a learning problem in which it is expected to uncover shoppers' behavior, as time passes this behavior may be affected by inflation, season, marketing campaigns, and other variables unknown to the learner. In practice, it is not necessary to discover the cause to adapt the learning hypothesis, i.e., it is the learner task to adapt itself to a drift, not to explain the reason behind it.

Another important trait of concept drift relates to the rate at which it happens. The rate at which drifts happen can be abrupt, incremental, gradual or reoccurring. Notice that noise or outliners ought not be confused with drift. The difference between noise/outliers and drifts is persistence.

Abrupt drifts are the most noticeable kind of drift because the prediction error and the data distribution vary greatly in a short period of time. A drift is considered abrupt if after one given instance the concept shifts persistently. For example, if the learning problem is to evaluate products as relevant or irrelevant for a single user, considering the user has been looking for a new smartphone, after he bought it, smartphones become irrelevant for him all of a sudden.

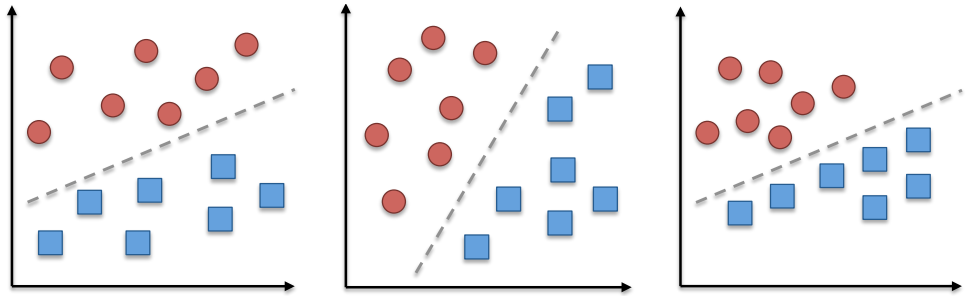
If the new concept takes a while to become persistent, then it is said that the drift is gradual. Precisely, during an unknown size window, instances from a previous concept appear less frequently, while instances from a new concept become predominant. For example, considering that a user has already bought a new smartphone, but he/she is still sporadically interested in smartphones up to the point when he/she completely loses interest on them.

Concepts that slowly evolve over time cause incremental drifts to happen, which

are not easily detectable.

Reoccurring concepts refers to concepts that alternates with certain regularity over time. For example, when the user behavior changes according to the season of the year.

The last classification at which a drift is susceptible relates to its location. There are many works with different naming conventions for the same phenomena. In this work we adopt the same definitions, and naming, as those presented in (GAMA et al., 2014). Generally, a drift can be either "real" or "virtual". A real concept drift happens when there are changes in the posterior probabilities of classes, $p(y|X)$, i.e., the output variable distribution changes affecting the upcoming predictions. A virtual concept drift is said to occur if the distribution of the incoming data $p(X)$ changes without affecting $p(y|X)$. In practice, there is not much interest in virtual drifts, because they do not change the conditional distribution of the output. Figure 2.1 illustrates a visual comparison of virtual and real drift.



(a) Original data. (b) Real drift. (c) Virtual drift

Figure 2.1: Types of drifts. Squares and circles represents instances with different classes, adapted from (GAMA et al., 2014)

It is important to notice that virtual drifts are observable by checking the input distribution, which is always available. This advantage may not be useful, as virtual drifts do not impact the output variable. Usually, detecting real drifts depends on the availability of labeled instances. In some contexts the label of instances may not be readily available after they have been used for testing. In this work we do not consider these contexts (see Section 2.1).

Different strategies have been developed to deal with drifts varying from active drift detection algorithms (detectors) to indirect adaptation methods (dynamic ensemble classifiers). According to (ŽLIOBAITĖ et al., 2010) there are mainly four strategies to deal with concept drift: Forgetting, Detectors, Contextual and Dynamic Ensemble Classifiers. Each of these strategies has tradeoffs, for example, Dynamic Ensemble Classifiers are usually good at recovering from gradual drifts, but they tend take longer to adapt to an abrupt drift.

One of the simplest strategies to adapt to concept drifts is to "forget" old instances and train the model only on the most recent data. This forgetting mechanism is based on the hypothesis that instances lose relevance as time passes. It is possible to forget one single instance at a time or whole chunks of instances (windows). On one hand, forgetting only one instance at a time is difficult because removing its influence from the classifier hypothesis is not trivial for non-instance based classifiers. On the other hand, determining the window size is difficult if the rate of change that drifts exhibit is unknown for the given stream. Smaller windows are appropriate in case of abrupt drifts, as the classifier is able to adapt itself faster. Although during times where there are no drifts the prediction accuracy will be impacted as the classifier cannot learn from a large amount of instances. A third option is to use dynamically sized windows, but then the problem turns to "when" and "how" to change the window sizes. A possible solution resides at drift detection algorithms.

It is possible to detect changes by observing the data distribution, the parameters of the classifier, or the incorrectly classified instances. Algorithms that actively check for drifts, namely drift detector algorithms, are used together with classifiers. Usually, a drift detector indicates the instance at which the drift had begun and the training is restarted at that instance. Drift detectors are often associated with abrupt drifts, as gradual and incremental drifts are very difficult to detect with existing techniques. Some well-known and widely used drift detectors are the DDM (GAMA et al., 2004), EDDM (BAENA-GARCÍA et al., 2006) and ADWIN (BIFET; GAVALDÀ, 2007).

Contextual strategies denote methods that build many hypothesis and alternate between them according to the current data. In other words, instead of discarding the current hypothesis, it is stored for future use when its corresponding concept becomes active again. Even though this strategy saves time and resources as there is no need to rebuild the hypothesis, it can be difficult to determine if a previously seen concept has returned.

Many popular methods to cope with evolving streams are based on dynamic ensemble classifiers. Learning methods based on ensemble of classifiers are widely used in traditional machine learning, the most prominent examples are Bagging (BREIMAN, 1996), Boosting (SCHAPIRE; FREUND, 2012) and Random Forests (BREIMAN, 2001). These ensemble classifiers tend to achieve accuracy that overcomes single classifiers, with other additional characteristics (see Chapter 3). For evolving streams, ensemble learners can be modeled to remove or add classifiers according to variations on the data distribution of the stream, along with other complex strategies that involves combining and weighting classifiers. This dynamic characteristic allows ensemble based classifiers to smoothly adapt

to gradual drifts, although they may fail to rapidly adapt to abrupt drifts. Despite its benefits, ensemble classifiers demand more resources, processing time and memory, than a single classifier. Thus, depending on the data stream setting, some ensemble classifiers are not viable. Some examples of dynamic ensemble classifiers are DWM (KOLTER; MALOOF, 2007), SEA (STREET; KIM, 2001), Flora (WIDMER; KUBAT, 1996), ADWIN and ASHT Bagging (BIFET et al., 2009), Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010), OAUE (BRZEZINSKI; STEFANOWSKI, 2014), SFNC (BARDDAL; GOMES; ENEMBRECK, 2014) and SAE (GOMES; ENEMBRECK, 2013).

2.3 Evaluation Procedures

Data stream classification has three main evaluation metrics: space, time and classification performance. Very often, especially on less recent research, evaluation was solely based on accuracy. Machine learning evaluation has received more attention from practitioners (JAPKOWICZ; SHAH, 2011; DEMŠAR, 2006), partially because of suspicious raised about statements like "algorithm A is better (more accurate) than algorithm B", which were made without the foundation of a reliable (and applicable) statistical test.

In a data stream context, evaluating classifiers can become even more complicated as it is necessary to observe not only "if" a prediction error happened, but also "when" it took place. Since learning from data streams is a process that happens over time, it makes sense to observe accuracy, or other metric of classifier quality, evolution over time, instead of outputting a single value at the end of the process. On top of that, data streams can be very large, potentially infinite, thus there is no clear "end of training" and "start of testing". Another aspect to be considered is that a stream may be non-stationary (evolving), thus the latest instances can better represent the current concept than instances that were processed a long time ago. Therefore, in order to have a consistent estimation of the current accuracy the most recent incorrect predictions must have a greater impact than older ones. On top of that, a data stream classifier might not be useful in practice if it cannot attend to specific constrains like memory and processing time. Therefore a consistent evaluation might include estimations of space and time required.

In this section we describe methods for evaluating data stream classifiers, including the three dimensions of interest: classification performance, space and time. Even though we use statistical tests to assess our findings (see Chapter 5), we refrain from discussing them here as the subject is vast and beyond the scope of this work. The reader is referenced to (JAPKOWICZ; SHAH, 2011) for a detailed discussion on the topic.

2.3.1 Evaluating Data Stream Classifiers

In traditional classification problems there is a concern about taking the maximum advantage of a limited amount of data. Some problems have very little labeled instances, sometimes because it is too expensive to label them, thus every instance counts to obtain a stronger hypothesis. One could suggest using all instances for training, but then it would not be possible to assess the hypothesis ability to correctly predict previously unseen instances (generalization). To address this issue, many strategies to split the database into train and test sets have been proposed over the last years, for example: holdout, cross-validation, leave-one-out, and others.

One of the widely used approaches, in batch learning, is the k-fold cross-validation. Although this strategy requires multiple passes over the database, thus it cannot be used in a data stream scenario because it is prohibitive to store all instances and then inspect them multiple times. For data stream classifiers evaluation, the preferred strategies include: periodic holdout, test-then-train and prequential.

In periodic holdout fixed size windows are defined for training and testing. For instance, the first 100 instances are used for training, the subsequent 60 instances are used for testing, and then the next 100 instances are used for training, and so forth. One good trait of this strategy is that the generalization capabilities of the classifier can be assessed, i.e., instances used for training are not used for testing. Also, the accuracy value given at the end of a test window is influenced only by the instances on that window, thus it is a good approximation of the current accuracy of the algorithm. Although not using some instances for training can be a drawback, even though in a data stream it is expected that data is abundant. To take maximum advantage of input data the test-then-train strategy can be used. Test-then-train uses all instances for testing and training, such that each incoming instance is first used for testing and right after for training. Even though, maximum advantage is obtained from data, test-then-train accuracy at a given point in time is influenced by old predictions. Therefore test-then-train may overestimate (or underestimate) the current accuracy.

The prequential (GAMA; RODRIGUES, 2009) strategy is similar to test-then-train, but maintain characteristics from holdout. Prequential uses a fading factor that diminishes the impact of older incorrect predictions. In (GAMA; RODRIGUES, 2009) authors demonstrate how prequential results converges to equivalent results obtained by periodic holdout.

2.3.2 Kappa, Kappa M and Kappa-Temporal statistics

It is important to consider alternatives to accuracy for measuring classifier fitness. The reason for that is based on the fact that accuracy can be misleading in many situations, for example, if the distribution of classes is imbalanced. In these cases, a valid option is to use Cohen's Kappa statistic (COHEN, 1960). For datasets that exhibit temporal dependencies, i.e. instances are not temporally independent of each other [2], it is advisable to evaluate Kappa Temporal since it replaces majority class classifier with the NoChange classifier (ŽLIOBAITĖ et al., 2014).

While determining which classifier to use for a given problem it is important to define performance baselines for the minimum meaningful performance, otherwise a lot of effort may be wasted (ŽLIOBAITĖ et al., 2014). A baseline classifier is a naive classifier that does not use any information about x, only the class labels y. If all the considered classifiers are considered worse than a baseline classifier, then none of them is appropriate for the given problem. In the absence of data, but knowing the number of possible class labels, then the best strategy for a baseline classifier is to assign class labels at random (p_{ran}). Even though, in most cases, input data is available, thus it is viable to estimate at least the prior probabilities. Therefore, a better baseline classifier would be the one that always predict the class label that has maximum prior probability (p_{maj}). In (BIFET et al., 2015) authors show that Kappa Statistic M (k_m) measure has advantages over Kappa statistic as it has a zero value for a majority class classifier.

Many efforts were made to detect or adapt to concept drift, as it is expected that data streams exhibit non-stationary behavior, i.e. data is not identically distributed. Identically distributed means that the joint probability of an observation and its label is the same at any time, i.e., $P(x^{t1}, y^{t1}) = P(x^{t2}, y^{t2})$, when $t1 \neq t2$. However, most existing works consider instances as independent, i.e., x^t class label does not influence x^{t+1}. Under the assumption of instances independency and class imbalance, a popular measure to assess classifiers is the Kappa M Statistic. Equation 2.1 presents the calculation of Kappa M, such that p^* is the baseline classifier (e.g. p_{maj}), p is the analyzed classifier accuracy. If the predictions of the classifier are perfectly correct, then $k = 1$, if the predictions coincide with the correct ones as often as by chance, then $k = 0$. Any classifier with $k \leq 0$ is not improving upon the baseline classifier and may not be adequate to the problem at hand.

[2] In this context, an independent distribution means that the probability of a class label does not depend on the previous class label, i.e., $P(y^t) = P(y^t|y^{t-1})$.

$$k = \frac{p - p^*}{1 - p^*} \tag{2.1}$$

Despite its benefits, k and k_m both fail to diagnose cases of poor performance due to temporal dependence. In these cases, it is necessary to use a baseline classifier that takes into account temporal information. In (BIFET et al., 2013) authors propose the No-Change classifier [3] (p_{per}), which assumes that the next class label is always equal to the previous class label. Equation 2.2 presents the Kappa-Temporal Statistic (or Kappa Plus Statistic, denoted by k_{per}), which is similar to k, but uses p_{per} as the baseline classifier.

$$k_{per} = \frac{p - p_{per}}{1 - p_{per}} \tag{2.2}$$

The problem is that if there are no temporal dependences, then p_{maj} may be more accurate than p_{per}, which is undesirable as the baseline must be the best possible naïve classifier. In (ŽLIOBAITĖ et al., 2014) authors propose a combined measure for classification performance, that takes into account k and k_{per}. Equation 2.3 presents the combined metric (geometric average).

$$k^+ = \sqrt{max(0, k) \times max(0, k_{per})} \tag{2.3}$$

In Equation 2.3, if any measure is zero or below, the combined measure will result in zero. This trait is to avoid the situation when both input measures are negative, but their product is positive. Alternatively, an arithmetic average could be used, but then a good performance in one criterion could fully compensate for a bad performance in the other. Thus the geometric average is used, as it punishes large differences in the two input measures.

2.3.3 RAM Hours and CPU Time

Even if the classifier is able to achieve a high accuracy or k^+, if it cannot do so in a timely fashion or within available memory, then it cannot be used in practice. It is possible to infer a classifier's resources demands through computational complexity analysis of its memory and processing time needs. Although it is easier, for comparison purposes, to estimate resources demands through empirical analysis. In this work we use CPU Time (BIFET et al., 2011) and RAM Hours (BIFET et al., 2010) to evaluate processing time and memory, respectively. CPU Time is the amount of time the classifier process is active in the processor. While RAM Hours is the amount of gigabytes deployed per hour by the

[3] In (ŽLIOBAITĖ et al., 2014) the No-Change classifier is named Persistent classifier.

classifier, e.g., one GB of memory deployed for one hour corresponds to one RAM-Hour. For consistent results both metrics must be assessed on the same environment for all considered classifiers, since they are hardware dependent.

2.4 Data stream Generators and Real Datasets

To evaluate if a classifier is able to work in different scenarios, e.g., streams with abrupt drift, noise, temporal dependences, and so forth, it is necessary to consider different datasets for evaluation. Synthetic data stream generators are frequently used to evaluate data stream classifiers. This preference is due to the flexibility that these generators offer, such as a precise specification of a drift type and location within the stream. In the next sections, we present some of the most widely used data stream generators and real datasets, which are later used for experimental evaluations.

2.4.1 LED

The LED data set simulates both abrupt and gradual drifts based on the LED generator, early introduced in (BREIMAN et al., 1984). This generator yields instances with 24 boolean features, 17 of which are irrelevant. The remaining 7 features corresponds to each segment of a seven-segment LED display. The goal is to predict the digit displayed on the LED display, where each feature has a 10% chance of being inverted. To simulate drifts in this data set the relevant features are swapped with irrelevant features.

2.4.2 RBF

This generator creates centroids at random positions and associates them with a standard deviation value, a weight and a class label. To create new instances one centroid is selected at random, where centroids with higher weight have more chances to be selected. The new instance input values are set according to a random direction chosen to offset the centroid. The extent of the displacement is randomly drawn from a Gaussian distribution according to the standard deviation associated with the given centroid. To simulate incremental drifts, centroids move at a continuous rate, effectively causing new instances that ought to belong to one centroid to another with (maybe) a different class.

2.4.3 Random Tree Generator

The Random Tree Generator (RTG) (DOMINGOS; HULTEN, 2000) builds a decision tree through randomly selecting attributes as split nodes and assigning random classes to each leaf. After the tree is build, new instances are obtained through the assignment of uniformly distributed random values to each attribute. The leaf reached after a traversal of the tree, according to the attribute values of an instance, determines its class value. RTG allows customizing the number of nominal and numeric attributes, as well as the number of classes. Originally, RTG does not incorporate a concept drift simulator. Since RTG is build upon a decision tree structure, it is assumed that decision tree based classifiers may obtain better results from it (BIFET et al., 2011).

2.4.4 SEA Generator

The SEA generator (STREET; KIM, 2001) can produce data streams with three continuous attributes (f_1, f_2, f_3). The range of values that each attribute can assume is between 0 and 10. Only the first two attributes (f_1, f_2) are relevant, i.e., f_3 does not influence the class value determination. New instances are obtained through randomly setting a point in a two dimensional space, such that these dimensions corresponds to f_1 and f_2. This two dimensional space is split into four blocks, each of which corresponds to one of four different functions. In each block a point belongs to class 1 if $f_1 + f_2 \leq \theta$ and to class 0 otherwise. The threshold θ used to split instances between class 0 and 1 assumes values 8 (block 1), 9 (block 2), 7 (block 3) and 9.5 (block 4). It is possible to add noise to class values, being the default value 10%, and to balance the number of instances of each class.

2.4.5 AGRAWAL Generator

The AGRAWAL generator (AGRAWAL; IMILIELINSKI; SWANI, 1993) can produce data streams with six nominal and three continuous attributes. There are ten different functions that map instances into two different classes. Table 2.1 presents the attributes domains. A perturbation factor is used to add noise to the data. This factor changes the original value of an attribute by adding a deviation value to it, which is defined according to a uniform random distribution.

Attribute	Type	Domain
Salary	Continuous	between 20k and 150k
Commission	Continuous	If salary $<$ 75k then 0 else between 10k and 75k
Age	Continuous	between 20 and 80
Elevel	Nominal	between 0 and 4
Car	Nominal	between 1 and 20
Zipcode	Nominal	between 9 different zipcodes
Hvalue	Continuous	between 0.5p100000 and 1.5p100000, such that $p \in 1 \ldots 9$ varying according to zipcode
Hyears	Continuous	between 1 and 30
Load	Continuous	between 0 and 500k

Table 2.1: AGRAWAL generator attributes, adapted from (BIFET et al., 2011)

2.4.6 HYPERPLANE Generator

The hyperplane generator was presented in (HULTEN; SPENCER; DOMINGOS, 2001). A hyperplane is a flat, $n-1$ dimensional subset of that space that divides it into two disconnected parts. It is possible to change a hyperplane orientation and position by slightly changing its relative size of the weights w_i. Formally, a hyperplane in a n-dimensional space is the set of points x that satisfy Equation 2.4.

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

Such that x_i is the ith coordinate of x. In a binary classification problem, instances where $\sum_{i=1}^{n} w_i x_i \geq w_0$ are labeled positive, and instances for which $\sum_{i=1}^{n} w_i x_i < w_0$ are labeled negative (HULTEN; SPENCER; DOMINGOS, 2001). This generator can be used to simulate time-changing concepts, by varying the values of its weights as the stream progresses (BIFET et al., 2011). The direction of changes alternate according to a probability σ.

2.4.7 Spam Corpus

The Spam Corpus dataset was developed in (KATAKIS et al., 2009) as the result of a text mining process on an online news dissemination system. The work presented in (KATAKIS et al., 2009) intended on creating an incremental filtering of emails classifying them as spam or ham (not spam), and based on this classification, deciding whether an email was relevant or not 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).

2.4.8 Forest Covertype

The covertype dataset (BLACKARD; DEAN, 1999) represents forest cover type for 30 x 30 meter cells obtained from the US Forest Service Region 2 Resource Information System (RIS) data. Each class corresponds to a different cover type. This data set contains 581,012 instances, 54 attributes (10 numeric and 44 binary) and 7 possible class labels.

2.4.9 Airlines

The Airlines dataset was inspired in the regression dataset from Ikonomovska[4]. The task is to predict whether a given flight will be delayed given information on the scheduled departure. Thus, it has 2 possible classes: delayed or not delayed. This dataset contains $539,383$ records with 7 attributes (3 numeric and 4 nominal).

2.4.10 Electricity

The Electricity dataset was collected from the Australian New South Wales Electricity Market, where prices are not fixed. These prices are affected by demand and supply of the market itself and set every five minutes. The Electricity dataset contains $45,312$ instances, where class labels identify the changes of the price (2 possible classes: up or down) relative to a moving average of the last 24 hours. An important aspect of this dataset is that it exhibits temporal dependencies.

2.4.11 GMSC

The Give Me Some Credit (GMSC) data set[5] is a credit scoring dataset where the objective is to decide whether a loan should be allowed. This decision is crucial for banks since erroneous loans lead to the risk of default and unnecessary expenses on future lawsuits. The data set contains historical data provided on 150,000 borrowers, each described by 10 attributes.

[4] <http://kt.ijs.si/elena_ikonomovska/data.html>
[5] Available at: https://www.kaggle.com/c/GiveMeSomeCredit. Last access in Dec. 9th, 2016.

2.4.12 KDD99

KDD'99 data set[6] is often used for assessing data stream mining algorithms' accuracy due to its ephemeral characteristics (AGGARWAL et al., 2003; AMINI; WAH, 2014). It corresponds to a cyber attack detection problem, i.e. attack or common access, an inherent streaming scenario since instances are sequentially presented as a time series (AGGARWAL et al., 2003). This data set contains $4,898,431$ instances and 41 attributes.

2.5 Final Considerations

In this chapter we have presented the main aspects of data stream learning, emphasizing the classification task. It is very important to situate the reader about the assumptions made, since they may not be true in some environments. For example, it is possible that labels are not readily available after an instance x^t before instance x^{t+1} is available. It would be unfruitful to develop novel classifiers for data streams without considering strategies that take into account concept drifts. The methods presented in this work are mainly based on dynamic ensemble classifiers, although they also include hybrid methods that combine drift detectors and dynamic ensemble classifiers. The evaluation procedures are utterly important, as weak experimental protocols cannot support strong statements. Therefore, we reviewed the current best practices on the field, including Prequential evaluation and the combined metric of k^+, which takes into account temporal dependence. In this work we present results based on the Prequential evaluation, but not yet results based on k^+. The reason for not using k^+ yet is because it is not clear how this measure behaves, in comparison with Prequential accuracy, if the stream does present temporal dependences. Instead we decided to report both Kappa M and Kappa T instead of using the geometric average. In the next chapter we discuss Ensemble Learning, first highlighting its fundamental concepts, and latter discussing its variations for data stream learning.

[6] <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

Chapter 3

Ensemble Learning

Ensemble classifiers are widely used as an alternative to obtain accurate models through the combination of weak classifiers. During recent years, well-known ensemble methods for traditional machine learning have been adapted to cope with data streams, as well as new methods.

There are two main components that must be taken into account while developing an algorithm based on an ensemble of classifiers (POLIKAR, 2006). Firstly, the building strategy of the ensemble must enforce diversity among its component classifiers. Secondly, the output combination method must highlight correct and obfuscate incorrect predictions. Also, in a stream learning context it is important to consider adaptation techniques, since drifts are expected to occur.

Despite the difficulty around proving the effectiveness (and relationship) between accuracy and diversity, in practice ensemble classifiers tend to overcome single classifiers in a multitude of problems. In (KUNCHEVA, 2004), three reasons to use ensemble classifiers are presented, i.e., Statistical, Computational and Representational.

Statistical. Assuming that D_k samples of a training data set D are used to train K classifiers, such that each of them obtain 100% accuracy on the subset of D_k used for training them. It is possible that the generalization capabilities of these classifiers will be different when they are applied to a test set disjoint from D, therefore their accuracy will vary. From a statistical point of view it is safer to use the mean of individual predictions from these K classifiers instead of using only one of them, since the chance of selecting the classifier with the worst generalization capabilities is eliminated. There is a chance that the ensemble accuracy is worst than that of the best of its members, but the goal here is to avoid selecting the worst classifier.

Computational. Some classifiers may converge to a local maximum. Suppose that the local maxima of K classifiers are close to the absolute maximum. Such that, there

is a way of combining them in a model even closer to the absolute maximum (optimal classifier) than any of them is capable individually.

Representational. The classifier used may not be able to represent the separation surface of the given problem. For example, a classifier based on decision trees, e.g., C4.5 (QUINLAN, 1993) is capable only of linear separations, therefore when applied to a problem where data is not linearly separable, a single decision tree is not capable of achieving good results. Although the combination of linear classifiers can approximate a non-linear separation surface, similarly to a non-linear classifier.

One of the main goals of an ensemble classifier is to permit that each classifier be as unique as possible, particularly with respect to classification errors (POLIKAR, 2006). If an ensemble is composed of classifiers that misclassify different instances, complementing each other, then this set is said to be diverse and it is probably going to have accuracy above any of its members individually. On the other hand, it is possible that by combining classifiers the overall accuracy is worst than any of the classifiers individually.

To illustrate the reasons previously mentioned, we now present a classical example, first presented in (KUNCHEVA, 2004), of ensemble application that motivates its use. Consider a problem with 10 instances and an ensemble of 3 classifiers, such that each classifier has an individual accuracy of 60% on those 10 instances, i.e., each classifier is capable of correctly classifying 6 out of 10 instances. Assuming all possible combinations there 28 possible cases in which 3 classifiers can obtain 60% accuracy for a data set of 10 instances. Table 3.1 presents these 28 combinations (rows) and the overall accuracy according to a simple majority vote. In Table 3.1 columns a to h presents the sum of correct predictions made to the same instance for the 3 classifiers simultaneously (column a), only for the first and last classifier (column b), and so forth. The optimal combination achieves an accuracy of 90% (row 1). In practice, it is unlikely to happen, but the chances of a good combination (rows 2 to 12) or at least the same accuracy as individual classifiers (rows 13 to 23) are more probable. There is a chance that the overall accuracy is worst than that obtained by any of the ensemble members, but it is less probable than the other (rows 24 to 28). Notice that, even though the chances are in favor of best or similar for 3 classifiers with 60% accuracy in a data set of 10 instances, there is not a general formula for higher orders of instances or classifiers and empirical tests are difficult since there are too many possible combinations, even though some experiments and discussion can be found at (KUNCHEVA, 2004).

In the following sections we describe the most distinctive characteristics of data stream ensemble classifiers, i.e., combination, diversity and update dynamics. Even though, specific algorithms may be referenced, detailed information about some of them

ID	a 111	b 101	c 011	d 001	e 110	f 100	g 010	h 000	Overall Accuracy	Overall - Individual
1	0	3	3	0	3	0	0	1	0.9	0.3
2	2	2	2	0	2	0	0	2	0.8	0.2
3	1	2	2	1	3	0	0	1	0.8	0.2
4	0	2	3	1	3	1	0	0	0.8	0.2
5	0	2	2	2	4	0	0	0	0.8	0.2
6	4	1	1	0	1	0	0	3	0.7	0.1
7	3	1	1	1	2	0	0	2	0.7	0.1
8	2	1	2	1	2	1	0	1	0.7	0.1
9	2	1	1	2	3	0	0	1	0.7	0.1
10	1	2	2	1	2	1	1	0	0.7	0.1
11	1	1	2	2	3	1	0	0	0.7	0.1
12	1	1	1	3	4	0	0	0	0.7	0.1
13	6	0	0	0	0	0	0	4	0.6	0
14	5	0	0	1	1	0	0	3	0.6	0
15	4	0	1	1	1	1	0	2	0.6	0
16	4	0	0	2	2	0	0	2	0.6	0
17	3	1	1	1	1	1	1	1	0.6	0
18	3	0	1	2	2	1	0	1	0.6	0
19	3	0	0	3	3	0	0	1	0.6	0
20	2	1	1	2	2	1	1	0	0.6	0
21	2	0	2	2	2	2	0	0	0.6	0
22	2	0	1	3	3	1	0	0	0.6	0
23	2	0	0	4	4	0	0	0	0.6	0
24	5	0	0	1	0	1	1	2	0.5	-0.1
25	4	0	0	2	1	1	1	1	0.5	-0.1
26	3	0	1	2	1	2	1	0	0.5	-0.1
27	3	0	0	3	2	1	1	0	0.5	-0.1
28	4	0	0	2	0	2	2	0	0.4	-0.2

Table 3.1: Three classifiers combination, adapted from (KUNCHEVA, 2004)

are discussed only in Section 3.5. A summary of over sixty ensemble classifiers for data streams is presented in Table 4.1 (Section 4.1) accompanied by our proposed taxonomy to organize ensemble learning for data streams.

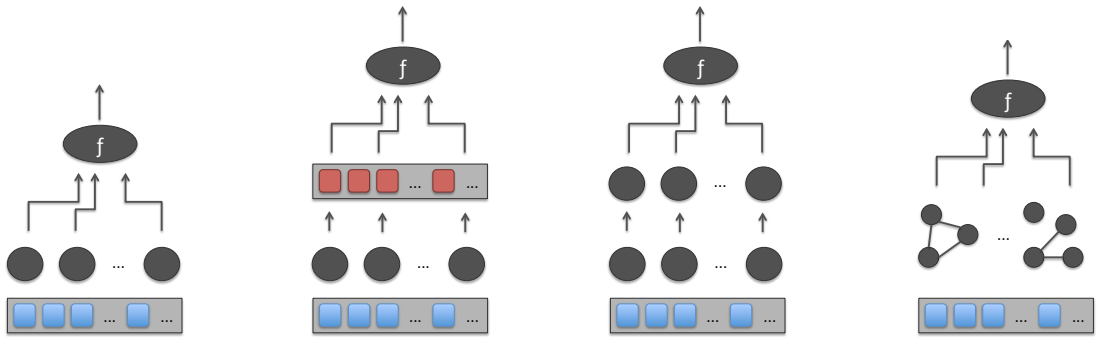
3.1 Combination

Combining ensemble members' predictions can either enhance the overall performance or jeopardize it. Through an appropriate combination method it is expected to correctly predict the class label of difficult to classify instances. However, a lot of effort has been dedicated to the generation of diverse sets of classifiers, and not so much on methods to combine classifiers outputs (TULYAKOV et al., 2008). For example, the original bagging algorithm (BREIMAN, 1996) emphasizes the construction of a diverse ensemble, while its combination method is a simple majority vote. There are many approaches to handle voting for ensemble classifiers, varying from simple methods (e.g. majority vote) to more complex approaches (e.g. Behavior-Knowledge Space (HUANG; SUEN, 1993)).

In this work we differentiate between the voting method and how ensemble members are arranged (architecture). Very often ensemble members' arrangement and voting method are intrinsically related, thus while explaining the inner workings of a specific algorithm it is sometimes much clearer to not separate them into different blocks. However, while analysing multiple ensemble methods it is reasonable to differentiate between voting and ensemble architecture, since it facilitates the understanding of algorithms in which the ensemble members' arrangement is not trivial. For example, SAE (GOMES; ENEMBRECK, 2013), SAE2 (GOMES; ENEMBRECK, 2014) and SFNC (BARDDAL; GOMES; ENEMBRECK, 2014) arrange the ensemble members in a network (graph) structure, while the ensemble of Restricted Hoeffding trees (BIFET et al., 2012) uses a meta-level combiner trained on the outputs of a set of decision trees trained on the input data. Also, there are situations in which the architecture can be used for more than voting, for example, it can be used to control the ensemble training (CHAN; STOLFO, 1995) or to enable operators, e.g., a redundant classifier removal method (GOMES; ENEMBRECK, 2013; GOMES; ENEMBRECK, 2014). In the following sections we discuss different architectures and voting methods for ensemble learning in a data stream context.

3.1.1 Architecture

The ensemble architecture defines how classifiers interact with one another. In (JAIN; DUIN; MAO, 2000), authors organize ensemble methods, for batch learning, into



(a) Flat (b) Meta-Learner (c) Hierarchical (d) Network

Figure 3.1: Ensemble structural arrangement (circles = classifiers; squares = instances)

three different architectures: parallel, cascading and hierarchical. In the parallel architecture each classifier output is aggregated by a combiner, such that the combiner can be a simple linear function (e.g. weighted vote) or another classifier (e.g. stacking (WOLPERT, 1992)). Cascading includes architectures in which the output of one classifier is the input of another for multiple levels (ALPAYDIN; KAYNAK, 1998; GAMA; BRAZDIL, 2000). Finally, an ensemble in which members are arranged in a tree-like structure is classified as hierarchical. Our taxonomy for ensemble architecture differs from that presented in (JAIN; DUIN; MAO, 2000), as we split parallel into two different architectures, combine cascading and hierarchical into one and include a new architecture. Concretely, we classify a given ensemble structure as either: flat (parallel with simple combiner), meta-learner (parallel with meta-learners), hierarchical (cascading or tree-like structure) or network (graph structure). Figure 3.1 presents a schematic view of these four different structural arrangements of ensembles.

Flat. The flat structure assumes that base models are trained on the input data and the decision fusion is delegated to a simple combination function (voting scheme) such as majority voting. In comparison to other arrangements this is the most widely used, partly because it is simple, but also because it makes fewer assumptions about individual classifiers. Examples of data stream ensemble classifiers that employ a flat structure includes: Online Bagging (OZA, 2005), DWM (KOLTER; MALOOF, 2007), Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010), and many others.

Meta-learner. In a meta-learning structure the combiner (meta-learner) is trained on meta-data, which may refer to properties of the learning problem (MINKU; YAO, 2012) or to the outputs of learners trained on the input data (BIFET et al., 2012). From a high-level perspective the flat and meta-learner approaches may seem very similar, however they are effectively different since the latter involves creating a meta-dataset and training a meta-learner on it. Despite meta-learning being feasible without an ensemble structure

(BRAZDIL et al., 2008; GAMA; KOSINA, 2009) in this work we focus on meta-learning for ensemble classifiers. A canonical example of meta-learning is the stacking algorithm (WOLPERT, 1992). Stacking creates a meta-dataset where every meta-instance corresponds to an instance in the original dataset. This meta-dataset replaces the original instances' inputs by the predictions of each ensemble member, while the class label remains the original. A meta-classifier is induced from the meta-dataset, which during predictions is responsible for combining component classifiers predictions into a final one. An example of stacking for data stream classification is the ensemble of Restricted Hoeffding trees (BIFET et al., 2012), where the first level of learners is composed of Hoeffding Trees, while the meta-learner level is formed by perceptrons (one per class label).

Hierarchical. We classify as hierarchical any ensemble that imposes a tree-like structure or a strict order (cascading) over its members. Examples of batch ensembles in which the structural arrangement adhere to this definition includes Arbiter Trees (CHAN; STOLFO, 1995), Combiner Trees (CHAN; STOLFO, 1997) and Hierarchical Misture of Experts (HME) (JORDAN; JACOBS, 1994). There are ensemble methods for data stream classification that use hierarchical structures, most notably HSMiner (PARKER; MUSTAFA; KHAN, 2012) and SluiceBox (PARKER; KHAN, 2013). HSMiner is a hierarchical additive weighted voting ensemble that boosts the accuracy of a set of weak learners by decomposing the learning problem. At the top tier of HSMiner's hierarchy stands a multi-class ensemble of k per-class ensembles, where k corresponds to the number of current class labels. Each per-class ensemble is composed of single class ensembles, which are further decomposed into single feature classifiers. Besides the top tier ensemble, all other classifiers that compose HSMiner's hierarchy of learners are all committed to distinguish one class from all others (i.e. single class learners), thus HSMiner performs a one-vs-all decomposition of multi-class problems. At the bottom of HSMiner's hierarchy, single feature classifiers learn a model that discerns between the class label it represents (positive label) from all others (negative label) using only one feature. To avoid pre-processing, if the feature domain is discrete then a Naive Bayes classifier is used, otherwise (domain is continuous) an AdaBoost ensemble of threshold learners is used. Finally, in our classification the main difference between meta-learner and hierarchical structures is that we consider the former to only include one level of learners which outputs are used to train second level learners, while the latter may organize learners hierarchically for other purposes, such as decomposing the input data.

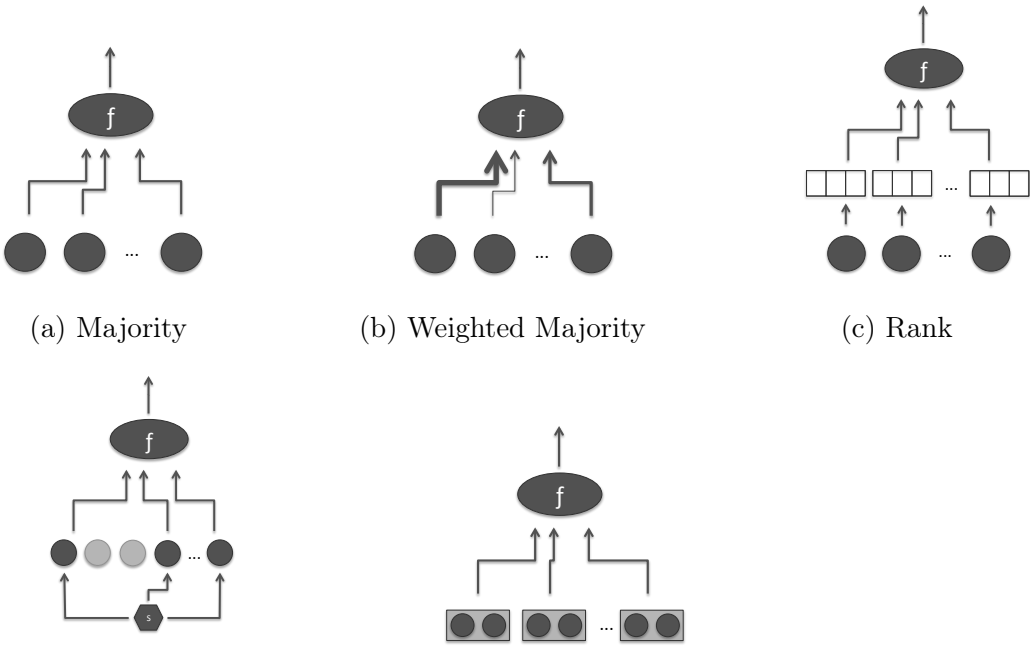
Network. The last ensemble structure in our taxonomy includes methods that arrange the ensemble members in a graph. We use the term network instead of graph to refer to these structures since they are often dynamic and thus most closely related to

complex networks (ALBERT; BARABÁSI, 2002) than to static graphs. In this structure, ensemble members are represented as vertices of a network whose connections are determined according to a specific criterion. In SFNC (BARDDAL; GOMES; ENEMBRECK, 2014) connections between classifiers are generated according to a Scale-Free Network model, such that classifiers with higher estimated accuracy are more likely to connect to recently added classifiers. During voting, classifiers weighting is directly proportional to a given centrality metric α, e.g., eigenvector, betweenness, degree, etc. Since high accuracy classifiers usually receive most of the connections, these are expected to have higher influence on the overall decision. This thesis is dedicated to network-based ensembles, thus to avoid redundancy we do not include further explanations here.

Our goal with the architectures presented is to be as general as possible while emphasizing distinguishable characteristics of actual algorithms. Although, some algorithms, which could be interpreted as ensembles, do not fit in any of these architectures. An example is the Option Hoeffding Trees (PFAHRINGER; HOLMES; KIRKBY, 2007), which is basically an algorithm that could be interpreted as either a decision tree or an ensemble. The ensemble architecture, most of the time, is chosen to accomplish a specific voting scheme.

3.1.2 Voting

Voting concerns how individual outputs from ensemble members are used during prediction. The ensemble structure influences voting, e.g., it is possible that some learners' outputs are only used as input for another level of meta-learners. There are many voting schemes for ensemble learning, which we organize into five categories: Majority, Weighted Majority, Rank, Classifier Selection and Relational. The voting categories can be organized in an hierarchy, such that: majority vote is the most basic method; weighted majority extends majority vote by allowing heterogeneous weights; rank is a form of weighted vote where all outputs from all learners are considered; classifier selection sets weights dynamically; and relational voting transform individual votes, through an intricate method, before applying majority or weighted voting. These categories are general enough to represent voting in batch learning, stationary and evolving data streams. However, in an evolving data stream scenario it is common that the voting method also plays an important role with respect to concept drift adaptation. For example, a simple strategy consists of weighting learners based on their age or on their performance restricted to the latest instances (GOMES; ENEMBRECK, 2013; GOMES; ENEMBRECK, 2014; BARDDAL; GOMES; ENEMBRECK, 2014; BRZEZINSKI; STEFANOWSKI, 2014). These



(a) Majority (b) Weighted Majority (c) Rank

(d) Classifier Selection (e) Relational

Figure 3.2: Ensemble basic voting methods

strategies can be build on top of the five voting categories, thus they should not be interpreted as a separate category. Figure 3.2 depicts our five basic voting categories and the following paragraphs discuss each of them.

Majority Vote. Majority vote assumes every classifier has the same weight on the overall ensemble decision. Thus, the final prediction is the class label that most classifiers predicted. To avoid ties in a binary classification setting, it is usual to define an odd number of base learners. In multiclass problems, ties can become a concern and the default approach is to randomly break them. Examples of data stream ensembles that use majority vote includes: Online Bagging and Boosting (OZA, 2005), Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010) and the MCIK-Ensemble (MASUD et al., 2008).

Weighted Majority. It is reasonable to weight classifiers' predictions according to some criteria. For example, it is possible to assign a score to each classifier based on its accuracy on a validation set. A more complex method is the Weighted Majority (WM) algorithm (LITTLESTONE; WARMUTH, 1994). WM weights the predictions of classifiers based on their past performance, such that every classifier has a weight β, which is decreased every time it predicts incorrectly. Majority and weighted vote share the characteristic of only considering one prediction per classifier, i.e., each classifier chooses one class label. The Dynamic Weighted Majority algorithm (KOLTER; MALOOF, 2007) as

well as other data stream ensemble classifiers rely on some form of weighted majority, to name a few: Accuracy Update Ensemble (AUE) (BRZEZIŃSKI; STEFANOWSKI, 2011), Online Accuracy Update Ensemble (OAUE) (BRZEZINSKI; STEFANOWSKI, 2014), Adaptive Classifiers-Ensemble (ACE) (NISHIDA; YAMAUCHI; OMORI, 2005), Weighted Ensemble Online Bagging (WEOB) (WANG; MINKU; YAO, 2015).

Rank. In situations where the base learner can output more than one class label per prediction a voting method, which is similar to the weighted majority approach, can be used to combine all predictions. For example, if the base learner prediction is a sorted list of class labels, then the Borda count method can be used. Borda count (BORDA, 1781) is a preferential voting system introduced in 1770 by Jean Charles de Borda. In ensemble learning, the overall decision when using Borda count is the class label with the highest rank sum. An example of batch ensemble that uses rank based voting (Borda count) is the Nearest Neighbor Ensemble (DOMENICONI; YAN, 2004).

Classifier Selection[1]. Classifier selection concerns selecting the most 'appropriate' classifiers for predicting the class label of an unknown instance. The selection can take place during training (SCHAFFER, 1993) or prediction (MERZ, 1996; WOODS; BOWYER; JR, 1996), which are commonly known as static (SCS) and dynamic classifier selection (DCS), respectively. Usually, DCS involves storing instances used for training each learner and then using a K nearest neighbor approach to determine which classifiers should be combined for predicting an unknown instance. This naive DCS approach is infeasible on a data stream setting as the impact of storing all instances may surpass available memory, or simply cause the algorithm to take too long to calculate all needed distances. On top of that, selecting an appropriate distance function and setting K are challenging tasks. DCS has been used in several ensemble methods for data stream classification, such as the Coverage Base Ensemble Algorithm (CBEA) (RUSHING et al., 2004), the Attribute-Oriented Dynamic Classifier Selection (AO-DCS) (ZHU; WU; YANG, 2004) and the Conceptual Clustering and Prediction (CCP) (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010). In CCP to avoid storing a large amount of instances to describe each classifier a clustering algorithm is used to 'summarize' the instances representation, i.e., only the instances' centroids are stored.

Relational. Instead of interpreting each ensemble member prediction individually, and literally, these can be translated to reflect the class label that they most likely represent. For example, suppose two classifiers c_i and c_j consistently choose class labels 0 and 1, respectively, whenever the true class label is 2. Then it would be reasonable to

[1] A related term to classifier selection is a gating network, this is often used in Artificial Neural Networks literature (JACOBS et al., 1991).

'translate' to class label 2 whenever classifier c_i predicts 0 and c_j predicts 1. This is a powerful voting strategy, as it allows a group of learners to indirectly predict the class label of hard to classify instances. This voting strategy is represented by the Behavior-Knowledge Space (BKS) (HUANG; SUEN, 1993), in batch learning, and a similar version appears for online learning as Pairwise Patterns (PP) (GOMES; BARDDAL; ENEMBRECK, 2015). The Learn^{++}.NC algorithm (MUHLBAIER; TOPALIS; POLIKAR, 2009) uses a voting method, namely the dynamically weighted consult and vote (DW-CAV), in which classifiers consult its peer decisions to check if its decision is aligned with them and with the classes it was trained on. If the classifier identifies discrepancies, it reduces its vote, or even refrains from voting. Other voting strategies could be classified as 'Relational', yet they differ from BKS, PP and DW-CAV significantly. For example, even though SAE (GOMES; ENEMBRECK, 2013), SAE2 (GOMES; ENEMBRECK, 2014) and SFNC (BARDDAL; GOMES; ENEMBRECK, 2014) extract relational data from ensemble members to generate networks, they are more closely related to a multilevel weighted majority vote, since these algorithms do not map original outputs to a different domain. Finally, relational voting can be related to the meta-learner structural arrangement (see Section 3.1.1), where the 'vote' translation is delegated to a learner trained on the first layer learners' outputs.

Different voting strategies are biased towards specific assumptions regarding the problem. For example, a voting method that takes into account the class label distribution can outperform another method that does not, especially for imbalanced data streams (WANG; MINKU; YAO, 2013; WANG; MINKU; YAO, 2015). However, a simple voting scheme may overcome a complicated method that considers a variety of factors, usually because the assumptions that the latter is based on do not hold for the problem at hand. For example, a weighted majority vote strategy will perform poorly if the weighting function fails to represent each classifier true prediction capabilities. Nevertheless, the ensemble structure and the voting method are useless unless ensemble members are diverse with respect to their outputs. Finally, some authors focus on determining the limits of majority voting (KUNCHEVA et al., 2003) or on comparing multiple voting methods using a probabilistic framework (KUNCHEVA; RODRÍGUEZ, 2014), yet conclusions obtained are often limited to specific cases.

3.2 Diversity

Diversity is often identified as one of the building blocks of ensemble-based classifiers. The motivation for the importance of diversity can be intuitively explained using the

anthropomorphic example of a group of individuals, such that their opinions are always homogeneous. This group can safely be replaced by any of its members if its only purpose is decision making.

Although some works are able to show correlations between accuracy and specific diversity measures for some special cases (KUNCHEVA; WHITAKER, 2003), theoretical guarantees are more complicated to obtain, and often inconclusive, for the general case. Unfortunately, it is not as simple as 'augment diversity measure d and the overall accuracy will improve'. This problem is even more complicated because there is no generally accepted definition of diversity (KUNCHEVA, 2004).

Only a few studies on ensemble learning for data streams are focused on measuring diversity and on diversity properties (MINKU; WHITE; YAO, 2010; MINKU; YAO, 2012). However, most ensemble learners proposals are accompanied with strategies to induce diversity. In the following sections we present different strategies to induce diversity and classical metrics to measure diversity. Appendix A presents experiments that illustrate how diversity can be monitored during stream execution for different ensemble methods.

3.2.1 Inducing diversity

For our purposes, a set of diverse classifiers is analogous to a set of non-trivial classifiers (i.e. consistent with the training data) that given the same instance output different predictions. This definition assumes that learners cannot be random guessers, although it does not consider diverse a set of learners with different internal representations that consistently predict the same class labels.

In this work we organize methods to induce diversity based on whether they manipulate the input data, the output predictions, the underlying classifiers or use a set of heterogeneous base learners. A similar classification of diversity inducing methods is presented in (ROKACH, 2010) with a slightly different nomenclature.

Input manipulation. Methods that manipulate the input are common and include training classifiers in different chunks of data (horizontal partitioning) or with different subsets of features (vertical partitioning). Training classifiers with different instances often includes some form of randomization, e.g., bagging uses resampling (BREIMAN, 1996). The problem with resampling in a data stream environment is that it requires multiple passes over data. That is infeasible, since streams are expected to provide an infinite amount of data. To solve this problem in (OZA, 2005) authors propose a method that approximates resampling for online processing. Besides sampling, a stream can be partitioned horizontally by "adding classifiers at different points of the stream" (KOLTER;

MALOOF, 2007; BRZEZINSKI; STEFANOWSKI, 2014; BARDDAL; GOMES; ENEMBRECK, 2014). For example, classifier c_1 is added at moment t and classifier c_2 is added at moment $t + \delta$, thus c_2 will be only trained with instances provided after $t + \delta$ while c_1 is trained with instances after t. This latter strategy can also be used to indirectly adapt to drifts, but it can potentially jeopardize diversity if there are no concept drifts because classifiers that have been added not too far apart become very similar after a while. Instead of training classifiers on different subsets of instances it is possible to train them on different subsets of features (HO, 1995; AMIT; GEMAN, 1997; HO, 1998b; BREIMAN, 2001). This strategy is known as the Random Subspace Method (RSM). For a feature space of M dimensions, there are $2^M - 1$ different non-empty subsets of features, thus it is infeasible to train one learner for each subset given a high dimensional dataset, especially on a data stream setting (BIFET et al., 2012). Nevertheless, Ho noted in (HO, 1998b), based in the theory of stochastic modeling, that highly accurate ensembles can be obtained far before all possible combinations of subspaces are explored. RSM is usually associated with decision trees, however in its general form it can be applied to different base learners, such as nearest neighbors (HO, 1998a) or linear classifiers (SKURICHINA; DUIN, 2002). The reason behind the association of decision trees with RSM is attributable to the Random Forests algorithm (BREIMAN, 2001), an ensemble method in which the random feature selection is intrinsically related to how its learners (decision trees) are trained, i.e., every node split is based on a (potentially) different random subset of features. Training ensembles using RSM yield several advantages, such as diversity enhancement and efficient training and prediction. The former depends upon the base learner's instability (see Section 3.3), while the latter may occur if ensemble member's training is independent, which permits training several learners in parallel. Also, on high dimensionality problems, such as functional magnetic resonance imaging (fMRI) classification, RSM can be used to ease the impact of the 'curse of dimensionality' by using small subsets of features per learner (KUNCHEVA et al., 2010). Examples of RSM for data stream classification includes: Streaming Random Forests (ABDULSALAM; SKILLICORN; MARTIN, 2007), Restricted Hoeffding Trees (BIFET et al., 2012), Dynamic Streaming Random Forests (ABDULSALAM; SKILLICORN; MARTIN, 2008) and the Ensemble Decision Trees for Concept-drifting data streams (EDTC) (LI et al., 2015).

Output manipulation. To manipulate the output of a classification problem one could decompose the original problem into smaller, potentially easier, problems. Afterwards, each problem can be mapped to a single classifier, and these classifiers would be diverse since they interpret the hypothesis space differently. One classifier that is capable of differentiating between multiple classes is difficult to achieve, while a set of

binary classifiers is relatively easy to obtain. Therefore, to cope with multiclass problems many ensembles use the One-Versus-All approach, i.e., decompose the original problem into $k(k-1)/2$ binary problems, and assign a different classifier to each class, such that instances associated with other classes are interpreted as negative examples by the given classifier. Decomposing the problem in tractable smaller problems is the main goal of this strategy, while diversity increase can be viewed only as a by-product. Examples of algorithms that use this strategy for data stream learning include the One-versus-All Decision Trees (HASHEMI et al., 2009) and HSMiner (PARKER; MUSTAFA; KHAN, 2012). There are some difficulties when applying this strategy to data stream learning, such as concept drifts, imbalanced class distributions and the high update cost (HASHEMI et al., 2009). A slightly different manipulation of the output can be achieved by using Error-Correcting Output Codes (ECOC) (DIETTERICH; BAKIRI, 1995). ECOC were inspired by the Error-Correcting Codes (ECC) presented in Shannon's information theory (SHANNON, 1948), and were originally used to decompose multiclass problems into binary problems. In (BIFET; HOLMES; PFAHRINGER, 2010), the authors experiment with a version of Leveraging Bagging that uses a variation of ECOC, namely random output codes. In random output codes class labels assigned to each example are modified to create a new binary classification of the data induced by a mapping from all possible labels to $\{0, 1\}$. Effectively, in this setting every classifier has a different view of the hypothesis space, e.g., one classifier may interpret class labels A and B as 0, while C and D as 1. For practical reasons, in (BIFET; HOLMES; PFAHRINGER, 2010) the algorithm balances the 0s and 1s for each classifier, to prevent them from mapping all original labels to 0 or to 1.

Base learner manipulation. In order to achieve diversity, it is possible to modify characteristics of each base model. For example, one could use multiple neural networks with different topologies, or with the same topology, but starting with different weights at the first layer (ROKACH, 2009). In (BIFET et al., 2009) authors propose the Adaptive Size Hoeffding Trees (ASHT) Bagging algorithm, which is an ensemble of decision trees of varying sizes. ASHT is based on the intuition that smaller trees are able to rapidly adapt to drifts, while bigger trees are useful during stable periods, thus mixing both yield different ensemble members, and may also contribute to drift recovery.

Heterogeneous base learners. Instead of varying parameters of the same base learner, it is possible to use heterogeneous base learners and obtain ensemble members with different biases. Heterogeneous ensembles for data stream learning includes BLAST (RIJN et al., 2015), HEFT-Stream (NGUYEN et al., 2012) and HSMiner (PARKER; MUSTAFA; KHAN, 2012). BLAST trains several different base learners and during pre-

diction selects one of them through a meta-learning approach. HEFT-Stream maintains an ensemble of decision trees and naive bayes learners, and in the occurrence of a sudden drift it adds a new learner, whose base learner matches the current learner with highest weight. HSMiner (PARKER; MUSTAFA; KHAN, 2012) uses two different base learners to avoid preprocessing features, naive bayes for discrete and threshold learners for continuous, thus if the feature set is heterogeneous with respect to features' domains then learners will also be heterogeneous.

Depending on which strategy is employed for inducing diversity into the ensemble, one must be aware that while processing a massive (potentially infinite) data stream, members' models may converge. That is especially true for methods that relies solely on adding (or resetting) models on different moments of the stream. Also, instead of committing to one or another strategy to induce diversity, it is possible to combine two or more strategies. For example, HEFT-Stream trains heterogeneous learners on different samples (online bagging) and subspaces of data[2]. To assess how effective one diversity inducing strategy is, one could choose to observe the ensemble overall accuracy. However, this analysis is biased since there may be other factors that influence accuracy. In the next section, we present some diversity measuring metrics and examples of their use to assess diversity in a data stream setting.

3.2.2 Measuring diversity

There are a few reasons to measure the diversity among members of an ensemble. The most obvious is based on the intuitive notion that an ensemble of homogeneous classifiers cannot achieve any better than any of its members alone can. Thus, it may seem logical to maximize diversity, since doing so will consequently have a good impact on the overall results. Although before optimizing for diversity, it is necessary to keep in mind that no general correlation between diversity and accuracy has been proven (KUNCHEVA et al., 2003). Although high diversity may not directly indicate high accuracy, measuring diversity can be useful to analyze the effectiveness of the diversity inducing method, and even to more specific tasks such as pruning ensemble members (MARGINEANTU; DIETTERICH, 1997; GOMES; ENEMBRECK, 2013; GOMES; ENEMBRECK, 2014).

Diversity can be measured in multiple levels, but it is usually calculated in pairs (pairwise) or for the complete ensemble (non-pairwise or aggregated). We now present a few pairwise metrics, but before we define some concepts that assist in the definition of

[2]HEFT-Stream periodically runs a feature selection algorithm (Fast Correlation-Based Filter - FCBF (YU; LIU, 2003)) and new learners are only trained with the latest subset of features deemed relevant.

Table 3.2: All possible outputs of a pair of classifiers h_u and h_v

	h_u correct (1)	h_u incorrect (0)
h_v correct (1)	N^{11}	N^{10}
h_v incorrect (0)	N^{01}	N^{00}

such metrics (KUNCHEVA; WHITAKER, 2003).

Let $X = x_1, \ldots, x_n$ be a labeled data set, $\hat{y}_v = [\hat{y}_v(x_1), \ldots, \hat{y}_v(x_n)]$ a n-dimensional binary vector that represents the output of a classifier h_v, such that $\hat{y}_v(x_j) = 1$, if h_v correctly predicts the class label for instance x_j, and 0, otherwise. Table 3.2 presents all the possible outcomes for a pair of classifiers h_u and h_v, such that $h_u \neq h_v$, where N^{ab} is the number of instances $x_j \in X$ for which $\hat{y}_u(x_j) = a$ and $\hat{y}_v(x_j) = b$.

The Yule's Q statistic (YULE, 1900) (Equation 3.1), or simply Q, is a widely used measure of diversity in many fields. Q varies between -1 and 1, such that for statistically independent classifiers the expectation of $Q_{v,u}$ is 0. Classifiers that tend to correctly predict the same instances yield positive values of Q, while those that commit errors on different instances render negative values.

$$Q_{v,u} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{3.1}$$

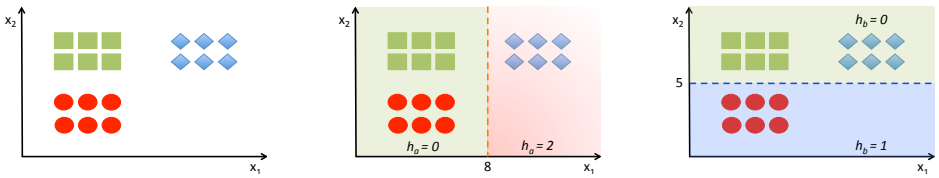
Another way of estimating the pairwise diversity is the correlation coefficient $p_{v,u}$ (Equation 3.2). For any two classifiers, Q and p have the same sign, and it can be proved that $|p| \leq |Q|$. Since Q is easier to calculate it is usually preferred.

$$p_{v,u} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11} + N^{10})\,(N^{01} + N^{00})\,(N^{11} + N^{01})\,(N^{10} + N^{00})}} \tag{3.2}$$

The disagreement measure $D_{v,u}$ (Equation 3.3) is used to characterize diversity between a base classifier h_v and a complementary classifier h_u. This metric is symmetrical, and correlated with $Q_{v,u}$ and $p_{v,u}$. $D_{v,u}$ represents the ratio between the number of instances on which one classifier is correct and the other is incorrect with respect to the total number of instances.

$$D_{v,u} = \frac{N^{01} + N^{10}}{N^{11} + N^{00} + N^{01} + N^{10}} \tag{3.3}$$

The measures presented so far are all based on concomitant correct or incorrect predictions. For binary classification problems the way matrix N is calculated (see Table 3.2) is sound, since if classifiers h_v and h_u incorrectly predict instance x class label, then h_v and h_u predictions must be equal, i.e., if the correct label was 0, then h_v and h_u both predicted 1. Although, for multiclass classification problems matrix N may fail to



(a) Classification space (b) Linear classifier h_a (c) Linear classifier h_b

Figure 3.3: 2-dimensional multiclass problem (labels: green squares = 0, blue diamonds = 1 and red circles = 2)

Table 3.3: Correct and incorrect counters N^{ab} for h_a and h_b according to Figure 3.3

	h_a correct (1)	h_a incorrect (0)
h_b correct (1)	6	0
h_b incorrect (0)	0	12

measure differences between classifiers that incorrectly predict the same instance using different labels. For example, given a 3-class classification problem, two linear classifiers h_a and h_b, such that

$$h_a = \begin{cases} 2, & x_1 \geq 8 \\ 0, & x_1 < 8 \end{cases}$$

$$h_b = \begin{cases} 0, & x_2 \geq 5 \\ 1, & x_2 < 5 \end{cases}$$

and the distribution of instances according to Figure 3.3, the classification errors of h_a and h_b will be accounted equally by the measures previously discussed. Table 3.3 shows the distribution of correct and incorrect predictions for h_a and h_b. If Q statistic is used to assess h_a and h_b diversity we obtain $Q_{a,b} = 1$, which indicates that both classifiers tend to correctly predict the same instances, yet it fails to express their divergences on incorrect predictions.

To precisely capture differences between classifiers in a multiclass problem context a possible approach is to keep track of the classifiers' exact predictions instead of only the dichotomy correct/incorrect. This can be achieved by constructing a contingency table C_{ij}, such that the value at the intersection of a row i and a column j stores the amount of instances $x \in X$ where $h_v(x) = i$ and $h_u(x) = j$. Table 3.4 shows an example of contingency table C_{ij} for a k-class problem. The diagonal in matrix C_{ij} contains the concomitant decisions of the pair, thus a naive approach to weight their similarity is to sum its values and divide it by the amount of instances n, as shown in Equation 3.4.

Table 3.4: All possible outputs of a pair of classifiers h_v and h_u for a multiclass classification problem with K possible labels

	$h_u(x) = 0$	$h_u(x) = 1$	...	$h_u(x) = (k-1)$
$h_v(x) = 0$	C_{00}	C_{01}	...	$C_{0(K-1)}$
$h_v(x) = 1$	C_{10}	C_{11}	...	$C_{1(K-1)}$
...	...	...	...	...
$h_v(x) = (K-1)$	$C_{(K-1)0}$	$C_{(K-1)1}$	...	$C_{(K-1)(K-1)}$

$$\Theta_1 = \frac{1}{n}\sum_{i=0}^{k} C_{i,i} \tag{3.4}$$

As noted in (MARGINEANTU; DIETTERICH, 1997) in problems where one class is much more common than others, all classifiers may agree by chance, so all pairs will obtain high values for Θ_1. Thus, it is more appropriate to use the Kappa κ statistic[3], since it corrects Θ_1 by considering the probability that two classifiers agree by chance according to the observed values in C_{ij}, namely Θ_2 (see Equation 3.5). The kappa κ statistic is shown in Equation 3.6.

$$\Theta_2 = \sum_{i=0}^{K}\left(\sum_{j=0}^{K}\frac{C_{i,j}}{n}\cdot\sum_{j=0}^{K}\frac{C_{j,i}}{n}\right) \tag{3.5}$$

$$k = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2} \tag{3.6}$$

The interpretation of κ is similar to Q, i.e., if h_u and h_v agree on every instance then $\kappa = 1$, and if their predictions coincide by chance, then $\kappa = 0$. Negative values of κ occurs when agreement is weaker than expected by chance, but this rarely occurs in real problems. The Kappa statistic has already been used to report diversity for data stream ensemble-based classifiers (BIFET et al., 2009; BIFET; HOLMES; PFAHRINGER, 2010; KUNCHEVA et al., 2010) and is often accompanied by a Kappa-Error diagram. The Kappa-Error diagram is a scatterplot where each point corresponds to a pair of classifiers. The x coordinate of the pair corresponds to the κ value, while the y coordinate is the average of the error rates of the two classifiers.

We now analyze our example concerning classifiers h_a and h_b (see Figure 3.3) from the perspective of κ statistic. Table 3.5 presents the contingency table for h_a and h_b, where the different predictions from h_a and h_b are clearly separated. Classifiers h_a and h_b scores $\kappa = -0.2$, such that the expected agreement by chance is $\Theta_2 = 0.\overline{4}$, while h_a

[3]Cohen's Kappa statistic (COHEN, 1960) measures inter-rater agreement for categorical variables and it was first used in (MARGINEANTU; DIETTERICH, 1997) as a pairwise diversity measure for ensemble learners.

Table 3.5: Contingency table for N^{ab} for h_a and h_b according to Figure 3.3

	$h_a(x) = 0$	$h_a(x) = 1$	$h_a(x) = 2$
$h_b(x) = 0$	6	0	6
$h_b(x) = 1$	6	0	0
$h_b(x) = 2$	0	0	0

and h_b effective agreement is only $\Theta_1 = 0.\overline{3}$. For this toy problem κ is able to represent the differences between h_a and h_b more precisely than Q, still for some problems it may be the case that κ and Q yield very similar results, perhaps because classifiers tend to commit prediction errors consistently.

The semantics behind κ and Q raise at the least two questions: (1) does it matter to measure error divergences as in κ? (2) Should κ be the preferred diversity metric in the analysis of every ensemble learner? Reinforcing the statement at the beginning of this section there is not a diversity metric that can be considered the 'best', it depends on the situation, thus when faced with a multiclass problem we may choose κ as it can differentiate between classifiers' errors. However, we may not be concerned whether classifiers commit prediction errors differently as long as we can identify how often they correctly predict the same instances, then it is reasonable to use Q or a similar metric.

To measure diversity for the whole ensemble one can either combine average pairwise measurements or use a non-pairwise measurement. Given symmetric diversity metrics, as is the case for κ and Q, we can calculate their average by summing all pairwise measures and dividing it by all possible pairs $2/\left(L\left(L-1\right)\right)$. The main problem of 'aggregating' statistics is that potentially interesting information is blurred in the midst of all possible combinations, thus it might be used and interpreted with caution. Other diversity measures have been studied for ensemble classifiers, including non-pairwise measures, for example: double-fault, entropy, Kohavi-Wolpert, difficulty θ, and others. We refer readers to Chapter 8 of Kuncheva's book (KUNCHEVA, 2004) and other works (KUNCHEVA; WHITAKER, 2003; BANFIELD et al., 2005; ZHOU, 2012) for an extensive discussion of diversity measures for ensemble learners.

Diversity is often identified as one of the building blocks of any ensemble-based classifier (ROKACH, 2010; ZHOU, 2012). While developing ensemble learners for data streams it is still considered a very important step to not only induce diversity into the ensemble, but also understand its implications in the overall ensemble performance (MINKU; WHITE; YAO, 2010).

Many techniques are used to induce diversity in a streaming environment, one of the simplest being the online bagging algorithm (OZA, 2005). Online bagging was used

in (MINKU; WHITE; YAO, 2010) to conduct diversity related experiments in evolving data streams for mainly 2 reasons: (1) "it does not present any specific behaviour to handle concept drift" (MINKU; WHITE; YAO, 2010); and (2) diversity in online bagging is 'controlled' through a single parameter λ[4].

In (MINKU; WHITE; YAO, 2010) experiments with online bagging (OZA, 2005) were specifically designed to analyze diversity before, during, and after concept drifts. Based on these experiments, authors found out that before a drift occurs, ensembles with less diversity obtain higher accuracy, but shortly after the drift occurs, highly diverse ensembles yield better accuracy despite the type of drift. Also, when there are no drifts, high diversity becomes less important.

3.3 Base Learner

Most ensemble classifiers for data stream learning were designed to work with any base learner (BIFET; HOLMES; PFAHRINGER, 2010; OZA, 2005; BRZEZINSKI; STEFANOWSKI, 2014; GOMES; ENEMBRECK, 2014) with open-ended constraints (e.g. any incremental base learner), still selecting an appropriate base learner according to the classification problem is an important step for obtaining an accurate ensemble. For example, very often classifiers can naturally deal with only one type of feature domain without resorting to input pre-processing. Thus, one can either select a base learner according to the input features domain, assuming all features have the same domain; use a base learner that deals with both discrete and continuous features, e.g., Hoeffding Tree (DOMINGOS; HULTEN, 2000); or use an ensemble method that combines heterogeneous base learners coherently with the feature domain, e.g., HSMiner (PARKER; MUSTAFA; KHAN, 2012). This last approach is more flexible as the ensemble can address problems where new features with different domains appear/disappear over time. It is still possible to achieve a diverse set of classifiers by using stable learners as long as the diversity induction strategy allows it. For example, one could use different subsets of features (see 'vertical partitioning' in Section 3.2.1) for training each classifier (BREIMAN, 2001; ABDULSALAM; SKILLICORN; MARTIN, 2007; NGUYEN et al., 2012).

The base learner must match the desired diversity induction strategy. If it is planned to obtain a diverse set of classifiers by training them on different instances, like

[4]Authors in (OZA, 2005) observe that the probability of each instance to be selected for a given subset is approximated by a Poisson distribution with $\lambda = 1$, thus it is feasible to "simulate" bagging in an online fashion by training each classifier k times on each instance, such that $k = poisson(\lambda = 1)$. In (BIFET; HOLMES; PFAHRINGER, 2010) authors presents the Leveraging bagging algorithm, which 'enhances' online bagging by using $\lambda = 6$, thus increasing the amount of instances presented to each classifier.

in Bagging (BREIMAN, 1996), then unstable learners (e.g. decision trees) should be preferred instead of stable learners (e.g. Naive Bayes). Stable learners must be trained on a large set of different instances for their models to differ from one another, while unstable learners tend to yield significantly different models even when trained on subsets of instances that overlap a lot (ZHOU, 2012).

Decision trees are the most common base learner for ensemble learning in a streaming setting. Specifically Hoeffding Trees[5] (DOMINGOS; HULTEN, 2000) or some of its variations that explicitly deals with concept drift as Adaptive Hoeffding Trees (BIFET; GAVALDÀ, 2009) and Concept-Adaptive Very Fast Decision Trees (CVFDT) (HULTEN; SPENCER; DOMINGOS, 2001), or that replaces majority class predictions by Naive Bayes models at the leaves of the tree (HOLMES; KIRKBY; PFAHRINGER, 2005). Other base learners often used in ensemble stream learning include naive bayes (KOLTER; MALOOF, 2007), perceptrons (CHEN; LIN; LU, 2012; PARKER; MUSTAFA; KHAN, 2012; PARKER; KHAN, 2013) and multilayer perceptrons (POLIKAR et al., 2001). Hoeffding Tree's preference over other base learners is attributable to its characteristic of being not only unstable, but also an incremental learner (DOMINGOS; HULTEN, 2000). We discuss incremental and batch (window) based learning in Section 3.4.2, however while explaining base learners it is important to emphasize that by using non-incremental learners, such as C4.5 (QUINLAN, 1993), the ensemble must incorporate a parameter to control the window (batch) size used for training its members. Ensemble methods that use incremental base learners may also include a widow size parameter, but then it is used to define when the ensemble is updated (e.g. adding new learners or recalculating statistics). This latter approach allows the development of algorithms in which the top level method (the ensemble) learns at a different rate than its members (GOMES; ENEMBRECK, 2013; GOMES; ENEMBRECK, 2014; BRZEZINSKI; STEFANOWSKI, 2014). Table 4.1 in Section 4.1 can be used for a quick overview of ensemble methods that use incremental or batch base learners.

3.3.1 Dependency

The training of one ensemble member may depend upon the output of other members. A canonical example of this approach is AdaBoost (FREUND; SCHAPIRE et al., 1996). Conversely, classifiers may be trained completely independently of one another, that is the case for Bagging (BREIMAN, 1996) and its variants. Intuitively, training one

[5]In (DOMINGOS; HULTEN, 2000) authors refer to their general method of inducing decision trees for data streams as Very Fast Decision Trees (VFDT) and to their implementation as Hoeffding Trees.

classifier while considering its peers' mistakes seems reasonable as it uses more information to guide the training process, for example, which instances to emphasize or which are already correctly mapped by the group. The drawback of this approach is that it may lead to overfitting, and on a stream environment it is not straightforward to train classifiers in this 'sequential' way. There are several proposals for adapting boosting for online classification (OZA, 2005; PELOSSOF et al., 2009; CHU; ZANIOLO, 2004; SCHOLZ; KLINKENBERG, 2007; BEYGELZIMER; KALE; LUO, 2015). Training classifiers independently is often preferred as it is easier, yields good results (BIFET et al., 2009), and allows training classifiers in parallel. The ability to train classifiers independently of one another is one characteristic that enables ensemble-based methods to cope with big data streams (MORALES; BIFET, 2015). An example of an algorithm that was developed to allow parallel training is HSMiner (PARKER; MUSTAFA; KHAN, 2012) and its subsequent enhancements presented in (HAQUE et al., 2014) that runs on top of Hadoop (WHITE, 2012).

3.4 Update Dynamics

There are specific characteristics of ensemble methods that are not directly related to diversity or combination methods, e.g., the ensemble cardinality. Learning from data streams requires algorithms that are not only accurate, but also efficient and able to adapt to changes in data. In this section we focus on the update dynamics of ensemble classifiers for data streams, i.e. how learning takes place in the ensemble.

3.4.1 Cardinality

Intuitively, it seems that by adding more classifiers it will cause the ensemble to achieve higher accuracy. Although it is not straightforward to exploit this in practice, since as the number of classifiers increases it becomes difficulty to maintain all classifiers minimally different from each other, i.e., a diverse set. Generating a great quantity of redundant classifiers cannot do any good to the overall decision quality, but will surely negatively impact the ensemble memory and processing consumption. In a data stream context, the cardinality of the ensemble can be either defined prior to the execution or vary during execution. There are good reasons to support both approaches, for example, the resources needed for a fixed set of classifiers are easier to estimate and control, while an ensemble that can selectively add or remove classifiers has more flexibility, e.g., remove redundant classifiers and save resources or add more classifiers to cover different parts of

the classification space.

Ensemble methods that vary its size dynamically, like SAE (GOMES; ENEMBRECK, 2013) and DWM (KOLTER; MALOOF, 2007), are intuitively better suited for a highly dynamic task, such as data stream classification. Although in practice these ensembles can yield too little or too many classifiers, because their heuristic method that dictates when to add or remove classifiers is not suitable for the given data stream. To avoid too many classifiers, some methods (KOLTER; MALOOF, 2005a; BARDDAL; GOMES; ENEMBRECK, 2014; GOMES; ENEMBRECK, 2014) include a parameter to define the 'maximum' number of classifiers of an ensemble. The complexity behind the definition of a heuristic method for adding and removing classifiers, and the success of existing stream ensembles (OZA, 2005; BIFET et al., 2009; BIFET; HOLMES; PFAHRINGER, 2010; BRZEZINSKI; STEFANOWSKI, 2014) that use a predefined number of classifiers explains the lack of interest in the development of strategies for dynamically sizing the ensemble (this is observable in Table 4.1 where algorithms' cardinality is often 'fixed').

3.4.2 Learning mode

The ability to learn new concepts (*plasticity*) while retaining previously learned knowledge (*stability*) is referred as the stability-plasticity dilemma (LIM; HARRISON, 2003). This dilemma is pervasive in the evolving data stream setting (GAMA et al., 2014) as it is expected that evolving data streams alternate between drifting and stable periods. Therefore any data stream classifier, including ensembles, must incorporate mechanisms to adapt its model to concept drifts, while accounting for periods of concept stability.

There are different types of concept drifts that may occur as well as several approaches for coping with them. For example, a classifier may periodically forget its model and learn a new model on the most recent n instances or reset the current model if a change has been triggered by a drift detector algorithm. These methods are different approaches for dealing with concept drift that can be adapted to either ensembles or single classifiers. Although ensemble-based classifiers have the advantageous characteristic of being flexible as its members can learn at different rates (see Section 3.3); new classifiers can be added and existing classifiers may be updated, replaced, reset or even removed selectively.

Most stream classifiers assumes that *recency* is analogous to *relevance* when it comes to weighting instances for training, and ensembles are no exception to that. The reasoning behind this assumption is simple: old instances are associated with previously outdated concepts, while new instances are committed to the most current concept. In

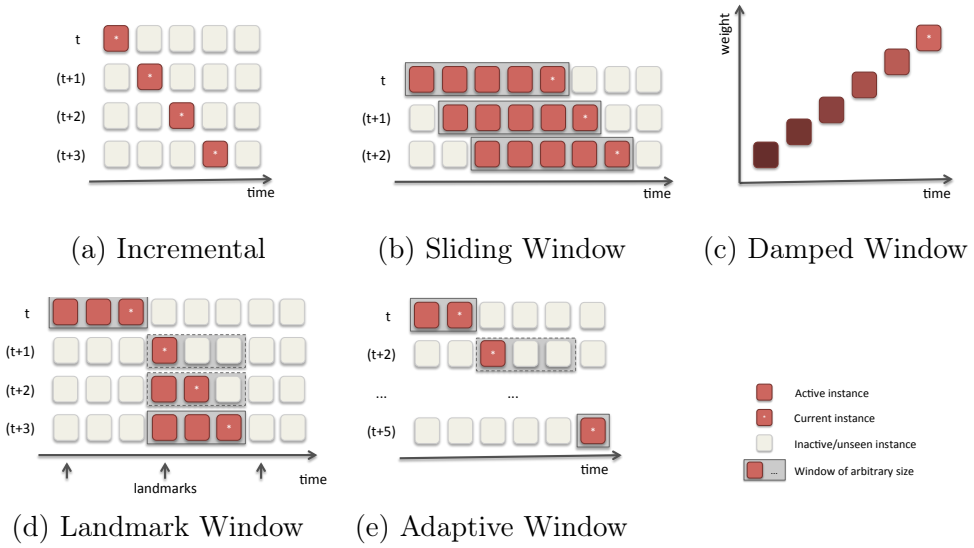
practice, real world problems may not adhere to this assumption as concepts can reoccur either periodically (e.g. seasons of the year) or erratically. If concepts reappear it is a waste of resources to re-learn old concepts over and over again. Therefore, tracking and dealing with recurring concept drifts is a difficult task in which the classifier must provide efficient answers to the following questions:

- When to store a previously learned model?
- When should a model be removed/updated?
- When (and how) to evaluate old models for 'activating' them?

One can only justify using a recurring concept drift strategy if storing and updating previous models is more efficient, with respect to resources and accuracy, than re-learning the model. FLORA3 (WIDMER; KUBAT, 1996) was one of the first algorithms that dealt with recurring concept drifts. More recently, several ensemble classifiers (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010; JABER; CORNUÉJOLS; TARROUX, 2013a; DÍAZ et al., 2015a) were designed for dealing with recurring concept drifts.

Ensemble-based algorithms can be more flexible with respect to concept drift adaptation. For example, several algorithms (STREET; KIM, 2001; GOMES; ENEMBRECK, 2013; BRZEZINSKI; STEFANOWSKI, 2014; GOMES; ENEMBRECK, 2014; GOMES; BARDDAL; ENEMBRECK, 2015) use a background learner, i.e., an auxiliary single learner trained only on the most recent instances alongside the other ensemble members, but that does not influence overall decisions. Whenever it is necessary to reset an ensemble member the background learner replaces it. There are at least two advantages that supports this approach. First, the background learner already has a trained model, thus it will not take too long until it starts positively impacting the overall ensemble decision. Second, assuming that recency implies relevance, the background learner model might overcome existing models as it has been trained only on the latest instances.

Previous works organize data stream learning in different categorizations according to which specific learning problem the work discusses (GAMA et al., 2014; SILVA et al., 2013; GAMA, 2010; READ et al., 2012). Authors in (GAMA et al., 2014) organize learning into three categories: **learning mode**: whether the algorithms retrain models or incrementally update them; **adaption methods**: concerns how adaptation to drifts happens, either pro-actively (blind strategy) or reactively (informed); and **model management**: divided into three aspects of ensemble learning (dynamic combination, continuous updates of learners, and structural updates). In (READ et al., 2012) the authors provide an extensive discussion of the advantages and disadvantages of incremental



(a) Incremental (b) Sliding Window (c) Damped Window

(d) Landmark Window (e) Adaptive Window

Figure 3.4: Learning modes for data streams

and batch learners, while in (SILVA et al., 2013) authors focus on learning modes for clustering algorithms, thus they discuss learning according to window based models, i.e., landmark window, sliding window and damped window.

Ensemble classifiers designed to deal with evolving data streams may combine more than one learning strategy, as explained in Section 3.3, through, for example, the combination of incremental learners and a window based approach. In the remainder of this section we present our attempt to classify how learning happens on ensemble classifiers. We subdivide learning into two general classes: incremental and window based. Many of the existing stream ensemble learners fall into the latter category, which is further divided into: sliding windows, damped windows, landmark windows, and adaptive windows. Figure 3.4 illustrates learning mode according to our categorization.

In Section 3.3 we have briefly discussed how the base learner and the ensemble may operate at different learning rates. Throughout the rest of this Section we discuss each learning mode individually and present, whenever possible, examples of ensemble classifiers that instantiate the corresponding learning mode.

Incremental. A batch learner must store a batch of instances before using them for training; conversely, an incremental learner is trained on instances as they arrive. As a consequence, incremental learners better adhere to the four constraints suggested in (BIFET et al., 2010) (see Section 2.1) and are often preferred on a data stream setting. As discussed in (READ et al., 2012), both approaches have advantages and disadvantages. There is a large amount of algorithms to choose from when using batch learners, while, comparatively, the amount of incremental learners available is small (READ et al., 2012). On the other hand, batch learners require parameterizing the amount of in-

stances to be used for training, i.e. the batch size, which is critical for obtaining accurate models, yet difficult to define for evolving data streams. Generally, incremental learners are more effective when applied to streams that exhibit gradual or incremental drifts or when combined with drift detectors. In the occasion of an abrupt drift, an incremental learner (without the aid of a drift detector) may take longer to recover as its model is influenced by the concepts it has previously been presented to, while a batch learner completely discards its previous models periodically. Examples of incremental learners include Bayesian classifiers (JOHN; LANGLEY, 1995), like Naive Bayes; decision trees, like Hoeffding Trees (DOMINGOS; HULTEN, 2000); Stochastic Gradient Descent variations (WANG; CRAMMER; VUCETIC, 2012); Instance-based (Lazy) methods (BERINGER; HÜLLERMEIER, 2007; ZHANG et al., 2011a); and ensemble classifiers (OZA, 2005; BIFET; HOLMES; PFAHRINGER, 2010) as long as its base learners are also incremental learners.

Landmark windows. Landmarks can be used to separate the stream into disjoint chunks of data. A landmark can be defined using the number of instances seen since the previous landmark or according to a specific time frequency. Whenever a new landmark is reached, all instances in the previous chunk are discarded. Some ensemble classifiers use landmark windows (batches) of a fixed size n to control the periodicity of the ensemble updates, such as classifiers' removals, resets, additions, or statistics reset. This approach was first introduced in the Dynamic Weighted Majority (DWM) algorithm (KOLTER; MALOOF, 2007), and later used in other algorithms, such as AddExpert (KOLTER; MALOOF, 2005a), AUE (BRZEZIŃSKI; STEFANOWSKI, 2011), SAE2 (GOMES; ENEMBRECK, 2014), OAUE (BRZEZINSKI; STEFANOWSKI, 2014), and others. It seems reasonable that if incremental learners are used, then using a predefined fixed landmark window is unnecessary. However, many ensemble classifiers for data streams combine landmark windows and incremental base learners. This design choice may allow reasonably fast adaptation to abrupt drifts (given small values of n), while it allows incremental updates of ensemble members, which help addressing gradual and incremental drifts. The fixed landmark window approach permits the use of traditional batch learning algorithms for stream learning. In this case, a batch learner is trained on instances from window w and its model is used to classify instances from the next window $w+1$. After window $w+1$ is over, the model learned on w is replaced by a model trained on $w+1$. If this approach is used for adapting a batch learner for stream learning, then some problems may arise, most notably: training is concentrated during the transitions between windows, therefore if new instances arrive in a fast pace, then it is necessary to account for prediction delays while a new model is being trained; very often batch learners needs to be trained on large

amounts of data in order to yield accurate models, thus the window must be very large or the learned model will be weak. Finally, if a concept drift happens, it will not be taken into account until the window ends and the new model is generated, thus adaptation to abrupt drifts will be slow. Despite the simplicity of using fixed landmark windows, it is difficult to define the landmark size parameter n. On one hand, if the stream has abrupt drifts, smaller values of n are better, since ensemble updates will happen more often. On the other hand, if the stream is stationary (no drifts) or if drifts are gradual, then larger window sizes are advised, since the ensemble members will be capable of training on a larger number of instances before an ensemble update takes place. Although if the stream exhibits different types of drifts interlaced with periods of stability, then any predefined window size will most likely be unsatisfactory. Finally, by using incremental base learners instead of batch base learners, the ensemble might be capable of adapting to gradual or incremental drifts simply because its members' learned models must not necessarily be discarded after every window.

Sliding windows. Sliding windows are similar to landmark windows in the sense that both define a window size n, although sliding windows discard only one instance at a time. Instance-based classifiers (KHAN; DING; PERRIZO, 2002; LAW; ZANIOLO, 2005; GABER; KRISHNASWAMY; ZASLAVSKY, 2005; BERINGER; HÜLLERMEIER, 2007; ZHANG et al., 2011a) can be viewed as incremental learners and as sliding window based methods. They are incremental learners since their 'models' are updated after every new instance (READ et al., 2012), but as memory is limited it is necessary to forget one instance to make room for another. Discarding the influence of a single instance from the model is easy for instance-based methods, viable for Bayesian classifiers and very difficult for decision trees.

Damped windows. The damped window, or time-fading model, associate weights to instances based on their age. Thus, instances that have been recently presented to the learner will have a higher weight than those that were presented a long time ago. It is possible to use a linear or exponential decay function to assign weights. The base learner must differentiate between instances weights, such that these must influence its learning, otherwise the damped window model degenerates into a simple sliding window.

Adaptive windows. The adaptive window model can be viewed as a landmark window with varying values of n. Assuming that the stream contains drifts with varying extents and rates, using windows of different sizes is a suitable strategy. The problem is how to dynamically adjust n according to observations of the stream. The FLORA2 algorithm (WIDMER; KUBAT, 1996) uses a heuristic (Window Adjustment Algorithm) to augment or shrink the window size based on yet another heuristic that guesses whether

a drift has occurred or not. This approach for adjusting the window size may be useful in practice, however it depends on fixed thresholds to define by 'how much' should the size be decreased or increased. On top of that, it depends on a heuristic to determine whether the current concept is stable or a drift is happening. A different approach is used by ADWIN Bagging (BIFET et al., 2009) and Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010), where both algorithms use the ADWIN drift detector to selectively reset classifiers. Concretely, in these algorithms a classifier is reset whenever its associate ADWIN detector signals that a drift has occurred. Thus, the ensemble may end up with classifiers with varying levels of commitment to the current concept.

3.5 Data Stream Ensemble Classifiers

This section presents some of the previous works for supervised learning in the context of data stream mining. The focus is on incremental methods, especially those based on ensemble of classifiers. Even though some of the presented methods have roots in traditional machine learning, i.e., Bagging and Adaboost, the algorithms presented here are the versions adapted for online learning.

3.5.1 Hoeffding Tree

Even though the Hoeffding Tree (DOMINGOS; HULTEN, 2000), also known as Very Fast Decision Trees (VFDT) [6], is not an ensemble, we include it in our review since Hoeffding Trees are not only a state-of-the-art single classifier, but also widely used as a base learner for ensembles.

In every decision tree, any given node will contain a test, based on the values of its attributes, to decide to which path an instance should be sent down the tree. This process is repeated until a leaf node is reached, where the predict label is decided. Therefore, there are at least two important decisions while building a decision tree: how nodes are split and how the predicted label is decided. Decision trees usually split nodes based on a criterion that selects the attribute that best divides the data according to the class labels. The most popular method for this endeavor is the information gain used in the C4.5 algorithm (QUINLAN, 1993). Information gain uses entropy to determine which attribute can better be used to predict the class label. The critical part for data

[6] Originally in (DOMINGOS; HULTEN, 2000) authors describe the Hoeffding Tree as the basic theoretical algorithm, while the VFDT is the actual implementation. Currently, authors usually refer to the implementation and slight variations of the original algorithm simple as Hoeffding Tree (BIFET et al., 2011).

streams is that entropy is calculated based on statistics obtained from data. Since it is not possible to observe all instances before generating the model, the traditional way to build decision trees is not applicable in a data stream environment [7]. In this matter is where Hoeffding Trees provides its most distinctive innovation, by using the Hoeffding Bound (HOEFFDING, 1963) to estimate the mean of a random variable with only a minimal amount of observations, which allows to build trees incrementally. In (DOMINGOS; HULTEN, 2000), authors demonstrate that a Hoeffding Tree performs very close to batch learned decision trees.

One very popular variation of the Hoeffding Tree is the Adaptive Naïve Bayes Hoeffding Tree (HNBT) (HOLMES; KIRKBY; PFAHRINGER, 2005). This variation works by adding a naïve Bayes predictor at each leaf, but it only uses it for predictions if it was more accurate during training than majority class.

There are more to Hoeffding Trees than what has been briefly discussed here. For information about memory management, computational complexity, and specifics on actual implementations the reader is directed to (BIFET et al., 2011). For theoretical guarantees and further discussion, the reader must investigate the original publication in (DOMINGOS; HULTEN, 2000).

3.5.2 FLORA

In (WIDMER; KUBAT, 1996) authors presents four versions of the FLORA basic approach. Each version is build on top of its predecessor and is meant to solve its previous drawbacks. Authors in (WIDMER; KUBAT, 1996) emphasize that an effective learner must be able to detect drifts without being explicitly informed about them; promptly recover from them by adjusting its hypothesis; re-use previous hypothesis if a concept reoccur. The general FLORA approach aimed at providing reasonable implementation to these characteristics. Thus, the general FLORA approach was defined according to the following properties:

- Maintains only a window of currently trusted examples and hypotheses;
- Store concept descriptions and re-use them if a previously seen concept reappears; and
- Control both functions using a heuristic that constantly monitors the algorithm performance.

[7] It is always possible to use batch learners by using training windows, but here we are referring to classifiers that have been developed specifically for data streams.

To satisfy these properties, an algorithm would need to implement the following functions:

1. Operators for the modification of the concept description in reaction to changes in the contents of the window

2. The ability to decide when and how many old examples should be deleted from the window (forgotten)

3. Store current and old (hypotheses) and ability to assess the relative merits of concept hypotheses for the current context.

The base learner in FLORA is restricted to a representation language based on attribute-value logic that does not include negation. In this setting, a hypothesis is analogous to a concept description. A concept description is composed of three description sets:

1. **ADES.** description of items that matches positive examples only;

2. **NDES.** summarizes all negative examples; and

3. **PDES.** all of its description items are too general, i.e. they match positive and negative examples concomitantly.

Description sets are conjunctions of attribute-value pairs. A description item is said to match an example if all of its literals occur in the description. For instance, $(color = white)^{(}temperature = low)$ *matches* 'snow', and (shape = cube) does not *match* 'Globe'.

When instances are added or removed from a window, the current description concepts are updated to reflect these changes.

The FLORA2 algorithm uses a heuristic to dynamically adjust the window size based on a strategy that actively monitors concepts drifts. This strategy depends on the extent of the drift and on the momentary state of the learner, which can only be determined during learning. FLORA2 detects drifts by (1) monitoring accuracy over a fixed number of past classification attempts, and by (2) interpreting syntactic properties of the evolving concept descriptions, which are identified by sudden increases of the number of description items in ADES. Fixed thresholds are used to control this drift detection, e.g., (1) is controlled by an acceptable predictive accuracy threshold p. FLORA2 uses a heuristic to dynamically shrink if a drift has been detected, keep the window size when a concept seems stable, or enlarge the window until a stable concept description can

be formed. The specific algorithm implemented in FLORA2 is the Window Adjustment Heuristic (WAH). WAH algorithm decrease the window size by 20% if a concept drift is suspected to have occurred. To avoid too many instances in memory, the window size is decreased by 1, i.e. add 1 new example and delete the 2 oldest examples, if the concept description seems extremely stable.

If neither of the previous conditions is met, then the current concept description seems stable enough and the window size is left unchanged. The authors note that it is "hopeless to make the heuristic general and free of parameters" (WIDMER; KUBAT, 1996), thus its parameters might vary for different datasets. The FLORA3 algorithm adds the capability of storing and reusing previously learned models, while maintaining the adaptation strategy of FLORA2. The authors claim that in many real world problems the hidden contexts that vary over time causing concept drifts are finite, and reappear either cyclically or in an unordered fashion. As a consequence, it is a waste to re-learn a concept description from scratch. One possible way that a learning algorithm could avoid this wasteful situation is by storing and reinstating concept descriptions when concept drifts are suspected to have occurred. Put in simple terms, the algorithm, in FLORA3, evaluate the current state of learning after every cycle and decide if some old concept description must be reconsidered. When a concept drift seems to occur, the storage of old concept descriptions is consulted. If a stable period is reached, the concept descriptions are saved to the storage, if they are not already represented there.

To decide which old concept description must be selected, FLORA3 uses a three step procedure, described below:

1. **Find best candidate.** A concept description becomes a candidate if it is consistent with the current example. All candidates are evaluated w.r.t the ratio of the number of positive and negative instances that they match in current window;

2. **Update the best candidate.** The best candidate is updated to the current data by setting all counters in the description sets to 0 and then re-processing all the examples in the window (re-run FLORA for each instance); and

3. **Comparison between best candidate and current.** The (updated) best candidate C_b is compared to the current concept description C by using some 'measure of fit' to decide whether C_b is better than C. If C_b is indeed better than C then replace C with C_b.

In FLORA3, the measure of fit is the relative complexity of the description, i.e. a concept description C_x is said to be 'better' than C_y if C_x's ADES set is more concise.

During experiments, authors noticed that datasets with gradual drifts, especially those with noticeable noise, destabilized FLORA3's performance more than expected. This observations lead to the development of FLORA4. FLORA4 avoid shortcomings due to decisions based on absolute values by basing decisions on confidence intervals. During experiments, FLORA4 performed better on noisy data than its predecessors, such behaviour was attributed to its use of statistical comparisons.

3.5.3 Streaming Ensemble Algorithm

In (STREET; KIM, 2001) authors presents one of the first ensemble-based approaches for evolving data streams, namely the Streaming Ensemble Algorithm (SEA). It discusses important concepts, like "any-time learning", limited memory, "single-pass" algorithms, and block processing classifiers for large datasets or streaming data. The proposed ensemble is not limited to a specific base learner. However authors only present results using the C4.5 decision trees. One very interesting trait of SEA is that it was one of the first ensemble-based methods for data stream processing to emphasize update dynamics, such as adding/removing and replacing classifiers during execution. On top of that, classifier training was aimed at inducing diversity into the ensemble by using different blocks of instances to train each classifier. SEA employed the terminology of a "candidate" classifier, as recent methods do, which represented a classifier that is not yet part of the ensemble and may be added to it. When the ensemble has reached its maximum size, the candidate would need to replace a member of the ensemble. Choosing between which classifiers must be kept and which must be removed depends on a score function. The classifier scores were calculated to represent how much the classifier contributes to the ensemble, not just how well it performed individually (e.g. individual accuracy). Specifically, there are four variables involved during classifier scoring:

- P_1 : percentage of votes received by the class with most votes.
- P_2 : similar to P_1, but for the class with the second highest number of votes.
- P_C : percentage of votes that the correct class received.
- P_T : percentage of votes that the class which classifier T choose received.

The scoring formulation can be applied to multiclass problems, although authors only report experiments with binary problems.

The reported experiments are limited to comparisons between SEA and single classifiers on binary classification problems. To simulate concept drifts, authors presents a synthetic data generator, which is described in details on Section 2.4.4.

3.5.4 Online Bagging

The Online Bagging algorithm was introduced in (OZA, 2005) as an adaptation of the batch ensemble classifier Bagging (BREIMAN, 1996). Originally, a bagging ensemble is composed of k classifiers, which are trained with subsets (bootstraps) N_j of the whole training set N. However, sampling usually is not feasible in a data stream configuration, since it requires storing all instances before creating subsets. Authors in (OZA, 2005) observe that the probability of each instance to be selected for a given subset is approximated by a Poisson distribution with $\lambda = 1$, thus it is feasible to "simulate" bagging in an online fashion by training each classifier k times on each instance, such that $k = poisson(1)$.

3.5.5 Online Boosting

In this section, we briefly discuss the adaptations by Oza and Russel (OZA, 2005) for online learning of the original AdaBoost (FREUND; SCHAPIRE et al., 1996), precisely the AdaBoost.M1 algorithm. The term 'boosting' is sometimes used to refer to the actual AdaBoost ensemble algorithm. For an extensive review of the 'boosting' meta-algorithm, and theoretically sound explanations, the reader is directed to (SCHAPIRE; FREUND, 2012).

AdaBoost generates base models sequentially, i.e., $h_1, h_2, h_3, \ldots, h_M$, by training them one after the other on weighted instances drawn from sets $D_1, D_2, D_3, \ldots, D_M$. The weights of instances in training set D_m depends upon the misclassified instances by h_{m-1}, such that the misclassified instances are given half the total weight, while the remaining instances (correctly classified) are given the other half. Intuitively, models h_m and h_{m-1} will differ significantly because h_{m-1} will emphasize its training on instances that were previously misclassified.

In Online Boosting, the Poisson distribution is used for deciding the random probability that an example is used for training, similarly to Online Bagging (OZA, 2005) (see Section 3.5.4). The difference to Online Bagging is that when model h_{m-1} misclassifies a training instance, the Poisson distribution parameter λ associated with that instance is increased when presented to h_m, otherwise it is decreased. Just as in AdaBoost, Online Boosting assigns the misclassified instances by h_{m-1} half the total weight for h_m. Conversely, the correctly classified instances receive the remaining half of the total weight.

Authors prove in (OZA; RUSSELL, 2001) that this procedure converge to the original algorithm when the number of models and training instances tend to infinity.

3.5.6 ADWIN Bagging

In (BIFET et al., 2009) authors present two new ensemble methods, Adaptive-Size Hoeffding Trees (ASHT) Bagging and ADWIN Bagging. Both methods combine Online Bagging with a strategy to cope with concept drift. The idea behind ASHT Bagging is that smaller decision trees (less levels) can adapt quickly to concept drifts, while bigger decision trees (more levels) perform better during periods with little to no changes in the underlying concept. ADWIN Bagging combines Online Bagging with the Adaptive Window (ADWIN) (BIFET; GAVALDÀ, 2007) algorithm for concept drift detection. ADWIN maintains a window of variable size with data in a histogram to reduce memory usage. This window has the maximum size statistically consistent with the hypothesis: "there were no changes in the average value within this window" (BIFET et al., 2011). A fragment of the window is removed if there is no evidence that its average value is different from the average value of the remainder of the window. Concept drifts are observed through analysis of changes in the window size.

3.5.7 ASHT Bagging

The idea behind the Adaptive-Size Hoeffding Tree (ASHT) Bagging algorithm (BIFET et al., 2009) is that smaller decision trees (less levels) can adapt faster to changes, while, larger trees (many levels) obtain better performance during periods with little or no changes. In ASHT Bagging, the base learner used is the Adaptive-Size Hoeffding Tree (ASHT), which is a derivation of the original Hoeffding Tree (DOMINGOS; HULTEN, 2000) that allows the user to specify the maximum three height through a parameter. In ASHT, after a node is split, if the maximum height has been reached, some nodes must be removed (operation known as the 'reset' of the tree). There are two approaches for the tree reset. The first option is to remove the root and all nodes, except for the node that started the split, which then becomes the new root. The second option is to remove all nodes, i.e., completely reset the tree.

In ASHT Bagging, the maximum size of the tree at position n is the double of the tree at position $n-1$. For example, in an ensemble with 5 trees, such that the first one has maximum height equals 2, then the maximum height of the other trees will be 4, 8, 16 and 32, respectively. Each tree has an assigned weight proportional to the reverse of its squared error, and its error is monitored through the exponential weighted moving

average (EWMA) with $\alpha = 0.01$.

It is important to notice that in ASHT Bagging, trees are always reset, independently if a drift has occurred or not. However, it is expected that this behavior does not negatively impact the overall ensemble prediction accuracy as threes are weighted according to their estimated accuracy. The smaller trees are reset more often and consequently can adapt faster if a drift occurs, while the larger trees takes longer to reset and are able to form more complex models.

3.5.8 Leveraging Bagging

Aiming at enhancing Online Bagging accuracy, authors in (BIFET; HOLMES; PFAHRINGER, 2010) introduce the Leveraging Bagging algorithm, which adds more randomization to the input of the ensemble by using a larger λ value. The standard Online Bagging algorithm uses $\lambda = 1$, which means that around 37% of the values output by the Poisson distribution are 0, another 37% are 1, and 26% are greater than 1. This implies that by using Poison(1) 37% of the instances are not used for training (value 0), 37% are used once (value 1), and 26% are trained with repetition (values greater than 1). By using Poisson(6) we obtain 0.25% of values 0, 45% lower than 6, 16% of values 6, and 39% greater than 6. Thus, by using values of $\lambda > 1$, we are effectively using more instances for training and, as a consequence, the base classifiers are able to produce more refined hypothesis of the input. This may lead to overfitting, even in a data stream context, although this has not been investigated yet.

Besides randomizing the input, in (BIFET; HOLMES; PFAHRINGER, 2010) authors also propose manipulating the output of the ensemble through random error correcting codes. For each ensemble member the class label of a training instance is mapped to a binary classification problem. For example, assuming a multiclass problem with classes 0, 1, 2 and 3, a given classifier C could interpret instances assigned with class label 0 and 3 as class 0, and those assigned to class labels 1 and 2 as class 1. By doing that, ensemble members may form different hypothesis, thus minimizing correlation between them, and enhancing diversity. In order to account for concept drifts, Leveraging Bagging uses the ADWIN algorithm (see Section 3.5.6) to detect drifts and reset ensemble members. Some variants of the described leveraging algorithm were presented in the paper as well, which are briefly described bellow:

- **Levearging Bagging MC.** Does not use Random Output Codes.
- **Levearging Bagging ME.** Adds more weight to misclassified instances. If an

instance is misclassified it is accepted with a weight of one, otherwise it is accepted with a probability $e_T/((1 - e_T))$, where e_T is the error estimate computed as a smoothed version of the proportion of misclassified instances using the estimation of ADWIN that is monitoring the error.

- **Leveraging Subbaging.** Resampling without replacement. Authors observe that this method uses less memory and is faster, but has low accuracy when compared to others.

- **Leveraging Half Subbaging.** Resampling without replacement half of the instances. Authors observe that this method is even fast and requires less memory that Leveraging Subbaging, but it has even lower accuracy.

- **Leveraging Bagging WT.** Bagging without taking out any instance. This is implemented using $1 + Poisson(1)$ instead of Poisson. Authors observe that if all instances are used, accuracy is improved at the expense of memory and speed.

3.5.9 Heterogeneous Ensemble with Feature Drift for Data Streams

The work of Nguyen, Woon and Wan introduces the Heterogeneous Ensemble with Feature Drift for Data Streams classifier (HEFT-Stream) (NGUYEN et al., 2012). Important traits of this ensemble method include allowing different base learners to compose the ensemble, which can induce more diversity to the ensemble. Also, HEFT-Stream uses a feature selection technique called FCBF (YU; LIU, 2003) to periodically select only the most relevant features for training. Authors claim that HEFT-Stream is suitable to adapt to gradual and sudden drifts, such that adaptation to gradual drifts occurs naturally because they only employ incremental classifiers as base learners, i.e. they used Hoeffding Trees (DOMINGOS; HULTEN, 2000) and Online Naive Bayes on their experiments. To further enhance diversity, ensemble members are trained according to online bagging, as a consequence besides each classifier learning from a different projected set of attributes, they also observe different instances. The overall ensemble prediction is given by a weighting combination according to each ensemble member individual probability distribution vectors.

Whenever the selected feature subset changes from one chunk of instances to another, a feature drift is signalled to the ensemble. It is assumed that if a feature drift has occurred, then a sudden drift has happened as well. In (NGUYEN et al., 2012) authors reason about the relationships between concept drifts and feature drifts, they conclude that concept drifts may or may not cause feature drifts, but when they do the feature drift

occurs at a slower rate than the concept drift. To adapt to sudden drifts, a new classifier is added to the ensemble. Its base learner must be the same as the classifier with the highest weight. On top of that, if the ensemble has reached its maximum size, the classifier with the lowest weight is removed to make place for the new classifier. Authors argue that this update policy allow the ensemble to select the fittest base learner dynamically.

The experiments presented shows that HEFT-Stream perform well w.r.t accuracy and resources usage, especially on datasets with high dimensional data (hundreds or thousands of features). Processing time and memory used was low due to ensemble members learning from projected sets of features, such that this fraction was for some experiments lower than 1% of the total features.

3.5.10 Fast Adapting Ensemble

The Fast Adapting Ensemble (FAE) (DÍAZ et al., 2015b) is an ensemble method, designed with the goal of dealing with abrupt, gradual and recurrent drifts concomitantly. FAE maintains a limited set of classifiers with two possible statuses: active or inactive. Active classifiers decisions are taken into account on the overall ensemble prediction. It is assumed that active classifiers have a better model of the current concept, which is reflected by their high weight. Inactive classifiers are only kept in expectation that they might become useful in the future if the concept that they model reappears.

In FAE, learning is divided into chunks of equal size according to a user given threshold. Classifiers are weighted according to a formula similar to that of AWE (WANG et al., 2003), such that weighted is relative to the classifier estimated accuracy calculated only in the current chunk of instances. Similarly to SAE2 (GOMES; ENEMBRECK, 2014), and differently from AWE, weights are updated after every new instance is presented according to $w(t)_j := \beta_1 w(t-1)_j + \beta_2 acc_{b,t}$, such that acc_{b_t} represents the estimated accuracy on the batch of instances b up to instance t, $w(t)_j$ is the weight associate with classifier j at moment t, and parameters β_1 and β_2 are used to control the trade-off between noise and rapid adaptations[8], respectively. Active classifiers' weight can be either decreased or increased, while inactive classifiers' weights are only updated if the value has increased. Authors justify this decision based on the argument that inactive classifiers represent concepts that are not currently active, thus it is not reasonable to decrease their weights based on their accuracy on the current concept.

FAE uses operators to add and remove classifiers based on specific criteria. A new classifier is added only if a drift is detected. The drift detector used in (DÍAZ et al.,

[8]By definition $\beta_1 + beta_2 = 1$.

2015b) is the EDDM (BAENA-GARCÍA et al., 2006), although authors note that any detector capable of outputting 3 levels of drift detection are suitable, i.e., no drift, warning and change. Classifiers are deleted whenever the ensemble has reached its maximum size (user given parameter). In order to remove a classifier, four characteristics are taken into account: (1) Active status; (2) Age; (3) Weight; (4) Number of classifiers associated with the concept. Preference for removal is given to classifiers that are not currently active, but if all classifiers are active then the other 3 criteria are used.

Overall, FAE combines many heuristics to deal with different types of drifts, and often save resources, e.g., it does not periodically (and blindly) add new classifiers. The experiments presented in (DÍAZ et al., 2015b) are limited to one real dataset (Electricity, see Section 2.4.10) and variations of two synthetic data generators (SEA and LED, see Section 2.4.4), thus in order to draw more general conclusions about FAE capabilities it might be necessary to carefully examine it on more datasets.

3.5.11 Ensemble of Restricted Hoeffding Trees

The work in (BIFET et al., 2012) uses the observation that it is not necessary to model complex attribute interactions to obtain good classification accuracy on practical problems. Therefore, a learner may be capable of obtaining an average performance only by training on a subset of attributes. Authors exploit this by building an ensemble of Hoeffding Trees (DOMINGOS; HULTEN, 2000), which have their learning restricted to a subset of features. These trees are exhaustive, i.e. all possible attribute subsets of a given size have one tree associated with it, thus if there are m attributes and all subsets of size k are to be generate, there will be $\binom{m}{k}$ subsets. As a consequence, only moderate values of k, or values of k very close to m, are feasible for high-dimensional data.

The trees' predictions are combined using a stacking approach, where the meta level data is composed of the log-odds of the class probabilities estimates (trees outputs) instead of discrete classifications, since probabilities estimates provide more information for the combiner.

The meta level combiner is formed by perceptrons (one per class value) with sigmoid activation functions, trained using stochastic gradient descent to minimize the squared loss with respect to the actual observed class labels in the data stream.

The reason for using log-odds instead of raw probabilities estimates is because the application of the sigmoid function presupposes a linear relationship between $log(f(a_i)/(1-f(a_i)))$ and a_i, assuming that a_i is a vector of log-odds for class i, and that $f(a_i)$ is the output of the sigmoid perceptron for class i according to $f(a_i) = 1/(1+e^{-(w_i a_i + b_i)})$. To avoid

zero-frequency problems, a small constant ϵ is added to the trees probabilities estimates.

The learning rate α is decreased as the amount of training data increases. This is to avoid situations where a large value of α prevent convergence. Assuming m as the total number of attributes, and n the current number of instances seen, the learning rate α is updated according to Eq 3.7. Figure 3.5 depicts a schematic view of the ensemble structure.

$$\alpha = 2/(2 + m + n) \tag{3.7}$$

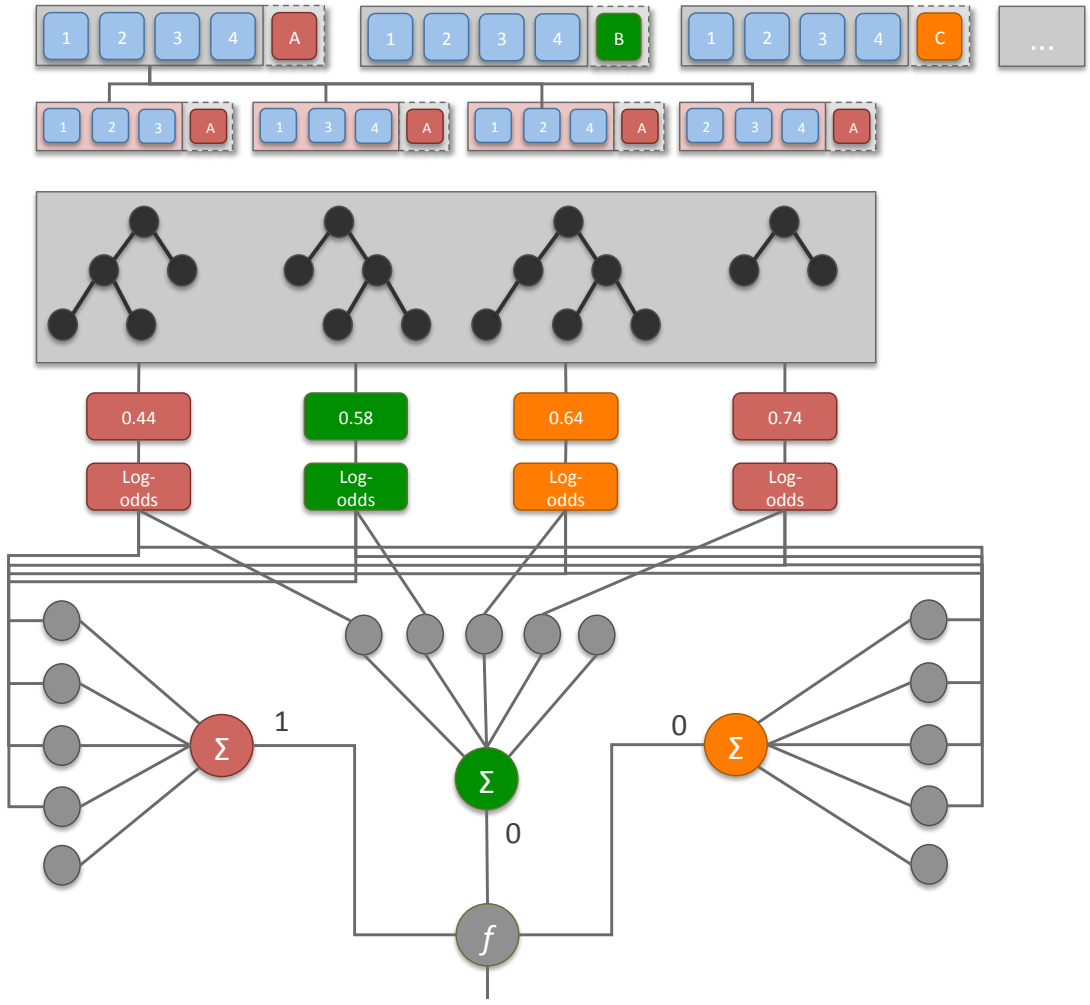


Figure 3.5: Ensemble of Restricted Hoeffding Trees for a problem with 3 classes (A,B and C), total features $m = 4$ and $K = 3$.

Although, this approach assumes that the training data is identically distributed and that is not true since the Hoeffding trees probability estimates change over time, usually becoming more and more accurate. Thus, setting the learning rate based solely on Eq. 3.7 means that the perceptrons will adapt too slowly after the initial data has

been processed (BIFET et al., 2012). To overcome this problem, authors interpret the meta level data as an evolving stream and use the ADWIN change detector (BIFET; GAVALDÀ, 2007) to determine when substantial changes have happened. Notice that not necessarily the input data has to be an evolving data stream for this approach to work. Once a drift is detected, the value of n is reset, and thus α value restart (see Eq 3.7). Finally, the effect of this approach is that the learning rate will be relatively large as long as the Hoeffding trees models keep evolving. When there are no drifts, n is incremented and as a consequence the value of α decreases.

The ADWIN detector is used to selectively reset trees whenever concept drifts that may happen in the input data. This approach is similar to the one presented in (BIFET et al., 2009) and (BIFET; HOLMES; PFAHRINGER, 2010). In practice, there is one ADWIN detector per Hoeffding tree, and whenever the change detector of a particular tree reports a significant decrease in accuracy, this tree is reset and the learning coefficients in the perceptrons associated with it are set to zero. There is another approach tested in the paper that consists in not resetting the tree in which the change was detected, but the tree with the lowest accuracy estimates.

The work presented in (BIFET et al., 2012) include several experiments, which were not only designed to verify the new method proposed in it, but also to test individual properties of it when they are applied to ADWIN Bagging. For example, an experiment is reported where an ADWIN Bagging based ensemble uses stacking as the combination method. Authors note that the stacking procedure merits further investigation, as it could not only improve on their new method, but also ADWIN Bagging results.

3.5.12 Dynamic Weighted Majority

The Weighted Majority (WM) algorithm (LITTLESTONE; WARMUTH, 1994) weights classifiers votes based on past performance, such that every classifier has a weight β, which is decreased every time it predicts incorrectly. Authors in (KOLTER; MALOOF, 2007) introduced the Dynamic Weighted Majority (DWM) algorithm, which uses the same voting strategy as WM with the addition of update heuristics to cope with evolving streams. The update method includes removing classifiers if their weight β is below a given user threshold α. The combined decision of the ensemble is a simple aggregation of each classifier prediction weighted by their β value. In DWM, the ensemble size varies due to removals and additions of classifiers. Originally, new classifiers are added whenever the ensemble prediction is incorrect. Although since this heuristic may cause too many additions of classifiers authors introduced a user given parameter p, which defines after

how many instances an ensemble update (additions and removals) will happen. DWM does not include an explicit drift detection method. Concretely, adaptation to drifts occur indirectly through the removals of old classifiers, as their β value degenerates in response to prediction errors on the current instances. This strategy is suitable for gradual drifts, although its efficiency depends directly on the parameter p. Despite the fact that p depends on the user, it is fixed throughout the whole execution, assuming that the stream drift characteristics can change as time goes on, this method fails to adapt the algorithm to the current state of the stream.

3.5.13 Online Accuracy Updated Ensemble

In (BRZEZINSKI; STEFANOWSKI, 2014) the Online Accuracy Updated Ensemble (OAUE) is presented as an evolution to existing block-based ensemble methods for data stream mining, namely Accuracy Weighted Ensemble (AWE) and Accuracy Updated Ensemble (AUE). A block-based classifier, as described in (BRZEZINSKI; STEFANOWSKI, 2014), is any classifier that needs to be trained in a block of instances before being able to perform predictions. The opposite of a block-based classifier is an incremental classifier, which is able to update its hypothesis after every instance. Even though it is possible to train block-based classifiers on blocks of one instance, the results would be suboptimal, since these classifiers need more instances in order to produce a consistent hypothesis. OAUE aims at combining the best characteristics of three adaptions to block-based ensembles presented in (BRZEZINSKI; STEFANOWSKI, 2014), namely: online evaluation of components; introduce an additional incremental learner; and addition of a drift detector algorithm. Even though authors claim that the best of each of these adaptations were included in OAUE, only the first two are actually implemented, i.e., OAUE does not include an active drift detector algorithm.

In OAUE, a window of d instances is used to determine how many instances are going to be used to train a new classifier, namely the candidate. At the beginning of every window a new candidate classifier is created and trained only on the next instances. Also, the candidate does participate on the ensemble predictions during the window in which it was created. When the window comes to an end, if the total number of component classifiers is less than a parameter k the current candidate is added to the ensemble, otherwise the candidate replaces the least accurate component classifier.

Since OAUE does not use an active drift detection technique, it relies on gradual resets of the ensemble through candidates to adapt to concept drifts. The weighting mechanism also contributes to the ensemble adaptation to concept drifts, since the weighting

function is designed to assign higher impact to predictions on recently presented instances. The weighting function is based on the equations, introduced in (WANG et al., 2003), originally developed to estimate prediction error for block-based classifiers. Equations 3.8, 3.9, 3.10, 3.11, 3.12 presents the adaptations to these functions, presented in (BRZEZINSKI; STEFANOWSKI, 2014) that allow incremental estimations of prediction error, thus permitting the weighting function usage for incremental classifiers.

$$MSE_r^t = \begin{cases} MST_i^t - 1 + \frac{e_i^t}{d} + \frac{e_i^{t-d}}{d}, & t - \tau < d \\ \frac{t-\tau_i-1}{t-\tau_i} MSE_i^t - 1 + \frac{e_i^t}{t-\tau_i}, & 1 \leq t - \tau_i \leq d \\ 0, & t - \tau_i = 0 \end{cases} \tag{3.8}$$

$$e_i^t = \left(1 - f_{iy}^t\left(x^t\right)\right)^2 \tag{3.9}$$

$$MSE_r^t = \begin{cases} MSE_r^{t-1} - r^{t-1}(y^t) - r^{t-1}(y^{t-d}) + r^t(y^t) + r^t(y^{t-d}), & t > d \\ \sum_y r^t(y), & t = d \end{cases} \tag{3.10}$$

$$r^t\left(y\right) = p^t\left(y\right)\left(1 - p^t\left(y\right)\right)^2 \tag{3.11}$$

$$w_i^t = \frac{1}{\left(MSE_r^t + MSE_i^{t+\epsilon}\right)} \tag{3.12}$$

Such that:

- d = window (period) size;
- t = sequential number assigned to each instance, also denotes the number of instances already seen;
- i = component classifier index in the ensemble;
- τ_i = time at which classifier i was created;
- f_{iy}^t = probability given by classifier C_i that instance x^t is an instance of class y^t;
- ϵ = a very small value to avoid division by zero;
- $r^t(y)$ = for instance x^t and class value y, the probability of t being randomly assigned y;

- $p^t(y)$ = class distribution of y up to instance x^t.

MSE_i^t estimates the prediction error of classifier C_i on the last d instances, and it is scaled according to the number of instances seen. MSE_r^t is the mean square error of a randomly predicting classifier, also trained on the d instances, and it is used as a reference point to predictions made based on the current class distribution.

Regarding time and memory demands, authors show in (BRZEZINSKI; STEFANOWSKI, 2014), through empirical experiments, that OAUE is able to achieve a comparable tradeoff between resources usage and accuracy obtained when compared to other state-of-the-art algorithms.

3.5.14 Scale-free Network Classifier

The Scale-free Network Classifier (SFNC) (BARDDAL; GOMES; ENEMBRECK, 2014) is an ensemble method that weights classifiers predictions based on an adaptation of the scale-free network construction model. In SFNC, classifiers are arranged in a graph structure, such that classifiers with higher accuracy are more likely to connect to recently added classifiers. Classifiers weighting is directly proportional to a user given centrality metric α, e.g., eigenvector. Since high accurate classifiers usually receive many connections, these are expected to have higher influence on the overall decision. Although this process is non-deterministic and, therefore low accuracy classifiers can become prominent, that is very rare in practice. The worst classifier, in terms of accuracy, is removed from the network every p instances, and that triggers a rewiring process to maintain the graph connected.

3.5.15 Social Adaptive Ensemble

The Social Adaptive Ensemble (SAE) (GOMES; ENEMBRECK, 2013) arranges ensemble members as a network (undirected graph), such that there is one node for each classifier and a connection between two nodes quantify their similarity with respect to past predictions. Precisely, connections are weighted according to the Similarity Coefficient (Sc) shown in Equation (3.13), where l represents the period length and $k_{i,j}$ accounts for how many instances i and j predicted the same class value.

$$Sc(i,j) = \frac{k_{i,j}}{l} \tag{3.13}$$

SAE uses a fixed window (period), denoted as l, to determine after how many instances a network updates happen, similarly to DWM. These updates happens at the end

of every period and includes: addition and removal of classifiers, and network structural changes. Many classifiers can be removed during a network update, but only one classifier can be added. The main difference between SAE and other ensemble methods is that it uncovers similarities among classifiers and uses this knowledge to update the ensemble and to combine individual predictions. The relationships among classifiers are depicted as a network, which is updated every period to better approximate the current state of its members' similarities. One example of usage of the network is to identify highly correlated (redundant) classifiers, i.e., almost always predict the same class value given the same instances. The identification of such pairs is followed by the removal of one of the pair, thus improving run time without drastically affecting overall accuracy.

To increase diversity among classifiers SAE uses Online Bagging (OZA, 2005) for training. Adaptation to concept drifts happens implicitly in SAE due to its adaptation strategies which includes removing classifiers with very low individual performance during last period and adding new classifiers trained on last period misclassified instances. These strategies are complemented by a combination method that combine similar classifiers decisions before applying majority vote to predict a new instance class value.

One of the main drawbacks in SAE is that new classifiers are only trained on instances that were misclassified during the last period. This method is appropriate if there were a large number of misclassified instances, possibly due to a concept drift, but it is not suitable if there were only a few misclassified instances, since the new classifier would be trained only on a small set of instances. Besides that, new classifiers are kept isolated from other classifiers in the network during the period subsequent to its addition, independently of their prediction similarities to other classifiers. Thus incorrectly highlight the new classifier predictions as if it were dissimilar to all other classifiers. Other issues related to SAE include an elevated amount of prediction ties and its unconstrained addition method, which can cause the ensemble to grow very large.

3.6 Final Considerations

This chapter contains the fundamentals of ensemble classifiers for data streams, including a discussion about diversity, combination and update dynamics. These concepts serve as basis for our current work, since we are proposing of a new kind of data stream ensembles. We also present state-of-the-art ensemble classifiers for data streams in this chapter. Most methods presented on Section 3.5 focus on one or two aspects of ensemble learning for data streams. For example, Online Bagging focus on adding diversity to the ensemble through a simulation of resampling, while OAUE uses background learners (or

candidates) to implicitly recover from concept drifts and uses a sophisticate combination (voting) method. In this work, we hypothesize that it is possible to achieve accurate predictions through the use of an ensemble classifier that exploits relational data extract from it component classifiers. The SAE algorithm (GOMES; ENEMBRECK, 2013) is a seminal work on this area, yet we observe that SAE does not use its network of classifiers to its maximum extend. For example, SAE simplifies its weighted connections to dichotomous relations based on a fixed threshold. In the next Chapter, we present the definition of a network-based ensemble, a preliminary implementation and proposals for subsequent developments.

Chapter 4

Methodology

To date, there are many ensemble classifiers for data stream learning. Some of these ensembles employ different approaches that have proved to be efficient, such as: implicit drift recovery, explicit drift detection, periodic updates, diversity enhancing training, and so forth. In this work, we propose a new family of data stream ensemble classifiers, namely the network-based ensembles. The determining characteristic of network-based ensembles is the exploration of relational data obtained from component classifiers. These relations can be used, for example, to define unique combination methods or to identify redundancy between classifiers.

We introduce network-based ensembles by first positioning them in relation to other stream ensemble classifiers. To achieve this, we propose a general data stream ensemble classifiers taxonomy, and outlines which portions of it are most representative to network-based ensembles. Subsequently, we present a formal definition, followed by a preliminary implementation of a network-based ensemble. We close our explanations with individual discussions around "Combination" and "Relation", followed by an outline of a novel network-based ensemble method.

4.1 A Taxonomy of Data Stream Ensemble Classifiers

Ensemble learning has been an active research topic in the last years. Thus, many taxonomies and classifications (BROWN et al., 2005; KUNCHEVA, 2004; ROKACH, 2009) have been proposed to provide a reasonable way to think about ensembles, and to identify opportunities for future work.

In this section, we present a taxonomy for data stream ensemble classifiers in order to address the specific objective 1 of this work (see 1.2). The taxonomy is presented in Figure 4.1. This taxonomy shares with previous proposals (for batch learning) the dis-

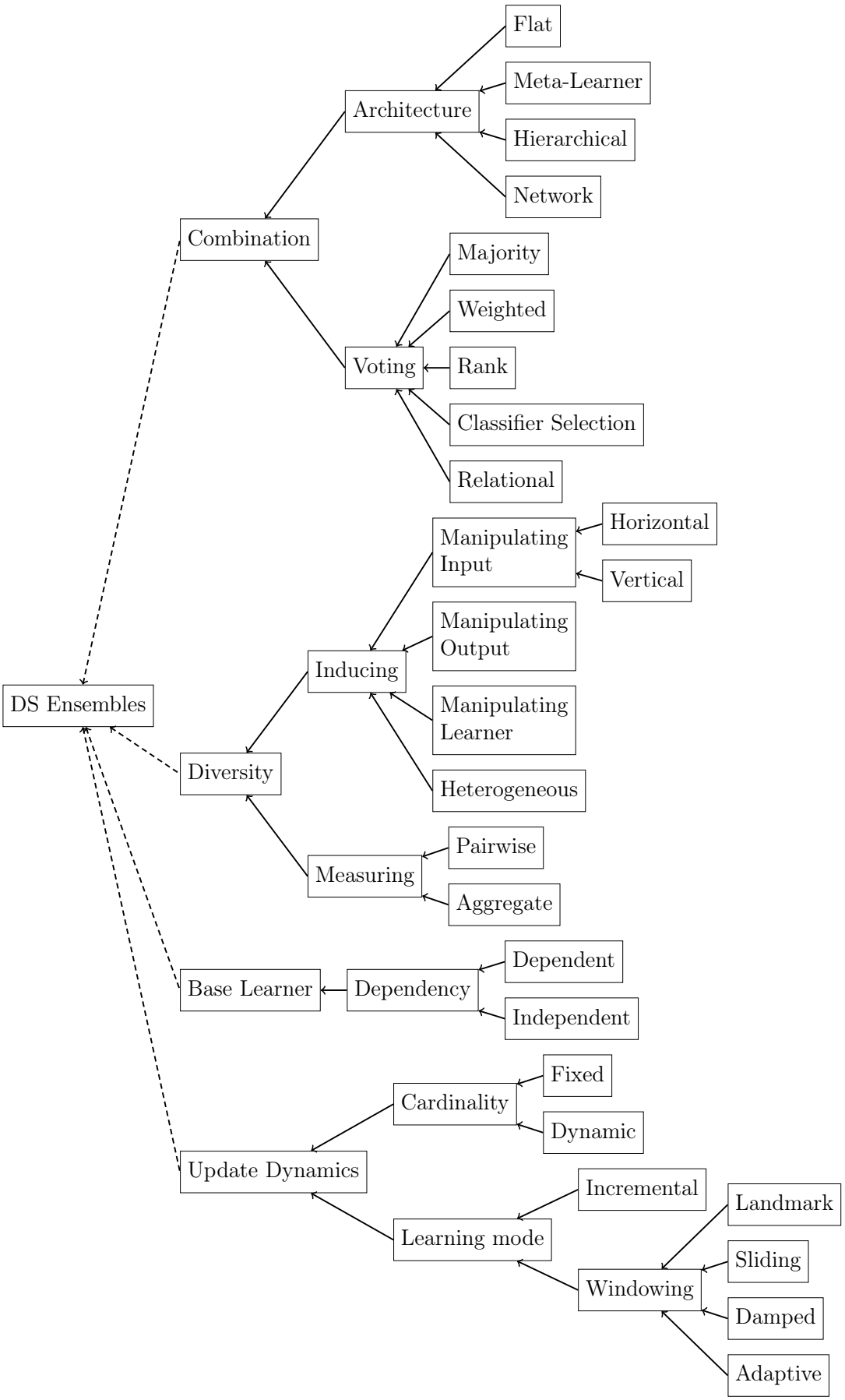


Figure 4.1: A taxonomy of data stream ensemble classifiers. Dashed edges = 'part of', Normal edges = 'a kind of'.

tinction between two very important concepts for ensemble learning, namely, combination and diversity. The difference to its predecessors resides in the inclusion of a dimension that is relevant to data stream learning, namely 'update dynamics'. This part of the taxonomy represents important methods for stream learning, e.g., strategies to cope with drifts. Generally, it is difficult to clearly separate different methods into a taxonomy, since some areas are blurred and grouped together by different algorithms. For example, one algorithm may use one technique to adapt to concept drifts that also induces diversity among classifiers.

The dimensions presented in the taxonomy are discussed in Sections 4.4, 3.2, 3.3, and 3.4. Therefore, in the following sections we focus the discussion around the dimensions that are relevant to network-based ensembles, i.e., "combination → structure → network" and "diversity → measuring". The latter is identified as a form of "relation" in our work.

Table 4.1 presents the classification of ensemble learners from the literature (see Section 3.5), alongside with the implementations presented in this work, according to the taxonomy presented. Our implementations are: SAE2 (Section 4.3), PA (Section 4.5.1), PP (Section 4.5.2) and CNE (Section 4.6). The columns and rows identifiers from Table 4.1 are presented below:

- **(A) Architecture**: f: flat, m: meta-learner, h: hierarchical, n: network;
- **(V) Voting**: m: majority, w: weighted, r: rank, s: classifier selection, re: relational;
- **(DI) Diversity Inducer**: he:heterogeneous, l: learner manipulation, v: vertical input, v^*: vertical input with instance weighting, h: horizontal input, o: output, t: time-based;
- **(B) Base Learner**: b: batch, i: incremental;
- **(D) Dependency**: d: dependent, i: independent, h: hybrid;
- **(C) Cardinality**: f: fixed, f^*: fixed (derived from feature set), m: maximum, d: dynamic;
- **(L) Learning Mode**: s: sliding window, l: landmark window, d: damped window, a: adaptive window, i: incremental.

Algorithm	(A)	(V)	(DI)	(B)	(D)	(C)	(L)	Reference
SEA	f	m	t	b	i	m	l	(STREET; KIM, 2001)
Learn^{++}	f	w	v	b	d	f	l	(POLIKAR et al., 2001)
AWE	f	r	t	b	i	f	l	(WANG et al., 2003)
CDC	f	w	t	i	i	m	i	(STANLEY, 2003)
FLBoost	f	w	v*	b	d	f	l	(CHU; ZANIOLO, 2004)
CBEA	f	s,m	t	b	i	m	l	(RUSHING et al., 2004)
AO-DCS	f	s	v	b	i	f	l	(ZHU; WU; YANG, 2004)
OzaBag	f	m	v	i	i	f	i	(OZA, 2005)
OzaBoost	f	w	v*	i	d	f	i	(OZA, 2005)
AddExpert	f	w	t	b	i	f	l	(KOLTER; MALOOF, 2005b)
ACE	f	w	t	i,b	i	d	l,i	(NISHIDA; YAMAUCHI; OMORI, 2005)
FAE	f	w	t,h	i	i	d	i	(WENERSTROM; GIRAUD-CARRIER, 2006)
DWM	f	w	t	b	i	d	l	(KOLTER; MALOOF, 2007)
BoostDC	f	w	v*	b	d	f	l	(SCHOLZ; KLINKENBERG, 2007)
ICEA	f	w	t	i	i	m	i	(YUE et al., 2007)
RDE	f	w	t	b	i	d	l	(RAMAMURTHY; BHATNAGAR, 2007)
Streaming RF	f	m	h	i	i	f	i	(ABDULSALAM; SKILLICORN; MARTIN, 2007)
Dynamic SRF	f	m	h	i	i	f	i	(ABDULSALAM; SKILLICORN; MARTIN, 2008)
MCIK-Ensemble	f	w	v	i,b	i	f	l	(MASUD et al., 2008)
ASHT Bag	f	w	l	i	i	f	a	(BIFET et al., 2009)
ADWIN Bag	f	m	v	i	i	f	a	(BIFET et al., 2009)
OVA Trees	f	w	o,v	i	i	f	i	(HASHEMI et al., 2009)
OCBoost	f	w	v*	i	d	f	i	(PELOSSOF et al., 2009)
Learn^{++}.NC	f	w,re	t	b	i	f	l	(MUHLBAIER; TOPALIS; POLIKAR, 2009)
FISH	f	s	v	i	d	f	a	(ŽLIOBAITĖ, 2009)
LevBag	f	m	v,o	i	i	f	a	(BIFET; HOLMES; PFAHRINGER, 2010)
CCP	f	s	v	i,b	i	d	l	(KATAKIS; TSOUMAKAS; VLAHAVAS, 2010)
Learn^{++}.UDNC	f	w,re	t	i	i	f	l	(DITZLER; MUHLBAIER; POLIKAR, 2010)
DXMiner	f	w	t	i,b	i	f	l	(MASUD et al., 2010)
ONSBoost	f	w	v*	i	d	f	l	(POCOCK et al., 2010)
AUE	f	r	v,t	i	i	f	l	(BRZEZIŃSKI; STEFANOWSKI, 2011)
AE	f	w	he,v	i,b	i	f	l	(ZHANG et al., 2011b)
BWE	f	w	t	b	i	f	a	(DECKERT, 2011)
Learn^{++}.NSE	f	w	t	b	i	f	l	(ELWELL; POLIKAR, 2011)
DDD	m	w	v,l	b	i	f	a	(MINKU; YAO, 2012)
RestrictedHF	m	w	v,h	i	d	f*	a	(BIFET et al., 2012)
HEFT-Stream	f	w	he,h,v	i	i	f	l	(NGUYEN et al., 2012)
AEBC	f	w	v	i	d	f	a	(WANKHADE et al., 2012)
HSMiner	h	w	he,h,o	i	h	f*	l	(PARKER; MUSTAFA; KHAN, 2012)
OSBoosting	f	w	v*	i	d	f	l	(CHEN; LIN; LU, 2012)
Woo	f	w	v	i	i	d	l	(RYU; KANTARDZIC; KIM, 2012)
OOB	f	m	v	i	i	f	i	(WANG; MINKU; YAO, 2013)
UOB	f	m	v	i	i	f	i	(WANG; MINKU; YAO, 2013)
SAE	n	m	v,t	i	h	d	l	(GOMES; ENEMBRECK, 2013)
DACC	f	m	t	b,i	i	f	l	(JABER; CORNUÉJOLS; TARROUX, 2013b)
ADACC	f	m	t	b,i	i	f	a	(JABER; CORNUÉJOLS; TARROUX, 2013a)
SluiceBox	h	w	he,h,o	i	i	f	l	(PARKER; KHAN, 2013)
Learn^{++}.CDS	f	w	t	i	i	f	l	(DITZLER; POLIKAR, 2013)
Learn^{++}.NIE	f	w	v,t	i	i	f	l	(DITZLER; POLIKAR, 2013)
RCD	f	m	t	i	i	m	l	(JR; BARROS, 2013)
OAUE	f	r	t	i	i	f	l,i	(BRZEZINSKI; STEFANOWSKI, 2014)
SFNC	n	w	t	i	i	m	l	(BARDDAL; GOMES; ENEMBRECK, 2014)
SAE2	n	w	v,t	i	h	m	l	(GOMES; ENEMBRECK, 2014)
M^3	f	w	he	i	i	f	l	(PARKER; KHAN; BIFET, 2014)
SE-PLS	h	w	v	i	i	f	i	(SETHI et al., 2014)
Fast-AE^6	f	w	t	i	i	m	l	(DÍAZ et al., 2015a)
IBEP	f	w	v	i	i	f	l	(ZHI et al., 2015)
PA/PP	f	re	v	i	i	f	l	(GOMES; BARDDAL; ENEMBRECK, 2015)
Online BBM.W	f	w	v*	i,b	d	f	i	(BEYGELZIMER; KALE; LUO, 2015)
AdaBoost.OL.W	f	w	v*	i,b	d	f	i	(BEYGELZIMER; KALE; LUO, 2015)
SluiceBox-AM	h	w	he,h,o	i	i	m	i	(PARKER; KHAN, 2015)
WEOB	f	w	v	i	i	f	i	(WANG; MINKU; YAO, 2015)
EDTC	f	m	h	i	i	f	i,s	(LI et al., 2015)
BLAST	m	w	he	i	i	f	i	(RIJN et al., 2015)
MOOB	f	r	v	i	i	f	i	(WANG; MINKU; YAO, 2016)
CNE	n	w	v,h,t	i	i	f	a	-

Table 4.1: Data stream ensemble classifiers according to our taxonomy

4.2 Formal Definition of Network-Based Ensembles

Actual implementations are useful for empirical evaluations, and for studying specific characteristics of algorithms. While formal abstract definitions can be used for reasoning in a higher level about a method. In this section we present a formal description of a general network-based ensemble, which specifically achieves objective 2 of this work (see Section 1.2).

Let $C = \{c_1, c_2, c_M\}$ be a diverse set of classifiers, R a relation that defines connections $\Phi = \{\phi_1, \phi_2, \phi_P\}$ between members of C, β a combination method that takes into account the structure formed by Φ, and f_ψ an adaptation function that updates C and Φ according to the current state of a data stream S.

Notice that it is expected that members of C are different from one another, i.e., diverse. The reason for that is to be consistent with the intuitive principle that a homogeneous subset of classifiers cannot contribute to the overall decision any better than any of them alone. For the sake of generality our definition is not bound to any specific method to induce diversity into the ensemble.

An important trait in this definition is that connections are not strictly defined to be between pairs of classifiers. Even though, pairs are the most intuitive way to group elements in a network (WASSERMAN; FAUST, 1994) or measure diversity between classifiers (KUNCHEVA; WHITAKER, 2003), we do not want to restrict other types of connections. Also, the relation R is not strictly defined to be a measure of diversity or similarity, so far we can only envision one of these two to be useful for a combination method (see Sections 4.3 and 4.6), although it would be a limiting factor if we added it to this general definition.

Recalling our hypothesis, we state that "An ensemble classifier can obtain accurate predictions with the aid of structural analysis of a weighted network of its component classifiers" (see Section 1.3). The "aid" referenced in our hypothesis is the combination method β, which must use the set of connections Φ as a source of information to achieve accurate predictions. For example, in SAE (GOMES; ENEMBRECK, 2013) classifiers deemed as similar are combined into subnetworks in order to obtain a voting method in which similar classifiers do not reinforce their homogeneous decisions. A subnetwork's decision is effectively interpret as a single vote at the ensemble level. In this case, we observe that the combination method used the information of "how similar" component classifiers are to avoid a large set of similar classifiers dominating the ensemble decisions.

The last component of our definition is the adaptation function f_ψ. This function must update the ensemble structure, either periodically or incrementally, to allow it to

adapt to drifts. These updates may include adding, removing, or replacing classifiers, and refreshing statistics extracted from classifiers, such as similar predictions counters. In SFNC (BARDDAL; GOMES; ENEMBRECK, 2014), a Scale-free network (ALBERT; BARABÁSI, 2002) model is used to update the ensemble, in which periodically a new classifier is added to the ensemble.

The following section presents an instantiation of the theoretical network-based ensemble, namely the Social Adaptive Ensemble 2 (SAE2). Latter, in Section 4.6, we present another network-based ensemble, namely the Complex Network Ensemble (cne).

4.3 A Preliminary Network-Based Ensemble: The Social Adaptive Ensemble 2

In Section 4.2 a formal general description network-based ensembles was introduced. This kind of formal abstract definition can be used for reasoning about high level properties, without bounding these to specific implementations. Although abstract methods cannot be tested or compared with existing algorithms. Thus, aiming at providing an initial attempt to solve objective 4 of this work (see Section 1.2), in this section we present the details of an instantiation of our generic network-based ensemble, namely the Social Adaptive Ensemble 2 - SAE2.

The original SAE (GOMES; ENEMBRECK, 2013) algorithm focused on three aspects of ensemble classifiers in a data stream scenario, which are: diversity, combination and adaptation. SAE2 is based on the same hypothesis as SAE, but its adaptation mechanisms have been significantly enhanced. The hypothesis behind SAE and SAE2 is that if it is possible to track component classifiers similarity, then it is feasible to quantify diversity; combine individual predictions in a way that similar classifiers predictions are grouped; remove redundant classifiers; and in the occurrence of a concept drift, emphasize recently added classifiers predictions.

SAE and SAE2 use relational information to refine the ensemble. The relational ties among ensemble members are weighted according to the relation "has similar predictions". We found it useful to employ Social Network terminology to refer to SAE2 functionalities. Hence on this Section we refer to new classifiers as candidates, similarity between two classifiers as a connection, subsets of similar classifiers as subnetworks and the whole ensemble as network. Figure 4.4a presents an example of a network of classifiers.

SAE2 divides training in periods (windows) of c instances, each of which are presented only once to the network and used to train classifiers and candidates using Online

Bagging. The network is updated at every period end to adapt it to the current concept. This update includes the following actions in order: classifiers' removal, candidate's addition and network restructure. There is only one candidate per period, but the number of classifiers that form the network vary from one to max_e. Figures 4.2 and 4.3 presents an overview of the periodical updates (periods) and an outline of a SAE2 period, respectively.

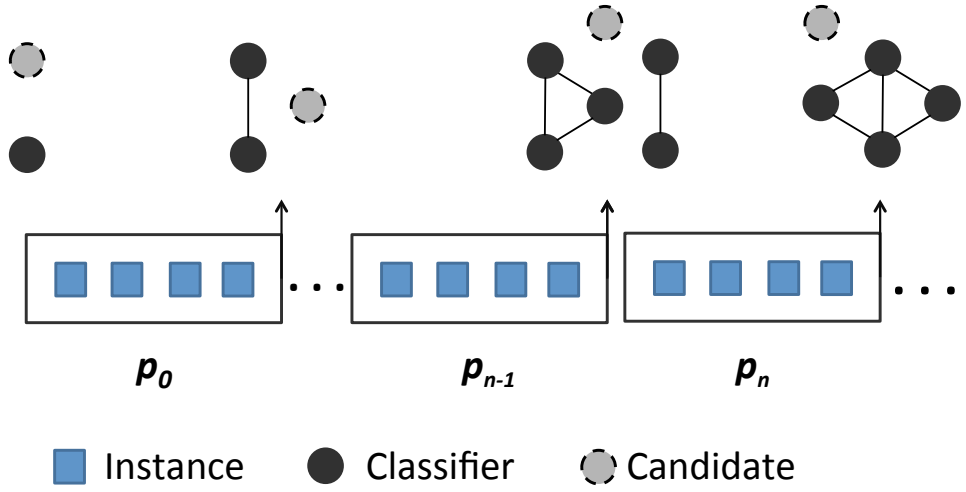


Figure 4.2: Overview of periods in SAE2 with the network structure obtained at every period end.

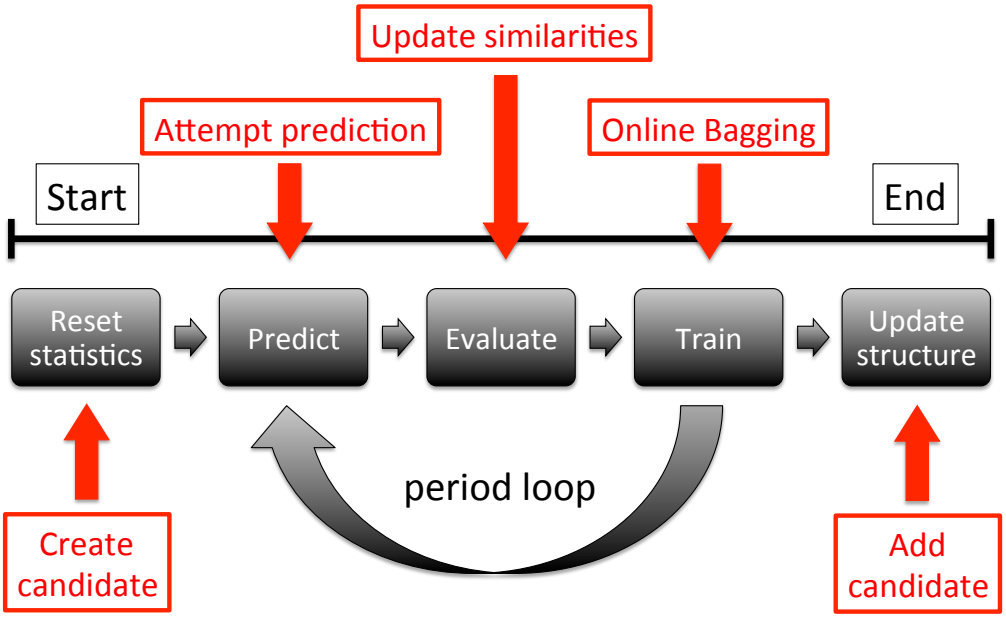


Figure 4.3: Schematic representation of a period in SAE2.

The candidate is trained along with the network during the period in which it was created, although it does not influence the network predictions until it is added to the network (similarly to OAUE (BRZEZINSKI; STEFANOWSKI, 2014)). One reason

to train the candidate before adding it is to identify its similarities with the rest of the ensemble. If it is very similar to others, it will become part of an already established component, otherwise, it will be left isolated and its predictions will have direct impact on network predictions. Also, if candidates were trained only on incorrectly classified instances in some cases there would be too little instances to train on, thus candidates would have low individual accuracy. The reason to reinforce candidates' learning on misclassified instances is to allow them to correctly classify them, thus enhancing network diversity and adapting the candidate to concept drifts.

SAE adds an classifier to the network if the network accuracy is below a minimum threshold ($Acc_{net.min}$). Therefore, if the parameter $Acc_{net.min}$ is too optimistic a classifier will be added every period and since there are no limits imposed on the ensemble size it could grow to an unreasonable size, e.g., more than fifty classifiers. In SAE, it was expected that the removal method could aid this situation by constantly removing classifiers, although there are some exceptional situations not covered by it. In (GOMES; ENEMBRECK, 2013), the authors present an experiment with a Random Tree Generator, identified as RTC, in which performance was decreased and accuracy not significantly increased, due to the addition of too many classifiers. To aid this situation, in SAE2 a threshold that limits ensemble size (max_e) has been added. Also, the $Acc_{net.min}$ threshold has been removed and the candidate classifier is added to the network if it has not been marked for removal by the removal method. If max_e has been reached, the classifier with lowest accuracy during last period is replaced by the candidate.

After being presented to a great volume of instances that belongs to the same concept it is possible that two classifiers that initially had different hypothesis become very similar with respect to their predictions. Consequently, it is useful to track similarity between classifiers and take actions based on that. These actions vary from removing one of two very similar (redundant) classifiers to combining decisions of similar enough classifiers. The subjective concepts of "very similar" and "similar enough" are based on the Similarity Coefficient (Equation (3.13)) and determined by the Maximum and Minimum Similarity Coefficient, Sc_{max} and Sc_{min}, respectively.

Maintaining a diverse set of classifiers is not enough to enhance accuracy. The combination method plays the important role of highlighting correct and obfuscating incorrect predictions. In SAE and SAE2, predictions are first combined within subsets of similar classifiers (subnetworks), and afterwards their outputs are combined to obtain the final prediction. There are mainly two reasons to combine predictions in two levels. First, it is reasonable to combine highly similar classifiers, such that they do not dominate predictions based only on their quantity; and second, it is faster to recover from concept

drifts, since the combination method tends to assigns more weight on isolated classifiers, which is exactly what happens to a candidate if a drift has occurred. The differences between SAE and SAE2 rely on how classifiers are combined (subnetworks generation) and weighted during voting.

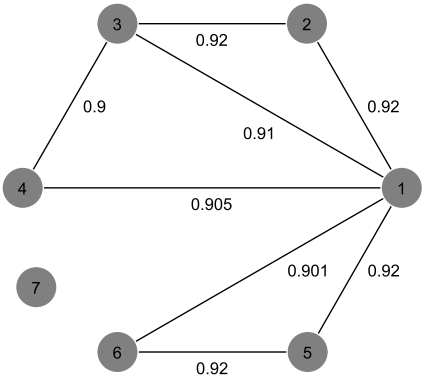
Identifying subnetworks (or communities) is a complex problem, precisely a NP-complete problem, and it has been thoroughly investigated by the complex network research community (FORTUNATO, 2010). For network-based ensembles besides the efficiency of the subnetwork generation process, it is also important to consider its semantics, i.e., what are the traits of the discovered subnetworks. In SAE, subnetworks are generated through identification of weakly connected components, while SAE2 uses maximal cliques to generate subnetworks. Figure 4.4 presents how subnetworks are generated based on the network shown in Figure 4.4a using both methods. By generating subnetworks through weakly connect components, classifiers that are dissimilar, may become part of the same component, for example, classifiers 6 and 4 in Figure 4.4c. This behavior is inconsistent with the notion of grouping similar classifiers, therefore SAE2 obtains subnetworks through identification of maximal cliques, i.e., members of a component must be connected to all other members as shown in Figure 4.4b.

SAE performs majority voting within subnetworks followed by a weighted majority voting based on subnetworks predictions. Subnetworks' votes are decreased if its members' predictions are split. For example, given an instance i and subnetwork $A = \{1, 2, 3, 4, 5, 6\}$ (Figure 4.4c), if all members of A predicted a class value of 1 for i, besides 5 that predicted 0, then subnetwork A will predict 1 with a weight of 5/6. This tends to increase isolated classifier's influence, since their prediction weights are always maximum. This voting method may cause too many ties and it can highlight low accuracy classifiers predictions, simply because they are isolated. In SAE2, voting is weighted according to classifiers and subnetworks accuracy during the current period. First, classifiers' predictions are combined within subnetworks based on a weighted majority vote, such that weight is given by their accuracy. Afterwards, a weighted majority vote is performed based on the subnetworks' predictions and weighted by the average accuracy of its members. Ties are diminished and unreasonable emphasis on low performance classifiers are avoided by this voting method.

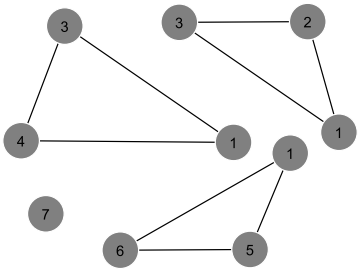
Classifiers and the candidate are subject to be marked for removal if their accuracy is bellow the min_{acc} threshold or if their predictions are redundant with respect to other classifier. This strategy assists the addition and combination methods in maintaining an ensemble of diverse classifiers, such that those that impact performance (redundant) and accuracy (low accuracy) are removed. When two classifiers are identified as redundant,

Algorithm 1: SAE2 algorithm. **Input:** Data stream S that provides an instance i every t moments. Period length l. Maximum and Minimum Similarity Coefficient, Sc_{max} and Sc_{min}, respectively. Max classifiers max_e. Minimum classifier accuracy min_{acc}. **Local variables:** Network of classifiers N. Instance $i = (\vec{x}_i, y_i)$. Set of misclassified instances I. Candidate classifier c.

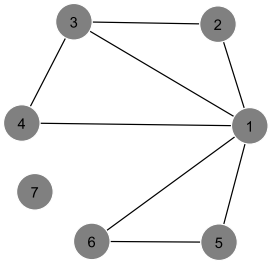
Require:
$max_e \geq 1 \wedge Sc_{min} \in (0,1) \wedge Sc_{max} \in (0,1) \wedge Sc_{min} < Sc_{max} \wedge min_{acc} \in (0,1) \wedge l \geq 1$
$e_0 \leftarrow new.classifier$
$N \leftarrow \{e_0\}$
while $has.next(S)$ **do**
  $reset.statistics(N)$
  $c \leftarrow new.classifier$
  $I \leftarrow \{\}$
  for $j \leftarrow 1$ **to** l **do**
    $i \leftarrow next(S)$
    $d_i \leftarrow predict(N, i)$
    if $d_i \neq y_i$ **then**
      $I \leftarrow I \cup \{i\}$
    end if
    for all $e \in N \cup \{c\}$ **do**
      $online.bagging(e, i, \lambda := 1)$
      for all $w \in (N \cup \{c\}) - \{e\}$ **do**
        $similarities.update(w, e)$
      end for
    end for
  end for
  $remove.classifiers(N \cup \{c\}, Sc_{max}, min_e)$
  if $not.removed(c)$ **then**
    $train(c, I)$
    if $size(N) = max_{exp}$ **then**
      $e_l \leftarrow lowest.accuracy(N)$
      $N \leftarrow N - \{e_l\}$
    end if
    $N \leftarrow N \cup \{c\}$
  end if
  $connections.update(N, Sc_{min})$
  $generate.subnetworks(N)$
end while



(a) Network composed of 7 experts and their Sc (Equation 3.13) between active connections



(b) Maximal cliques



(c) Weakly Connected Components

Figure 4.4: Network and its subnetworks using two different combination methods

that which was added first to the network is removed. This design decision is based on the assumption that recently added classifiers are less committed to older concepts (oldest first), thus they may adapt to new concepts faster. In the occasion of a concept drift many classifiers might be removed at once, while during periods without drifts redundant classifiers are more likely to be removed. If all classifiers are marked for removal, then the one with higher accuracy is kept.

The main function for SAE2 is presented in Algorithm 1. At the beginning of every period, all the statistics from the last period are discarded. These statistics account for the individual accuracy of classifiers, the similarity counter between pairs of classifiers $k_{i,j}$ and the set of misclassified instances I. The classifiers' hypotheses are kept across periods, but network structure may change. Subnetworks are generated only once at period end.

In the following sections we deepen our analysis about network-based ensembles by exploring the combination and relation definition processes.

4.4 Combination

Combination plays a very important role in an ensemble classifier. Ideally, the combination method is responsible for obfuscating incorrect and highlighting correct pre-

dictions. For conventional ensembles, combination is analogous to voting, but for network-based ensembles it is worthwhile to separate the combination into two separate processes: building the structure and voting. The building block of the structure is the connections between classifiers. The connections adhere to the relation defined for the ensemble. In SAE2, connections are created between pairs of classifiers based on their similarity, afterwards this structure is filtered and a network of classifiers is obtained. It is expected that voting exploit the network structure. For example, in SAE2 voting is performed in two levels, such that the first one is internal to subgroups of classifiers as specified by the network structure. One could argue that conceptually building the structure is also part of voting, but for the existing methods (SAE, SAE2, SFNC), it has been considered a separate step, as it involves complex steps that, different from voting, are not executed every time a new prediction is requested.

In SAE (GOMES; ENEMBRECK, 2013) empirical reasons are presented that justify the exploration of an ensemble beyond simply assigning individual weights or reinforcing diversity. Although it is not clear that those results were due to the network or due to other characteristics, such as periodical additions and removals of classifiers, time dependent weighting, or diversity enhancing training. To investigate the benefits of using a network of classifiers, in Section 5.2, we present results of SAE2 with 3 different variations of combination method, including a "no network" version. Also, a more in-depth analysis of how combination influence the overall results is presented in Section 5.4.

4.5 Relation R

The relation R is the fundamental block that defines how connections are formed and weighted. Intuitively, to produce useful information for the combination step the relationship between a pair of classifiers must express their similarities or dissimilarities. The most generic way to measure similarity or dissimilarity between a pair of classifiers is to observe their outputs given the same input. Notice that there may be other forms of defining relations for a set of classifiers, therefore we do not bound our general definition to "similarity/diversity" weighting (see Section 4.2). In SAE2, and SAE, the Similarity Coefficient (Sc) is used to define the relation between classifiers. Sc measures the similarity between classifiers according to the number of instances that both predicted the same class label. One benefit of this approach is that SAE and SAE2 do not depend on a specific base learner.

In the following sections we present two different relation definitions that are associated with objective 3 of this work (see Section 1.2). These relations are used to define

voting strategies that focus almost completely on the relational data extracted from the component classifiers. These strategies are mainly based on pairwise combination of component classifiers. Despite efforts to build a diverse ensemble, there is always some degree of overlap between component classifiers models. In these methods, we hypothesize that by combining pairs of classifiers it is possible to alleviate incorrect individual predictions that would otherwise negatively impact the overall ensemble decision. The first strategy, Pairwise Accuracy (PA), combines the shared accuracy estimation of all possible pairs in the ensemble, while the second strategy, Pairwise Patterns (PP), record patterns of pairwise decisions during training and use these patterns during prediction. We also present the adaptations needed for an ensemble to incorporate PA and PP.

4.5.1 Pairwise Accuracy

Pairwise Accuracy (PA) combine classifiers into pairs, and weights the predictions of these pairs based on their shared estimated accuracy in the most recent instances. To estimate accuracy, we use a similar approach to the weighting function in SAE2 (see Section 4.3), such that individual and pairs of classifiers are weighted incrementally and reset periodically according to a fixed window (period). Formally, let U_n be the set of all instances from window n, I and J be subsets of U_n, which classifiers c_i and c_j were able to correctly classify, respectively, and I_e and J_e instances drawn from U_n that c_i and c_j incorrectly classified, respectively. Also, $U_n^t = I \cup J \cup I_e \cup J_e$ is the set of all instances from window n already presented to c_i and c_j up to instance t. We define the pairwise estimated accuracy, namely $S_{acc}(c_i, c_j)$ (Equation 4.1), of classifiers c_i and c_j, as the ratio of instances from U_n^t that both correctly classifies. Conversely, c_i and c_j shared estimated error rate, namely $S_{err}(c_i, c_j)$ (Equation 4.2), is defined as the ratio of instances from U_n^t that both incorrectly classifies.

$$S_{acc}(c_i, c_j) = \frac{|I \cap J|}{|U_n^t|} \tag{4.1}$$

$$S_{err}(c_i, c_j) = \frac{|I_e \cap J_e|}{|U_n^t|} \tag{4.2}$$

We also define the accuracy estimation for a single classifier c_i during window n up to instance t, namely $acc(c_i)$ (Equation 4.3), as the ratio of instances from U_n^t that c_i was able to correctly classify.

$$acc(c_i) = \frac{|I|}{|U_n^t|} \tag{4.3}$$

Figure 4.5 shows a graphical representation of how sets I, J, I_e, J_e could overlap.

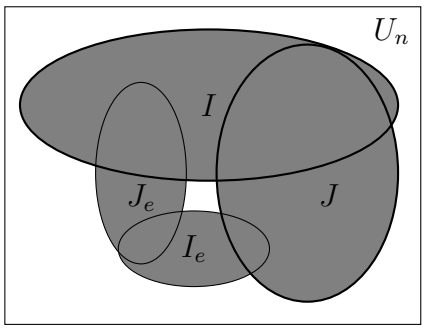


Figure 4.5: Venn diagram representation of window n and classifiers c_i and c_j correctly and incorrectly classified subsets of instances

In order to use S_{acc} and S_{err} during voting, we employ a weighting function that prioritizes equal pairwise predictions over individual predictions, if and only if, their pairwise predictions are accurate. For all c_i classifiers in C at window n, voting is performed in pairs, such that $\binom{|C|}{2}$ pairs are formed. Using a vector $\vec{v}$ to store weights during voting, such that every position in $\vec{v}$ corresponds to a possible label and is initialized with 0, and assuming individual predictions are denoted by $h(x)$, each pair of classifiers (c_i, c_j) contributes to the overall prediction through Equation 4.4, if c_i prediction matches c_j prediction, i.e., $h_i(x) = h_j(x)$, or Equations 4.5 and 4.6, if c_i and c_j predictions diverge for an instance x, i.e., $h_i(x) \neq h_j(x)$.

$$\vec{v}(h_i(x)) := \vec{v}(h_i(x)) + S_{acc}(c_i, c_j) - S_{err}(c_i, c_j) \tag{4.4}$$

$$\vec{v}(h_i(x)) := \vec{v}(h_i(x)) + acc(c_i) - S_{acc}(c_i, c_j) \tag{4.5}$$

$$\vec{v}(h_j(x)) := \vec{v}(h_j(x)) + acc(c_j) - S_{acc}(c_i, c_j) \tag{4.6}$$

After inspecting every pair, the overall ensemble prediction is obtained by Equation 4.7.

$$\arg\max_i \vec{v}(i) \tag{4.7}$$

Through Equations 4.4, 4.5 and 4.6, it is possible to observe that given two classifiers with high S_{acc}, their split predictions will be degraded, while their equal predictions will obtain a high weight, which in turn is decreased by their shared mistakes S_{err}. If pairs disagree on their predictions, their individual decisions will be taken into account, but their weights will be decreased according to their estimated shared accuracy (S_{acc}). The intuition behind PA is: "if i and j are accurate when their decisions match, then when they disagree it might be a mistake, conversely, if they are not accurate together, them their individual predictions must be taken into account". We note that this formulation does not take into account concept drifts directly, although since all pairs of classifiers

are considered, classifiers committed to previous concepts might be obfuscated when their votes are combined with classifiers trained only in the latest instances.

When and how classifiers are added to the ensemble is beyond the scope of the voting method, but PA was conceived to work with either periodically additions/removals of classifiers (SAE, SAE2, SFNC, OUAE) or drift detection based classifier resets (Leveraging bagging, ADWIN bagging). In comparison to existing weighting methods, PA resembles OAUE and SAE2 weighting mechanism as its formulation normalizes weights based on the number of instances that a classifier has been trained on. The main difference between existing voting methods and PA, is that the later combines classifiers pairwise, which may reduce the impact of classifiers with low accuracy on the overall decision.

4.5.2 Pairwise Patterns

Similarly to PA, Pairwise Patterns (PP) voting scheme uses all possible pairs of classifiers $\binom{|C|}{2}$, but instead of estimating shared accuracy, PP records predictions patterns, during training, and use these patterns to weight decisions while predicting the label of an unknown instance x. To achieve that, PP uses a vector $\vec{p}$ with all possible $d = k^2$ prediction patterns given two classifiers and k classes, and a matrix $M_{d,k}$ which is updated during training and used to determine which labels corresponds to each pattern. $M_{d,k}$ has one column for each possible label and one line for each pattern. During training, classifiers c_i and c_j predict the label of an instance x independently, their predictions are then combined into a pattern which is used to find the corresponding line index in $M_{d,k}$, while the correct label y determines the column index that must be incremented in $M_{d,k}$. The overall ensemble prediction for an unknown instance x is the label that receives more votes based on the observed patterns from all pairs of classifiers for instance x. Figure 4.6 presents an example of how $M_{d,k}$ and $\vec{p}$ are related for a given pair. It is important to notice that the whole ensemble just needs one vector $\vec{p}$, but every pair of classifiers must have a distinct matrix $M_{d,k}$. Using the example from Figure 4.6, while predicting the label of an unknown instance x, if $h_i(x) = 0$ and $h_j(x) = 1$, then pair p_{c_i,c_j} combined prediction weight is going to be 3 for label 0, 16 for label 1, and so forth.

The intuition behind PP is: "If i and j decisions are A and B, it might be the case that the correct class label is C". For instance, consider a classifier i that always predict A for instances with real class label C, and a classifier j that always predict class B for instances with real class label C. When i decides for A and j for B this individual outputs can be corrected as C through PP voting method. PP is similar to PA as it is able to effectively work with different ensemble learning strategies, i.e., periodical resets or drift

$\vec{p}$	Corr. 0	Corr. 1	...	Corr. (k-1)
(0,0)	12	4	...	0
(0,1)	3	16	...	1
⋮	⋮	...	...	...
(k-1,k-1)	3	5	...	18

Figure 4.6: Example of the data stored for a given pair of classifiers c_i and c_j for a classification problem with k classes. Every entry in $\vec{p}$ has a one-to-one relation to a line in M.

detection resets. The main difference between PP and PA is that PP ignores individual predictions completely. Only combined predictions are considered. Theoretically, the individual accuracy does not matter in PP as long as classifiers are consistent with their decisions, i.e., classifier i is still useful if it consistently incorrectly predict class values. This characteristic brings PP fairly close to error-correcting output codes (ECOC) for multiclass problems (DIETTERICH; BAKIRI, 1995), even though PP is designed to combine multiple classifiers decisions and not decompose multiclass problems into binary problems.

4.5.3 Ensemble Adaptations for Pairwise methods

To make it possible to test PA and PP, we use an ensemble structure based on existing methods. We denote this ensemble by Generic Ensemble (GE) and briefly describe it. GE updates its component classifiers periodically based on a predefined window, such that the worst or oldest classifier is replaced by a background learner (also known as candidate (BRZEZINSKI; STEFANOWSKI, 2014; GOMES; ENEMBRECK, 2014)) trained during the last window. The default method and the adaptation for PA replaces the worst classifier, according to Equation 4.3, while the adaptation for PP replaces the oldest classifier, since the latter does not assess classifiers individually. During the first window GE only has one active classifier and a background classifier. In the second window the background classifier becomes active and a new background classifier is created. This process continues until the maximum number of classifiers has been reached (user threshold), and after that new classifiers can only replace existing classifiers. There are mainly two reasons to perform this process. First, classifiers are trained on different chunks of data and this assists in creating a diverse set. Second, the background classifier is adapted to the latest concept, and therefore it can gradually "adapt" the ensemble if concept drifts occurs. The background classifier do not take part of voting during predictions, since it could potentially degrade the overall performance as it may have a "weaker" hypothesis.

The default voting strategy of GE is based on the combination of the individual classifiers weight, such that weights are assigned according to Equation 4.3. Besides using GE to test PP, we have also adapted the Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010) algorithm to use PP for voting. The changes made to Leveraging Bagging includes the addition of a vector of patterns $\vec{p}$ to the ensemble, and for each pair of classifiers a matrix M. In order to account for the fact that Leveraging Bagging uses ADWIN to detect concept drifts, whenever a drift is detected, and a classifier c is reset, all matrices M corresponding to classifier c are reset as well. As a consequence, values in different matrices M may vary in scale. To adjust for that, each pair vote is weighted according to the total amount of instances that both classifiers have predicted together. We have not adapted Leveraging Bagging to work with PA, since PA weight functions depends on a fixed window size.

The empirical analysis of PA and PP are presented in Section 5.3.

4.6 The Complex Network Ensemble Classifier

In this section we present a novel network-based ensemble aiming at fully accomplishing objective 4 (see Section 1.2) of this thesis, namely the Complex Network Ensemble (CNE). We define CNE in terms of our general model presented in Section 4.2, thus we explain it in terms of the relation R, the combination β used to induce a network structure from the set of connections according to R, and the adaptation function f_ψ.

Whenever possible we tried to address SAE2 shortcomings in CNE. SAE2 uses a fixed period length parameter to define when to perform multiple updates to the network, including: removing/adding models and resetting estimations, etc. Setting this parameter correctly is crucial to obtain good classification performance as smaller values will lead to many updates while larger values may jeopardize the ability to adapt to drifts. Besides that, SAE2 demands the specification of other fixed parameters, for example, to define when to remove models based on their estimated accuracy (min_{acc}) or redundancy (Sc_{max}). In overall, SAE2 heavily depends upon the following hyperparameters:

- l period length;
- Sc_{max} maximum similarity coefficient;
- Sc_{min} minimum similarity coefficient;
- min_{acc} minimum classifier accuracy;
- max_e maximum number of classifiers.

The main problem with these parameters is that most of them must ideally be tuned for each stream. For example, min_{acc} is set to 0.7 in our experiments (see Section 5.2), however assuming that for a given data set there is no single base model able to achieve this accuracy the ensemble will constantly reset models. The same argument can be used to define the maximum (Sc_{max}) and minimum (Sc_{min}) similarity coefficients, which control when to remove redundant classifiers and when to activate connections, respectively.

In CNE, we have decided to remove the l parameter and use an active drift detection strategy. The main benefit is that it is not necessary anymore to fine tune the period length, yet it increases the complexity of the algorithm as effectively there are multiple adaptive windows (one for each base model). In SAE2 the relation R was defined as the similarity coefficient (Sc) between a pair of classifiers, connections were activated if they surpassed Sc_{min} and a network was induced based on the maximal cliques. In SAE2 predictions were obtained by combining the weighted votes (based on current period accuracy) first at the subnetwork level and latter at the network level, which contributed to an indirect drift adaptation technique as recently added classifiers, probably better adapted to the current concept, tend to receive higher weights. In CNE we define the relation R as either the Kappa statistic between pairs of base models outputs or their Jaccard similarity of the features used to induce the model, and the network is build using a k nearest strategy as presented in (SILVA; ZHAO, 2016). CNE uses an adaptation strategy based on one drift/warning detector per base model, training background learners whenever warnings are detected, and weighting votes based on accuracy calculated on adaptive windows. Also, CNE uses two diversity inducing techniques: vertical (similar to Leveraging Bagging) and horizontal (random subspaces). The rest of this section discusses each aspect of CNE in details.

4.6.1 CNE Relation R

We have tried two different similarity weighting functions to define the relation R, namely Kappa statistic κ and Jaccard Index. The reason for using Kappa instead of Similarity Coefficient is that Kappa account for agreements that might happen by chance, while also precisely measuring divergence votes on multiclass problems (See Section 3.2.2 for a thorough discussion on Kappa).

Jaccard Index is used to estimate the similarity between finite sample sets (LEVANDOWSKY; WINTER, 1971), and is defined as the size of the intersection divided by the size of the union of the sample sets as shown in Equation 4.8, where A and B represents subsets of

features used to induce models a and b. The intuition behind using Jaccard to measure the similarity among base models is that models induced using approximately the same features might as well generate very similar models even if bagging is used. There are other set distance metrics that could be used instead of Jaccard, such as Sorensen-Dice index (DALIRSEFAT; MEYER; MIRHOSEINI, 2009), however our problem matches the ideal scenario for applying Jaccard, i.e., it is defined in terms of a binary set membership and element identity (features either belong to the subset or not), and two features are either completely equal or not at all.

$$J(A, B) = \frac{A \cap B}{A \cup B} \tag{4.8}$$

There are three important factors to take into account when comparing Kappa and Jaccard for measuring similarity in CNE:

1. Input data to estimate similarity: Kappa is calculated on the output predictions, while Jaccard is calculated on the feature subset;

2. Domain: Kappa ranges from -1 (Inverse dependency) to 1 (Dependency), such that 0 represents independency. Jaccard ranges from 0 (No features are shared between models) to 1 (Exactly the same subset or one subset is a superset of the other[1]).

3. Update frequency: To maintain an updated estimation Kappa must be recalculate after every new instance is used for training, while Jaccard is updated only when the subspaces are defined for each model and when subspaces are reset.

In overall, Kappa provides a more accurate similarity estimation as it is based on the actual outputs. For example, it may happen that a completely different subset of features is used to induce two models, yet the features that compose these subsets may be correlated, thus both models will output very similar predictions. The main concern about using Kappa is that CNE does not use a fixed update period length to control network updates, thus it is necessary to recalculate Kappa after every new instance is used for training, which requires a lot of computational resources. Optionally, we could have defined a grace period after which Kappa would be recalculate and the network rebuild, however we would then be tied to a similar parameter as the period length l in SAE2.

[1]In CNE the number of subspaces is fixed, thus it is not possible that the number of features diverges from one classifier to another.

4.6.2 CNE Combination β

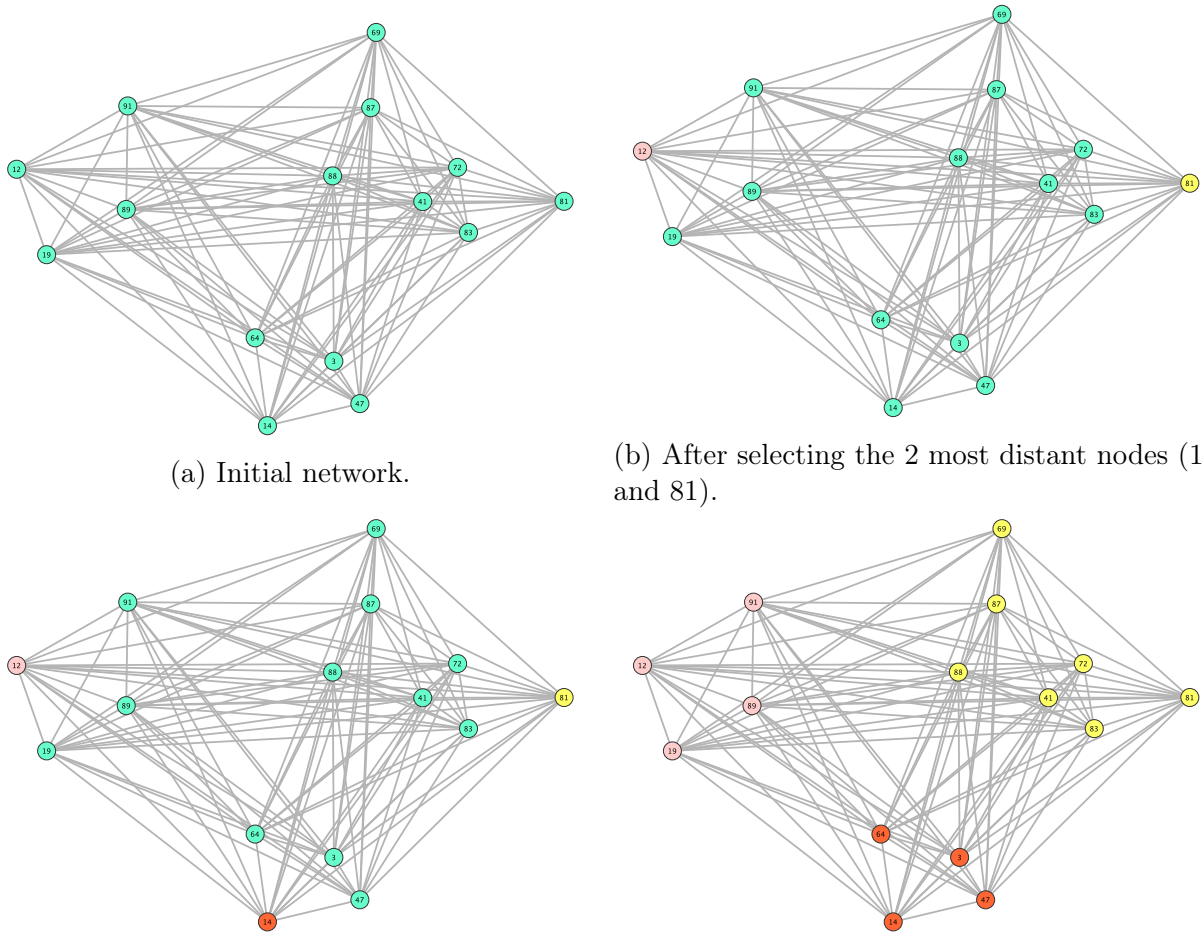
Beyond defining relations, it is also necessary to specify how they will be explored by the ensemble. In this case, how the structure induced by them will be used to boost predictions. In our formal framework this is equivalent to defining the combination method β. In SAE2, classifiers were combined based on dichotomous connections created based on the Sc_{min} parameter. The goal was to first decide within a set of highly similar classifiers a class label, and then use this decision in a secondary level in which all subsets of classifiers decisions were combined to form the overall ensemble decision.

In CNE we use a similar voting strategy, i.e., first combine votes within subnetworks and then combine the subnetworks votes to obtain the overall prediction. Precisely, when an unknown instance x is to be classified each component model will yield one vote weighted by its current estimate accuracy. These individual votes are then combined into an overall subnetwork vote, which is weighted by the average accuracy estimation of its members. This final vote per subnetwork is used to define the overall decision.

The network structure is created based on a variation of the k nearest neighbors network construction technique as proposed in (SILVA; ZHAO, 2016). This method must not be confused with the classical k nearest neighbor learner. In (SILVA; ZHAO, 2016) authors present a deterministic approach to construct a network given an arbitrary distance function. Basically, once set a reference vertex, the remaining non-reference vertices are ordered according to the their distance to it. Then, the reference vertex creates a connection with the top k vertices, i.e., closest, from the ordered list.

The base algorithm (SILVA; ZHAO, 2016) does not specify how the reference vertices are selected. Thus, we have changed it to accommodate a more intuitive network construction approach given our problem. First, we define the k reference vertices, which we name as seed models/nodes, to maximize the overall distance among them. Intuitively, our goal is to create subnetworks as diverse as possible from one another. To do that, we maximize the dissimilarity among seed nodes in an iterative process: first we select the 2 most distant nodes, then the node that is most distant from the previously selected 2, and continue until k is reached. For example, assuming $k = 3$ and that nodes are arranged in Figure 4.7 with distance corresponding to their Kappa (or Jaccard) measure, the nodes selected as seeds would be first 14 and 81, and then 12.

Finally, there are a multitude of algorithms for finding subgroups on networks (BOCCALETTI et al., 2006). In SAE (GOMES; ENEMBRECK, 2013) we have used weakly connected components, while in SAE2 we used maximal cliques. In CNE we have decided to use a method that was already used in the context of complex networks for



(a) Initial network.

(b) After selecting the 2 most distant nodes (12 and 81).

(c) After selecting the node (14) most distant from the previously selected.

(d) Assigning remaining nodes to the nearest seed node.

Figure 4.7: CNE network formation example.

machine learning, though not exactly for the same purpose as in (SILVA; ZHAO, 2016) authors show how to use the k-nearest neighbors algorithm to transform vector-based data into networks.

This network formation strategy still depends on an hyperparameter (k). However, it is an improvement over SAE2 Sc_{min} parameter as it is independent of the connections weight scale. For example, assuming each connection in the example from Figure 4.7 were 25% "closer", the resulting subnetworks would be the same.

4.6.3 CNE Adaptation f_ψ

Following our definition of a network-based ensemble, we have to define the adaptation function f_ψ, responsible for matters such as: how training takes place and when/how the ensemble structure is updated. As explained in Section 4.2, the general definition of a network-based ensemble does not explicitly defines the training method used. Although

the definition specifies that component classifiers must be diverse, thus for implementations this must be taken into account. In SAE2, diversity is induced into the base models by training using Online Bagging (OZA, 2005) and by removing/adding new models.

In CNE we decided to simulate Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010) instead of Online Bagging. Both methods train each model using a randomly selected subset of instances. As discussed in Section 3.5.4 and 3.5.8 this is simulated in an online setting by using a poisson distribution. The Online Bagging algorithm uses $Poisson(\lambda = 1)$, which means that around 37% of the values output by the Poisson distribution are 0, another 37% are 1, and 26% are greater than 1. This implies that by using Poison(1) 37% of the instances are not used for training (value 0), 37% are used once (value 1), and 26% are trained with repetition (values greater than 1). Leveraging Bagging uses $Poisson(\lambda = 6)$, which implies that 0.25% of values are 0, 45% are lower than 6, 16% are 6, and 39% are greater than 6. Effectively, base models are trained using more instances in Leveraging Bagging, still more resources are used in comparison to Online Bagging.

Besides training classifiers on different subsets of instances, in CNE we also train them on different *subsets of features*. This strategy is known as the Random Subspace Method (RSM). As discussed in Section 3.2.1 there are $2^M - 1$ different non-empty subsets of features, which makes it not practical (sometimes impossible) to try every possible combination. However, it is possible to achieve good classification performance in the aggregated ensemble even if only a subset of all possible combinations is explored. The main reason for using random subspaces to train CNE component classifiers is to enhance diversity among them. One secondary reason would be to allow processing high dimensionality data streams, however this is not exactly the case as CNE requires too many resources to execute because of the network updates.

As previously commented, instead of using a fixed window approach for updates, CNE uses *drift detectors*. Each component classifier is associated with one drift/warning detector. When detector d_j signals a drift warning, then a background learner $c_{bkg}(j)$, is initialized. When d_j outputs the drift signal, then $c_{bkg}(j)$ replaces the current classifier c_j. This approach is similar to that used in Leveraging Bagging (BIFET; HOLMES; PFAHRINGER, 2010), although instead of just resetting the classifier, we also start training a replacement beforehand, i.e., when a drift warning is detected. CNE is not bounded to a specific drift detection method, however for our actual implementation we used ADWIN (BIFET; GAVALDÀ, 2007), which demands a unique parameter that specify the drift confidence level δ. Thus we have two separate parameters, one for warning detection δ_w and another for drift detection δ_d.

Given our reset strategy based on drift detectors there is no fixed window to estimate accuracy or any other metric. Therefore, to *weight classifiers* we use the estimated accuracy for the given classifier window. This provides a good estimation of the classifier as long as it has seen enough instances, i.e., it will underestimate or superestimate its classification performance if the classifier has seen just a few instances. This is somewhat aided by using background learners as by the time a classifier is added to the ensemble it will already been trained on a few instances (hundreds or thousands) and will have a good estimation of its accuracy.

4.6.4 CNE Subspace resets

We learned through empirical experiments (see Section 5.5) that not resetting the subspace when drifts were detected was not a good approach for CNE. The intuition behind this is that some feature subsets might be irrelevant either temporally or throughout the whole stream, thus we present two approaches for resetting them whenever background learners are created. The first approach simply assign a new randomly generate subset of features whenever a new background learner is created. The second approach tries to find the best subset to be assigned by taking into account the current learners accuracy. Similar approaches have been tested in different works for batch learning, either to assign weights to component classifiers based on their subspace (LI; ZHAO, 2009) or (more similar to our approach) to assign weights or select feature subsets (JOHN et al., 1994; NGUYEN et al., 2015). Basically, our second approach assign each feature a probability of being selected to compose the subspace of a newly created background learner. For each feature, we sum over the estimated accuracy of all learners that are using it, then we normalize these weights by dividing each feature weight by the number of learners where they are used and end up with a vector w which is then used to selected the features for the new subspace.

4.6.5 CNE Overview

It is very difficult to achieve a parameter-free ensemble classifier. For example, we were not able to eliminate the parameter to limit the number of classifiers n and introduced a few, other parameters. The following list presents CNE parameters accompanied by their short descriptions.

- n defines the total amount of active based models that the ensemble will have at any time. In a stream learning context it is very important to limit the number

of classifiers, since memory and processing time are limited assets. n is different from max_c from SAE2, as CNE starts execution with n component classifiers and maintain it during the whole execution, while in SAE2 max_c defines the maximum number of component classifiers, thus at some point of the execution there might be less than max_c active classifiers.

- k defines the number of seed nodes to build the network. As discussed in Section 4.6.2 this parameter serves a similar purpose as that from Sc_{min} in SAE2, i.e., it guides the connections and subnetworks creation. However, k is easier to set as it is independent from the similarity metric used scale.

- m is the subspace size, which defines the percentage of features randomly assigned to be used for training each base model. Lower values adds more diversity into the ensemble as it lower the chances of the same component classifiers being assigned exactly the same subspaces. However, it can also be detrimental to performance as low subspace sizes for a high dimensional problem may incur that some important features are never selected.

- δ_d and δ_w representing the ADWIN drift and warning confidence levels. These parameters effectively replace SAE2 l period length, since they define individually the periodicity of when each base model is updated. Effectively, each base model has its own adaptive window

The pseudocode for CNE is presented in Algorithm 2. One important aspect of the algorithm depends upon the connection weighting function. If Kappa is used, then the algorithms executes as illustrates in Algorithm 2, i.e., after every instance $UpdateEdges(N)$ and $RegenerateSubnetworks(k, N)$ are executed as in lines 19 and 20. However, if Jaccard is used instead, then only if a $ResetSubspace(c, m)$ takes place it will be necessary to update edges and recreate the subnetworks, since Jaccard is calculated on the feature subset instead of learners outputs.

4.7 Final Considerations

In this section we have presented our efforts to address the overall and specific objectives (see Section 1.2). Albeit there is room for improvements in our solutions, for example, CNE presented in Section 4.6 still demands a lot of computational resources as is shown in Section 5.5.

Algorithm 2: CNE algorithm. **Input:** S: data stream that provides an instance (x_i, y_i) every t moments; n: total base models; m: subspace size; k: number of seed models; δ_d and δ_w: drift and warning detection confidence levels. **Local variables:** N: set of classifiers; R: set of subspaces. W: set of classifiers' weights; $P(\cdot)$: learning performance estimation function (e.g. accuracy); E: set of edge weights; BKG: set of background learners; u number of times a classifier will be trained on the same instance (given by $poisson(\lambda = 6)$); $\hat{x}$: input feature vector limited to a given subspace.

Require: $n \geq 1 \wedge \delta_w > \delta_d$

$R \leftarrow RandomSubspaces(m, n)$
$N \leftarrow CreateNetwork(k, n)$
while $HasNext(S)$ **do**
  $(x, y) \leftarrow next(S)$
  for all $c \in N$ **do**
    $\hat{x} \leftarrow PrepareSubspace(R, x, c)$
    $\hat{y} \leftarrow predict(c, \hat{x})$
    $W(c) \leftarrow P(W(c), \hat{y}, y)$
    $u \leftarrow poisson(\lambda = 6)$
    $Train(c, \hat{x}, y, u)$
    if $ADWIN(\delta_w, c, \hat{x}, y)$ **then**
      $BKG(c) \leftarrow CreateClassifier()$
      $ResetSubspace(c, m)$
    end if
    if $ADWIN(\delta_d, c, \hat{x}, y)$ **then**
      $c \leftarrow BKG(c)$
    end if
  end for
  $UpdateEdges(N)$
  $RegerateSubnetworks(k, N)$
end while

In the next section we present empirical results aiming at a deeper level of understanding of SAE2, PA, PP and CNE.

Chapter 5

Experiments

We segment the experiment sections according to the methods presented in sections 4.3, 4.5.1, 4.5.2 and 4.6. Assessing all methods concomitantly would be confusing. Therefore, we organize this section into four subsections in order to analyze SAE2, PA, PP and CNE separately. We added Section 5.4 to specifically evaluate the combination method used in SAE2, and also investigate the general idea of "combination" through relational data.

For all experiments, we present empirical results comparing ensemble classifiers in both real and synthetic datasets, with and without concept drifts. The datasets used are described in Section 2.4. The experimental protocol for each experiment might differ as they are aligned with their original publications. For an overall assessment of all the methods presented in this work against the literature state-of-the-art, the reader is directed to the last section 5.5.

5.1 Datasets configuration

The data used for the experiments presented on this chapter include six real datasets and nineten variations of synthetic data streams. The synthetic data are either evolving (abrupt, gradual and incremental drifts) or stationary (no drifts) streams. Table 5.1[1] presents which types of drifts were simulated for every synthetic data stream used. Synthetic data streams configurations generated 1 million instances, while real datasets have varying sizes. Table 5.2 presents a summary of the real datasets used.

We do not use all datasets in all experiments, for example, it was not feasible to

[1]Notice that SAE2, PA and PP have already been published during the writing of this work, to avoid "clash" of the acronyms used to refer to each dataset presented on each paper we have renamed some configurations here. Concretely, AGR1 and AGR2 in PA/PP (GOMES; BARDDAL; ENEMBRECK, 2015) are AGR3 and AGR4 here.

ID	Data generator	# Attributes	# Classes	Drifts
LED_a	LED	24	10	A/A/A
LED_g	LED	24	10	G/G/G
RTS	RTG	10	2	N
RTC	RTG	10	2	N
AGR1	AGRAWAL	9	2	A
AGR2	AGRAWAL	9	2	G
AGR3	AGRAWAL	9	2	A/A
AGR4	AGRAWAL	9	2	G/G
AGR_a	AGRAWAL	9	2	A/A/A
AGR_g	AGRAWAL	9	2	G/G/G
SEA1	SEA	3	2	A
SEA2	SEA	3	2	G
SEA3	SEA	3	2	A/A
SEA4	SEA	3	2	G/G
SEA_a	SEA	3	2	A/A/A
SEA_g	SEA	3	2	G/G/G
RBF_m	RBF	10	5	I_m
RBF_f	RBF	10	5	I_f
HYPE	Hyperplane	10	2	I_f

Table 5.1: Synthetic data streams configurations (A: Abrupt Drift, G: Gradual Drift, I: Incremental Drift (m: moderate, f:fast), N: None).

ID	# Instances	# Attributes	# Classes	MF Label	LF Label
Airlines	539,383	8	2	55.46%	44.54%
Electricity	45,312	8	2	57.55%	42.45%
Covertype	581,012	54	7	48.76%	0.47%
GMSC	150,000	11	2	93.32%	6.68%
KDD99	4,898,431	41	23	57.32%	0.00004%
Spam	9,324	39,917	2	74.4%	25.6%

Table 5.2: Real datasets. MF Label and LF Label stands for Most Frequent and Less Frequent class label, respectively.

use SPAM on the experiments listed on Section 5.4 simply because it is computationally impractical. Thus, in each of the following sections we specify which datasets were used for the current experiments.

All experiments were configured and executed on MOA (Massive On-line Analysis) framework (BIFET et al., 2011). To evaluate accuracy on all experiments we apply the Prequential (GAMA; RODRIGUES, 2009) evaluation procedure. The reason to use Prequential is because it gives accuracy estimates that approximates those of a holdout evaluation and allows using all instances for testing and training (see Section 2.3). For the final experiments in section 5.5, we also report Kappa M and Kappa Temporal. Experiments that measure processing time and memory used proceed as follows: Processing time is measured in seconds and is based on CPU time; and memory is measured in MB and based on RAM-hours (BIFET et al., 2010), e.g., one GB of memory deployed for one

hour corresponds to one RAM-Hour.

To account for random decisions in some methods, we repeat evaluations third times and average the results varying the random seeds. Thus, reported accuracy, time and memory cost are based on averages of third repetitions.

5.2 SAE2 Experiments

In this experimental evaluation we assess three aspects of data stream classification: average accuracy, processing time and memory.

In order to evaluate how SAE2 performs when compared to other ensemble classifiers we present results for DWM, ASHT Bagging, ADWIN Bagging, Leveraging Bagging and SAE. The parameters defined for these ensemble methods were those defined in their original papers. SAE and SAE2 parameters are set as follows: $Sc_{max} = 0.99$, $Sc_{min} = 0.90$, $min_{acc} = 0.70$, $max_e = 10$ (SAE2) and $Acc_{net.min} = 0.9$ (SAE). The period length was set to 10,000 for synthetic data streams for SAE and SAE2 to DWM. For real data sets, SAE, SAE2 and DWM had their period lengths set to approximately 1/100 of total instances. We present results for 3 versions of SAE2, varying its combination method, SAE2-c (maximal cliques), SAE2-w (weakly connected components) and SAE2-f ("free" combination). The last version, SAE2-f, considers every classifier as a subnetwork, i.e., it does not group classifiers based on their connections. All ensembles presented, besides ASHT Bagging, uses Hoeffding Naive Bayes Tree (HNBT) (HOLMES; KIRKBY; PFAHRINGER, 2005) as base learner. It is reasonable to use decision trees as base learners since they are unstable classifiers, i.e., given two slightly different sets of instances the decision trees induced from them will differ significantly. ASHT Bagging uses a variation of the Hoeffding Tree (DOMINGOS; HULTEN, 2000), named Adaptive Size Hoeffding Tree (ASHT), which imposes a limit on the tree height.

To perform statistical comparisons between SAE2-c and other classifiers we employ non-parametric tests (DEMŠAR, 2006). First, the Friedman test has been performed ($p = 0.05$) and differences were found among classifiers. Since we are performing with multiple comparisons among multiple datasets to one control classifier (SAE2-c), the Friedman test was followed by the post-hoc test of Bonferroni-Dunn. The Bonferroni-Dunn test indicates, with a confidence level of 95%, that for the given experiments SAE2-c $\succ$ {ASHT Bagging, Online Bagging, DWM, HFNB, SAE} and that there are no significant differences between SAE2-c and SAE2-f, SAE2-w, ADWIN Bagging or Leveraging Bagging.

Through analysis of Table 5.3, it is evident that different versions of SAE2 yields

	ADWIN Bag	ASHT Bag	Leveraging Bag	Online Bag	DWM	HNBT	SAE2-f	SAE2-c	SAE2-w	SAE
AGR1	93.79 ± 0.31	91.68 ± 0.54	93.94 ± 0.25	92.04 ± 0.29	78.98 ± 1.14	91.36 ± 0.55	94.21 ± 0.37	**94.3** ± 0.29	94.18 ± 0.33	86.5 ± 5.45
AGR2	94.89 ± 0.44	91.14 ± 0.76	94.16 ± 0.61	89.78 ± 0.61	85.28 ± 1.39	88.36 ± 1.15	95.23 ± 0.43	**95.25** ± 0.35	95.12 ± 0.51	87.26 ± 9.52
AGR3	88.84 ± 1.38	88.33 ± 0.96	**91.64** ± 0.32	88.47 ± 0.35	75.83 ± 1.21	86.44 ± 0.5	91.44 ± 0.75	91.28 ± 0.64	90.95 ± 0.73	75.48 ± 1.42
AGR4	91.72 ± 1.11	88.86 ± 0.72	**94.09** ± 0.42	85.58 ± 0.37	83.29 ± 1.27	82.76 ± 0.54	93.84 ± 0.97	93.54 ± 0.76	93.44 ± 0.81	68.25 ± 0.43
RTC	**76.58** ± 4.72	70.49 ± 4.49	54.27 ± 3.84	76.37 ± 4.93	67.49 ± 6.91	75.37 ± 5.34	74.34 ± 5.74	74.33 ± 5.72	74.32 ± 5.7	63.64 ± 8.11
RTS	**96.68** ± 1.74	89.73 ± 2.01	80.25 ± 10.78	**96.68** ± 1.74	92.32 ± 5.17	94.92 ± 2.66	95.8 ± 2.35	95.72 ± 2.3	95.66 ± 2.27	89.83 ± 11.41
SEA1	89.46 ± 0.57	89.28 ± 0.47	89.65 ± 0.49	89.03 ± 0.48	88.57 ± 0.48	88.57 ± 0.53	89.72 ± 0.45	89.72 ± 0.44	89.72 ± 0.45	**89.77** ± 0.45
SEA2	88.62 ± 0.61	88.68 ± 0.38	89.25 ± 0.35	86.23 ± 0.39	88.41 ± 0.38	85.69 ± 0.41	89.46 ± 0.36	**89.52** ± 0.34	89.48 ± 0.37	88.49 ± 0.35
SEA3	89.38 ± 0.37	88.94 ± 0.39	89.39 ± 0.37	88.7 ± 0.39	88.26 ± 0.42	88.25 ± 0.45	89.4 ± 0.35	89.4 ± 0.35	89.4 ± 0.35	**89.44** ± 0.38
SEA4	89.4 ± 0.34	88.66 ± 0.37	89.41 ± 0.32	86.39 ± 0.39	88.4 ± 0.34	85.83 ± 0.37	89.33 ± 0.34	**89.42** ± 0.3	89.36 ± 0.3	88.57 ± 0.35
Syn. Avg	89.94	87.58	86.61	87.93	83.68	86.76	**90.28**	90.25	90.16	82.72
Syn. Avg Rank	4.23	6.64	5	6.23	8.68	8.05	2.73	**2.45**	3.64	7.36
AIRL	65.82	**68.02**	62.97	64.11	64.13	64.45	61.09	61.09	61.09	62.56
COVT	85.31	86.29	88.88	84.34	**91.28**	82.91	84.69	87.09	86.36	84.77
ELEC	84.7	84.54	**89.31**	84.76	84.69	81.17	85.44	85.35	84.92	85.44
SPAM	88.27	81.61	**92.85**	81.58	88.21	78.84	86.86	87.83	87.76	86.53
Real Avg	81.03	80.12	**83.5**	78.7	82.08	76.84	79.52	80.34	80.03	79.83
Real Avg Rank	4.25	5.75	**2.5**	7.25	4	8.25	6.38	5	5.75	5.88
Total Avg	87.39	85.45	85.72	85.29	83.22	83.92	87.2	**87.42**	87.27	81.9
Total Avg Rank	4.25	6.43	4.07	6.61	7.29	8.21	3.89	**3.21**	4.29	6.75

Table 5.3: Comparison of average accuracy. Best accuracies per data stream are indicated in boldface. Bottom line indicates average ranking for each classifier.

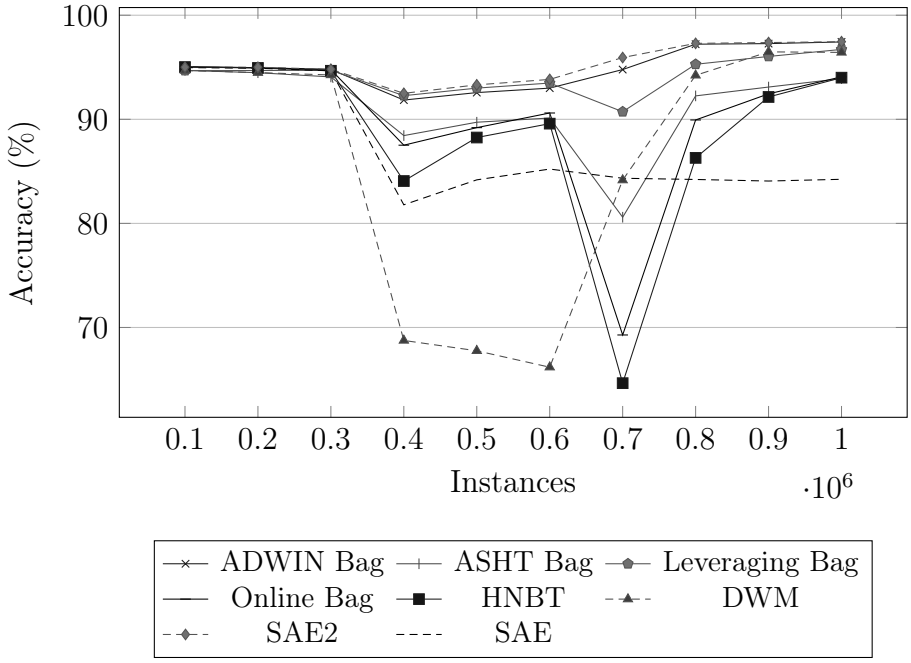


Figure 5.1: Accuracy on the AGRAWAL experiment AGR3 (2 abrupt drifts).

similar results with respect to overall classification accuracy. Among SAE2 versions compared, SAE2-c obtains a slightly better performance on average. Even though, the Bonferroni-Dunn test indicates that there are no significant differences between SAE2-c and Leveraging Bagging or ADWIN Bagging, if we turn our attention to Table 5.4, it is noticeable that SAE2-c demands less processing time and memory than these two algorithms. For example, SAE2-c model cost is around 2.19 MB/Hour for RTC experiment, while ADWIN Bagging and Leveraging Bagging model costs are 28.67 MB/Hour and 49.5 MB/Hour, respectively. We also note that SAE2 does not outperform its base learner (HNBT) in non-evolving data streams, such as RTS and RTC. This characteristic is shared by other ensemble methods, such as Leveraging Bagging, ASHT Bagging and

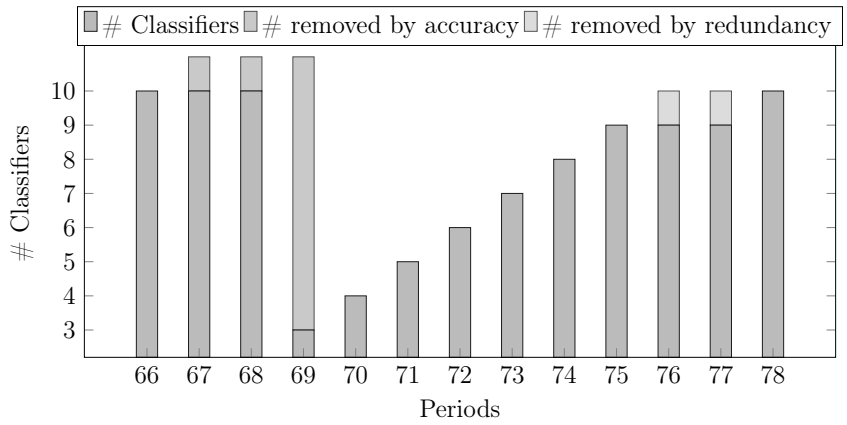


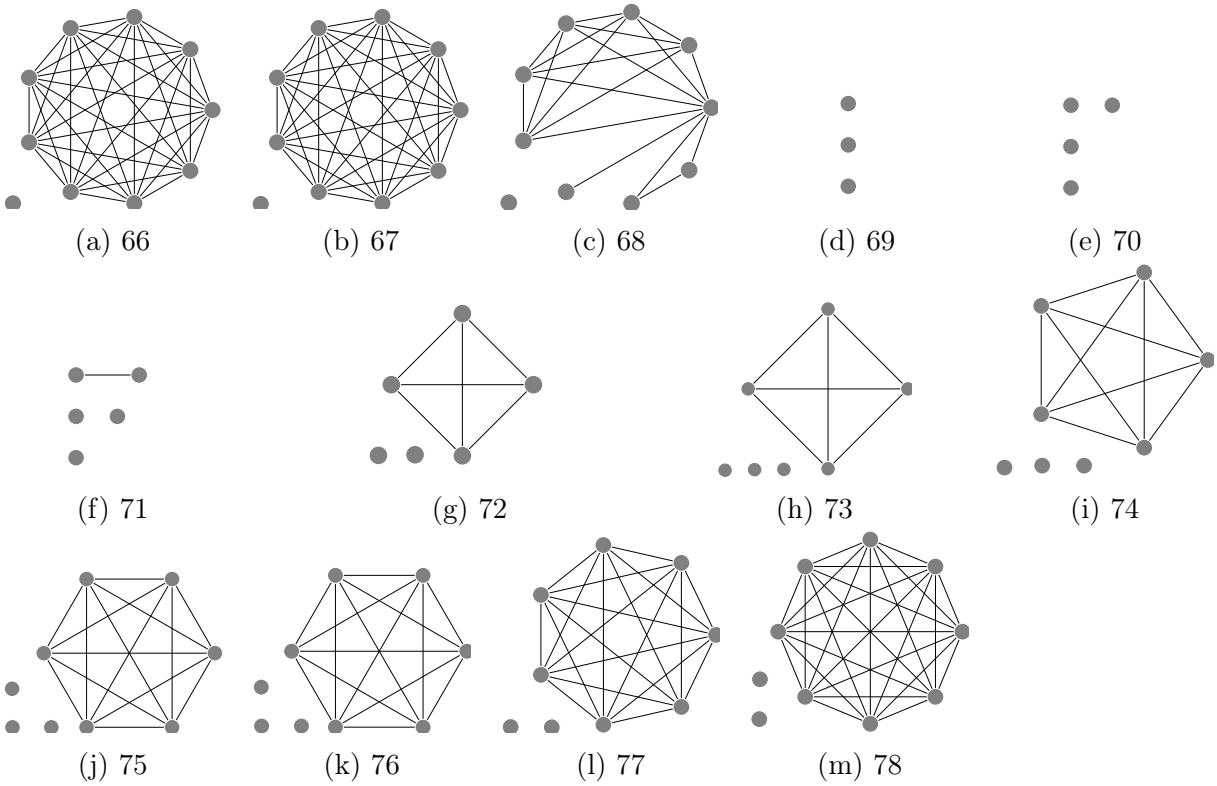
Figure 5.2: Classifier counter for SAE2-c in AGR3 (period 66 through 78).

	ADWIN Bag		**Leveraging Bag**		**SAE2**	
	Time	Mem.	Time	Mem.	Time	Mem.
AGR1	55.47	0.16	189.35	5.01	42.24	0.08
AGR2	59.76	0.21	212.91	6.28	40.44	0.05
AGR3	59.27	0.15	206.59	5.78	38.57	0.06
AGR4	56.99	0.11	203.33	3.71	36.94	0.03
RTC	528.08	28.67	1130.75	49.5	129.33	2.19
RTS	77.95	0.47	335.64	16.85	40.82	0.16
SEA1	37.66	0.03	123.58	1.55	23	< 0.01
SEA2	36.01	0.03	112.85	1.07	22.96	< 0.01
SEA3	36.4	0.02	118.14	1.28	22.95	< 0.01
SEA4	34.39	0.01	102.13	0.66	23	< 0.01
AIRL	180.02	1.97	480.15	4.41	37.15	0.02
COVT	146.42	0.01	210.95	0.1	157.43	0.01
ELEC	4.53	< 0.01	9.1	< 0.01	3.74	< 0.01
SPAM	1405.46	76.68	3429.94	922.67	772.14	22.13

Table 5.4: Comparison of time and memory of SAE2-c, Leveraging Bagging and ADWIN Bagging. Time is measured in seconds and memory in megabytes per hour.

DWM. SAE2 versions inefficiency in non-evolving streams can be attributed to its period length parameter. If we augment its period length, it may perform better in non-evolving streams, but it will perform poorly in evolving data streams.

Besides evaluating average accuracy and resources used (time and memory), it is also interesting to evaluate accuracy before, during and after a concept drift happens and the state of the network during these periods. Figures 5.1, 5.2 and 5.3, presents classifiers accuracy every 100 thousand instances, SAE2-c network size at every 10 thousand instances and SAE2-c network snapshots around a concept drift, respectively. Experiment AGR3 has an abrupt concept drift at instance 666,666, i.e., end of period 67. In the of period68 (Figure 5.3c), 7 classifiers are removed due to low accuracy, which in turn is caused by the drift at period 67 (Figure 5.3b). After the new concept becomes stable, classifiers start being removed due to redundancy, e.g., period 76 (Figure 5.3k).



(a) 66 (b) 67 (c) 68 (d) 69 (e) 70

(f) 71 (g) 72 (h) 73 (i) 74

(j) 75 (k) 76 (l) 77 (m) 78

Figure 5.3: SAE2-c Network state for AGR2. Period 66 (instance 650000 to 660000) to 78 (instance 770000 to 780000)

5.3 Pairwise methods experiments

In this section, we present empirical results comparing ensemble classifiers with our ensemble adaptations for PA and PP (see Section 4.5.3). Our analysis indicates that pairwise voting is able to enhance overall performance for PP, especially on real datasets, and that PA is useful whenever there are great differences in accuracy estimates among ensemble members, which is common during concept drifts.

On these experiments, we assess only the accuracy of classifiers. Even though, other aspects such as processing time and memory are important for data stream evaluation, we do not present these here, as the specific goal of these experiments is to evaluate the methods accuracy and not resources usage. In overall, the overhead caused by PA and PP only marginally affects memory and processing time for either PA or PP, as long as the number of possible class labels is low. By "low" number of possible class labels we mean a number below ten class labels. The number of classes has no impact on PA, but on PP it can yield too many combinations and cause it to generate large pattern matrices.

We compare our modified ensemble classifiers with PA and PP to their default methods (without pairwise voting), and also to other ensemble classifiers. The reason to compare the modified ensembles against default implementations is to check whether (and

Dataset	**LevBag-PP**	**LevBag**
AGR3	93.64 ± 0.15	**93.69** ± 0.83
AGR4	90.95 ± 1.17	**91.07** ± 1.29
AIRL	**63.7** ± 0.28	62.67 ± 0.25
COVT	**92.95** ± 0.26	92.19 ± 0.28
ELEC	90.67 ± 0.24	**90.82** ± 0.21
RTS	97.99 ± 0.1	**98.21** ± 0.09
SEA3	**89.91** ± 0.04	89.64 ± 0.34
SEA4	**90.46** ± 0.03	90.45 ± 0.04
SPAM	**93.89** ± 0.55	93.11 ± 0.35
HYPE	**90.75** ± 0.11	90.29 ± 0.12

Table 5.5: Average accuracy for LevBag and LevBag-PP. The best accuracies per data stream are indicated in boldface.

Dataset	**GE-PA**	**GE-PP**	**GE**
AGR3	**94.2** ± 0.27	87.43	92.04 ± 0.08
AGR4	**92.42** ± 0.41	82.7	90.87 ± 0.01
AIRL	**66.38** ± 0.15	62.67	66.21 ± 0.02
COVT	87.72 ± 0.43	**89.67**	88.31 ± 0.16
ELEC	**86.03** ± 0.17	84.76	85.97 ± 0.15
RTS	95.13 ± 0.08	**96.57**	95.19 ± 0.02
SEA3	89.32 ± 0.22	86.54	**89.33** ± 0
SEA4	89.41 ± 0.07	86.51	**89.43** ± 0
SPAM	87.19 ± 0.06	**88.76**	87.1 ± 0.02
HYPE	**91.16** ± 0.06	86.42	91.15 ± 0.06

Table 5.6: Average accuracy for GE, GE-PA and GE-PP. The best accuracies per data stream are indicated in boldface. GE-PP standard deviation was below 10^{-9} for all experiments.

when) these pairwise voting strategies are able to enhance the overall ensemble accuracy.

Tables 5.5 and 5.6 presents the results for the adapted methods (LevBag-PP, GE-PA and GE-PP) with their default methods (LevBag and GE). LevBag-PP was able to boost LevBag accuracy most notably in 3 real datasets (SPAM, AIRL and COVT). We observed that the best results for LevBag-PP were achieved when classifiers with low individual accuracy for a given class, when combined translated their prediction pattern into the most often correct label. For example, classifiers c_i and c_j predict label 1 for an unknown instance x, but it is more likely that the correct label is 0 when both decide for 1.

Through analysis of Table 5.6 it is noticeable that GE-PA was able to achieve higher or comparable accuracy in most datasets, while GE-PP was more unstable. In order to understand in which situations PA voting surpassed that of GE we analyzed in details experiments AGR3 and AGR4. Figures 5.4 and 5.5 shows that GE-PA was able to

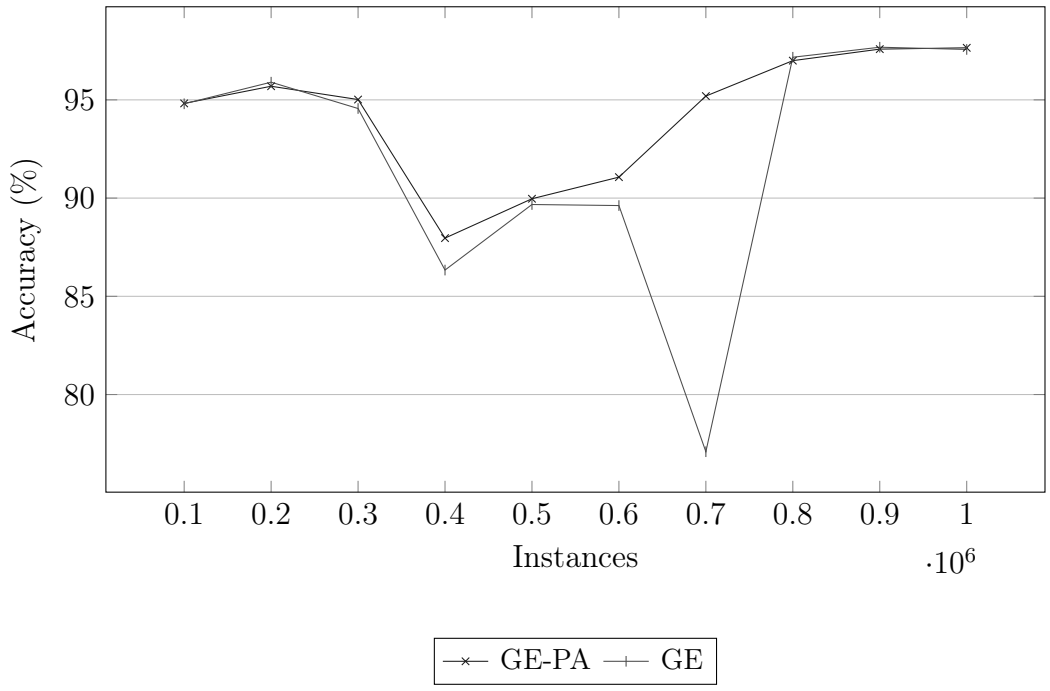


Figure 5.4: Accuracy on the AGR3 experiment.

adapt itself to the second concept drift (around instance 666,666) faster than GE. GE-PA performance reported at 0.7×10^6 (see Figure 5.5) cannot be credited solely to background classifiers replacing older classifiers, as this same mechanism took place on GE during the same period.

By analyzing the period comprehended between instances $666,000$ and $700,000$ for AGR1, we observed a scenario, unusual during other periods, in which the amount of disagreement between pairs was high, and it was for the better. In this scenario, during prediction, classifiers with low individual weight (adapted to the previous concept) "transferred" their condition to other classifiers that predicted the same label as them, since their estimated shared accuracy (S_{acc}) were low as well, while classifiers with high individual weight (adapted to the current concept) were able to achieve significant weight in split decisions when combined with these low weight classifiers. Table 5.7 presents the average accuracy for LevBag-PP and GE-PA in comparison to other ensemble methods. To verify if there were statistically significant differences between algorithms, we performed non-parametric tests using the methodology from (DEMŠAR, 2006). First, we used the Friedman test with $\alpha = 0.05$ and the null hypothesis "there were no statistical difference between given algorithms" was rejected. We proceeded with the Nemenyi post-hoc test to identify these differences. The Nemenyi test indicates, with a confidence level of 95%, that for the given experiments {LevBag-PP, OAUE} $\succ$ {DWM}.

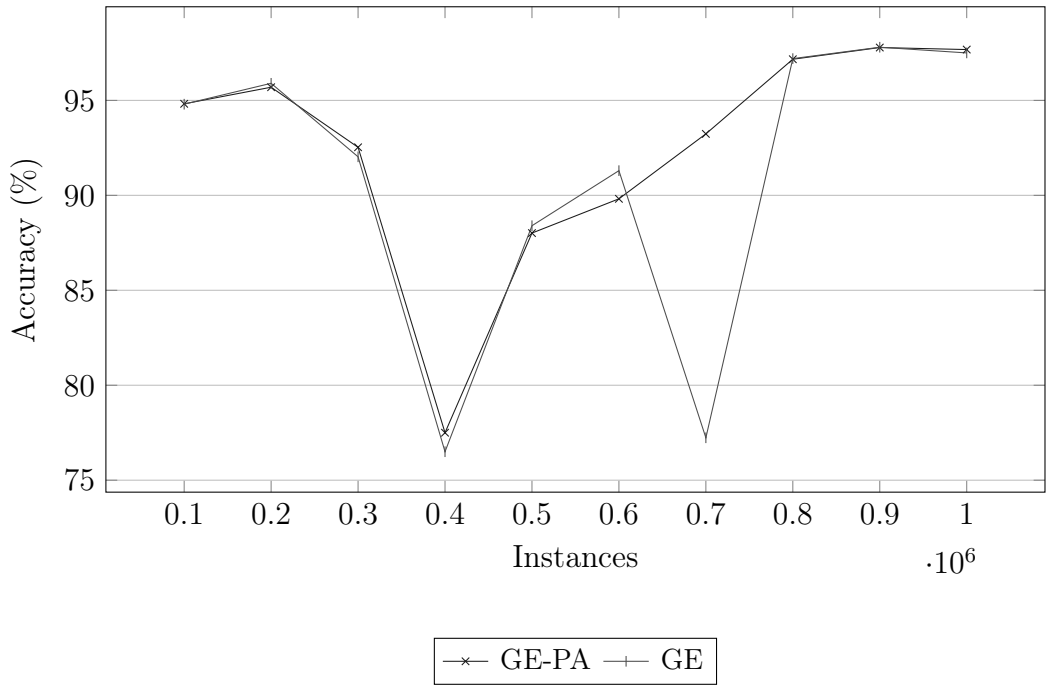


Figure 5.5: Accuracy on the AGR4 experiment.

5.4 Combination Methods in SAE2

The objective of the experiments presented on this section is to verify one fundamental constituent of our hypothesis: "Is it worthwhile to use a combination method based on a network of classifiers?" We use the SAE2 algorithm presented in Section 4.3 to illustrate our experiments.

Experiments were performed on almost all data sets presented in Section 5.1. The reason for using different data sets is because we want to observe characteristics that hold beyond one specific data set. We could not use all data sets because the processing time on SPAM and RTC were prohibitively high. The following invariants were obeyed throughout all the experiments in this section:

1. There are no classifiers removals based on accuracy or redundancy;
2. SAE2 period (window) size was set to 1/100 of the data stream size;
3. The default SAE2 combination method is maximal cliques, which was referenced as SAE2-c in Section 5.2;
4. All other SAE2 parameters were set as presented in Section 4.3.

To keep the ensemble, whenever possible, with all the classifiers we ignored the removal based on low accuracy or redundancy. The main goal of these experiments is to compare multiple ways of combining the set of classifiers, and if many removals happened

Dataset	**LevBag-PP**	**GE-PA**	**ADWBag**	**DWM**	**OAUE**	**SFNC**	**SAE2**
AGR1	93.64±0.15	94.2±0.27	94.36±0.2	86.5	93.77	93.33	**94.68**±0.15
AGR2	90.95±1.17	92.42±0.41	90.69±1.32	82.41	**93.24**	92.31	89.79±2.09
RTS	**97.99**±0.1	95.13±0.08	95.6±0.09	93.64	97.35	95.09	95.06±0.12
SEA1	89.91±0.04	89.32±0.22	88.63±0.48	88.6	**90.02**	89.54	89.86±0.17
SEA2	**90.46**±0.03	89.41±0.07	90.15±0.08	88.63	90.25	89.16	90.12±0.1
HYPE	90.75±0.11	**91.16**±0.06	90.5±0.12	88.2	90.41	90.91	90.88±0.12
Syn. Avg	92.28	91.94	91.66	88	**92.51**	91.72	91.73
Syn. Avg Rank	2.83	3.33	4	7	**2.67**	4.33	3.83
AIRL	63.7±0.28	66.38±0.15	66.05±0.32	61.46	64.48	**66.42**	60.8±0.58
COVT	92.95±0.26	87.72±0.43	85.67±0.25	91.28	**93.55**	85.85	86.59±0.53
ELEC	**90.67**±0.24	86.03±0.17	85.05±0.33	84.69	89.38	85.38	85.8±0.48
SPAM	**93.89**±0.55	87.19±0.06	88.34±0.9	88.21	67.23	86.5	87.76±0.42
Real Avg	**85.3**	81.83	81.28	81.41	78.66	81.04	80.24
Real Avg Rank	**2.25**	3.5	4.5	4.75	3.5	4.5	5
Total Avg	**89.49**	87.9	87.5	85.36	86.97	87.45	87.13
Total Avg Rank	**2.6**	3.4	4.2	6.1	3	4.4	4.3

Table 5.7: Comparison of average accuracy. The best accuracies per data stream are indicated in boldface. SFNC standard deviation were below 10^{-14} for all experiments.

then it would not be possible to achieve the expected scenario. After every period end, SAE2 rebuild the network of classifiers according to the relations observed during the last period. In the experiment, all possible network structures were generated, including the default SAE2 network build method (maximal cliques) and the "Free combination", i.e. no connections considered. The output of the experiment is the accuracy obtained in each period by the "Best combination", "Worst combination", SAE2 default method (maximal cliques – SAE2-c) and "Free combination" (SAE2-f).

We could not extend our experiments to generate all possible networks for ensembles with more than 6 classifiers. The time to generate these networks was prohibitive, since each experiments requires that all possible network structures be build 100 times (number of periods). Since the number of networks is given by $2^{(n*(n-1)/2)}$, where n is the number of classifiers, thus for $n = 6$ there are $32,768$ networks per period and for $n = 7$ there are $2,097,152$ networks per period. Table 5.8 shows the number of graphs that must be checked per period given the number of classifiers. To illustrate the difficulty in running experiments with 7 or more classifiers, it took fifteen days to finish the experiments for AIRL dataset with 6 classifiers.

Figures 5.6, 5.7 and 5.8 presents for ELEC dataset the comparison between the overall accuracy obtained over 100 SAE2 evaluation periods for each of the four networks of interested: Best, Worst, SAE2-c and SAE2-f. The first periods do not have a full network, as SAE2 adds one classifier per period. Thus, the first 4, 5 or 6 periods present very similar results for all different combinations.

The plots for the other datasets have a very similar behavior, thus they were moved

# **Classifiers**	# **Networks**
1	1
2	2
3	8
4	64
5	1024
6	32768
7	2097152
8	268435456
9	6.8719×10^{10}
10	3.5184×10^{13}

Table 5.8: The amount of possible networks given n classifiers

to Appendix B.

It is important to observe that the conclusions presented cannot be extrapolated to the general case for two reasons. First, we could not use more than 6 classifiers for testing. Second, SAE2 is an implementation of a network-based ensemble, thus it has its own bias and does not represent all possible network-based ensembles. Our conclusions about these experiments are presented below:

- The difference between Worst and Best shows that there are ways to combine classifiers that either negatively or positively affect the ensemble;
- The proximity between the results obtained by SAE2-c and SAE2-f, indicate that the current default combination method for SAE2, i.e. SAE2-c, does not improve significantly over SAE2-f. Although when we look at the overall accuracy using ten classifiers, SAE2-c outperforms SAE2-f (see Section 5.2);
- SAE2-c and SAE2-f are usually closer to Best than they are to Worst;
- The Best network was selected for each period, thus the topology of this network was different in every period for the same experiment. Unfortunately, it is difficult to observe a concise pattern from the best networks generated. We have tried to verify each Best network individually, but all we could learn was that almost all the times Best is not SAE2-f, i.e., the Best network almost never has an empty set of edges.

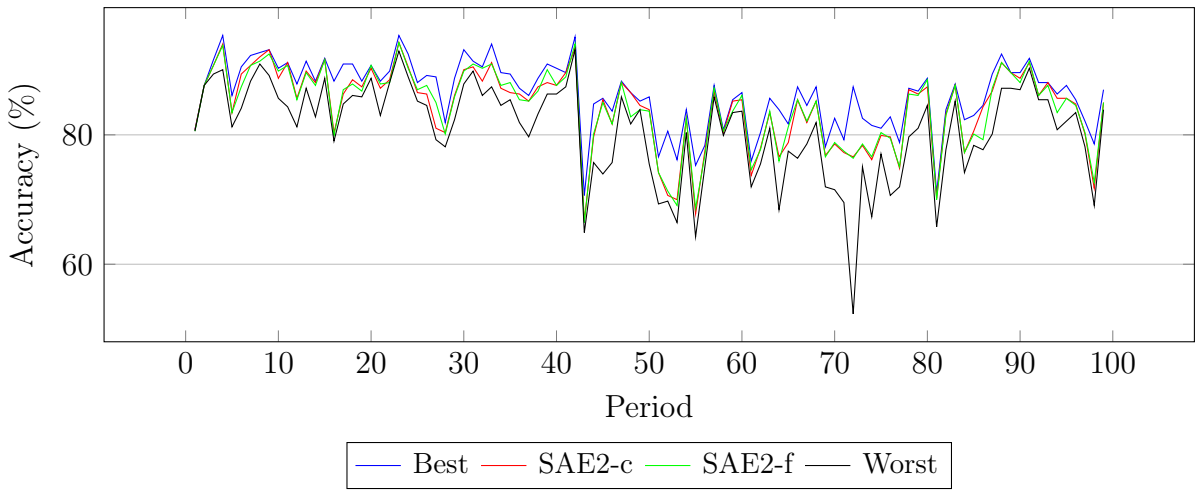


Figure 5.6: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset ELEC, max classifiers = 4

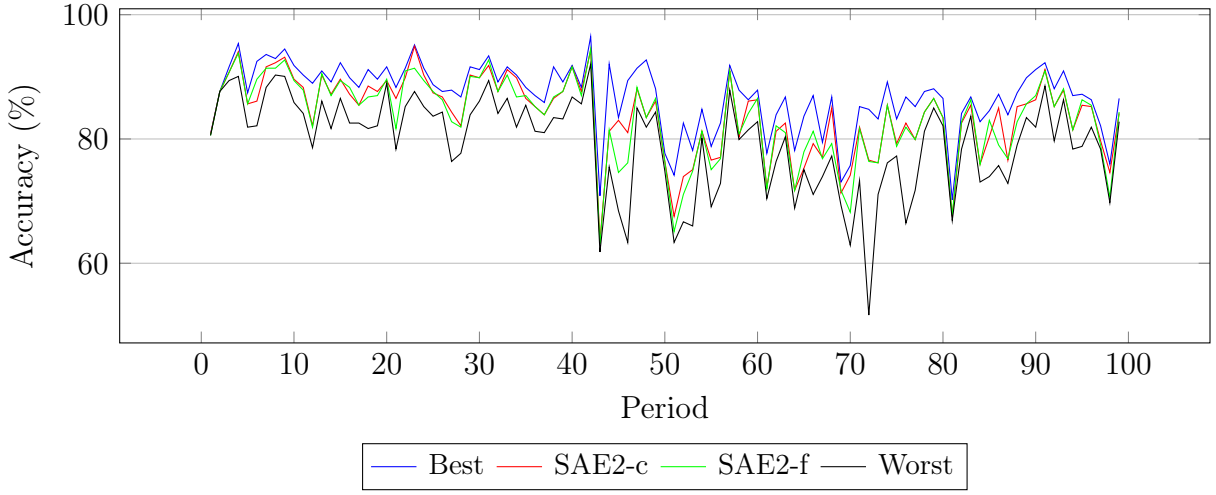


Figure 5.7: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset ELEC, max classifiers = 5

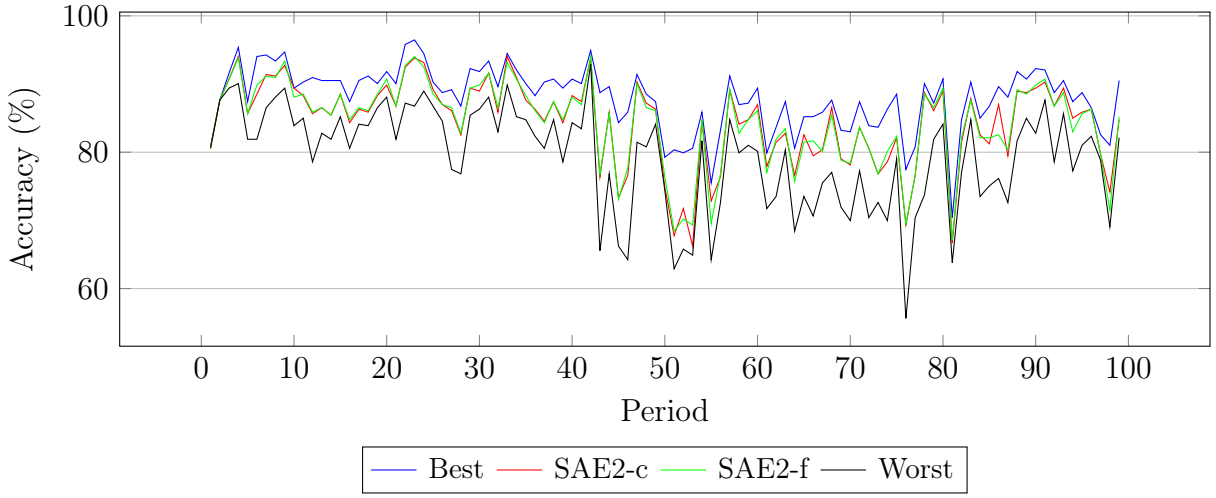


Figure 5.8: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset ELEC, max classifiers = 6

5.5 CNE Experiments

In this section we discuss the final experiments concerning the CNE algorithm. The experiments in this section includes challenging problems such as multi-class tasks or datasets that exhibits temporal dependences, multiple drifts or class imbalanced. To better assess temporally dependent problems and imbalanced datasets we use Kappa Temporal and Kappa M. In (BIFET et al., 2015) authors show that Kappa Statistic M measure has advantages over Kappa statistic as it has a zero value for a majority class classifier. For datasets that exhibit temporal dependences it is advisable to evaluate Kappa Temporal since it replaces majority class classifier with the NoChange classifier (ŽLIOBAITĖ et al., 2014).

All experiments in this section use 100 base models[2]; the base learner for all ensembles is the Hoeffding Naive Bayes Tree (HNBT) (HOLMES; KIRKBY; PFAHRINGER, 2005) (grace period = 50 and split confidence = 0.01); specific parameter values for methods other than CNE were set according to their original publications. Some datasets that were not used in previous experiments are included in this section to provide a broader benchmark, which includes abrupt, gradual and incremental drifts, configured on several different synthetic generators. The real world datasets GMSC and KDD99 were added here, both exhibit class label imbalance (see Table 5.2 for details). Given the number of base models it was unfeasible to include SPAM corpus (KATAKIS et al., 2009) dataset as part of this experimentation as it requires too much memory. This section does not include a thorough analysis of parameters such as the drift/warning confidence levels (All experiments uses $\delta_d = 0.00002$ and $\delta_w = 0.0002$) or the number of features per random subspace (All experiments uses $m = 85\%$). The reader is directed to Appendixes C and D for further analysis of these parameters.

5.5.1 Jaccard and Kappa networks

We start the experiments comparing the two connection weighting functions described in section 4.6, i.e. Jaccard and Kappa. Specifically, we present the results for $k = 5, 10, 20, 30$ using Jaccard or Kappa. As previously mentioned, k stands for the number of seed base models used to create subnetworks according to either Kappa or Jaccard measures. Table 5.9 presents the results for these experiments.

The results in Table 5.9 suggests that CNE_{kap30} is the most effective, yet the non-parametric Friedman test indicates that there are no differences among the methods.

[2]DWM is an exception as it does not include a maximum or target number of base models (see Section 3.5.12)

Table 5.9: Accuracy - CNE Jaccard and Kappa varying k. KDD99 did not finish for some variations of Kappa versions, thus KDD99 is not used for the average and ranking calculations.

Dataset	CNE_{kap5}	CNE_{kap10}	CNE_{kap20}	CNE_{kap30}	CNE_{jac5}	CNE_{jac10}	CNE_{jac20}	CNE_{jac30}
LED_a	73.77	73.8	**73.8**	73.78	73.73	73.74	73.76	73.76
LED_g	73.1	73.11	73.1	**73.12**	73.06	73.07	73.09	73.11
SEA_a	89.49	89.49	89.49	**89.49**	89.49	89.49	89.49	89.48
SEA_g	**89.08**	89.07	89.07	89.07	89.07	89.07	89.07	89.07
AGR_a	90.16	90.38	90.66	**90.85**	90.71	90.21	90.27	90.33
AGR_g	86.25	86.63	86.99	87.09	**87.24**	86.48	86.55	86.65
RTS	97.33	97.31	97.06	96.94	**97.39**	97.37	97.36	97.32
RBF_m	86.46	86.47	86.47	86.49	86.36	86.5	86.51	**86.52**
RBF_f	77.11	77.16	77.29	**77.34**	76.9	77.14	77.18	77.22
HYPER	85.08	85.05	85.03	85.08	85.18	85.24	85.26	**85.29**
Syn. Avg	84.78	84.85	84.9	**84.92**	84.91	84.83	84.85	84.87
Syn. Avg Rank	5.44	4.78	4.44	**3.11**	4.67	4.56	4.78	4.22
AIRL	64.94	65.04	**65.14**	65.14	65	64.87	64.92	64.96
ELEC	89.66	89.61	89.65	**89.75**	89.58	89.73	89.67	89.67
COVT	95.11	95.12	95.14	95.12	95.1	95.12	95.14	**95.15**
GMSC	93.55	93.55	93.55	93.55	**93.57**	93.55	93.54	93.55
KDD99	99.96	-	-	-	99.96	99.96	**99.96**	99.96
Real Avg	85.82	85.83	85.87	**85.89**	85.81	85.82	85.82	85.83
Real Avg Rank	5.63	4	3.63	**3**	5.25	5.5	5.13	3.88
Total Avg	85.95	86	86.05	**86.08**	86.05	85.99	86	86.02
Total Avg Rank	5.5	4.54	4.19	**3.08**	4.85	4.85	4.88	4.12

Figure 5.11 presents the average CPU Time and RAM-Hours for each variation in Table 5.9, where it is clear that Kappa does not scale well as it requires recomputing the subnetworks after every new instance. Jaccard variations vary only slightly in terms of resources used as the subnetworks are generated only once at the beginning of the execution after the random subspaces were selected. Thus we choose the simplest, yet accurate, CNE_{jac30} for further experimentation.

5.5.2 Subspace reset

Further analysis of the network state for CNE_{jac30} shows that using a fixed subspace and only resetting the model is inflexible as if a set of features ceases to be relevant this will not be reflected on the base models subspaces associated with it. This can be observed in Figures 5.12, 5.13 and 5.14 where:

- Seed nodes are depicted in the center of the image;
- Blue edges represent connections between seeds and illustrate how far apart are they (these connections are not active);
- Node fill color represent the model accuracy, varying from light red (worse accuracy) to light green (best accuracy);

Figure 5.9: CPU Time

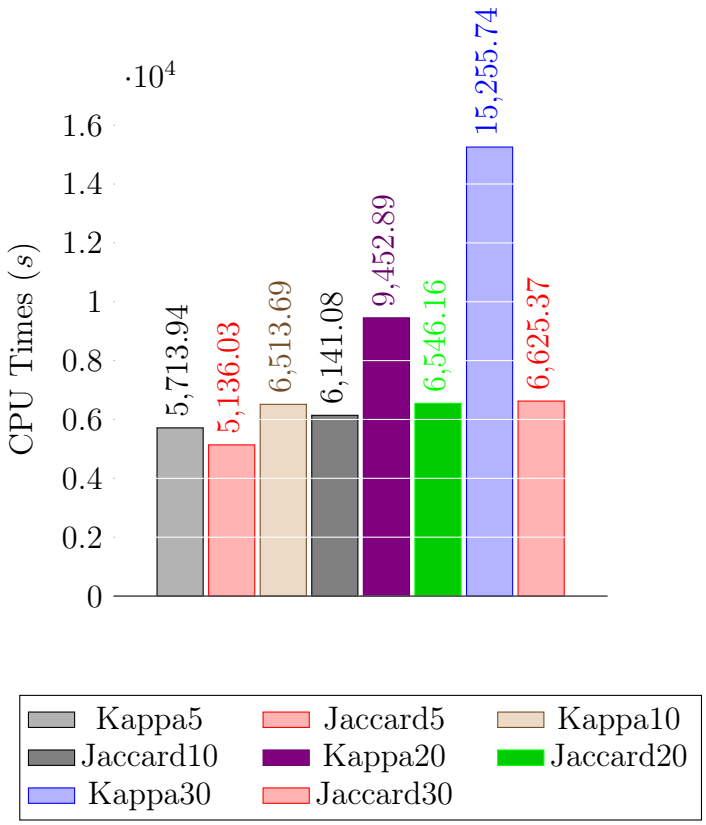


Figure 5.10: RAM-Hours

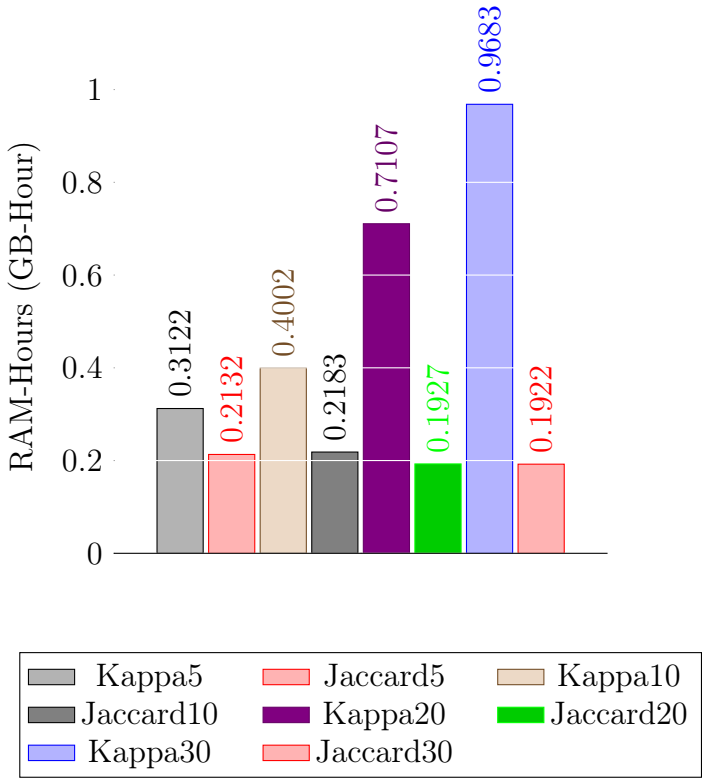
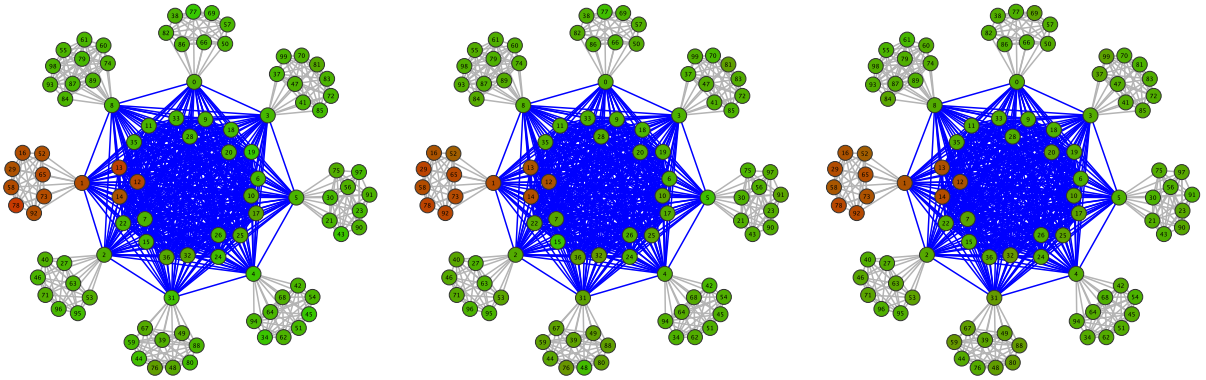
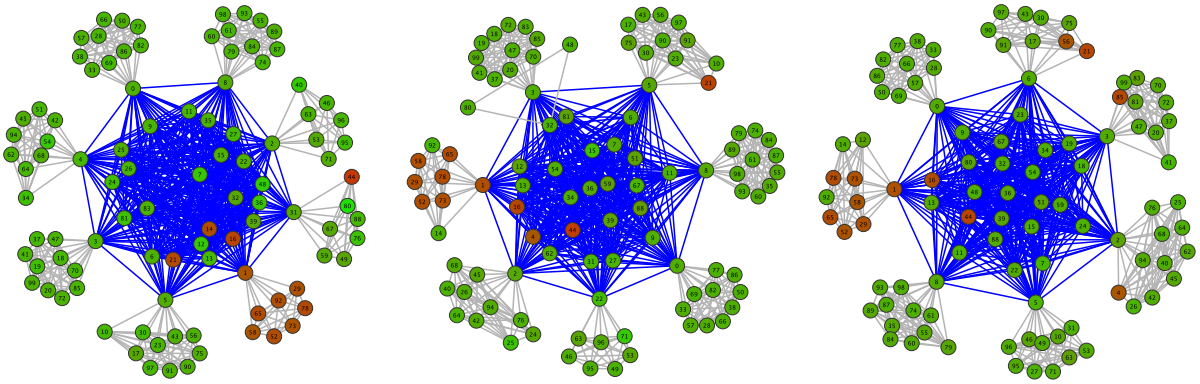


Figure 5.11: Comparison in terms of Average CPU Time and RAM-Hours (100 classifiers).



(a) after 4532 instances. (b) after 9064 instances. (c) after 13596 instances.

Figure 5.12: CNE$_{jac30}$ executing on ELEC without resetting subspaces.



(a) after 4532 instances. (b) after 9064 instances. (c) after 13596 instances.

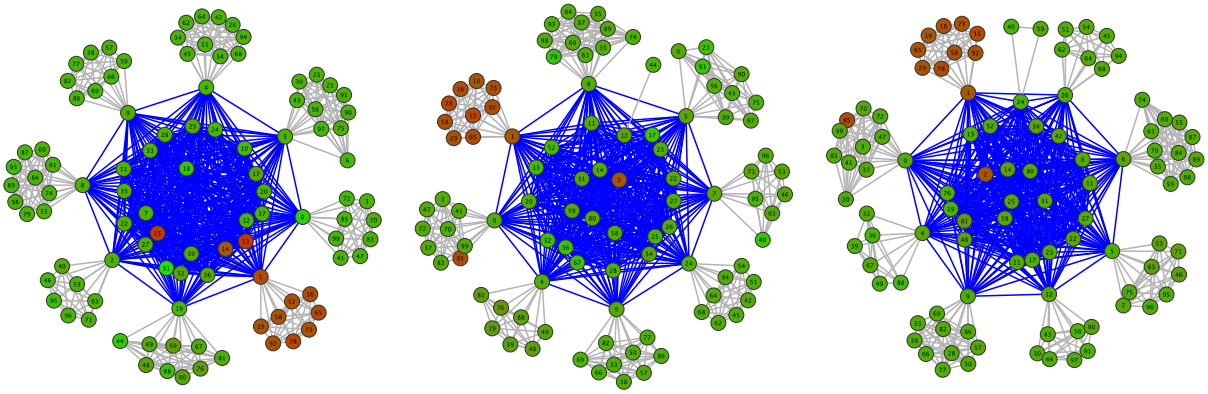
Figure 5.13: CNE$_{jac30}$ executing on ELEC using random subspace resets.

- The distance between nodes is the result of a stress minimization layout[3].

The intuition behind applying subspace resets is related to the fact that some subsets of features might be less useful to represent the underlying concept. For example, in Figure 5.12 the subnetwork on the left is committed to a subset of features which models trained on it are consistently worse than the rest throughout the execution of the stream. Based on this verifications we developed two strategies to reset the subspaces as described in Section 4.6, i.e. randomly resetting the subspace or reset it based on accuracy. Figures 5.13 and 5.14 depict the network state using random resets and accuracy resets, respectively. It can be observed that using these strategies avoid consistent bad subspaces of low accurate models as in the default strategy.

Figures 5.15, 5.16 and 5.17 report the networks for CNE$_{jac30}$, CNE$_{jac30}(rand)$ and CNE$_{jac30}(acc)$, respectively, for the LED$_g$ dataset (3 gradual drifts). These networks represent the period immediately before the first drift starts, during the drift window and after it. We observe that CNE$_{jac30}(rand)$ (Figure 5.16) and CNE$_{jac30}(acc)$ (Figure

[3]These network visualizations were created using the Visone software (BAUR et al., 2001)



(a) after 4532 instances. (b) after 9064 instances. (c) after 13596 instances.

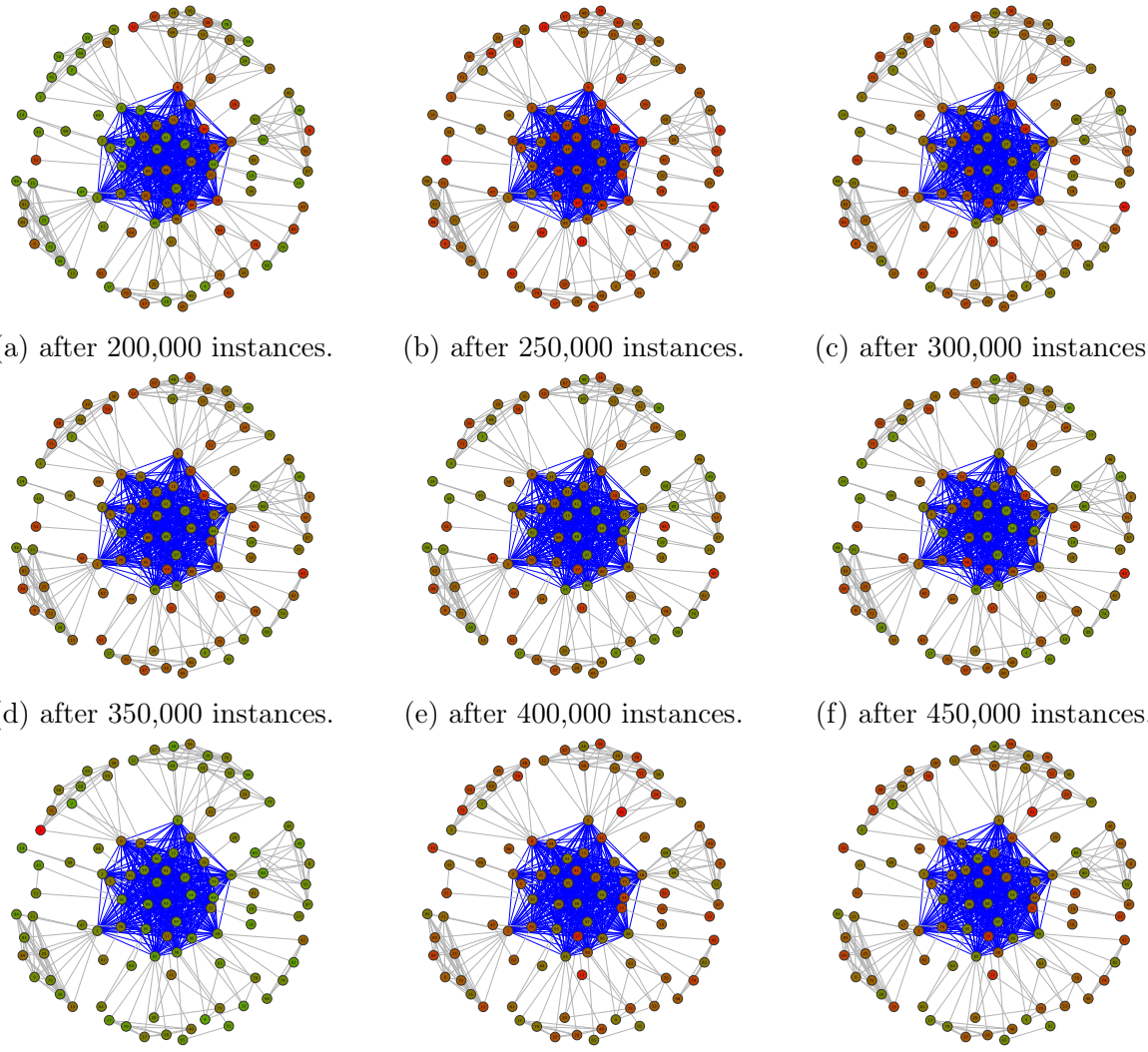
Figure 5.14: CNE$_{jac30}$ executing on ELEC using accuracy based subspace resets.

5.17) are able to recover faster from the drift than the default version CNE$_{jac30}$ (Figure 5.15), which is also shown in Figure 5.18 where the overall accuracy of these 3 variations are plotted overtime. Drifts are simulated on LED dataset by changing which of the 24 features correspond to the 7 digit segments that can be used for determining the number displayed on the LED, i.e., the class label, when the ensemble can reset the subspaces it becomes easier, especially for CNE$_{jac30}(acc)$, to find the most suitable subsets of features that correspond to the current concept.

5.5.3 CNE compared to other ensembles

In this section we compare CNE against state-of-the-art ensemble learners using Accuracy, Kappa M and Kappa Temporal. As discussed in Section 2.3.2 Kappa M and Kappa Temporal are useful for evaluating learners for datasets containing imbalaced class labels and temporal dependences, respectively. Thus, we present the results in terms of accuracy in Table acompanied by the results of Kappa M and Kappa Temporal in Tables 5.11 and 5.12, respectively.

We observe that SAE2, as well as other ensembles, classification performance seems degraded in these experiments in comparison to previously reported results. There are mainly two reasons for this behavior: (1) The update period length is set to $5,000$ for all experiments, which was previously set as $1/100$ of the total number of instances per dataset; (2) The experimental protocol is different, while previously we split the dataset into 10 equal sized chunks and averaged the prequential performance of classifiers in each chunk, here we report the overall prequential performance over the whole dataset. The first reason impacts SAE2 as the period length is sometimes "too large" or "too small", for example, to be able to adapt to incremental drifts, such as those in RBF$_f$ and RBF$_m$,

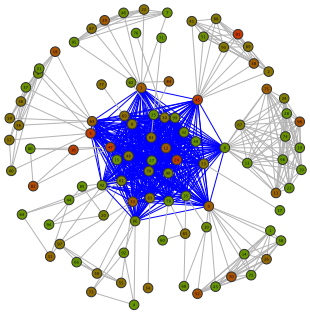


(a) after 200,000 instances. (b) after 250,000 instances. (c) after 300,000 instances.

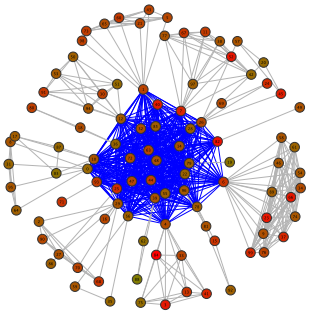
(d) after 350,000 instances. (e) after 400,000 instances. (f) after 450,000 instances.

(g) after 500,000 instances. (h) after 550,000 instances. (i) after 600,000 instances.

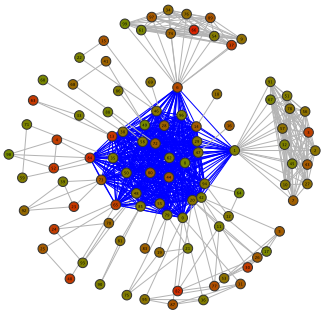
Figure 5.15: CNE_{jac30} executing on LED_g without any subspace resets.



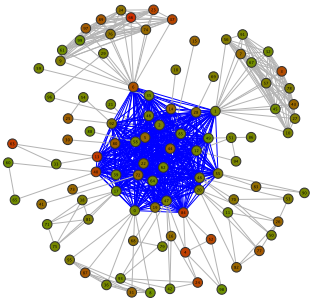
(a) after 200,000 instances.



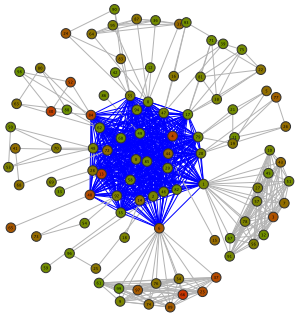
(b) after 250,000 instances.



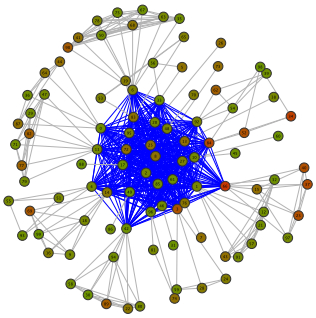
(c) after 300,000 instances.



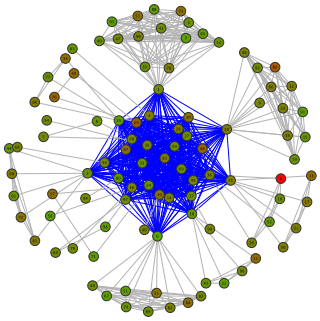
(d) after 350,000 instances.



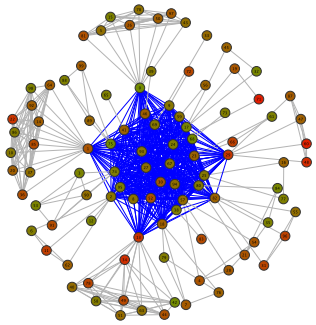
(e) after 400,000 instances.



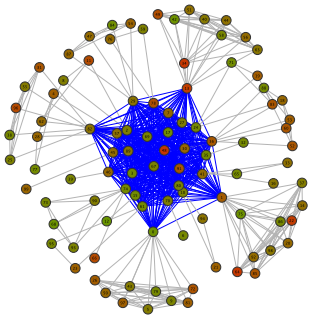
(f) after 450,000 instances.



(g) after 500,000 instances.

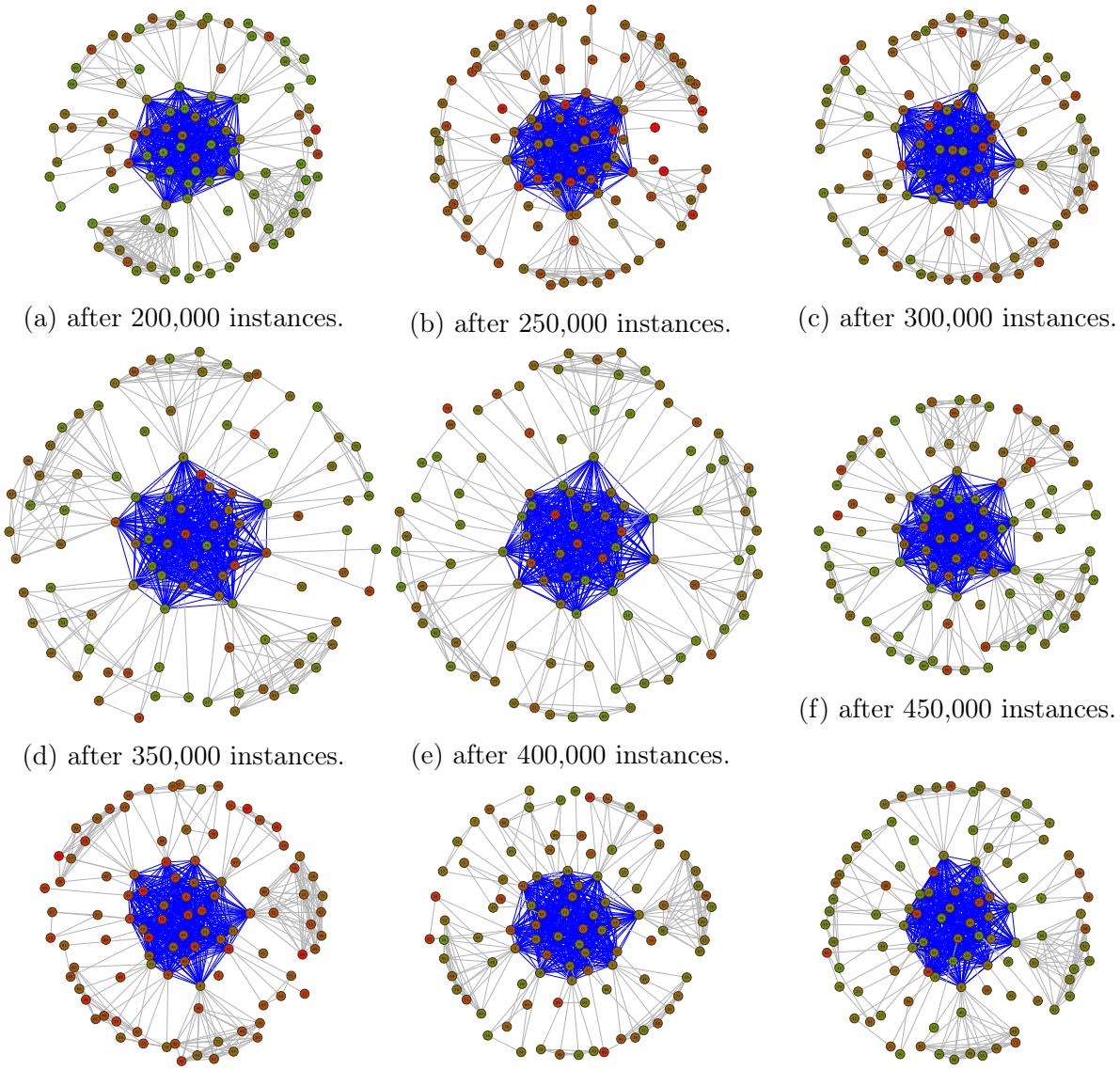


(h) after 550,000 instances.



(i) after 600,000 instances.

Figure 5.16: $\text{CNE}_{jac30}(rand)$ executing on LED_g using random based subspace resets.



(a) after 200,000 instances.

(b) after 250,000 instances.

(c) after 300,000 instances.

(d) after 350,000 instances.

(e) after 400,000 instances.

(f) after 450,000 instances.

(g) after 500,000 instances.

(h) after 550,000 instances.

(i) after 600,000 instances.

Figure 5.17: $\text{CNE}_{jac30}(acc)$ executing on LED_g using accuracy based subspace resets.

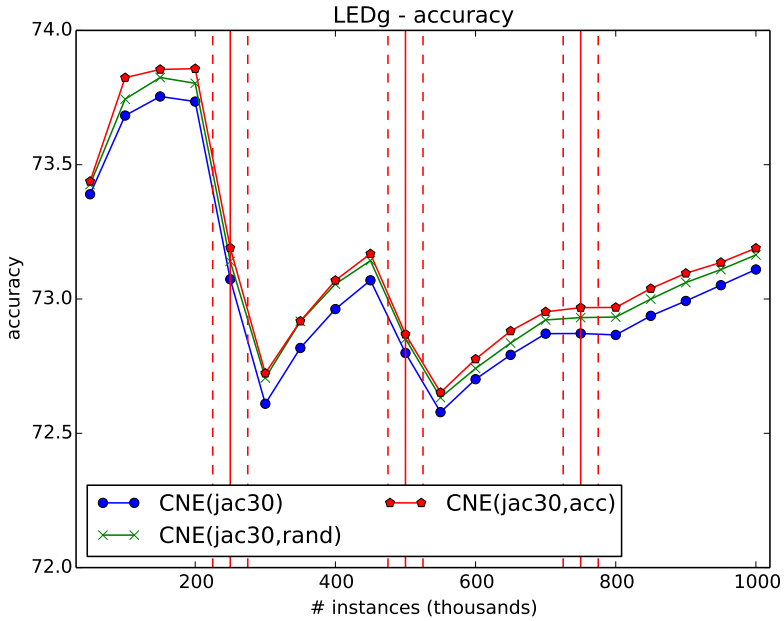
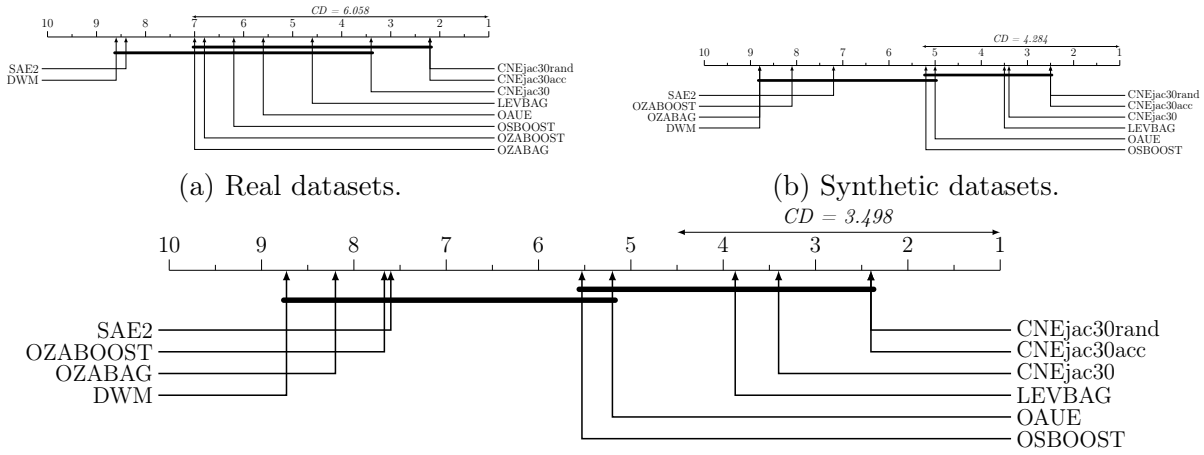


Figure 5.18: CNE_{jac30}, $\text{CNE}_{jac30}(rand)$ and $\text{CNE}_{jac30}(acc)$ executing on LED_g. Solid and dashed vertical lines indicates drifts and drift window start/end, respectively.

SAE2 would require a smaller period length. In this aspect, we note that CNE is able to adapt better than SAE2 and even to methods that include active drift detection, such as LevBag.

The usefulness of the analysis based on Kappa M and Kappa Temporal is visible when we analyze ELEC, COVT, GMSC and KDD99. In COVT all learners obtain accuracy over 90%, however when we observe the Kappa Temporal metric, only a few (CNE variations and LevBag) are able to surpass the baseline NoChange classifier, i.e. yield non-negative values. Due to the severe imbalance in both GMSC and KDD99, it is difficult to visualize their differences by inspecting only accuracy. For example, $\text{CNE}_{jac30}(acc)$ and SAE2 yields 93.58% and 93.46%, respectively, which suggests that there is not much difference between these two, but when we observe Kappa M, then $\text{CNE}_{jac30}(acc)$ and SAE2 obtains 3.88% and 2.14%, which better illustrate their differences in predicting the minority class.

We highlight CNE stability in comparison to other ensembles. For example, OAUE obtain good results in general, but fails to obtain a reasonable model for KDD99 (-128.58% while others obtain a minimum of 98.81% Kappa M). The same happens for OzaBoost, which obtains an outstanding result for ELEC dataset (36.39% Kappa Temporal), yet obtains the worse results for LED_a, LED_g, RBF_m and RBF_f. Using the same parametrization (besides varying the subspace reset strategy) CNE obtain the best or reasonable results for all datasets considered in this experiment. This is an interesting fact, since



(a) Real datasets.

(b) Synthetic datasets.

(c) All datasets.

Figure 5.19: Nemenyi posthoc with 95% confidence.

there is a multitude of different problems represented in this benchmark.

We followed these experiments with a non-parametric Friedman test, which indicate that there are significant differences among the evaluated classifiers for these datasets, both when we evaluate all datasets at once and when we conduct the test separately for synthetic and real datasets. Figures 5.19a, 5.19b and 5.19c presents the results of applying the nemenyi posthoc test to identify the statistically relevant differences. We observe that when considering all datasets at once we cannot find relevant differences among CNE variations, LevBag, OAUE and OSBoost. When we analyse the synthetic and real datasets separately, then we observe that for the synthetic datasets the results are similar to those obtained in the general analysis, but when we limited our analysis to the real datasets there are no longer differences among CNE variations, LevBag, OAUE, OSBoost, OzaBoost and OzaBag.

Finally, we compare CNE and the other ensembles in terms of CPU time and RAM Hours. Figure 5.22 presents the average CPU time and RAM Hours for the previous experiments (Tables 5.10, 5.11 and 5.12). The overall good classification performance of CNE comes at the expense of a high resources demand, especially the version based on accuracy resets. On the other hand, SAE2 is very efficient, however it may demand specific parameter tuning for each dataset to become part of the most accurate methods as shown in the previous analysis. CNE's inefficiency is attributable mainly to three aspects of it implementation: (1) it is rare that all base models in the ensemble maintain a background learner at the same time, however in the worst case it is necessary to maintain 2 versions of each base model concomitantly; (2) when a drift is detected it triggers a change in the ensemble, effectively replacing the learner where it was detected by its background learner and causing a recalculation of the network; (3) the subspace

reset based on accuracy demands further calculation to estimate each feature probability. For CNE$_j ac30$ (2) and (3) are not applicable, since there are no subspace resets in this version and thus it is not necessary to recalculate the network structure, neither perform any estimation on the features.

5.6 Final Considerations

In this section we presented a multitude of experiments concerning the methods implemented in this work. Our experiments with PA and PP revealed many interesting patterns, such as situations where a small set of classifiers could correct misclassifications from a larger set using PA (see experiments AGR3 and AGR4 in Section 5.3). The experiments in Section 5.4 contemplates comparisons between SAE2-c, SAE2-f, the best and the worst possible combinations. In these experiments we learned that the differences between SAE2-c and SAE2-f may be due to chance, and thus performance obtained by SAE2 may be attributable to its adaptation methods, instead of its combination function, however it was unfeasible to scale the experiments using larger networks, thus the results were inconclusive. The results for SAE2 shows that it is able to achieve accuracy comparable to existing ensembles, while demanding minimum resources. However, using a larger and more general set of datasets we observe that SAE2 depends too much on its parametrization, especially the period length. This motivated us to develop a method where learners were updated according to an active drift detection method, which culminates on CNE and its variations. For CNE, we present experiments with different edge weighting functions (Kappa and Jaccard), subspace reset strategies and finally a strategy to save resources by limiting the number of drift detectors. In the next chapter we present our final conclusions alongside future work.

Figure 5.20: CPU Time

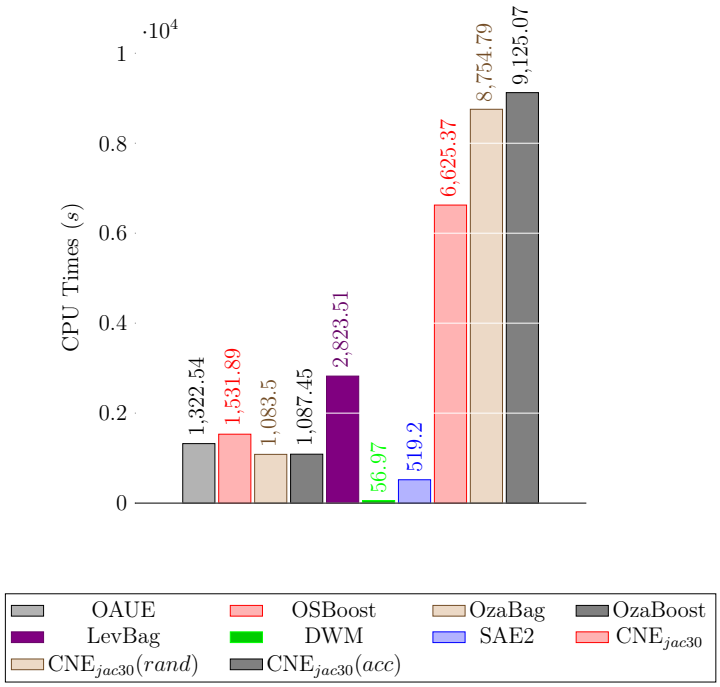


Figure 5.21: RAM-Hours

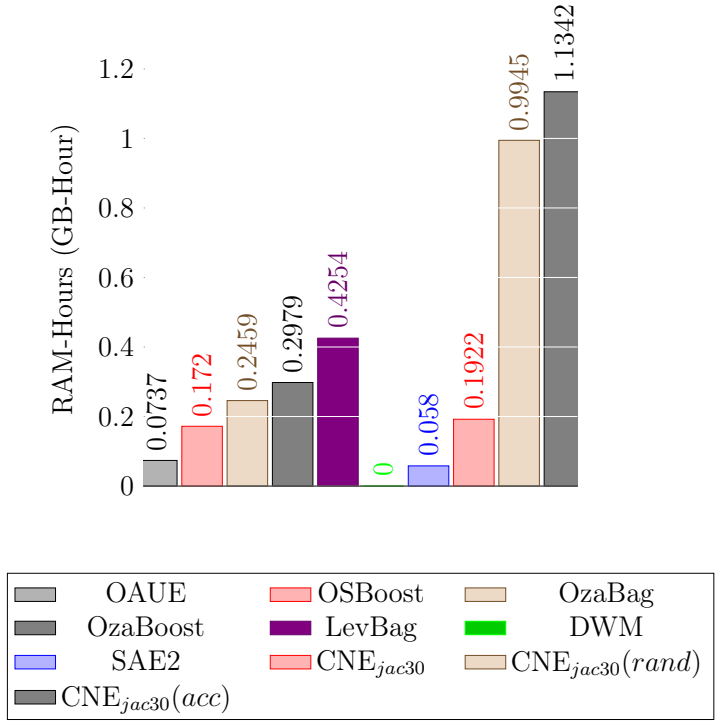


Figure 5.22: Average CPU Time and RAM-Hours for CNE and others. DWM RAM Hours is less than 10^4.

Table 5.10: Accuracy - CNE x others.

Dataset	*OAUE*	*OSBoost*	*OzaBag*	*OzaBoost*	*LevBag*	*DWM*	*SAE2*	CNE_{jac30}	$CNE_{jac30}(rand)$	$CNE_{jac30}(acc)$
LED_a	73.39	72.47	69.04	68.9	**73.95**	71.69	72.53	73.76	73.85	73.83
LED_g	72.58	72.12	68.71	68.54	**73.22**	70.72	72.07	73.11	73.16	73.18
SEA_a	88.8	89.16	87.21	88.25	88.44	86.81	88.94	**89.48**	**89.48**	**89.48**
SEA_g	88.19	88.94	87.11	87.92	**89.09**	86.38	88.72	89.07	89.07	89.07
AGR_a	90.16	90.37	83.83	88.12	88.72	76.91	88.17	90.33	**91.31**	90.85
AGR_g	85.24	87.83	79.25	84.66	83.71	76.3	82.4	86.65	87.93	**87.96**
RTS	96.81	94.78	95.12	96.93	**97.85**	94.76	95.68	97.32	97.4	97.43
RBF_m	84.26	74.51	73.06	65.23	84.34	73.51	65.43	**86.52**	86.47	86.48
RBF_f	57.15	48.7	43.54	26.16	76.77	53.88	39.83	**77.22**	77.15	77.14
HYPER	**87.8**	86.96	79.37	85.3	85.74	81.04	85.11	85.29	85.51	85.5
Syn. Avg	82.44	80.59	76.62	76	84.18	77.2	77.89	84.87	**85.13**	85.09
Syn. Avg Rank	5	5.2	8.8	8.1	3.5	8.8	7.2	3.4	**2.5**	**2.5**
AIRL	65.23	64.56	64.89	60.63	62.82	61.67	59.03	64.96	66.3	**66.44**
ELEC	87.41	89.51	85.08	**90.67**	89.51	82.19	83.66	89.67	89.82	90.01
COVT	92.86	92.69	91.49	94.82	95.1	90.03	91.98	95.15	**95.18**	95.14
GMSC	93.57	93.05	93.52	92.32	93.54	92.92	93.46	93.55	93.55	**93.58**
KDD99	2.45	99.94	99.93	99.49	**99.97**	99.93	99.88	99.96	99.96	99.96
Real Avg	68.3	87.95	86.98	87.59	88.19	85.35	85.6	88.66	88.96	**89.02**
Real Avg Rank	5.6	6.2	7	6.8	4.6	8.6	8.4	3.4	**2.2**	**2.2**
Total Avg	77.73	83.04	80.08	79.86	85.52	79.92	80.46	86.14	**86.41**	86.4
Total Avg Rank	5.2	5.53	8.2	7.67	3.87	8.73	7.6	3.4	**2.4**	**2.4**

Table 5.11: Kappa M - CNE x others.

Dataset	*OAUE*	*OSBoost*	*OzaBag*	*OzaBoost*	*LevBag*	*DWM*	*SAE2*	CNE_{jac30}	$CNE_{jac30}(rand)$	$CNE_{jac30}(acc)$
LED_a	70.39	69.37	65.55	65.39	**71.01**	68.49	69.43	70.79	70.9	70.88
LED_g	69.48	68.97	65.18	64.98	**70.2**	67.41	68.91	70.07	70.13	70.15
SEA_a	72.06	72.96	68.09	70.7	71.16	67.11	72.41	**73.77**	**73.77**	**73.77**
SEA_g	70.55	72.41	67.84	69.87	**72.8**	66.03	71.86	72.73	72.73	72.73
AGR_a	79.15	79.59	65.72	74.82	76.08	51.04	74.92	79.49	**81.59**	80.6
AGR_g	68.72	74.21	56.01	67.48	65.47	49.75	62.68	71.7	74.42	**74.47**
RTS	92.44	87.62	88.43	92.71	**94.91**	87.58	89.75	93.65	93.83	93.9
RBF_m	77.51	63.59	61.51	50.32	77.62	62.16	50.61	**80.74**	80.67	80.68
RBF_f	38.77	26.7	19.34	-5.5	66.81	34.11	14.04	**67.45**	67.36	67.34
HYPER	**75.56**	73.88	58.67	70.54	71.44	62.02	70.16	70.52	70.96	70.95
Syn. Avg	71.46	68.93	61.63	62.13	73.75	61.57	64.48	75.09	**75.64**	75.55
Syn. Avg Rank	5	5.2	8.8	8.1	3.5	8.8	7.2	3.4	**2.5**	**2.5**
AIRL	21.94	20.43	21.17	11.61	16.53	13.94	8.02	21.34	24.33	**24.66**
ELEC	70.33	75.3	64.85	**78.02**	75.28	58.04	61.5	75.66	76.02	76.46
COVT	86.06	85.74	83.39	89.89	90.44	80.55	84.35	90.53	**90.59**	90.51
GMSC	3.78	-4.02	3.1	-14.89	3.37	-5.88	2.14	3.48	3.54	**3.88**
KDD99	-128.58	99.85	99.84	98.81	**99.92**	99.85	99.72	99.91	99.91	99.9
Real Avg	10.71	55.46	54.47	52.69	57.11	49.3	51.15	58.18	58.88	**59.08**
Real Avg Rank	5.6	6.2	7	6.8	4.6	8.6	8.4	3.4	**2.2**	**2.2**
Total Avg	51.21	64.44	59.25	58.98	68.2	57.48	60.03	69.46	70.05	**70.06**
Total Avg Rank	5.2	5.53	8.2	7.67	3.87	8.73	7.6	3.4	**2.4**	**2.4**

Table 5.12: Kappa Temporal - CNE x others.

Dataset	*OAUE*	*OSBoost*	*OzaBag*	*OzaBoost*	*LevBag*	*DWM*	*SAE2*	CNE_{jac30}	$CNE_{jac30}(rand)$	$CNE_{jac30}(acc)$
LED_a	70.42	69.4	65.59	65.43	**71.05**	68.53	69.47	70.83	70.94	70.92
LED_g	69.52	69	65.22	65.02	**70.23**	67.45	68.95	70.1	70.16	70.19
SEA_a	76.5	77.26	73.16	75.36	75.74	72.34	76.8	**77.94**	**77.94**	**77.94**
SEA_g	75.24	76.81	72.96	74.67	**77.13**	71.44	76.34	77.07	77.07	77.07
AGR_a	78.48	78.94	64.62	74.01	75.32	49.48	74.12	78.83	**81**	79.98
AGR_g	68.14	73.73	55.19	66.88	64.83	48.81	61.99	71.17	73.94	**73.99**
RTS	93.46	89.3	90	93.7	**95.6**	89.26	91.14	94.51	94.66	94.73
RBF_m	79.64	67.02	65.15	55.01	79.74	65.73	55.28	**82.56**	82.5	82.5
RBF_f	44.56	33.62	26.96	4.46	69.95	40.33	22.15	**70.53**	70.44	70.43
HYPER	**75.61**	73.93	58.75	70.6	71.49	62.09	70.22	70.58	71.02	71.01
Syn. Avg	73.16	70.9	63.76	64.51	75.11	63.54	66.64	76.41	**76.97**	76.87
Syn. Avg Rank	5	5.2	8.8	8.1	3.5	8.8	7.2	3.4	**2.5**	**2.5**
AIRL	17.11	15.51	16.29	6.14	11.37	8.62	2.33	16.47	19.65	**20**
ELEC	14.16	28.52	-1.72	**36.39**	28.48	-21.42	-11.39	29.56	30.6	31.88
COVT	-44.63	-47.91	-72.32	-4.93	0.86	-101.81	-62.32	1.73	**2.35**	1.57
GMSC	48.43	44.25	48.06	38.42	48.21	43.25	47.55	48.27	48.3	**48.48**
KDD99	-770648.87	-401.94	-431.29	-3900.97	**-175.81**	-417.42	-850.32	-211.94	-210.32	-225.65
Real Avg	-154122.76	-72.31	-88.19	-764.99	**-17.38**	-97.75	-174.83	-23.18	-21.88	-24.74
Real Avg Rank	5.6	6.2	7	6.8	4.6	8.6	8.4	3.4	**2.2**	**2.2**
Total Avg	-51325.48	23.16	13.11	-211.99	**44.28**	9.78	-13.85	43.21	44.02	43
Total Avg Rank	5.2	5.53	8.2	7.67	3.87	8.73	7.6	3.4	**2.4**	**2.4**

Chapter 6

Conclusions

The problem that we tackle in this work is beyond defining a new network-based ensemble algorithm. Concretely, the problem extends to providing enough evidence that justifies the use of relational data in ensemble classifiers. Thus, beside propose and evaluate various relation definitions and structural analysis methods, we also had to provide sustainable evidence to validate network-based ensembles use for evolving data stream classification.

Besides network-based ensembles related innovations, it would be unfruitful to not explore existing adaption methods for data streams, such as drift detection algorithms or temporal aware weighting functions. Although if we employ such methods and limit our empirical analysis to the classification performance obtained by the proposed methods, we can actually blur the impact of network-based methods. Therefore, whenever possible we also set up experiments to verify the contribution of network approaches for ensemble classifiers (e.g. Section 5.4).

Effectively, in this work we present four distinct implementations that use relational data alongside ensemble classifiers for evolving data stream classification: SAE2, PA, PP and CNE. Each of them was empirically tested, and we were able to achieve good performance in terms of classification accuracy (CNE) and resources usage (SAE2). We also present a general strategy for developing network-based ensembles and a thorough analysis of existing ensemble classifiers as well as a comprehensible taxonomy of methods.

For future work we are going to pursue more efficient implementations of CNE, the most accurate and flexible method, and explore the combination methods from PA and PP. Also, we plan on using the network to guide which instances are used to train each base classifier, besides using it solely for performance pruning and voting. Training classifiers on different instances based on the network will give more control to how classifiers are build, thus we may build operators to make sure important subspaces of the classification

space are covered. However, this needs more investigation as it is not trivial to decide the criteria to build the network, especially for evolving data streams.

6.1 Published Work

In this section we present papers already published/recommended for publication during the development of this work. We highlight the papers "SAE2: Advances in the Social Adaptive Ensemble"[1], "Pairwise Combination of Classifiers for Ensemble Learning on Data Streams" and "A Survey on Ensemble Learning for Data Stream Classification", as these form substantial portions of the current work.

GOMES, HEITOR MURILO; BARDDAL, JEAN PAUL ; ENEMBRECK, FABRÍCIO; BIFET, ALBERT. A Survey on Ensemble Learning for Data Stream Classification. ACM Computing Surveys, *Submitted: Dec 2015, Recommended for publication with minor revision: Jan 2017.*

BARDDAL, JEAN PAUL ; **GOMES, HEITOR MURILO** ; ENEMBRECK, FABRÍCIO; BARTHES, JEAN-PAUL. SNCStream+: Extending A High Quality True Anytime Data Stream Clustering Algorithm. Information Systems (Oxford), v. 1, p. 1, 2016.

GOMES, HEITOR MURILO ; BARDDAL, JEAN PAUL ; ENEMBRECK, FABRÍCIO. Pairwise Combination of Classifiers for Ensemble Learning on Data Streams. Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, 2015. p. 941.

BARDDAL, JEAN PAUL ; **GOMES, HEITOR MURILO** ; ENEMBRECK, FABRÍCIO. SNCStream: A Social Network-based Data Stream Clustering Algorithm. Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, 2015. p. 935.

BARDDAL, JEAN PAUL ; **GOMES, HEITOR MURILO** ; ENEMBRECK, FABRÍCIO. Advances on Concept Drift Detection in Regression Tasks Using Social Networks Theory. International Journal of Natural Computing Research (IJNCR), 5(1), 2015. p. 26.

[1]Best paper award information systems.

GOMES, HEITOR MURILO ; ENEMBRECK, FABRÍCIO. SAE2: Advances in the Social Adaptive Ensemble. Proceedings of the 29th Annual ACM Symposium on Applied Computing, Gyeongju, Korea, 2014. p. 798.

BARDDAL, JEAN PAUL ; **GOMES, HEITOR MURILO** ; ENEMBRECK, FABRÍCIO. SFNClassifier: a scale-free social network method to handle concept drift. Proceedings of the 29th Annual ACM Symposium on Applied Computing, Gyeongju, Korea, 2014. p. 786.

GOMES, HEITOR MURILO ; ENEMBRECK, FABRICIO. SAE: Social Adaptive Ensemble classifier for data streams. IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Singapore, 2013. p. 199.

References

ABDULSALAM, H.; SKILLICORN, D. B.; MARTIN, P. Streaming random forests. In: IEEE. *Database Engineering and Applications Symposium, 2007. IDEAS 2007. 11th International*. [S.l.], 2007. p. 225–232.

ABDULSALAM, H.; SKILLICORN, D. B.; MARTIN, P. Classifying evolving data streams using dynamic streaming random forests. In: SPRINGER. *International Conference on Database and Expert Systems Applications*. [S.l.], 2008. p. 643–651.

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 - Volume 29*. VLDB Endowment, 2003. (VLDB '03), p. 81–92. ISBN 0-12-722442-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1315451.1315460>.

AGRAWAL, R.; IMILIELINSKI, T.; SWANI, A. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, v. 5, n. 6, p. 914–925, Dec. 1993.

ALBERT, R.; BARABÁSI, A. L. Statistical mechanics of complex networks. In: THE AMERICAN PHYSICAL SOCIETY. *Reviews of Modern Physics*. [S.l.], 2002. p. 139–148.

ALPAYDIN, E.; KAYNAK, C. Cascading classifiers. *Kybernetika*, Institute of Information Theory and Automation AS CR, v. 34, n. 4, p. 369–374, 1998.

AMINI, A.; WAH, T. Y. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, Springer US, v. 29, n. 1, p. 116–141, 2014. ISSN 1000-9000.

AMIT, Y.; GEMAN, D. Shape quantization and recognition with randomized trees. *Neural computation*, MIT Press, v. 9, n. 7, p. 1545–1588, 1997.

BAENA-GARCÍA, M.; CAMPO-ÁVILA, J. del; FIDALGO, R.; BIFET, A.; GAVALDÀ, R.; MORALES-BUENO, R. Early drift detection method. 2006.

BANFIELD, R. E.; HALL, L. O.; BOWYER, K. W.; KEGELMEYER, W. P. Ensemble diversity measures and their application to thinning. *Information Fusion*, Elsevier, v. 6, n. 1, p. 49–62, 2005.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F. Sfnclassifier: A scale-free social network method to handle concept drift. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2014. (SAC '14), p. 786–791. ISBN 978-1-4503-2469-4. Disponível em: <http://doi.acm.org/10.1145/2554850.2554855>.

BAUR, M.; BENKERT, M.; BRANDES, U.; CORNELSEN, S.; GAERTLER, M.; KÖPF, B.; LERNER, J.; WAGNER, D. Visone software for visual social network analysis. In: SPRINGER. *International Symposium on Graph Drawing*. [S.l.], 2001. p. 463–464.

BERINGER, J.; HÜLLERMEIER, E. An efficient algorithm for instance-based learning on data streams. In: SPRINGER. *Industrial Conference on Data Mining*. [S.l.], 2007. p. 34–48.

BEYGELZIMER, A.; KALE, S.; LUO, H. Optimal and adaptive algorithms for online boosting. ACM, New York, NY, USA, p. 2323–2331, 2015.

BIFET, A.; FRANK, E.; HOLMES, G.; PFAHRINGER, B. Ensembles of restricted hoeffding trees. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM, v. 3, n. 2, p. 30, 2012.

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

BIFET, A.; GAVALDÀ, R. Adaptive learning from evolving data streams. In: SPRINGER. *International Symposium on Intelligent Data Analysis*. [S.l.], 2009. p. 249–260.

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.; KIRKBY, R.; PFAHRINGER, B. *MOA Data Stream Mining - A Practical Approach*. [S.l.]: Centre for Open Software Innovation, 2011. <http://heanet.dl.sourceforge.net/project/moa-datastream/documentation/StreamMining.pdf>.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: *PKDD*. [S.l.: s.n.], 2010. p. 135–150.

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.; HOLMES, G.; PFAHRINGER, B.; FRANK, E. Fast perceptron decision tree learning from evolving data streams. In: *PAKDD*. [S.l.]: Springer, 2010. (Lecture Notes in Computer Science), p. 299–310.

BIFET, A.; MORALES, G. de F.; READ, J.; HOLMES, G.; PFAHRINGER, B. Efficient online evaluation of big data stream classifiers. In: ACM. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.], 2015. p. 59–68.

BIFET, A.; READ, J.; ŽLIOBAITĖ, I.; PFAHRINGER, B.; HOLMES, G. Pitfalls in benchmarking data stream classification and how to avoid them. In: *Machine Learning and Knowledge Discovery in Databases*. [S.l.]: Springer, 2013. p. 465–479.

BLACKARD, J. A.; DEAN, D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, Elsevier, v. 24, n. 3, p. 131–151, 1999.

BOCCALETTI, S.; LATORA, V.; MORENO, Y.; CHAVEZ, M.; HWANG, D.-U. Complex networks: Structure and dynamics. *Physics reports*, Elsevier, v. 424, n. 4, p. 175–308, 2006.

BORDA, J. C. de. *Memoire sur les elections au scrutin*. Paris: Historie de l'Academie Royale des Sciences, 1781.

BRAZDIL, P.; CARRIER, C. G.; SOARES, C.; VILALTA, R. *Metalearning: Applications to data mining*. [S.l.]: Springer Science & Business Media, 2008.

BREIMAN, L. Bagging predictors. *Machine Learning*, Kluwer Academic Publishers, v. 24, n. 2, p. 123–140, 1996.

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

BREIMAN, L.; FRIEDMAN, J.; STONE, C. J.; OLSHEN, R. A. *Classification and regression trees*. [S.l.]: CRC press, 1984.

BROWN, G.; WYATT, J.; HARRIS, R.; YAO, X. Diversity creation methods: a survey and categorisation. *Information Fusion*, Elsevier, v. 6, n. 1, p. 5–20, 2005.

BRZEZIŃSKI, D.; STEFANOWSKI, J. Accuracy updated ensemble for data streams with concept drift. In: *Hybrid Artificial Intelligent Systems*. [S.l.]: Springer, 2011. p. 155–163.

BRZEZINSKI, D.; STEFANOWSKI, J. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, v. 265, p. 50–67, 2014.

CHAN, P.; STOLFO, S. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, Kluwer Academic Publishers, v. 8, n. 1, p. 5–28, 1997. ISSN 0925-9902. Disponível em: <http://dx.doi.org/10.1023/A%3A1008640732416>.

CHAN, P. K.; STOLFO, S. J. A comparative evaluation of voting and meta-learning on partitioned data. In: *ICML*. [S.l.: s.n.], 1995. p. 90–98.

CHEN, S.-T.; LIN, H.-T.; LU, C.-J. An online boosting algorithm with theoretical justifications. In: *Proceedings of the International Conference on Machine Learning (ICML)*. [S.l.: s.n.], 2012.

CHU, F.; ZANIOLO, C. Fast and light boosting for adaptive mining of data streams. In: *Advances in knowledge discovery and data mining*. [S.l.]: Springer, 2004. p. 282–292.

COHEN, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, Sage Publications, 1960.

DALIRSEFAT, S. B.; MEYER, A. da S.; MIRHOSEINI, S. Z. Comparison of similarity coefficients used for cluster analysis with amplified fragment length polymorphism markers in the silkworm, bombyx mori. *Journal of Insect Science*, BioOne, v. 9, n. 71, p. 1–8, 2009.

DECKERT, M. Batch weighted ensemble for mining data streams with concept drift. In: *Foundations of Intelligent Systems*. [S.l.]: Springer, 2011. p. 290–299.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, JMLR.org, v. 7, p. 1–30, dez. 2006. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=1248547.1248548>.

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

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

DIETTERICH, T. G.; BAKIRI, G. Solving multiclass learning problems via error-correcting output codes. *arXiv preprint cs/9501101*, 1995.

DITZLER, G.; MUHLBAIER, M. D.; POLIKAR, R. Incremental learning of new classes in unbalanced datasets: Learn++.udnc. In: *Multiple Classifier Systems*. Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 5997). p. 33–42. ISBN 978-3-642-12126-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-12127-2_4>.

DITZLER, G.; POLIKAR, R. Incremental learning of concept drift from streaming imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, IEEE, v. 25, n. 10, p. 2283–2301, 2013.

DOMENICONI, C.; YAN, B. Nearest neighbor ensemble. In: IEEE. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. [S.l.], 2004. v. 1, p. 228–231.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: ACM SIGKDD. *Proc. of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2000. p. 71–80.

ELWELL, R.; POLIKAR, R. Incremental learning of concept drift in nonstationary environments. *Neural Networks, IEEE Transactions on*, IEEE, v. 22, n. 10, p. 1517–1531, 2011.

FORTUNATO, S. Community detection in graphs. *Physics Reports*, abs/0906.0612, n. 3-5, p. 75 – 174, 2010.

FREUND, Y.; SCHAPIRE, R. E. et al. Experiments with a new boosting algorithm. In: *ICML*. [S.l.: s.n.], 1996. v. 96, p. 148–156.

GABER, M. M.; KRISHNASWAMY, S.; ZASLAVSKY, A. On-board mining of data streams in sensor networks. In: *Advanced methods for knowledge discovery from complex data*. [S.l.]: Springer, 2005. p. 307–335.

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

GAMA, J.; BRAZDIL, P. Cascade generalization. *Machine Learning*, Kluwer Academic Publishers, v. 41, n. 3, p. 315–343, 2000. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1023/A%3A1007652114878>.

GAMA, J.; KOSINA, P. Tracking recurring concepts with meta-learners. In: *Progress in Artificial Intelligence*. [S.l.]: Springer, 2009. p. 423–434.

GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. Learning with drift detection. In: *Advances in Artificial Intelligence–SBIA 2004*. [S.l.]: Springer, 2004. p. 286–295.

GAMA, J.; RODRIGUES, P. Issues in evaluation of stream learning algorithms. In: ACM SIGKDD. *15th ACM SIGKDD*. [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. Disponível em: <http://doi.acm.org/10.1145/2523813>.

GOMES, H. M.; BARDDAL, J. P.; ENEMBRECK, F. Pairwise combination of classifiers for ensemble learning on data streams. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC)*. [S.l.]: ACM, 2015. (SAC 2015), p. 941–946.

GOMES, H. M.; ENEMBRECK, F. Sae: Social adaptive ensemble classifier for data streams. In: *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*. [S.l.: s.n.], 2013. p. 199–206.

GOMES, H. M.; ENEMBRECK, F. Sae2: Advances on the social adaptive ensemble classifier for data streams. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)*. [S.l.]: ACM, 2014. (SAC 2014).

HAQUE, A.; PARKER, B.; KHAN, L.; THURAISINGHAM, B. Evolving big data stream classification with mapreduce. In: IEEE. *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. [S.l.], 2014. p. 570–577.

HASHEMI, S.; YANG, Y.; MIRZAMOMEN, Z.; KANGAVARI, M. Adapted one-versus-all decision trees for data stream classification. *Knowledge and Data Engineering, IEEE Transactions on*, IEEE, v. 21, n. 5, p. 624–637, 2009.

HO, T. K. Random decision forests. In: IEEE. *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. [S.l.], 1995. v. 1, p. 278–282.

HO, T. K. Nearest neighbors in random subspaces. In: SPRINGER. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. [S.l.], 1998. p. 640–648.

HO, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 20, n. 8, p. 832–844, 1998.

HOEFFDING, W. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, Taylor & Francis Group, v. 58, n. 301, p. 13–30, 1963.

HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. Stress-testing hoeffding trees. In: *PKDD*. [S.l.: s.n.], 2005. p. 495–502.

HUANG, Y.; SUEN, C. The behavior-knowledge space method for combination of multiple classifiers. In: IEEE. *Proc. of IEEE Computer Vision and Pattern Recog.* [S.l.], 1993. p. 347–352.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: ACM. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2001. p. 97–106.

JABER, G.; CORNUÉJOLS, A.; TARROUX, P. A new on-line learning method for coping with recurring concepts: The adacc system. In: SPRINGER. *Neural Information Processing*. [S.l.], 2013. p. 595–604.

JABER, G.; CORNUÉJOLS, A.; TARROUX, P. Online learning: Searching for the best forgetting strategy under concept drift. In: SPRINGER. *Neural Information Processing*. [S.l.], 2013. p. 400–408.

JACOBS, R. A.; JORDAN, M. I.; NOWLAN, S. J.; HINTON, G. E. Adaptive mixtures of local experts. *Neural computation*, MIT Press, v. 3, n. 1, p. 79–87, 1991.

JAIN, A. K.; DUIN, R. P.; MAO, J. Statistical pattern recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 22, n. 1, p. 4–37, 2000.

JAPKOWICZ, N.; SHAH, M. *Evaluating learning algorithms: a classification perspective*. [S.l.]: Cambridge University Press, 2011.

JOHN, G. H.; KOHAVI, R.; PFLEGER, K. et al. Irrelevant features and the subset selection problem. In: *Machine learning: proceedings of the eleventh international conference*. [S.l.: s.n.], 1994. p. 121–129.

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

JORDAN, M. I.; JACOBS, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, MIT Press, v. 6, n. 2, p. 181–214, 1994.

JR, P. M. G.; BARROS, R. S. M. de. Rcd: A recurring concept drift framework. *Pattern Recognition Letters*, Elsevier, v. 34, n. 9, p. 1018–1025, 2013.

KATAKIS, I.; TSOUMAKAS, G.; BANOS, E.; BASSILIADES, N.; VLAHAVAS, I. An adaptive personalized news dissemination system. *Journal of Intelligent Information Systems*, v. 32, n. 2, p. 191–212, 2009.

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

KHAN, M.; DING, Q.; PERRIZO, W. K-nearest neighbor classification on spatial data streams using p-trees. In: *Advances in Knowledge Discovery and Data Mining*. [S.l.]: Springer, 2002. p. 517–528.

KOLTER, J. Z.; MALOOF, M. A. Using additive expert ensembles to cope with concept drift. In: *Proceedings of the 22Nd International Conference on Machine Learning*. New York, NY, USA: ACM, 2005. (ICML '05), p. 449–456. ISBN 1-59593-180-5.

KOLTER, J. Z.; MALOOF, M. A. Using additive expert ensembles to cope with concept drift. In: *Proceedings of the 22Nd International Conference on Machine Learning*. New York, NY, USA: ACM, 2005. (ICML '05), p. 449–456. ISBN 1-59593-180-5.

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.

KUNCHEVA, L. I. *Combining pattern classifiers: methods and algorithms*. [S.l.]: John Wiley & Sons, 2004.

KUNCHEVA, L. I.; RODRÍGUEZ, J. J. A weighted voting framework for classifiers ensembles. *Knowledge and Information Systems*, Springer, v. 38, n. 2, p. 259–275, 2014.

KUNCHEVA, L. I.; RODRÍGUEZ, J. J.; PLUMPTON, C. O.; LINDEN, D. E.; JOHNSTON, S. J. Random subspace ensembles for fmri classification. *IEEE transactions on medical imaging*, IEEE, v. 29, n. 2, p. 531–542, 2010.

KUNCHEVA, L. I.; WHITAKER, C. J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, Springer, v. 51, n. 2, p. 181–207, 2003.

KUNCHEVA, L. I.; WHITAKER, C. J.; SHIPP, C. A.; DUIN, R. P. Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis & Applications*, Springer, v. 6, n. 1, p. 22–31, 2003.

LAW, Y.-N.; ZANIOLO, C. An adaptive nearest neighbor classification algorithm for data streams. In: *Knowledge Discovery in Databases: PKDD 2005*. [S.l.]: Springer, 2005. p. 108–120.

LEVANDOWSKY, M.; WINTER, D. Distance between sets. *Nature*, Nature Publishing Group, v. 234, n. 5323, p. 34–35, 1971.

LI, P.; WU, X.; HU, X.; WANG, H. Learning concept-drifting data streams with random ensemble decision trees. *Neurocomputing*, Elsevier, 2015.

LI, X.; ZHAO, H. Weighted random subspace method for high dimensional data classification. *Statistics and its Interface*, NIH Public Access, v. 2, n. 2, p. 153, 2009.

LIM, C. P.; HARRISON, R. F. Online pattern classification with multiple neural network systems: an experimental study. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, IEEE, v. 33, n. 2, p. 235–247, 2003.

LITTLESTONE, N.; WARMUTH, M. K. The weighted majority algorithm. *Information and computation*, Elsevier, v. 108, n. 2, p. 212–261, 1994.

MARGINEANTU, D. D.; DIETTERICH, T. G. Pruning adaptive boosting. In: CITESEER. *ICML*. [S.l.], 1997. v. 97, p. 211–218.

MASUD, M. M.; CHEN, Q.; GAO, J.; KHAN, L.; HAN, J.; THURAISINGHAM, B. Classification and novel class detection of data streams in a dynamic feature space. In: *Machine Learning and Knowledge Discovery in Databases*. [S.l.]: Springer, 2010. p. 337–352.

MASUD, M. M.; GAO, J.; KHAN, L.; HAN, J.; THURAISINGHAM, B. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In: IEEE. *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. [S.l.], 2008. p. 929–934.

MERZ, C. J. Dynamical selection of learning algorithms. In: *Learning from Data*. [S.l.]: Springer, 1996. p. 281–290.

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

MINKU, L. L.; YAO, X. Ddd: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 24, n. 4, p. 619–633, 2012. ISSN 1041-4347.

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

MUHLBAIER, M. D.; TOPALIS, A.; POLIKAR, R. Learn^{++}.nc: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes. *Neural Networks, IEEE Transactions on*, IEEE, v. 20, n. 1, p. 152–168, 2009.

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*. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7302). p. 1–12. ISBN 978-3-642-30219-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-30220-6_1>.

NGUYEN, T.-T.; ZHAO, H.; HUANG, J. Z.; NGUYEN, T. T.; LI, M. J. A new feature sampling method in random forests for predicting high-dimensional data. In: SPRINGER. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. [S.l.], 2015. p. 459–470.

NISHIDA, K.; YAMAUCHI, K.; OMORI, T. Ace: Adaptive classifiers-ensemble system for concept-drifting environments. In: *Multiple Classifier Systems*. [S.l.]: Springer, 2005. p. 176–185.

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.

OZA, N. C.; RUSSELL, S. *Online ensemble learning*. [S.l.]: University of California, Berkeley, 2001.

PARKER, B.; KHAN, L. Rapidly labeling and tracking dynamically evolving concepts in data streams. *2013 IEEE 13th International Conference on Data Mining Workshops*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 1161–1164, 2013.

PARKER, B.; MUSTAFA, A. M.; KHAN, L. Novel class detection and feature via a tiered ensemble approach for stream mining. In: IEEE. *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*. [S.l.], 2012. v. 1, p. 1171–1178.

PARKER, B. S.; KHAN, L. Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2015.

PARKER, B. S.; KHAN, L.; BIFET, A. Incremental ensemble classifier addressing non-stationary fast data streams. In: IEEE. *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*. [S.l.], 2014. p. 716–723.

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

PFAHRINGER, B.; HOLMES, G.; KIRKBY, R. New options for hoeffding trees. In: *Proceedings of the 20th Australian Joint Conference on Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer-Verlag, 2007. (AI'07), p. 90–99. ISBN 3-540-76926-9, 978-3-540-76926-2.

POCOCK, A.; YIAPANIS, P.; SINGER, J.; LUJÁN, M.; BROWN, G. Online non-stationary boosting. In: *Multiple Classifier Systems*. [S.l.]: Springer, 2010. p. 205–214.

POLIKAR, R. Ensemble based systems in decision making. *Circuits and systems magazine, IEEE*, IEEE, v. 6, n. 3, p. 21–45, 2006.

POLIKAR, R.; UPDA, L.; UPDA, S. S.; HONAVAR, V. Learn++: An incremental learning algorithm for supervised neural networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, IEEE, v. 31, n. 4, p. 497–508, 2001.

QUINLAN, J. R. C4.5: Programs for machine learning. *The Morgan Kaufmann Series in Machine Learning*, v. 1, 1993.

RAMAMURTHY, S.; BHATNAGAR, R. Tracking recurrent concept drift in streaming data using ensemble classifiers. In: IEEE. *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*. [S.l.], 2007. p. 404–409.

READ, J.; BIFET, A.; PFAHRINGER, B.; HOLMES, G. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In: *Advances in Intelligent Data Analysis XI*. [S.l.]: Springer, 2012. p. 313–323.

RIJN, J. N. van; HOLMES, G.; PFAHRINGER, B.; VANSCHOREN, J. Having a blast: Meta-learning and heterogeneous ensembles for data streams. In: IEEE. *Data Mining (ICDM), 2015 IEEE International Conference on*. [S.l.], 2015. p. 1003–1008.

ROKACH, L. Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography. *Computational Statistics & Data Analysis*, Elsevier, v. 53, n. 12, p. 4046–4072, 2009.

ROKACH, L. Ensemble-based classifiers. *Artificial Intelligence Review*, Springer, v. 33, n. 1-2, p. 1–39, 2010.

RUSHING, J.; GRAVES, S.; CRISWELL, E.; LIN, A. A coverage based ensemble algorithm (cbea) for streaming data. In: IEEE. *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. [S.l.], 2004. p. 106–112.

RYU, J. W.; KANTARDZIC, M. M.; KIM, M.-W. Efficiently maintaining the performance of an ensemble classifier in streaming data. In: *Convergence and hybrid information technology*. [S.l.]: Springer, 2012. p. 533–540.

SCHAFFER, C. Selecting a classification method by cross-validation. *Machine Learning*, Springer, v. 13, n. 1, p. 135–143, 1993.

SCHAPIRE, R. E.; FREUND, Y. *Boosting: Foundations and algorithms*. [S.l.]: MIT press, 2012.

SCHOLZ, M.; KLINKENBERG, R. Boosting classifiers for drifting concepts. *Intelligent Data Analysis*, IOS Press, v. 11, n. 1, p. 3–28, 2007.

SETHI, T. S.; KANTARDZIC, M.; ARABMAKKI, E.; HU, H. An ensemble classification approach for handling spatio-temporal drifts in partially labeled data streams. In: IEEE. *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*. [S.l.], 2014. p. 725–732.

SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, v. 27, n. 3, p. 379–423 and 623–656, 1948.

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

SILVA, T. C.; ZHAO, L. *Machine learning in complex networks*. [S.l.]: Springer, 2016. v. 2016.

SKURICHINA, M.; DUIN, R. P. Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis & Applications*, Springer, v. 5, n. 2, p. 121–135, 2002.

STANLEY, K. O. Learning concept drift with a committee of decision trees. *Informe técnico: UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA*, Citeseer, 2003.

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

TSYMBAL, A. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, Citeseer, v. 106, 2004.

TULYAKOV, S.; JAEGER, S.; GOVINDARAJU, V.; DOERMANN, D. Review of classifier combination methods. In: *Machine Learning in Document Analysis and Recognition*. [S.l.]: Springer, 2008. p. 361–386.

WANG, H.; FAN, W.; YU, P. S.; HAN, J. Mining concept-drifting data streams using ensemble classifiers. In: ACM. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2003. p. 226–235.

WANG, S.; MINKU, L. L.; YAO, X. A learning framework for online class imbalance learning. In: IEEE. *Computational Intelligence and Ensemble Learning (CIEL), 2013 IEEE Symposium on*. [S.l.], 2013. p. 36–45.

WANG, S.; MINKU, L. L.; YAO, X. Resampling-based ensemble methods for online class imbalance learning. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 27, n. 5, p. 1356–1368, 2015.

WANG, S.; MINKU, L. L.; YAO, X. Dealing with multiple classes in online class imbalance learning. In: *Proc. 25th Int. Joint Conf. Artificial Intelligence, IJCAI/AAAI Press*. [S.l.: s.n.], 2016. p. 2118–2124.

WANG, Z.; CRAMMER, K.; VUCETIC, S. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *The Journal of Machine Learning Research*, JMLR. org, v. 13, n. 1, p. 3103–3131, 2012.

WANKHADE, K. K.; DONGRE, S. S.; MANKAR, K. A.; ADAKANE, P. K. A new adaptive ensemble boosting classifier for concept drifting stream data. In: *2011 3rd International Conference on Computer Modeling and Simulation (ICCMS 2011)*. [S.l.: s.n.], 2012.

WASSERMAN, S.; FAUST, K. *Social network analysis: Methods and applications*. [S.l.]: Cambridge university press, 1994. v. 8.

WENERSTROM, B.; GIRAUD-CARRIER, C. Temporal data mining in dynamic feature spaces. In: IEEE. *Data Mining, 2006. ICDM'06. Sixth International Conference on*. [S.l.], 2006. p. 1141–1145.

WHITE, T. *Hadoop: The Definitive Guide*. [S.l.]: O'Reilly Media, Inc., 2012. ISBN 1449311520, 9781449311520.

WIDMER, G.; KUBAT, M. Learning in the presence of concept drift and hidden contexts. *Machine learning*, Springer, v. 23, n. 1, p. 69–101, 1996.

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

WOODS, K.; BOWYER, K.; JR, W. P. K. Combination of multiple classifiers using local accuracy estimates. In: IEEE. *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*. [S.l.], 1996. p. 391–396.

YU, L.; LIU, H. Feature selection for high-dimensional data: A fast correlation-based filter solution. In: *ICML*. [S.l.: s.n.], 2003. v. 3, p. 856–863.

YUE, S.; GUOJUN, M.; XU, L.; CHUNNIAN, L. Mining concept drifts from data streams based on multi-classifiers. In: IEEE. *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*. [S.l.], 2007. v. 2, p. 257–263.

YULE, G. U. On the association of attributes in statistics: with illustrations from the material of the childhood society, &c. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, JSTOR, p. 257–319, 1900.

ZHANG, P.; GAO, B. J.; ZHU, X.; GUO, L. Enabling fast lazy learning for data streams. In: IEEE. *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. [S.l.], 2011. p. 932–941.

ZHANG, P.; ZHU, X.; SHI, Y.; GUO, L.; WU, X. Robust ensemble learning for mining noisy data streams. *Decision Support Systems*, Elsevier, v. 50, n. 2, p. 469–479, 2011.

ZHI, W.; GUO, H.; FAN, M.; YE, Y. Instance-based ensemble pruning for imbalanced learning. *Intelligent Data Analysis*, IOS Press, v. 19, n. 4, p. 779–794, 2015.

ZHOU, Z.-H. *Ensemble methods: foundations and algorithms*. [S.l.]: CRC Press, 2012.

ZHU, X.; WU, X.; YANG, Y. Dynamic classifier selection for effective mining from noisy data streams. In: IEEE. *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. [S.l.], 2004. p. 305–312.

ŽLIOBAITĖ, I. Combining time and space similarity for small size learning under concept drift. In: SPRINGER. *International Symposium on Methodologies for Intelligent Systems*. [S.l.], 2009. p. 412–421.

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*, Springer, p. 1–28, 2014.

ŽLIOBAITĖ, I.; RAUDYS, Š.; JUOZAPAVIČIUS, A.; TAMOŠIŪNAITĖ, M.; VAITKUS, P.; BASTYS, A.; ŽILINSKAS, J.; PRANEVIČIUS, H.; SIMUTIS, R. *Adaptive Training Set Formation*. Tese (Doutorado) — Vilniaus universitetas, 2010.

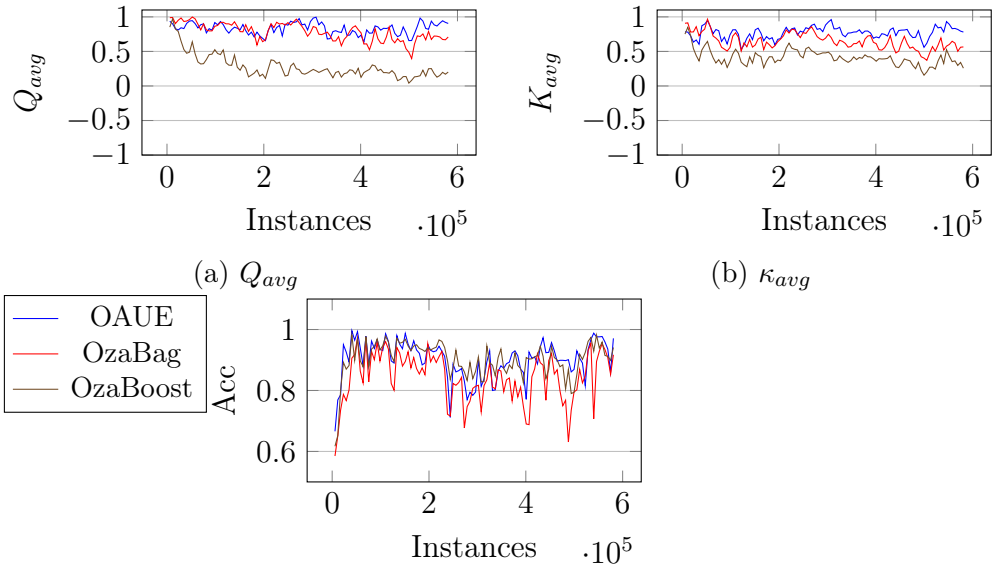
Appendix A

Diversity monitoring sample experiments

As a way to provide further understanding on how efficient different diversity inducing techniques are with respect to diversity maintenance we set up an experiment with one real multiclass dataset (COVT) and two synthetic generators with 1 million instances each, one containing two abrupt drifts (AGR) and the other without any drifts (RTG). To evaluate diversity we report Q_{avg} and κ_{avg} for every evaluation window, while accuracy is measured using prequential evaluation (GAMA; RODRIGUES, 2009). For these experiments we choose 3 algorithms that use different methods for inducing diversity: OzaBagging (vertical split and independent training), OzaBoosting (vertical split and dependent training) and Online Accuracy Updated Ensemble (vertical time-based split and independent training).

The results of the experiments are presented in Figures A.1, A.2 and A.3. In these experiments we observed that OzaBoosting (OZA, 2005) is closer to achieving a set of statistically independent classifiers ($Q_{avg} = 0$ or $\kappa_{avg} = 0$) than OzaBagging or OAUE. Also, right after a drift has happened (around instances 3.3×10^5 and 6.6×10^5 in Figure A.3) the Q_{avg} and κ_{avg} value decreases for all algorithms, but more dramatically for OAUE. This behavior is attributable to OAUE update operators, such that after a drift models might be replaced by others that are more adapted to the current concept, which in turn enhances the overall diversity.

Unfortunately, it is impossible to draw strong conclusions (e.g. find strong relationships between diversity and accuracy) from experiments that use the same protocol that we employed in this section (KUNCHEVA et al., 2003). Although, individuals using (or developing) ensemble-based algorithms for data streams may find it useful to use this experimental protocol to observe how diversity measures changes as a result of a drift, or to verify if the diversity inducing strategy is working as expected.

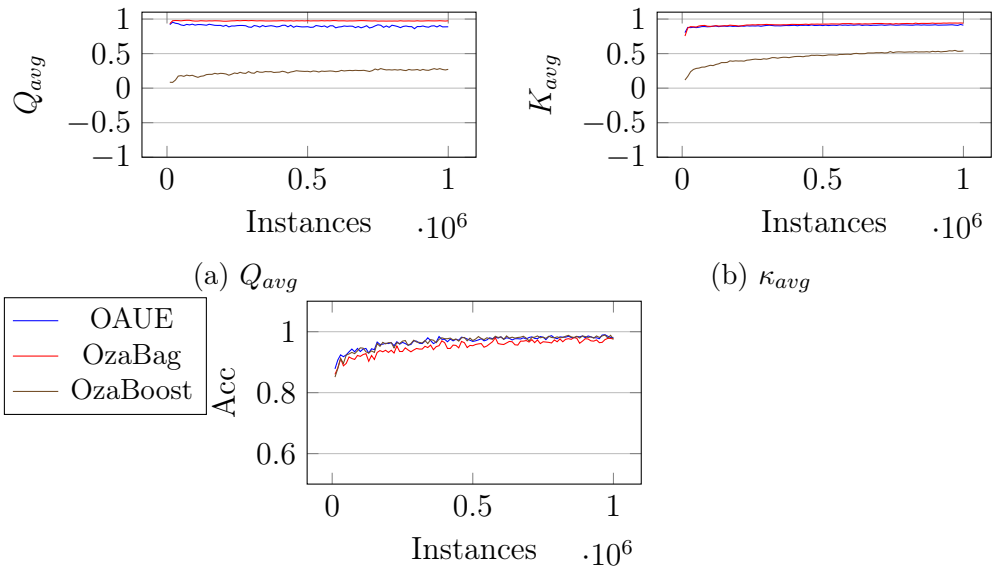


(a) Q_{avg}

(b) κ_{avg}

(c) Prequential Accuracy

Figure A.1: Q_{avg}, κ_{avg} and Prequential Accuracy for OAUE, OzaBag and OzaBoost on the real dataset Covertype (Multiclass)

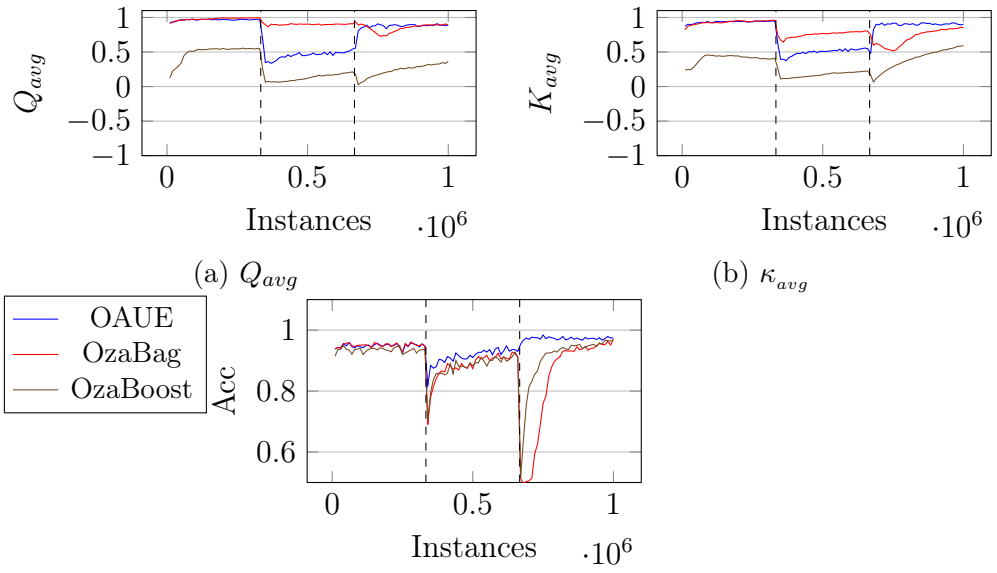


(a) Q_{avg}

(b) κ_{avg}

(c) Prequential Accuracy

Figure A.2: Q_{avg}, κ_{avg} and Prequential Accuracy for OAUE, OzaBag and OzaBoost on the dataset RTG (No Drifts)



(a) Q_{avg}

(b) κ_{avg}

(c) Prequential Accuracy

Figure A.3: Q_{avg}, κ_{avg} and Prequential Accuracy for OAUE, OzaBag and OzaBoost on the dataset AGR2 (2 Abrupt Drifts: dashed vertical lines)

Appendix B

All plots of the combination methods in SAE2

The following plots emphasize the experiments reported and discussed in Section 5.4, such that we observe that the differences between SAE2-c and SAE2-f are not too different, yet both methods are closer to the line representing the Best possible combination and far from the Worst combination in all plots.

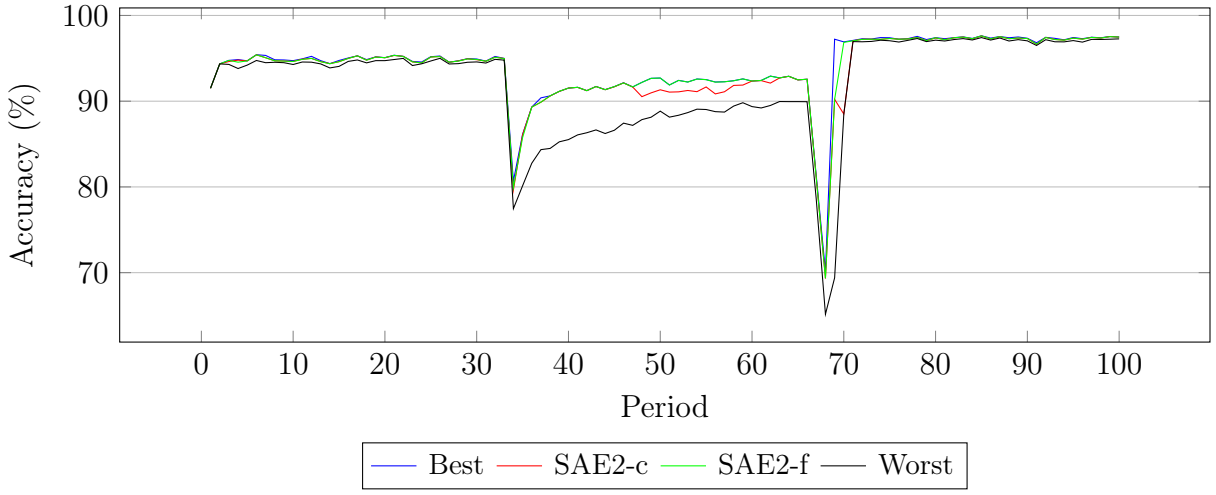


Figure B.1: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AGR1, max classifiers = 4

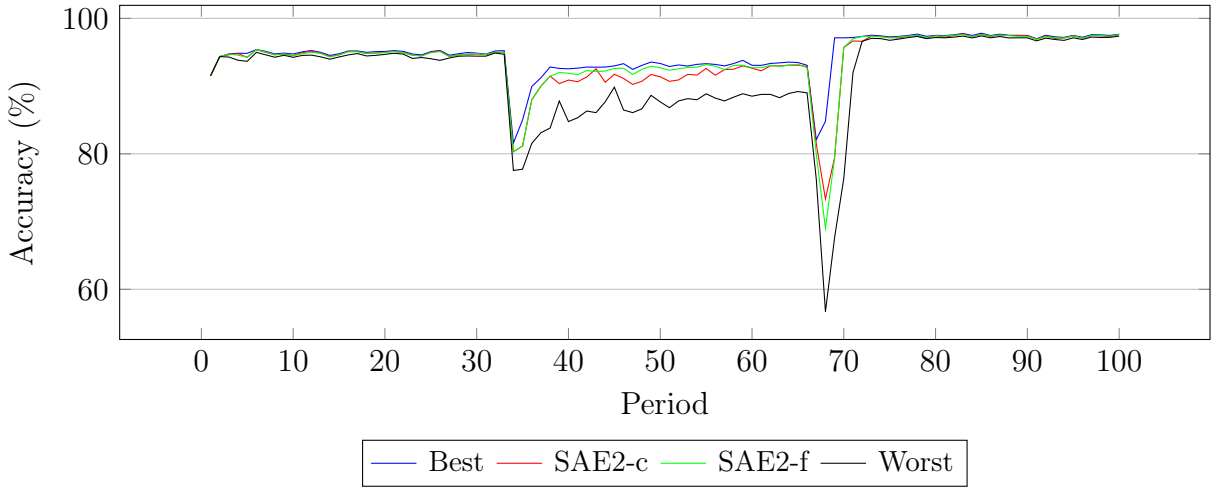


Figure B.2: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AGR1, max classifiers = 5

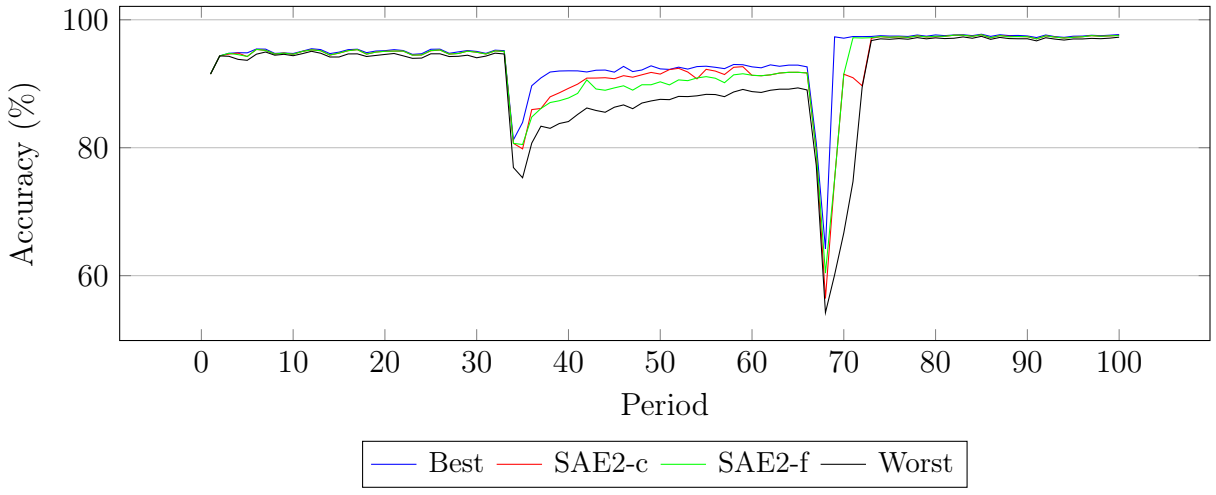


Figure B.3: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AGR1, max classifiers = 6

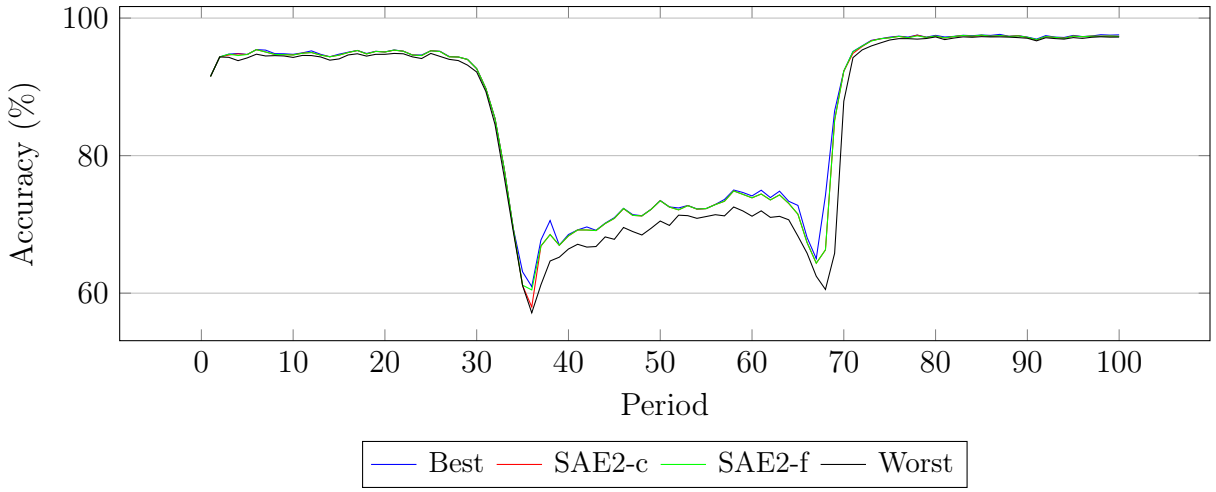


Figure B.4: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AGR2, max classifiers = 4

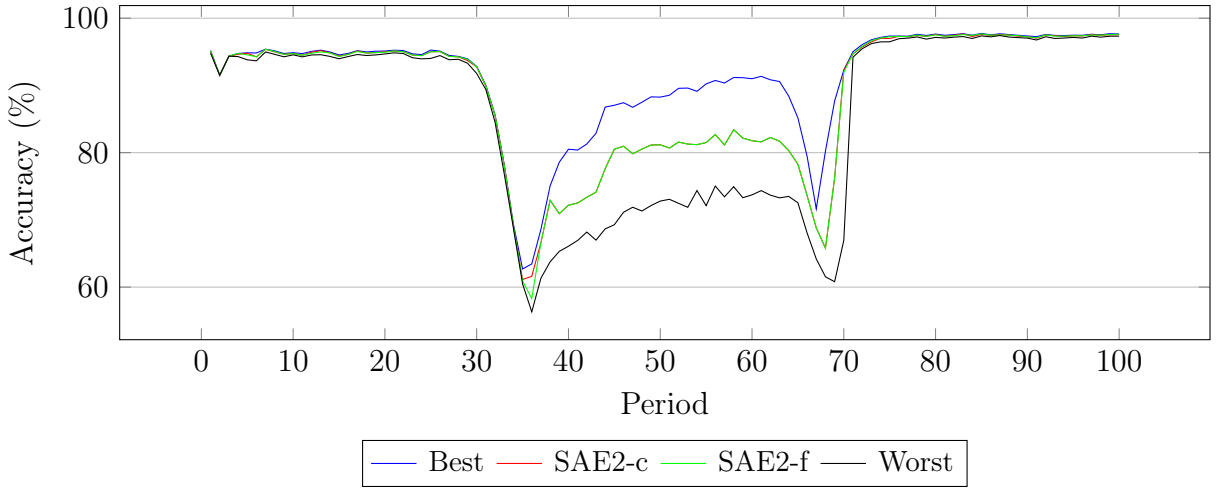


Figure B.5: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AGR2, max classifiers = 5

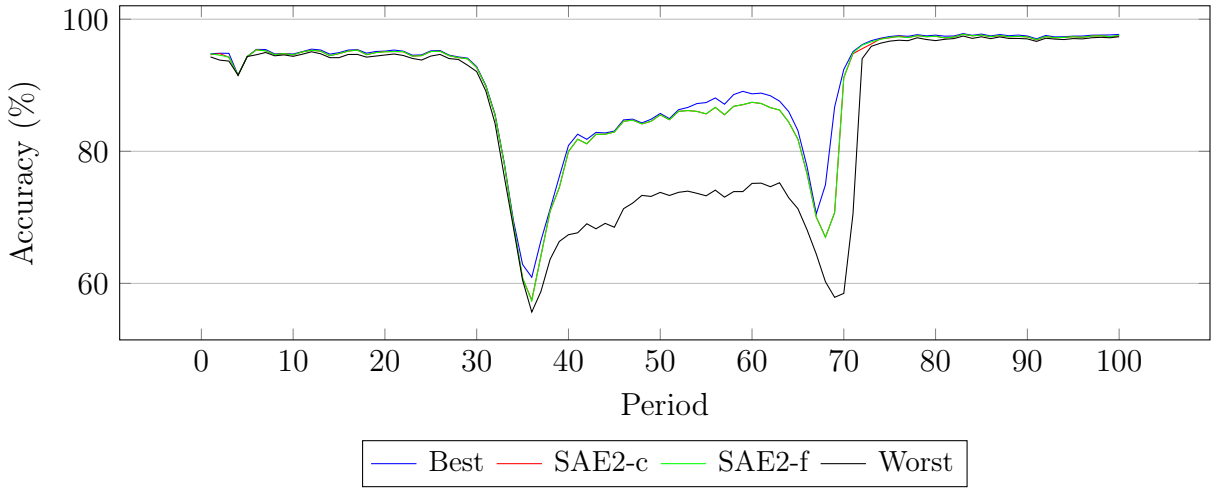


Figure B.6: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AGR2, max classifiers = 6

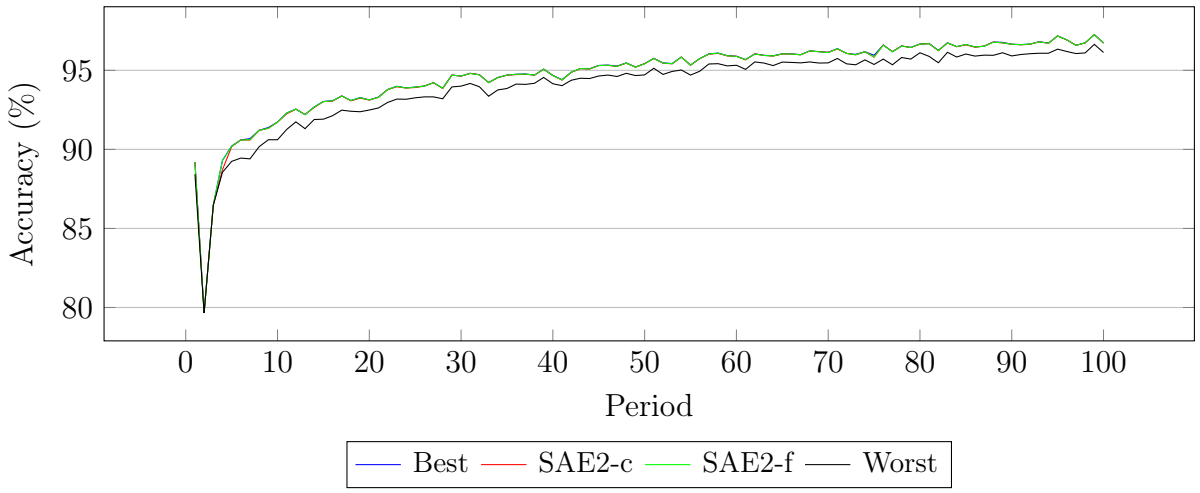


Figure B.7: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset RTS, max classifiers = 4

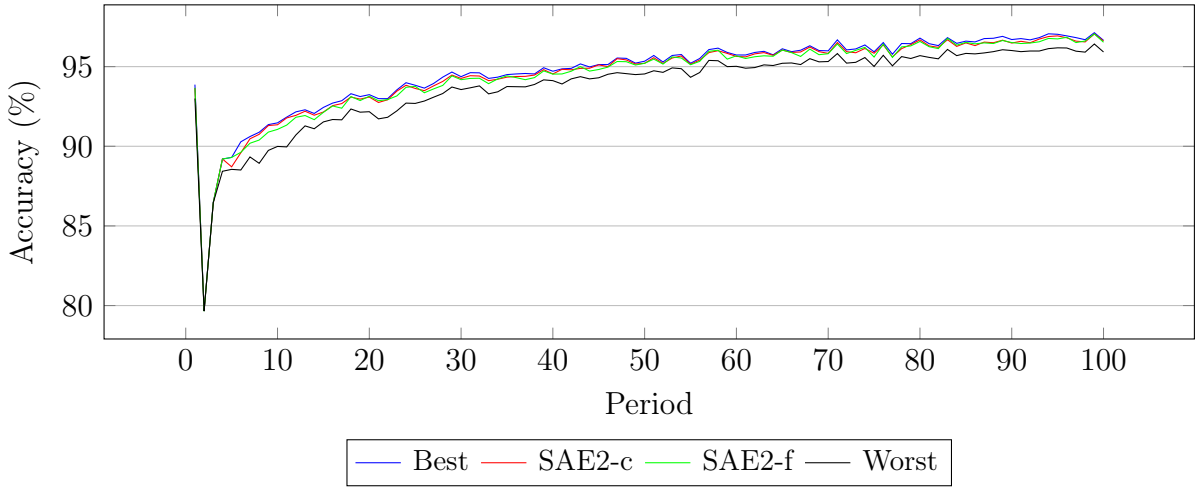


Figure B.8: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset RTS, max classifiers = 5

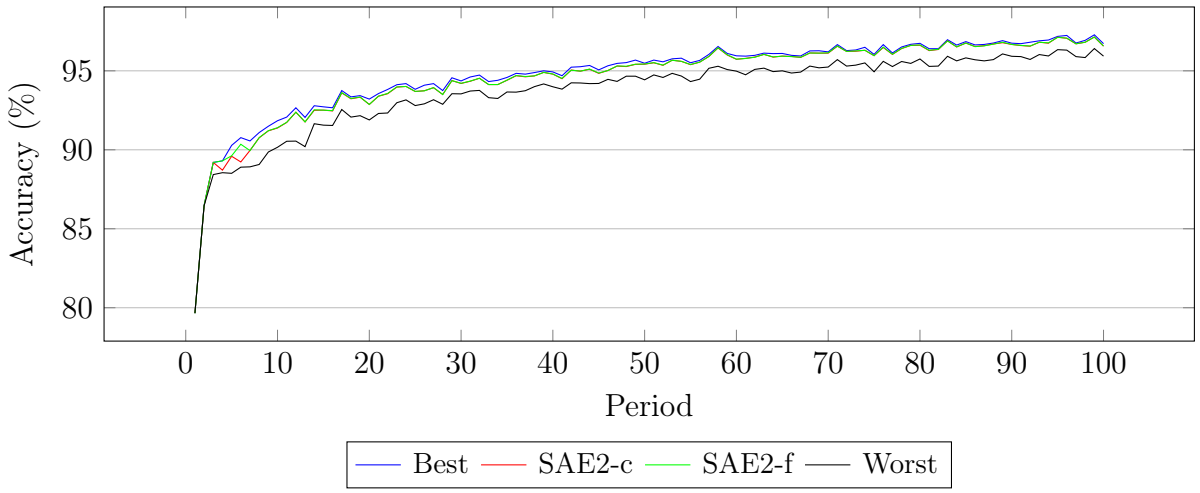


Figure B.9: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset RTS, max classifiers = 6

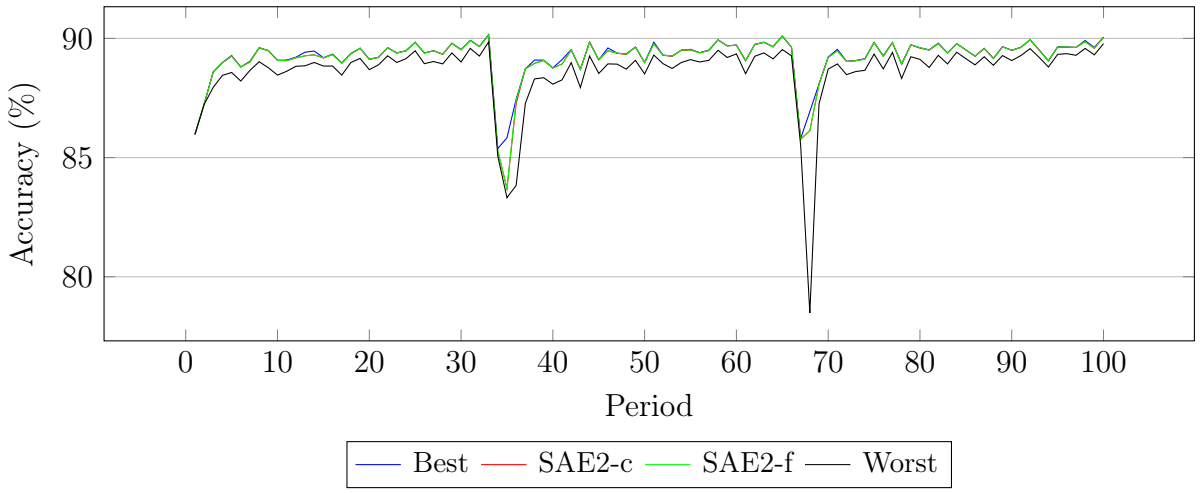


Figure B.10: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset SEA1, max classifiers = 4

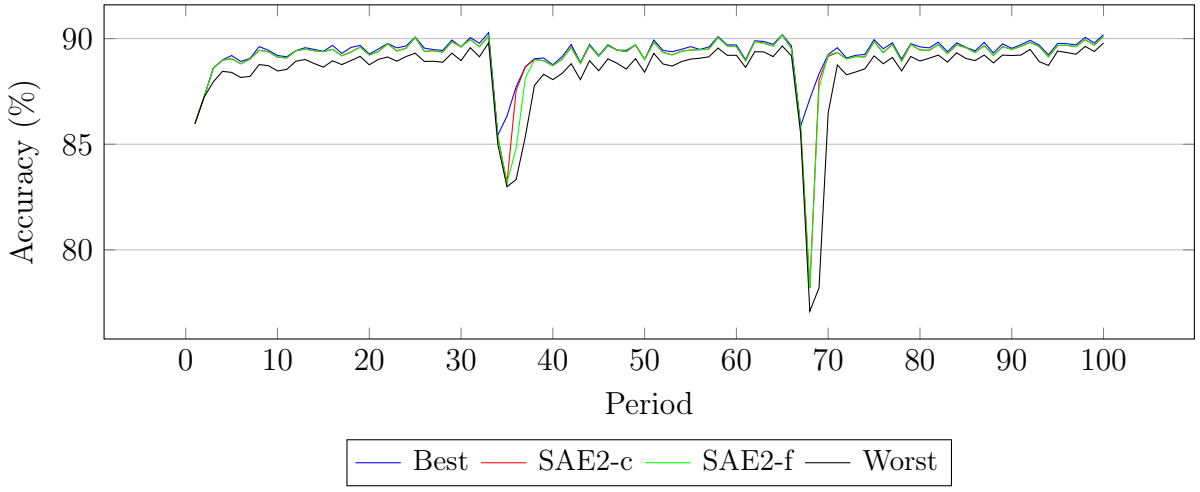


Figure B.11: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset SEA1, max classifiers = 5

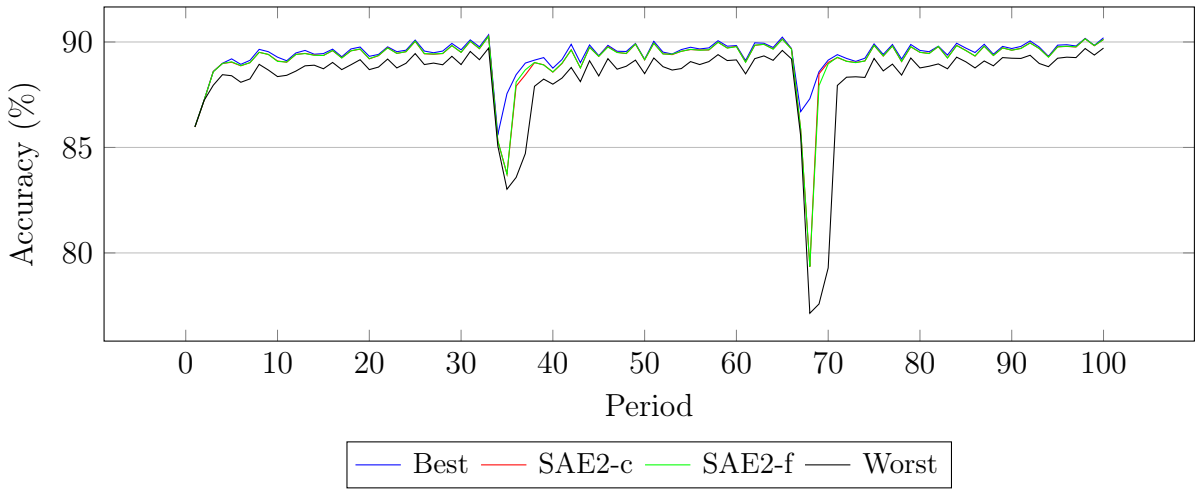


Figure B.12: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset SEA1, max classifiers = 6

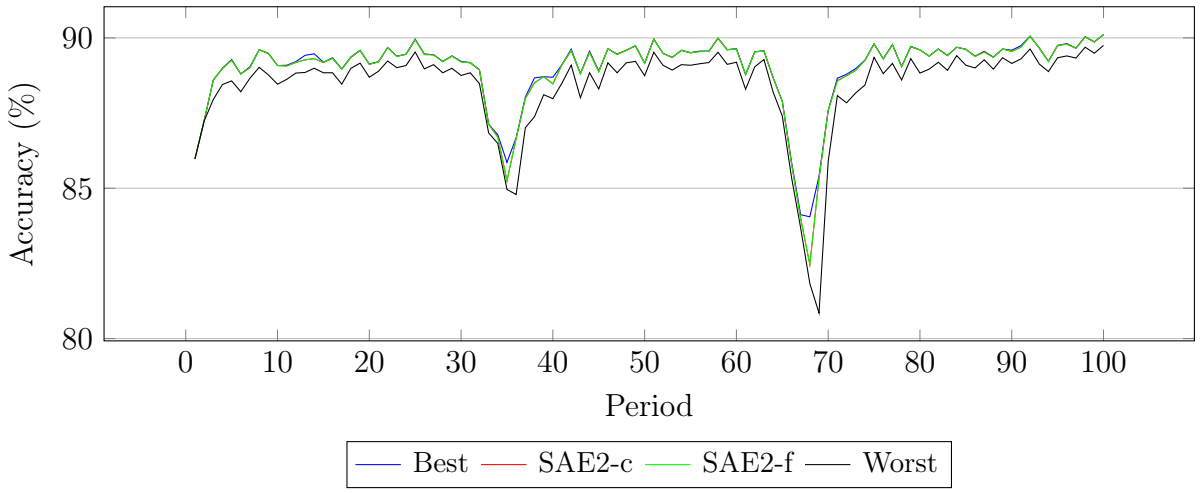


Figure B.13: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset SEA2, max classifiers = 4

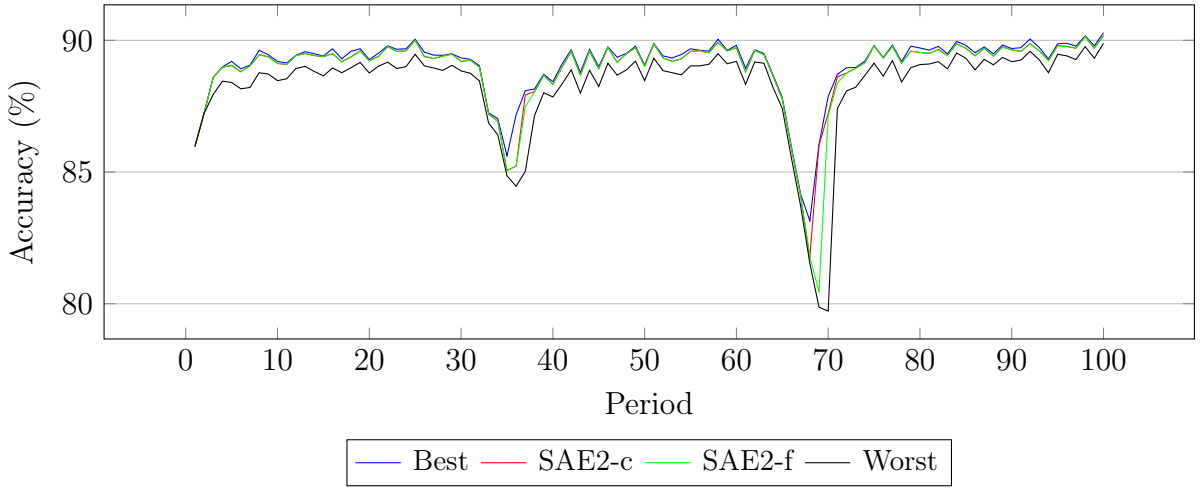


Figure B.14: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset SEA2, max classifiers = 5

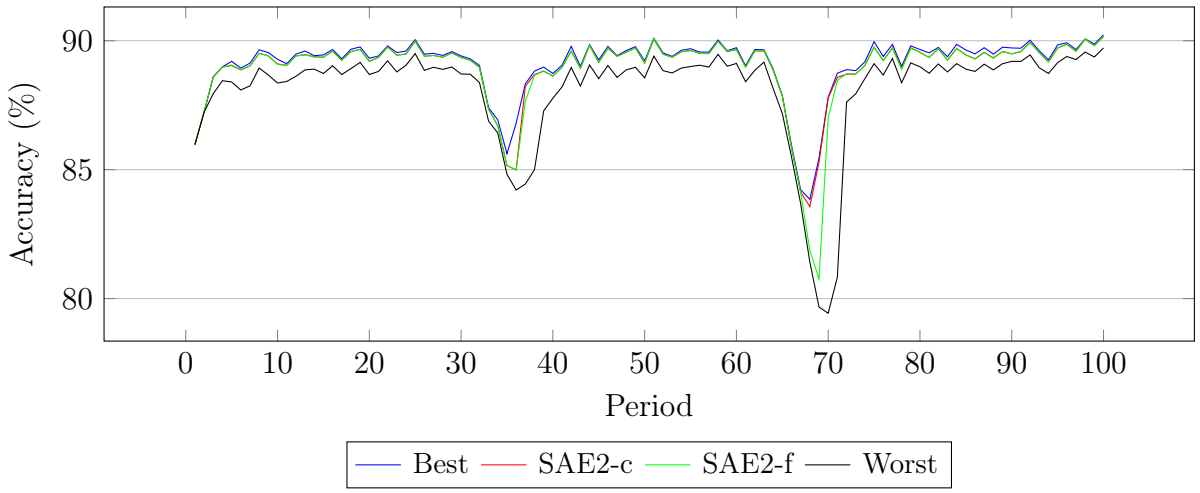


Figure B.15: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset SEA2, max classifiers = 6

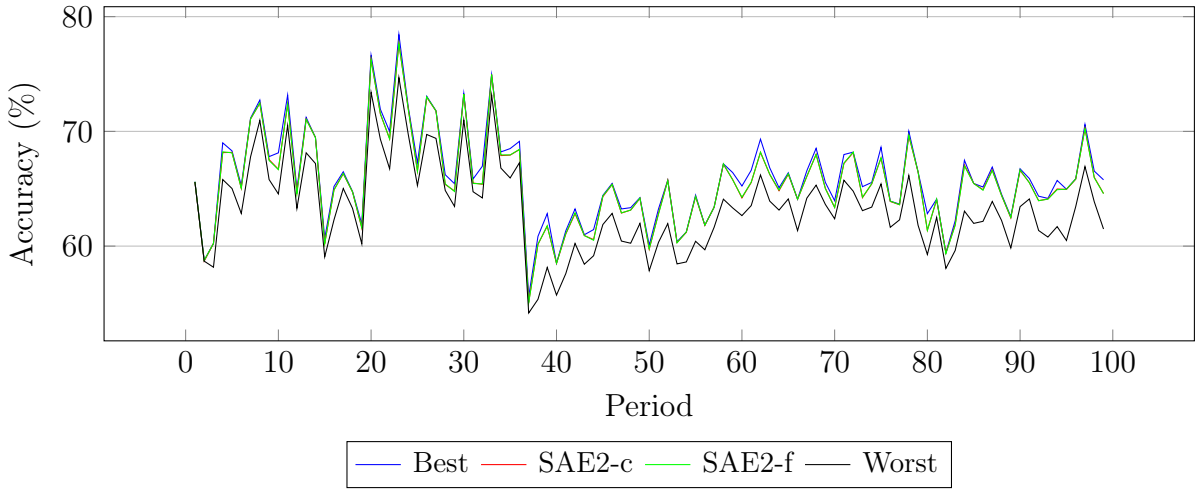


Figure B.16: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AIRL, max classifiers = 4

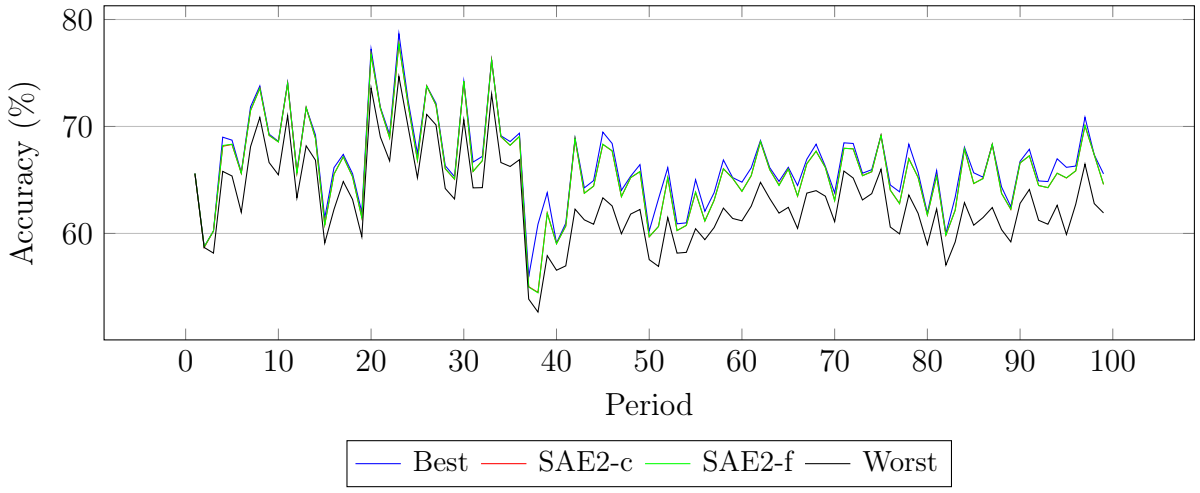


Figure B.17: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AIRL, max classifiers = 5

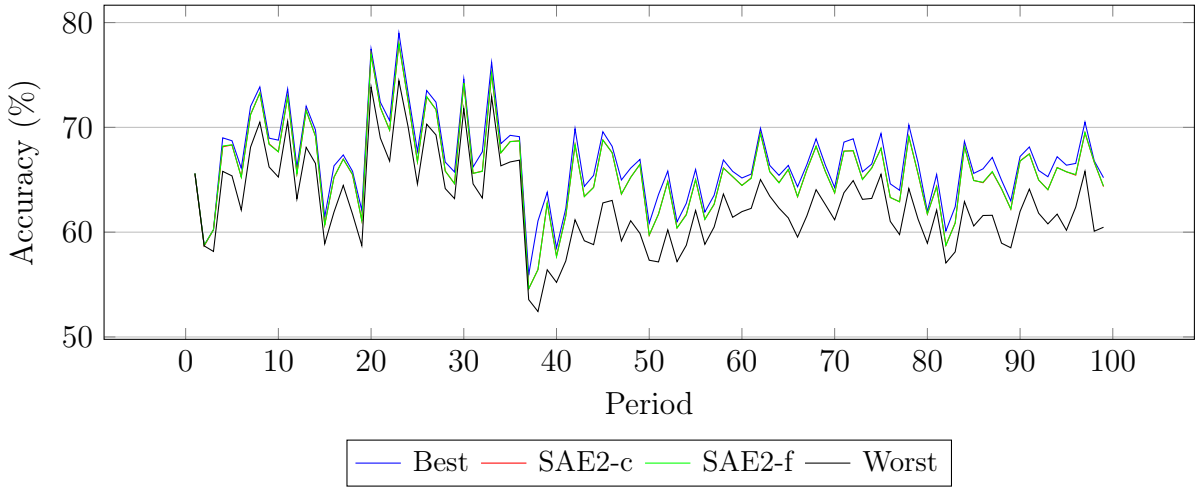


Figure B.18: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset AIRL, max classifiers = 6

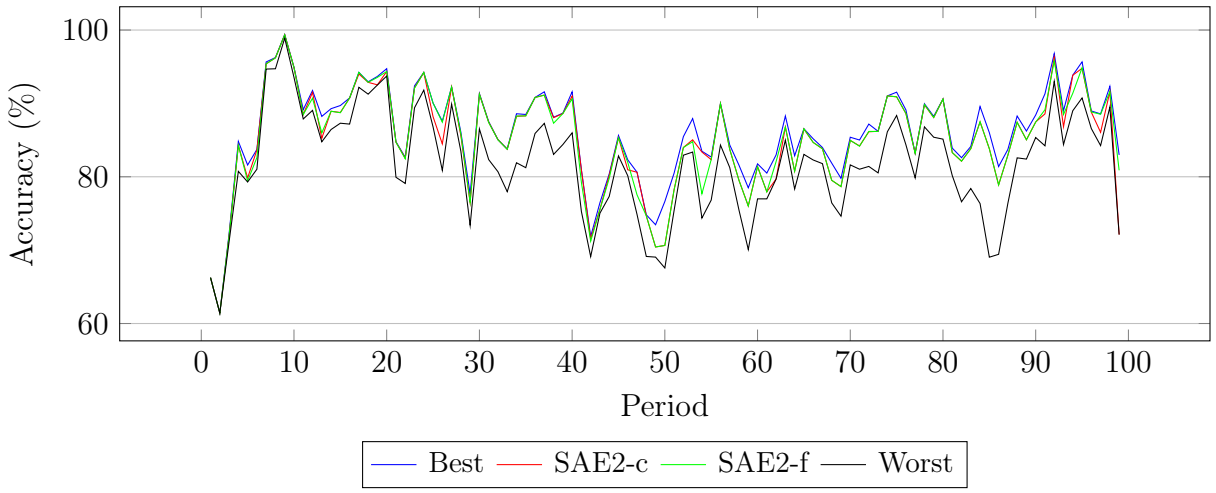


Figure B.19: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset COVT, max classifiers = 4

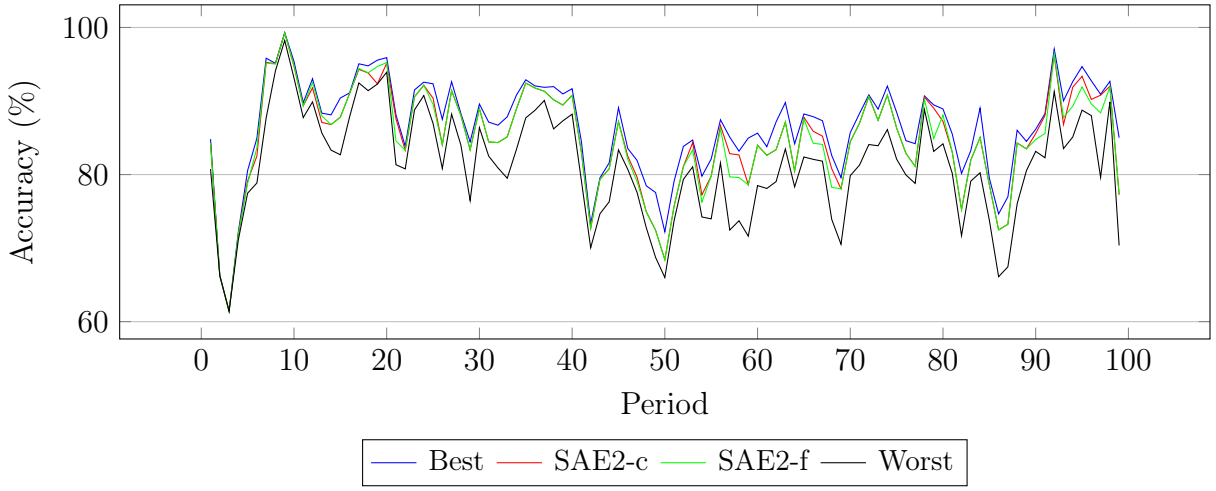


Figure B.20: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset COVT, max classifiers = 5

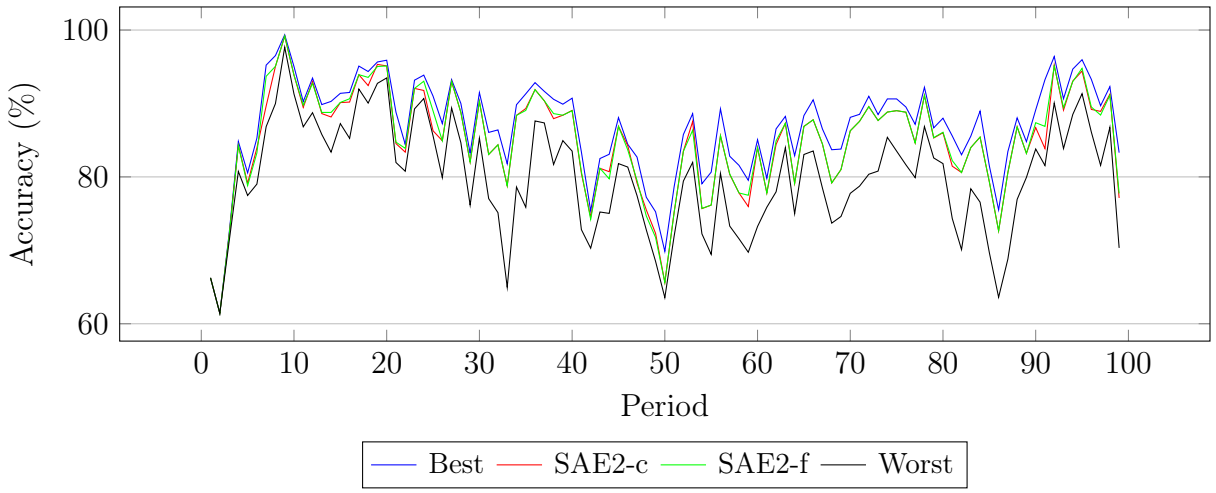


Figure B.21: Comparison between Best, Worst, SAE2-c and SAE2-f, dataset COVT, max classifiers = 6

Appendix C

CNE Drift and Warning Detection Exploration

In this Appendix we explore two different parametrisations of CNE_{jac30} used in Section 5.5. The experiments here use the same parameters for CNE_{jac30} (without any subset reset) and are identified as CNE_{fast} and CNE_{none}. CNE_{fast} uses higher confidence values ($\delta_d = 0.001$ and $\delta_w = 0.01$) in comparison to CNE_{jac30} ($\delta_d = 0.00002$ and $\delta_w = 0.0002$) for the drift and warning detector resulting in early detection of warnings and drifts and also increasing the false positives w.r.t detections. CNE_{none} does not use any drift or warning detection, thus it does not reset base models and never add background learners. Table C.1 presents a comparison in terms of accuracy of the three variations.

As can be observed in Table C.1, the version without any drift detection can only achieve slightly better results when there are no drifts (e.g. RTS dataset). CNE_{jac30} uses a more moderate approach to detect drifts, thus it outputs better results for gradual drift scenarios such as LED_g, SEA_g and AGR_g. Even though CNEfast*doobtainslightlybetterresultsinovera* parametrization in terms of drift/warning detection for the other experiments in this work, basically because CNE_{fast} also yield too many learners resets due to false positives, effectively increasing the execution time and memory used.

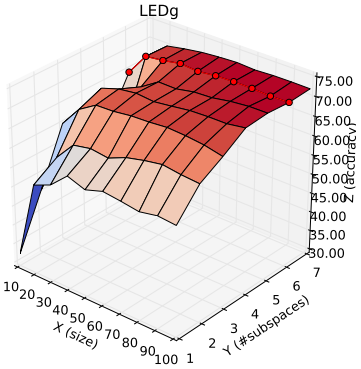
Table C.1: Accuracy - CNE_{jac30}, CNE_{fast} and CNE_{none}.

Dataset	CNE_{jac30}	CNE_{fast}	CNE_{none}
LED_a	**73.756**	73.59	70.064
LED_g	**73.11**	72.87	69.899
SEA_a	89.484	**89.5**	87.281
SEA_g	**89.066**	89.06	87.211
AGR_a	90.325	**90.38**	85.373
AGR_g	**86.649**	86.6	81.309
RTS	97.324	97.3	**97.497**
RBF_m	86.523	**86.83**	76.17
RBF_f	77.221	**77.82**	51.74
HYPER	85.287	**85.56**	78.846
Syn. Avg	84.874	**84.95**	78.539
Syn. Avg Rank	**1.6**	**1.6**	2.8
AIRL	64.962	65.02	**65.032**
ELEC	89.667	**90.5**	87.553
COVT	**95.147**	94.95	94.627
GMSC	93.549	**93.56**	93.549
KDD99	**99.961**	99.957	99.959
Real Avg	88.657	**88.79**	88.144
Real Avg Rank	1.9	**1.8**	2.3
Total Avg	86.135	**86.23**	81.741
Total Avg Rank	1.7	**1.67**	2.633

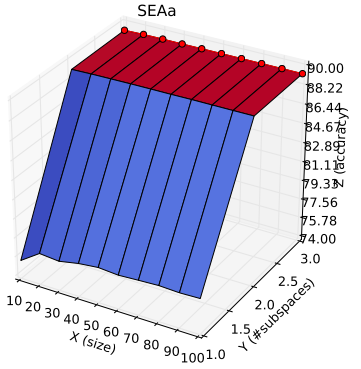
Appendix D

CNE Subspace Exploration

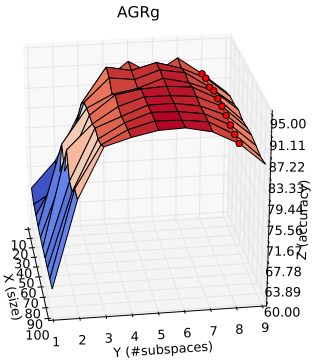
Figures D.1 and D.2 present an experiment where multiple executions of CNE were made varying the subspace m and ensemble size n. We can observe through these experiments that except for AIRLINES dataset it is advisable to use high values of m. Also, there is a small improvement by increasing m, yet using an appropriate m value has a higher impact in the overall classification performance.



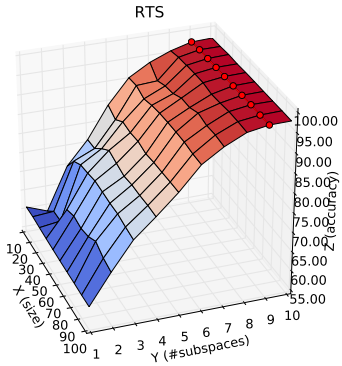
(a) LED_g



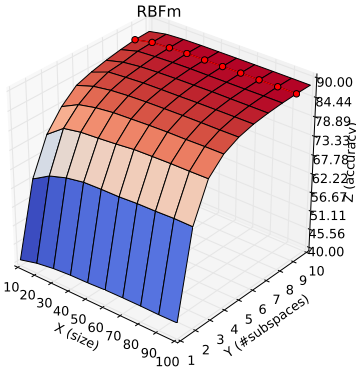
(b) SEA_a



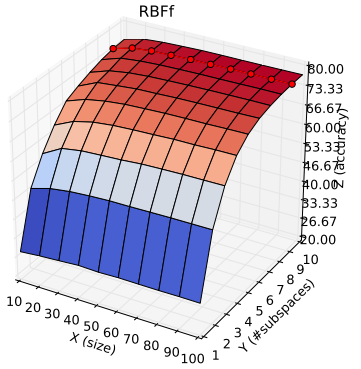
(c) AGR_g



(d) RTS

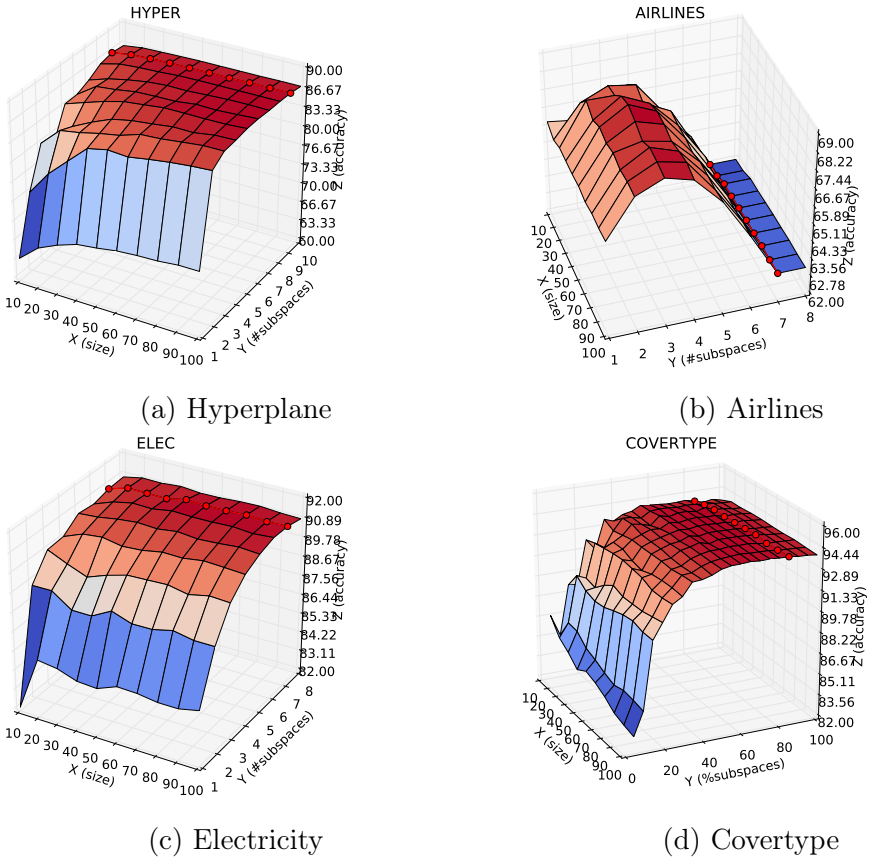


(e) RBF_m



(f) RBF_f

Figure D.1: CNE surfaces (LED_g, SEA_a, AGR_g, RTS, RBF_m and RBF_f): accuracy x ensemble size (n) x subspace size (m). Marked lines highlights $m = 85\%$



(a) Hyperplane (b) Airlines

(c) Electricity (d) Covertype

Figure D.2: CNE surfaces (Hyperplane, Airlines, Electricity, Covertype): accuracy x ensemble size (n) x subspace size (m). Marked lines highlights $m = 85\%$