UNIVERSIDADE FEDERAL DO PARANÁ

PAULO RICARDO LISBOA DE ALMEIDA

ADAPTING THE DYNAMIC SELECTION OF CLASSIFIERS APPROACH FOR CONCEPT DRIFT SCENARIOS

CURITIBA

2017

PAULO RICARDO LISBOA DE ALMEIDA

ADAPTING THE DYNAMIC SELECTION OF CLASSIFIERS APPROACH FOR CONCEPT DRIFT SCENARIOS

Thesis presented as partial requirement for the degree of Doctor. Graduate Program in Informatics, Sector of Exact Sciences, Universidade Federal do Paraná.

Supervisor: Luiz Eduardo S. de Oliveira

Co-supervisors: Alceu de Souza Britto Jr. and Robert Sabourin

CURITIBA

2017

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós Graduação em INFORMÁTICA
Código CAPES: 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Tese de Doutorado de **PAULO RICARDO LISBOA DE ALMEIDA**, intitulada: "**ADAPTING THE DYNAMIC SELECTION OF CLASSIFIERS APPROACH FOR CONCEPT DRIFT SCENARIOS**", após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.
A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 09 de Novembro de 2017.

LUIZ EDUARDO SOARES DE OLIVEIRA
Presidente da Banca Examinadora (UFPR)

DANIEL WEINGAERTNER
Avaliador Interno (UFPR)

ROBERT SABOURIN
Coorientador - Avaliador Externo (ETS)

PAULO RODRIGO CAVALIN
Avaliador Externo (IBM)

GEORGE DARMITON DA CUNHA CAVALCANTI
Avaliador Externo (UFPE)

RAFAEL MENELAU OLIVEIRA E CRUZ
Avaliador Externo (ETS)

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

To my wife, Camila

Acknowledgements

Agradeço aos meus pais, Maria e Paulo Almeida, e à minha irmã Natali Almeida, pelo apoio incondicional e ensinamentos que moldaram meu caráter.

O meu muito obrigado à minha esposa Camila Oliveira, por me mostrar que existem outras coisas além do peculiar mundo dos meus pensamentos e do trabalho em frente ao computador. Se ainda acredito que existe um mundo lá fora, devo muito disso à ela.

Agradeço aos meus orientadores, Professores Luiz Oliveira, Alceu Britto e Robert Sabourin, não só pelo fundamental incentivo e auxílio para a conclusão desse trabalho, mas também pelo exemplo de profissionalismo, o qual utilizo como guia em minha carreira. Agradeço especialmente ao Professor Alceu Britto, o qual me acompanhou durante toda minha carreira acadêmica, desde meu primeiro dia de aula de algoritmos, ainda na graduação, até a obtenção do meu título de Doutor.

Agradeço também aos membros da banca, os Doutores Paulo Cavalin, George Cavalcanti e Rafael Cruz, por aceitarem avaliar este trabalho, e por suas valorosas contribuições.

Agradeço à todos meus grandes amigos do Departamento de Informática da UFPR, pelos momentos de descontração e pelo diário café com bobagem que, contraditoriamente, muitas vezes me levou a pensar nos mais diversos temas relevantes.

Finalmente, por acreditar não ser possível citar a todos que participaram de minha vida e que, de alguma forma fizeram parte deste trabalho, estendo meu muito obrigado a todos que contribuíram para a conclusão desta Tese.

"Ao contrário", continuou Tweedledee, "se foi assim, poderia ser; e se fosse assim, seria; mas como não é, então não é. Isso é Lógico".

"Contrariwise", continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic".

(Lewis Carroll, Through the Looking-glass: And what Alice Found There, 1875)

Abstract

Many environments may suffer from distributions or *a posteriori* probabilities changes over time, leading to a phenomenon known as concept drift. In these scenarios, it is crucial to implement a mechanism to adapt the classification system to the environment changes in order to minimize any accuracy loss. Under a static environment, a popular approach consists in using a Dynamic Classifier Selection (DCS)-based method to select a custom classifier/ensemble for each test instance according to its neighborhood in a validation set, where the selection can be considered region-dependent. In order to handle concept drifts, in this work the general idea of the DCS method is extended to be also time-dependent. Through this time-dependency, it is demonstrated that most neighborhood DCS-based methods can be adapted to handle concept drift scenarios and take advantage of the region-dependency, since classifiers trained under previous concepts may still be competent in some regions of the feature space. The time-dependency for the DCS methods is defined according to the concept drift nature, which may define if the changes affects the *a posteriori* probabilities or the distributions only. By taking the necessary modifications, the Dynse framework is proposed in this work as a modular tool capable of adapting the DCS approach to concept drift scenarios. A default configuration for the Dynse framework is proposed and an experimental protocol, containing seven well-known DCS methods and 12 concept drift problems with different properties, shows that the DCS approach can adapt to different concept drift scenarios. When compared to state-of-the-art concept drift methods, the DCS-based approach comes out ahead in terms of stability, i.e., it performs well in most cases, and requires almost no parameter tuning.

Key-words: Pattern Recognition. Concept Drift. Virtual Concept Drift. Real Concept Drift. Ensemble. Dynamic Classifier Selection. Local Accuracy.

Resumo

Muitos ambientes podem sofrer com mudanças nas distribuições ou nas probabilidades *a posteriori* com o decorrer do tempo, em um problema conhecido como *Concept Drift*. Nesses cenários, é imperativa a implementação de algum mecanismo para adaptar o sistema de classificação às mudanças no ambiente a fim de minimizar o impacto na acurácia. Em um ambiente estático, é comum a utilização da Seleção Dinâmica de Classificadores (*Dynamic Classifier Selection* - DCS) para selecionar classificadores/ensembles customizados para cada uma das instâncias de teste de acordo com sua vizinhança em um conjunto de validação, onde a seleção pode ser vista como sendo dependente da região. Neste trabalho, a fim de tratar *concept drifts*, o conceito geral dos métodos de Seleção Dinâmica de Classificadores é estendido a fim de se tornar não somente dependente de região, mas também dependente do tempo. Através da adição da dependência do tempo, é demonstrado que a maioria dos métodos de Seleção Dinâmica de Classificadores podem ser adaptados para cenários contendo *concept drifts*, beneficiando-se da dependência de região, já que classificadores treinados em conceitos passados podem, em princípio, se manter competentes no conceito corrente em algumas regiões do espaço de características que não sofreram com mudanças. Neste trabalho a dependência de tempo para os métodos de Seleção Dinâmica é definida de acordo com o tipo de *concept drift* sendo tratado, que pode afetar apenas a distribuição no espaço de características ou as probabilidades *a posteriori*. Considerando as adaptações necessárias, o framework Dynse é proposto como uma ferramenta modular capaz de adaptar a Seleção Dinâmica de Classificadores para cenários contendo *concept drits*. Além disso, uma configuração padrão para o framework é proposta e um protocolo experimental, contendo sete Métodos de Seleção Dinâmica e doze problemas envolvendo *concept drifts* com diferentes propriedades, mostra que a Seleção Dinâmica de Classificadores pode ser adaptada para diferentes cenários contendo *concept drifts*. Quando comparado ao estado da arte, o framework Dynse, através da Seleção Dinâmica de Classificadores, se sobressai principalmente em termos de estabilidade. Ou seja, o método apresenta uma boa performance na maioria dos cenários, e requer quase nenhum ajuste de parâmetros.

Key-words: Reconhecimento de Padrões. Concept Drift. Concept Drift Virtual. Concept Drift Real. Conjunto de Classificadores. Seleção Dinâmica de Classificadores. Acurácia Local.

List of Figures

List of Tables

List of abbreviations and acronyms

ADWIN Adaptive Windowing

AO-DCS Attribute-Oriented Dynamic Classifier Selection

ASHT Adaptive-Size Hoeffding Tree

AUC Area Under Curve

AUE Accuracy Updated Ensemble

AWE Accuracy-Weighted Ensembles

CD KNORA Concept Drifts KNORA

CVFDT Concept-adapting Very Fast Decision Tree learner

DCS Dynamic Classifier Selection

DCEPU Dynamic Classifier Ensemble for Positive and Unlabeled text stream

DES Dynamic Ensemble Selection

DCS-LA Dynamic Classifier Selection by Local Accuracy

DCS-LA OLA DCS-LA Overall Local Accuracy

DCS-LA LCA DCS-LA Local Class Accuracy

DDD Diversity for Dealing with Drifts

DDM Drift Detection Method

DWM Dynamic Weighted Majority

DPSO Dynamic Particle Swarm Optimization

Dynse Dynamic Selection Based Drift Handler

EDDM Early Drift Detection Method

ECDD EWMA for Concept Drift Detection

EWMA Weighted Moving Average

FEDD Feature Extraction for Explicit Concept Drift Detection

GLVQ Generalized Learning Vector Quantization

ISVM Incremental Support Vector Machine

JIT Just-in-time

KNORA K-Nearest Oracles

K-E KNORA-ELIMINATE

K-EW KNORA-ELIMINATE-W

K-U KNORA-UNION

K-UW KNORA-UNION-W

KNN k-nearest neighbors

1NN One-nearest neighbor

LED Light-emitting Diode

LBP Local Binary Patterns

MDD MODL Drift Detection

MLP Multilayer Perceptron

MOA Massive Online Analysis

NEVE Neuro-Evolutionary Ensemble

NNPrune Nearest Neighbors Based Pruning

PDF Probability Density Function

PROC Precision-Recall Operating Characteristic

PSO Particle Swarm Optimisation

QIEA Quantum-inspired Evolutionary Algorithm

RBF Radial Basis Function

RCD Recurring Concept Drifts Framework

ROC Receiver Operating Characteristic

SEA Streaming Ensemble Algorithm

SOL Staged Online Learning

SPLL Semiparametric Log-Likelihood

SPRT Sequential Probability Ratio Test

VFDT Very Fast Decision Tree learner

SVM Support Vector Machine

UCI University of California, Irvine

WEKA Waikato Environment for Knowledge Analysis

List of symbols

B_t	A batch of data received at time t
CD	Critical Difference
D	Pool P maximum size ($\|P\| \leq D$)
f_c	Score factor of the classifier c kept by the Card Counting Pruning
G	Training/Testing set size used to simulate a virtual concept drift
k	Number of k nearest instances
l	Knora-Eliminate Slack Variable
M	Accuracy Estimation Window size ($\|W\| = M$)
P	Classifiers Pool
$P(\mathbf{x})$	Feature unconditional Distribution
$P(y)$	Class distribution (*a priori* probabilities)
$P(y\|\mathbf{x})$	*A posteriori* probabilities
Q	Labeled set used in the Dynamic Selection Methods
t	Moment in time
W	Accuracy Estimation Window of the Dynse Framework
$\mathbf{x}$	Feature Vector
y	Target class

Contents

1 Introduction

When dealing with classification problems, most methods assume that the environment is static. Under this assumption, a supervised classification method can be designed using historical labeled data, which is considered to contain all the information necessary, and then be used in the classification task of unlabeled instances during an indefinitely amount of time. Even thought the static environment assumption is valid in many applications, it can be unrealistic in some situations. In these scenarios the distributions or the *a posteriori* probabilities may change over time in a phenomenon called *concept drift*, requiring a mechanism to adapt the classifier to the changes in the environment. Some examples of applications that must consider this phenomenon are the intrusion, spam and fraud detection, medical decision support and climate data analysis [1, 2, 3].

To illustrate the concept drift problem, consider the social media networks. Since social media applications hold a vast collection of user images, they could employ these images to recognize user faces in future images. Under this scenario, some challenges may arise as a result of the fact that users' faces are constantly changing due to aging factors, or even through artificial aspects such as the use of makeup, beard growth/shaving, different haircuts, etc.. This scenario may therefore suffer from a concept drift, since images collected from the user in the past may not be suitable for recognizing him/her in the present.

An example of a person changing its face over time due to an aging factor is depicted in Figure 1, where we may ask how a learner trained with images collected when the subject was younger can recognize him in the present. A possible approach is to employ only the most recent labeled images (e.g. images from the last year only) to train the classifier. Nevertheless, this naive approach is not optimal, since it could discard important data from the past (e.g. the user face did not changed in the last 3 years), and it would still have problems when the user changes its face artificially in an "abrupt" fashion (e.g. the user stopped using a beard last week).

(a) 27 years ago (b) 21 years ago (c) Nowadays

Figure 1 – Images of the same person over the years.

In this work the Dynamic Classifier Selection (DCS) is studied as an alternative to deal with the concept drift problem. A DCS method basically involves selecting the best classifier/ensemble from a pool of classifiers P based on a local region of the feature space, usually defined as the neighborhood of the test instance in a validation set. The rationale behind this is the possibility that we may have a pool of classifiers specialist in different regions in the feature space, and the classifiers are chosen according to the test instance location. Thus, the DCS can be defined as region-dependent.

However, the region dependency alone is not sufficient for scenarios containing concept drifts, as will be demonstrated in the course of this work. Since under a concept drift scenario the problem evolves over time, the DCS must be adapted to be not only region-dependent, but also time-dependent. In the example of the Figure 1, a pool could have classifiers trained at different times, and each classifier could be specialist in a different region of the feature space. A DCS method provided with a time-dependency should then, for instance, be able to detect that any classifier specialized in the region of the feature space that define the colors can be a good candidate to classify the current test instances (e.g. the hair and eye colors of the user did not change over time). Nevertheless the DCS should be able to detect that the classifiers specialized in the skin texture trained in the past may not be suitable in the present.

In other words, when just part of the feature space has a change between different concepts (local changes), in this work it is hypothesized that most neighborhood DCS-based approaches, provided with a time dependency, can present a good adaptation to the changes, since it should be possible to detect the feature space areas that did not change, where a classifier from a past concept may still be used. The DCS approach can be also specially useful in scenarios where past concepts may reoccur in the future (recurrent concepts), where classifiers trained in the past concept may just be reactivated when a concept reoccurs.

The time dependency is considered in order to add new classifiers in the knowledge base (i.e. pool) over time as new supervised information becomes available, and to define the local region (i.e. neighborhood) of the test instances accordingly to the current concept. In this Thesis it is also hypothesized that the time dependency must be modeled accordingly to the concept drift nature, where the local region should be computed using a set containing only the latest supervised information received under changes in the *a posteriori* probabilities, or using a set that contains as much supervised information as possible under changes in the distribution $P(\mathbf{x})$. Another hypothesis is that, by correctly modeling the time dependency when computing the local region of the test instances, a pool containing as much classifiers as possible is beneficial under a concept drift scenario when using a DCS-based approach (i.e. no information discard in the pool is necessary to adapt to the new concepts).

The user face recognition problem is just a simple example of the concept drift phenomenon (which is used just for illustration purposes in the introduction of this work). In the course of this work the necessary adaptations to the DCS approach to deal with concept drifts are presented and discussed in order to create a general framework, named Dynamic Selection Based Drift Handler (Dynse) framework. The framework is put to test under a range of real world, and artificial well known benchmarks designed specifically to test methods that deal with concept drifts, where it is demonstrated that almost all of the tested DCS approaches can be adapted to concept drift problems, and generate results comparable to the state-of-the-art.

1.1 Problem Definition

Nowadays it is a common sense that in many problems the assumption that the environment is static and the training and testing data follows the same distribution is violated, thus leading to the concept drift phenomenon [4, 5, 6]. In these environments, where often the data arises in batches or in a stream fashion, methods based on a simple online learning strategy may not be suitable, since the knowledge acquired by the method over time may be conflicting, thus leading to a poor, or even unacceptable, classifier performance.

Under a concept drift scenario, several complexities often arise, such as the unpredictability, speed, severity, nature (e.g. a concept drift may affect only the distributions, of the *a posteriori* probabilities), presence of recurrences and many other properties of the concept drift. A DCS-based approach may be considered a natural answer for some of these problems, such as the severity of the concept drift (i.e. if the concept drift does not affect some region of the feature space, a DCS approach should be able to identify it), and the reactivation of old classifiers in the presence of recurrences when a concept reoccurs. Nevertheless, since a concept drift imposes that the environment can change over time, a DCS-based approach must be modified in order to add a time-dependency to make it possible to adapt to changes. Besides its popularity in stationary environments, just a few works use a DCS-based approach to deal with concept drifts, where in these works just a specific DCS-based method is used.

Thus, a general study on the time-dependency, neighborhood size definition and pool generation for DCS-based strategies when facing concept drift problems is still an open issue that, when dealt, may lead to a general DCS-based framework, capable to adapt a vast number of well known DCS-based methods to deal with most of the concept drift complexities, which may help the development of many systems that are inserted in non-stationary environments.

1.2 Objectives

In this work the DCS approach is proposed as an alternative to deal with the concept drift problem. The necessary adaptations to the DCS methods are discussed and presented as a general framework, named the Dynse framework. The framework must be flexible, where each adaptation made to deal with the concept drift problem must be presented as a interchangeable module (e.g. a module for the DCS strategy used, a module for defining the most recent information, a module for pruning classifiers, etc.). One of the most important adaptation that must be made in the DCS approach is the add of a time dependency, where new classifiers should be trained over time, and the local region should be computed considering only the current scenario (e.g. use only the latest supervised information to select the local region).

Thought this time dependency, it is expected that the knowledge discard (i.e. trained classifiers discard) should not be necessary to adapt to concept drifts. This can be an interesting behavior under a concept drift scenario, since past information may still be useful in the present in some areas of the feature space, where the DCS approach should detect the regions where the classifiers are still relevant. The past information can also become completely compatible with the current scenario in a phenomenon known as a *recurrent concept*. However, the knowledge discard must implemented in the framework to minimize computational resources consumption, like memory and CPU time.

The proposed framework must be able to deal with distribution changes (i.e. changes in $P(\mathbf{x})$) and changes in the *a posteriori* probabilities, both of any speed and extent. Changes in the *a priori* distributions (i.e. changes in $P(y)$) are not in the scope of this work. Besides the main objective, which is the creation of a DCS based framework to deal with concept drifts, the following secondary objectives should be accomplished by this work:

- To test the Dynse framework under concept drift scenarios using several well known DCS based methods. This test must be made in order to check the ability of the framework to adapt different DCS based approaches to deal with concept drifts.
- To assess the proposed framework in real world and artificial well known datasets found in the literature in order to verify the performance of the proposed method in a large variety of environments containing concept drifts of several natures.
- To check the impact of the classical classifiers pruning strategies when dealing with concept drifts using a DCS based approach. Besides that several authors already studied and also proposed pruning strategies for environments containing concept drift, there is a lack of evidence of the efficacy of those methods when using the Dynamic Classifier Selection. A pruning approach that keeps only the best performing

classifiers considering only the current concept may be a suboptimal solution, since the DCS should be able to take advantage of a plural pool of classifiers (e.g. classifiers trained under different concepts).

- To define the expected characteristics of a diverse pool for a DCS approach when facing a concept drift problem. Such a pool should contain classifiers trained under different concepts, and possibly covering different regions of the feature space. This pool is defined as a *Concept Diverse* pool in this work, and it may lead to better results when using a DCS approach to deal with concept drifts.

- To propose a default configuration of the proposed framework that must keep an acceptable performance under a wide range of concept drift scenarios. This is important since under some concept drift scenarios it may be difficult to acquire relevant data in order to fine tune each possible module of the proposed framework.

1.3 Challenges

The development of a DCS based framework that is capable to deal with different levels and types of concept drifts by adjusting its parameters and modules is the real challenge of this work. To fully accomplish this task, the following challenges had to be overcome in this work:

- To review the state-of-the-art in order to check the different concept drifts that could happen in an environment and verify how the authors deal with these concept drifts. This task can be specially difficult since the literature does not agree in many aspects and different works can even have conflicting information.

- To check the impact of changing/adjusting the framework modules under different concept drift scenarios to provide a guideline to researchers who wish to use the proposed Dynse framework in a specific concept drift scenario.

- To define a metric of diversity of classifiers based on the feature region of expertise of each classifier, considering also the concept used for the training.

1.4 Hypotheses

The main hypothesis of this work is that through a time dependency add to the DCS approach, any neighborhood-based DCS method can represent a natural answer to the concept drift problem, especially for problems where some portions of the feature space are not affected by the concept drift (local changes), and for recurrent scenarios. In this work is also hypothesized that the time dependency should be modeled according to the

nature of the concept drift problem (i.e. changes in the distribution or in the *a posteriori* probabilities) when using a DCS based approach. Another hypothesis of this work is that the DCS approach can benefit from a pool that is kept with as many classifiers as possible, where this pool may be contain classifiers specialists in different regions of the feature space and trained under different concepts.

1.5 Contributions

Besides the proposed framework, this work will contribute with the scientific community with the following items:

- The proposed framework was transformed in an open source project issued under the GNU General Public License 3 [7]. The framework is fully and freely available for anyone at https://web.inf.ufpr.br/vri/software/dynse/.
- A comprehensive description of the datasets used in the literature to test concept drift handling methods, and the main properties of the methods that use these datasets, which can make easier the selection of the datasets to test future approaches based on their characteristics.
- The proposal of a protocol to use the PKLot [8] dataset as a concept drift benchmark, which may help researchers to put their methods to test under a challenging real world scenario, where the distributions and the *a posteriori* information may change due to weather, camera position and capture environment (i.e. parking lot) changes.
- The original K-E method was improved in this work in order to add a mechanism to deal with noisy environments.
- The basic schematics of the proposed framework and experimental results present in this work was published in [9].

1.6 Document Structure

This work is further organized in six chapters. Chapter 2 contains the theoretical foundation about the DCS methods and concept drift problems, including the different properties of the concept drift problems and the most popular artificial and real datasets used to test concept drift handling methods. In Chapter 3 the state-of-the-art is described, where the methods proposed in the literature are categorized in several families, like window, gradual forgetting and ensemble based methods. Chapter 3 also contains a review about the state-of-the-art and a mapping between the most common datasets employed in the literature and the authors methods. The proposed framework is described and

discussed in Chapter 4 altogether with the definition of a *Concept Diverse* pool and the proposal of a pruning algorithm able to keep the pool concept diverse. In Chapter 4 a modification for the K-E for noisy environments is also proposed, and the experimental protocol is defined. Chapter 5 contains a series of experiments conducted in order to validate the proposed approach. Finally, the work conclusions are described in Chapter 6.

2 Theoretical Foundation

2.1 Concept Drift

In the pattern recognition field, a concept refers to the target variable that needs to be modeled (learned), which is often the class variable [10]. A concept drift may happen when there is a change in the *hidden context*, which denotes one or more features that could give the true and static description of the problem, but for some reason they are unknown or unobservable [6, 11, 12]. For instance, in an outdoor surveillance application concept drifts may occur due to weather and environment changes. In this case, the *hidden context* could contain information about climate conditions and objects positions in the background that would help the system to correctly classify objects, but due to some limitation, such information is unavailable. In a spam e-mail detection problem [13], the impossibility of modeling the *hidden context* becomes evident, since it would probably include information about the user mind state to track his preference changes.

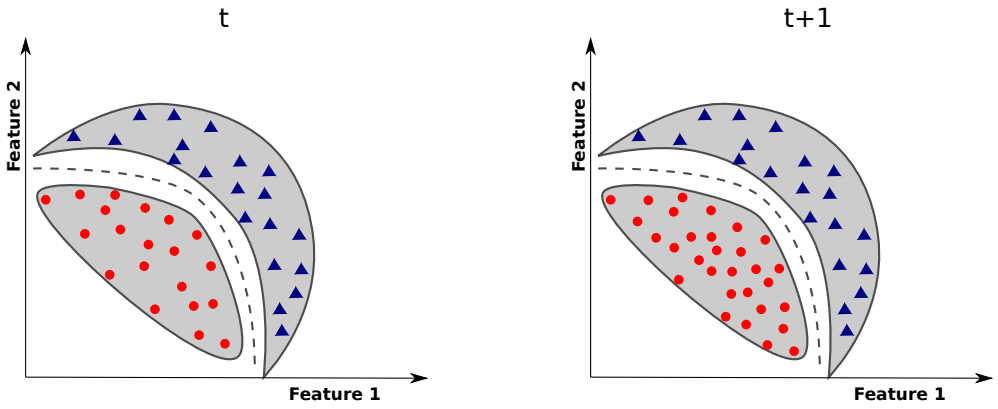
The outdoor surveillance and spam e-mail detection problems are just some examples of environments presenting concept drifts. In Section 2.2 the different types of concept drifts are described. The characteristics of the concept drifts are also explained in this work, including the concept drift speed (Section 2.3), recurrence (Section 2.4), and severity (Section 2.5). The data arrival forms for concept drift problems and datasets containing concept drifts commonly used in the literature are presented in Sections 2.6 and 2.7, respectively. Since this work proposes to modify the DCS approach to deal with concept drifts, in Section 2.8 the DCS idea for static environments is explained, altogether with some classical DCS methods. Finally, in Section 2.9 methods for comparing multiple classifiers over multiple datasets used in this work are explained.

2.2 Types of Concept Drifts

In a pattern recognition problem, an object can be described by a feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_L]$ containing L features, which is used to determine the object's class y by means of the *a posteriori* probabilities $P(y|\mathbf{x})$ [14]. For the analysis of the concept drifts in this work, also consider the features (unconditional) distribution as $P(\mathbf{x})$, the class conditional distribution as $P(\mathbf{x}|y)$, and the *a priori* probabilities as $P(y)$. Given these definitions, Subsection 2.2.1 describes the virtual concept drift phenomenon, which can be caused by changes in the features distribution $P(\mathbf{x})$ or in the class priors $P(y)$. Changes in the *a posteriori* probabilities $P(y|\mathbf{x})$, known as real concept drift, are discussed in Subsection 2.2.2.

2.2.1 Virtual Concept Drifts

In a virtual concept drift, the instances distribution may change over time, while the *a posteriori* probabilities remains unaltered. Figure 2 illustrates a virtual concept drift with class prior probabilities $P(y)$ changes, where the red class becomes more prevalent at time $t+1$. The problem illustrated in Figure 2 has equal costs associated to the misclassification of objects of all classes, causing no changes in the best boundary between times t and $t+1$, denoted by the gray dashed line. However, when dealing with cost sensitive problems, changes in $P(y)$ can become specially harmful [1, 15, 4, 16].

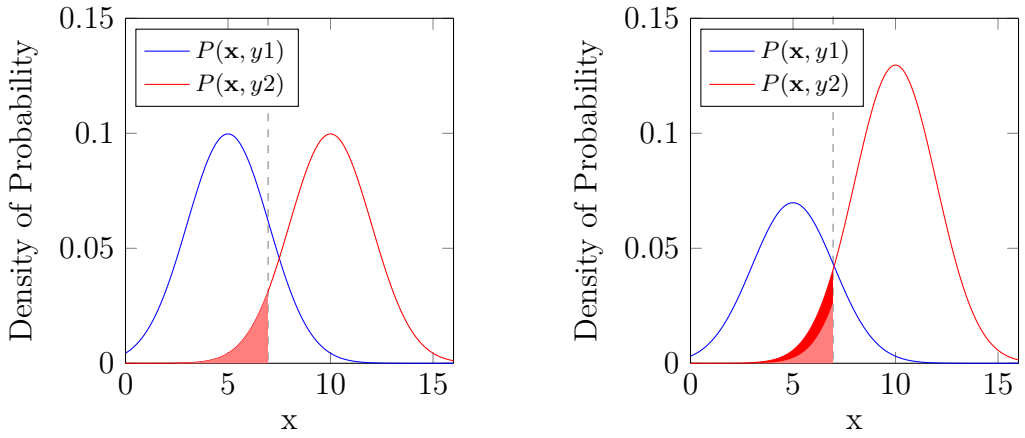


(a) Class distributions at time t (b) Class distributions at time t+1

Figure 2 – Class prior probabilities change in a virtual concept drift, where $P_t(y) \neq P_{t+1}(y)$. Adapted from [1].

To better illustrate the class prior changes impact, consider the distributions of a single feature with values contained in $\mathcal{N}(5, 1.8)$ for the blue class, and $\mathcal{N}(10, 1.8)$ for the red one, at both times t and $t+1$, in the cost sensitive problem depicted in Figure 3. In Figure 3a (time t), both classes are equiprobable and the threshold that separates them, represented by the gray vertical dashed line, is at position 7 in the feature axis. The red area in Figure 3a represents the proportion of the red class objects that will be classified as the blue class and, despite the problem being equiprobable, the threshold is displaced in order to decrease this area, considering that the misclassification cost of the red class is higher than the blue one. In Figure 3b (time $t+1$) the mean and standard deviations of both distributions remains unchanged, however the probability of finding objects of the blue and red classes are 35% and 65%, respectively. By keeping the threshold in the same position (i.e. keeping the same classifier unchanged), the probability of classifying a red object as a blue one is increased by the proportion shown in the dark red area.

The concept drift caused by the change in the distribution $P(\mathbf{x})$ between times t and $t+1$ is also called virtual. Figures 4a and 4b show a virtual concept drift caused by changes in the distribution $P(\mathbf{x})$ between times t and $t+1$. In Figure 4b $(t+1)$ it is possible to notice a change in the circle class boundary, denoted by the leftmost gray region. As in the class prior changes case, this concept drift should not alter the best

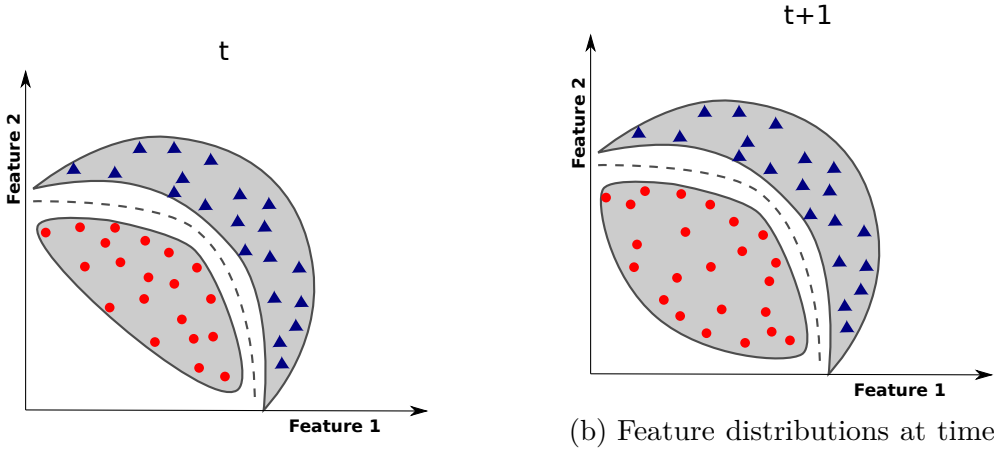


(a) Class distribution at time t

(b) Class distribution at time $t+1$

Figure 3 – Class prior probabilities change in an one feature cost sensitive problem. The dark red area in (b) denotes the extra probability of classifying a red class object as blue after the drift.

boundary in a cost insensitive equiprobable problem, however it may cause problems in cost sensitive methods, or force the classifiers remodeling to avoid performance losses due to inaccuracies when estimating the distributions at earlier times (i.e. some regions of the feature space were not known at time t)[1, 15, 4, 17, 18, 16].



(a) Feature distributions at time t

(b) Feature distributions at time $t+1$

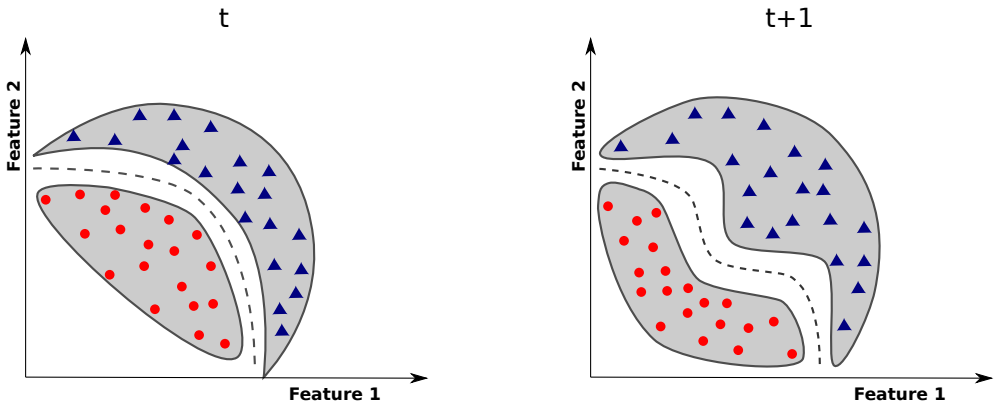
Figure 4 – Distributions change in a virtual concept drift, where $P_t(x|y) \neq P_{t+1}(x|y)$. Adapted from [1].

Even though changes in class priors and feature distributions will be referred as a virtual concept drift in this work due to its popularity [19, 20, 17, 12, 1, 15, 16], there are some equivalent nomenclatures in the literature like *Population Drift* [21, 20, 15, 18], *Covariate Shift* [4, 22], *Temporary Drift* [23], *Sampling Shift* [24] and *Feature Change* [25].

2.2.2 Real Concept Drifts

A real concept drift happens when the *a posteriori* probabilities $P(y|\mathbf{x})$ change over time, with or without changes in $P(\mathbf{x})$ or $P(y)$. In this kind of concept drift the relation between the target classes and the feature vectors may change over time, as in the spam e-mail filtering problem described in [26], where an e-mail represented by its feature vector $\mathbf{x_e}$ can belong to the class "spam" at a given time t, and can belong to the class "non-spam" at a time $t+1$ due, for instance, user behavior changes [15, 1, 4, 16].

To better illustrate this concept drift behavior, consider the two class problem in Figure 5. It shows a *a posteriori* probabilities change, causing a modification in the problem boundaries between the times t and $t+1$, showed in dashed lines in Figures 5a and 5b, respectively. The frontiers change forces an update in the classifier, since the boundaries learned by it at time t become obsolete at time $t+1$, which may cause strong accuracy drops.



(a) Boundaries at time t (b) Boundaries at time $t+1$

Figure 5 – Boundaries change in a real concept drift, where $P_t(y|\mathbf{x}) \neq P_{t+1}(y|\mathbf{x})$. Adapted from [1].

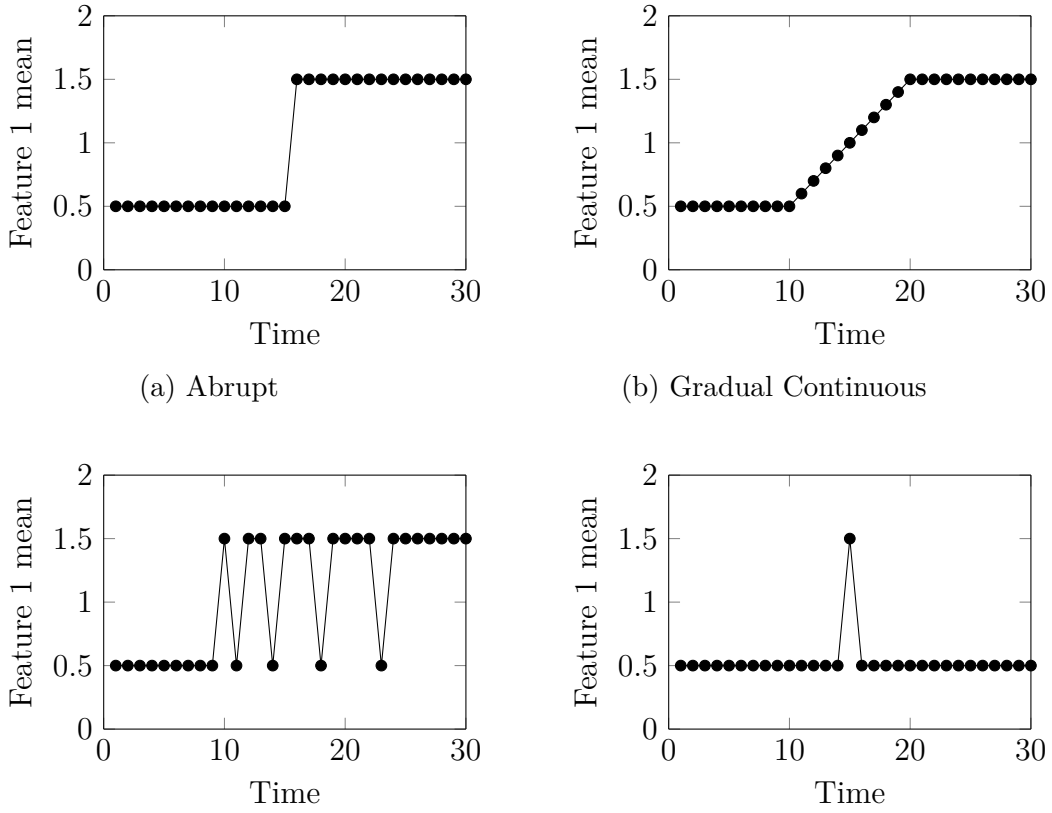
Real concept drifts can also be referenced in the literature as *Concept Shift* [4, 27], *Concept Substitution* [27, 28], *Conditional Change* [25] or simply *Concept Drift* [29, 30, 11]. In this work the term real concept drift is used to describe the change in the *a posterior* probabilities since it is widely employed [1, 2, 15, 20, 6, 17, 16], and authors may disagree with the use of other terms like the *Concept Shift* which is used as a real concept drift synonym in [4], and represents real concept drifts combined with abrupt changes in [27].

2.3 Concept Drift Speed

In some cases, the new concept will take place abruptly when, for instance, the user changes its face by shaving its beard in the user face recognition problem (Chapter 1). In other scenarios, the concept can change incrementally or gradually, taking several steps to the new concept take place, thus creating a period of uncertainty between stable

states (e.g. a sensor that wears off and start to lose accuracy gradually [15]). Formally, the concept drift speed can be defined as the inverse of the number of steps taken for a new concept completely replace the old one [1, 31, 16].

Figure 6 exemplifies different concept drift speeds over time in a one dimensional problem. In Figure 6a the feature mean changes at once, characterizing an *abrupt* concept drift. Figure 6b represents a *gradual continuous*, or *incremental*, concept drift, where the population moves to the new concept gradually, adding small changes at every step until the concept stabilizes. A *gradual probabilistic*, or *Continuous*, concept drift is represented in Figure 6c, where it is possible to find objects from both concepts when the new one starts to take place. The probability of finding an object from the old concept decreases over time until the change is complete [15, 31, 32, 33]. Although Figures 6b and 6c distinguish between two different *gradual* concept drifts, many authors [1, 2, 34, 27, 28, 35] do not make this differentiation, considering both Figures examples of a *gradual* concept drift.



(a) Abrupt

(b) Gradual Continuous

(c) Gradual Probabilistic

(d) Outlier - No concept drift

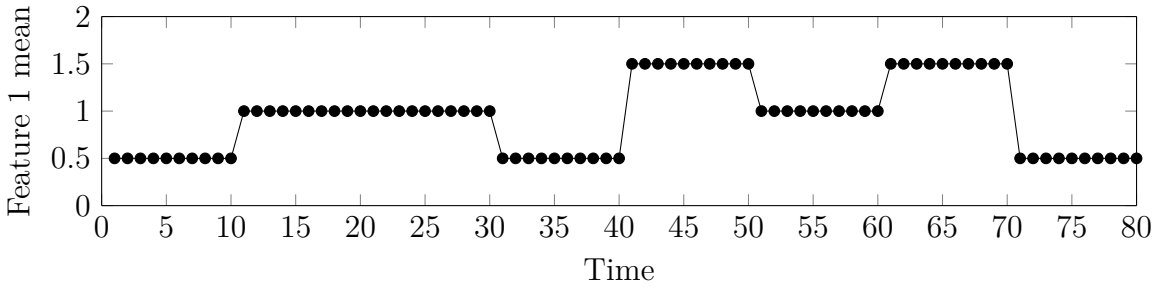
Figure 6 – Concept drift speeds. Figures a, b and c represents Abrupt, Gradual Continuous and Gradual Probabilistic concept drifts, respectively. Figure d represents an outlier, that should not be interpreted as a concept drift. Adapted from [5].

Finally, an outlier, that should not be regarded as a concept drift, is exemplified in Figure 6d, and represents one of the main challenges in concept drift problems, which is distinguishing drifts from outliers [15]. This problem is related to the *stability-plasticity dilemma*, which asks how an entity capable of learning (e.g. the human brain or a machine

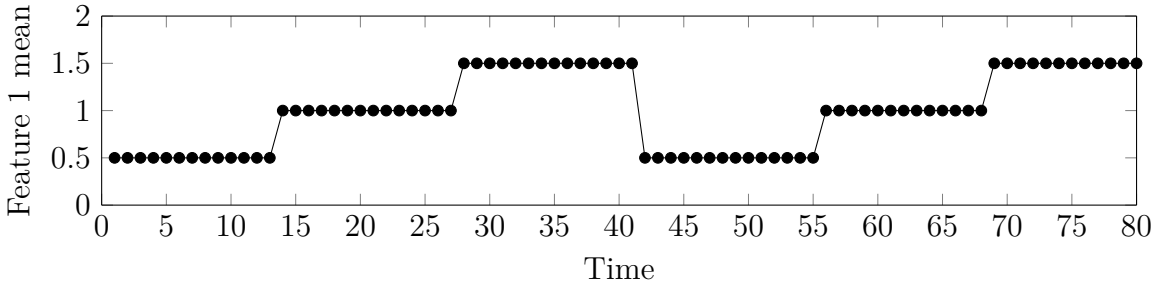
learning method) can be stable to irrelevant events, like outliers, maintaining the useful information, and still be able to learn with new relevant information [36, 6, 27].

2.4 Concept Recurrence

Many applications may have a concept recurrence, where old concepts may reoccur in the future [15, 31, 1, 33, 16]. Recurrent concepts are often related to seasonal changes, e.g., an application that must detect people in an outdoor environment, that needs to adapt to a new concept due to the snow in the winter season, and return to the old concept in the spring. The recurrence can also be classified as cyclic, when the concepts repeat in an ordered manner, or unordered (acyclic) [31, 16]. Keeping the information (i.e. trained models or training samples) acquired in older concepts when dealing with recurrent concepts can be a good practice, since this knowledge can be reused or refined when the concept reoccurs.



(a) Recurrent Acyclic Concept Drift



(b) Recurrent Cyclic Concept Drift

Figure 7 – Cyclic versus acyclic recurrent concept drifts. In (a) the drifts reoccur in an unordered manner, while in (b) the concepts follows a cyclic pattern.

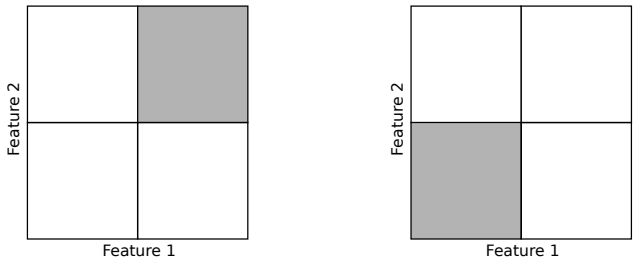
Figure 7 exemplifies recurring concepts in a one dimensional toy problem containing three different concepts, where the data points in the plots represents the feature mean. In Figure 7a the recurrence is random, while Figure 7b represents cyclic recurrent concepts, since the concepts repeats always following the same order. In Minku et al.[31], the different types of drifting sequences are further categorized according to their predictability (random

or predictable) and according to its frequency (periodic for drifts that happens for every t times, and nonperiodic otherwise).

2.5 Concept Drift Severity

Minku et al.[31] categorizes a concept drift based on its severity. A concept drift is considered *Severe* if most of the classes change their labels in the next concept, otherwise the drift can be considered *Intersected*. A similar definition is used in [12, 37], where a change in a sub-region of the instance space is called a *local concept drift*. Figure 8 represents a two dimensional classification problem, where the feature space was divided in four equally sized areas. The gray areas represent objects classified as the positive class, and the white ones represent the negative class. Using Figure 8 as an example, two different forms of measuring the drift severity can be considered [31]:

- Percentage of classes that changed its labels between the old and new concepts. Considering the drift between Figures 8a and 8b, 50% of input space had its target class changed.
- Consider the maximum percentage of a class that had its target concept changed between the times t and $t + 1$. Considering the drift between 8a and 8b, the positive class (gray area) has 100% of its target concept changed, while the negative class changed approximately 33% of its target concept. Considering this measure, the maximum percentage of concept change would be 100%.



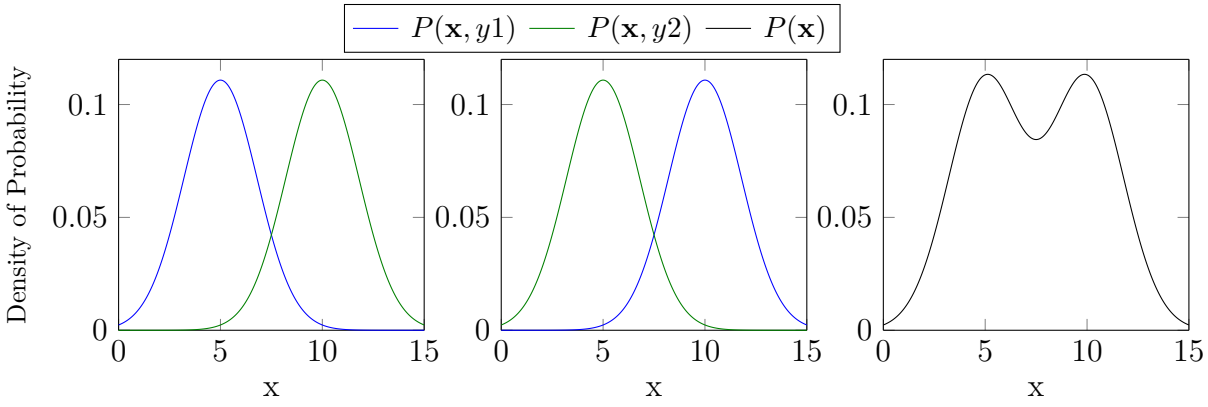
(a) Time t (b) Time $t + 1$

Figure 8 – Example of intersected drift in a 2D problem. The gray area represents the negative class objects while the white areas represents positive ones. Figure adapted from [31].

Also according to Minku et al.[31], changes in the unconditional and class-conditional Probability Density Functions (PDFs), which defines the Feature Severity, could be tracked using, for instance, the difference between the areas of the old and new concepts unconditional PDFs, or calculating the percentage of the input space that has its probability modified [32].

2.6 Data Arrival Forms

In order to detect and adapt to concept drifts, the incoming data needs to be analyzed, wherein this data can be available in several forms. For virtual concept drifts, one possible approach is to detect changes in the incoming unlabeled instances distribution $P(\mathbf{x})$ to detect possible drifts, like in [38, 28, 39]. Using this method, called *Novelty Detection* [40, 26], the classification method could request new labeled instances to perform a model update only when a concept drift is detected [26]. Nevertheless, analyzing only the unlabeled instances is unfeasible in environments that suffer from real concept drifts, since in this case changes in the *a posteriori* probabilities $P(y|\mathbf{x})$ cannot be tracked [16]. To illustrate this problem, consider the class swap scenario presented in Figure 9, where the class conditional distribution swaps between $y1$ and $y2$ classes from the time t (Figure 9a) to $t+1$ (Figure 9b). Despite the swap, the feature distribution $P(\mathbf{x})$ remains the same (Figure 9c), making methods that only checks the data distribution blind to this rather drastic change.



(a) Class conditional distribution at time t

(b) Class conditional distribution at time $t+1$

(c) $P(\mathbf{x})$ for both t and $t+1$

Figure 9 – Class swap between times t (a) and $t+1$ (b)

Methods created to handle real concept drifts need to rely not only in the feature distribution $P(\mathbf{x})$ (and in many cases the distribution is not even considered in the detection), but also in some supervised (labeled) data. This data must be fed to the system regularly, and it represents the current concept. In some applications, it is considered that a few supervised instances will be given to the system from time to time in order to detect drifts or adapt the models [20, 19, 6]. In other applications it is possible to assume that the true label of all instances will be known at some time [41, 17, 42] (e.g.: an application that predicts the weather for tomorrow will have the true state of the weather next day).

Applications that have manually labeled instances to detect the concept drifts can receive the data representing the current environment before classifying the batch of data collected in the current concept for classification, but with the drawback that

they often receive only a few instances (e.g. a human supervisor may label some samples from the current test batch before handling it to the classification system). Methods that consider that all instances will be labeled may have plenty data to adjust the model and detect drifts, but at the cost that the supervised data will often arrive with a delay. The supervised data can arrive to the system in a batch of N supervised instances form (e.g. 100 instances for every month) [43, 41, 44, 20, 19], or in a stream, were the labeled instances are given to the system one by one [27, 32, 29, 45] (e.g. one supervised instance for every hour).

2.7 Datasets Containing Concept Drifts

One major problem when defining a method to deal with concept drift problems is to find representative benchmarks. Despite the fact that there is a reasonable number of proposed datasets that contain concept drifts, specially when considering the artificial ones, not all of them can be employed for all methods. For instance, the *STAGGER Concepts* [29] dataset contains severe changes from one concept to another, whilst the *SEA Concepts* [46] dataset introduces smaller (intersected) changes for every new concept.

This section describes the most common datasets containing concept drifts found in literature and its main features, like the drift type, speed and severity (Sections 2.2, 2.3 and 2.5, respectively). The number of samples available, or the methodology employed for the dataset creation for the artificial datasets case, are also described. Subsections 2.7.1 and 2.7.2 contains the descriptions and the summary for the artificial datasets, respectively, whilst subsections 2.7.4 and 2.7.3 contains, respectively, the description and summary for the real datasets.

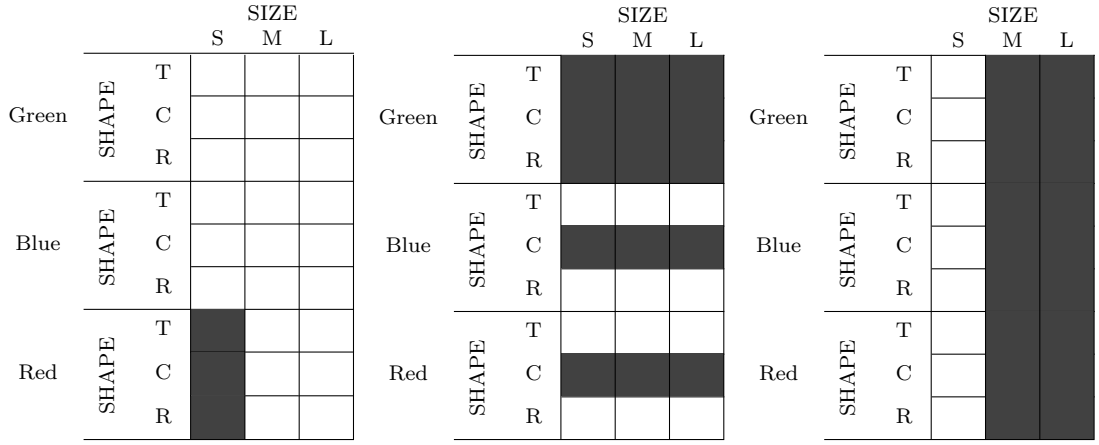
2.7.1 Artificial Datasets

A common practice adopted in the literature is to employ an artificial dataset to test the concept drift handling methods. Artificial datasets allow for a deeper analysis of the problems and methods, since the nature and moment of the concept drifts are known and fully controllable. This subsection describes some of the most popular artificial datasets used in the literature.

STAGGER Concepts: Introduced in [29], this synthetic dataset contains abrupt real concept drifts. Its instances are represented by three features, each with three possible discrete values: $color \in \{red, green, blue\}$, $shape \in \{circle, triangle, rectangle\}$ and $size \in \{small, medium, large\}$. There are also two possible classes, $y \in \{positive, negative\}$[1] . In Maloof & Michalski[47], a guideline for using the Stagger Con-

[1] Although originally the authors in [29] only defines the *target concepts*, not defining explicitly that the objects can belong to the positive or negative classes, these classes were introduced to better conform

cepts including severe concept drifts is given, where the number of training steps t for the dataset is 120, and the classification system receive one supervised sample at each step. For the first 40 time steps, the *positive* class is represented by $color = red \wedge size = small$. In the next 40 times steps, the *positive* class is represented by $color = green \vee shape = circle$. In the final 40 time steps, the positive class has the properties $size = medium \vee size = large$ (the true positive classes definition employed in [47] are the same of [29]). The three concepts described can be seen in Figure 10, where the gray areas denotes positive class objects.



(a) Steps 1-40 (b) Steps 41-80 (c) Steps 81-120

Figure 10 – Stagger concepts [47]. The gray areas represents the positive class.

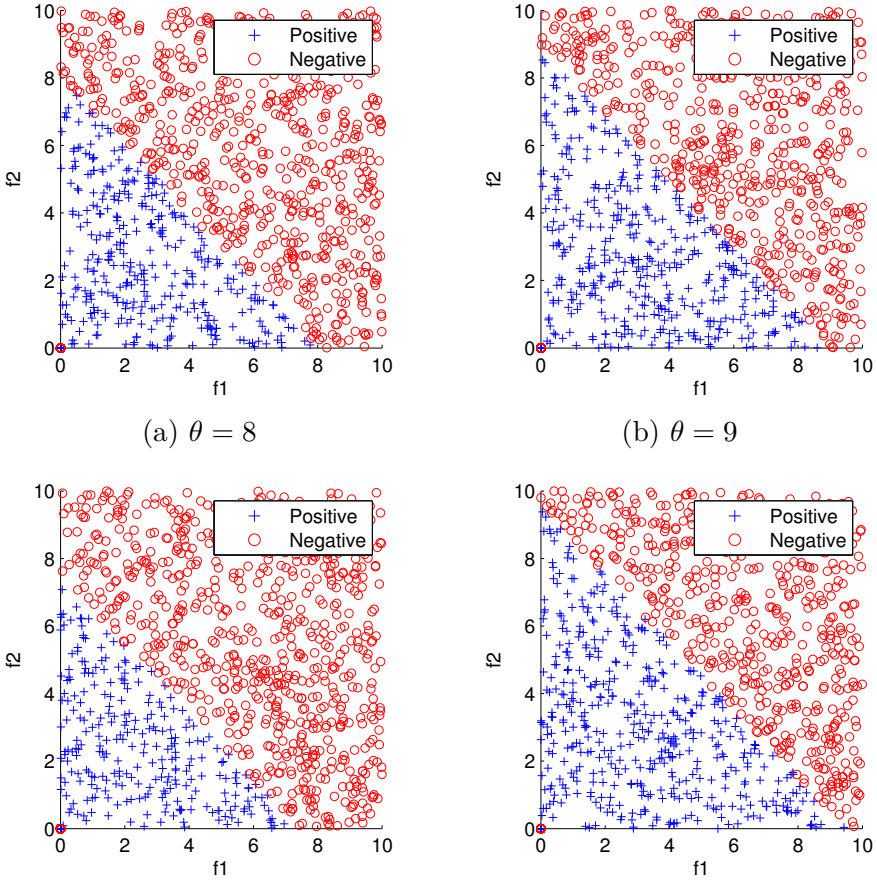
At evaluation phase, at each time step, 100 instances of the current concept are randomly generated and presented to the classification system, in order to check its performance (e.g. the number of instances correctly classified). In Chen et al.[43], a different approach for using the STAGGER Concepts dataset is used, where for every positive class definition (i.e. each new target concept), ten data batches are generated with 300 examples each. In each batch 100 examples are used or training and the other 200 examples for testing. Also in Chen et al.[43], a forth concept equal to the first one is added to simulate recurrences. Although the original configuration of the STAGGER Concepts contains only three distinct concepts, it can be easily modified to include more concepts, besides recurrences and different drift speeds, as done in [32, 43].

SEA Concepts: Developed in [46], this artificial dataset contains three randomly generated real features f_1, f_2 and f_3 in the range $[0, 10]$, where just f_1 and f_2 are relevant, and two possible classes $y \in \{positive, negative\}$. Four data blocks containing 15.000 instances each are generated. The boundary that separates the classes in all blocks is given by $f_1 + f_2 \leq \theta$, where the instances that respect this rule belong to the *positive* class, or to the *negative* class otherwise. Concept drifts are introduced by varying the θ threshold for each block. From the first to the last block, the thresholds used are $8, 9, 7$

with the other datasets studied.

and 9.5, respectively. The training data is composed by 12.500 samples of each block, and the remaining samples are employed for testing. Class noise is inserted by swapping the classes of 10% of the instances in the training data.

Figure 11 contains 1,000 samples of each concept for the SEA Concepts dataset. In order to simplify the data visualization, the non representative $f3$ feature is not present in the plots, and no noise was added. As can be seen in Figures 11a to 11d, the concept drift occurs abruptly, and much of the information is shared between concepts (i.e. for many objects the *a posteriori* probability does not change between concepts). Thus, in this document the SEA Concepts dataset will be considered as containing real abrupt concept drifts with intersected changes.



(a) $\theta = 8$ (b) $\theta = 9$

(c) $\theta = 7.5$ (d) $\theta = 9.5$

Figure 11 – 1,000 instances for each Concept of the SEA dataset without noise. The irrelevant f_3 feature is not represented in the plots

In [6] a different approach is employed to use the SEA Concepts benchmark, where for each time step, a supervised batch containing 250 samples is given for training, and another batch containing 250 samples from the same concept is generated for testing. The concept is changed for each 50 steps, thus generating a test with 200 time steps. It is worth pointing out that in [6] only the training instances contain the 10% of noise.

Moving Hyperplane: This two-class dataset was first used in [30], and it consists of a $d - dimensional$ real space containing samples generated uniformly in a predefined range. Samples are classified as belonging to the positive or negative classes according to Equation 2.1.

$$class = \begin{cases} positive & \text{if } \sum_{i=1}^{d} w_i x_i > w_0, \\ negative & \text{if } \sum_{i=1}^{d} w_i x_i \leq w_0 \end{cases} \tag{2.1}$$

In Equation 2.1, x_i is the i^{th} attribute value, and w_i is the i^{th} attribute weight. Concept drifts are introduced through the modification of the weights w_i, that needs to satisfy $w_0 = \sum_{i=1}^{d} w_i x_i$ in order to keep the prior probabilities fixed. By changing the weights with different values and at different times, the Moving Hyperplane can be employed to generate $d - dimensional$ datasets containing several kinds of concept drifts like intersected, severe, abrupt and gradual drifts. Noise can also be modeled by swapping classes of randomly chosen objects [15, 30].

Rotating Checkerboard Dataset: this dataset consists of two real attributes, uniformly distributed in the $[0 \times 1][0 \times 1]$ domain. To build the dataset, N objects are created, and a checkerboard, containing four squares of side 0.5, is responsible to assign the objects classes (e.g. objects that are in the dark squares will belong to the positive class). Real concept drifts are introduced by rotating the checkerboard by and angle α, as exemplified in Figure 12 [48, 6].

Figure 12 shows that the angle and location of the decision boundaries can change drastically, depending on the rotation angle α and rotation axis position, affecting the drift severity and speed (e.g. for small values of α the drift could be considered intersected gradual continuous, since the boundaries are always changing by small portions). In the configuration shown in Figure 12, it is also noticeable that the concept will reoccur after a rotation of π (half a rotation). In order to prevent training on identical snapshots of data when the concept reoccurs, Elwell & Polikar[6] add 10% of random noise. In the same work, drift variability is introduced by changing the rotation angle α, which can be constant or vary in a exponential, pulse or sinusoidal fashion. Elwell & Polikar[6] employed this dataset by taking 25 training and 1024 testing samples from the window at each time step.

Random RBF Generator: Bifet et al.[42] proposed to use a Radial Basis Function (RBF) generator to create an alternate complex concept type that is not straightforward to approximate with a decision tree model. The generator works by creating N centroids in a real space. Each centroid has a random position, a single standard deviation, a class label and a weight, which defines its prior probability. New samples are generated by picking a centroid randomly, considering its weights (centroids with greater weights are more likely to be chosen). The direction of the new sample offset is also determined at

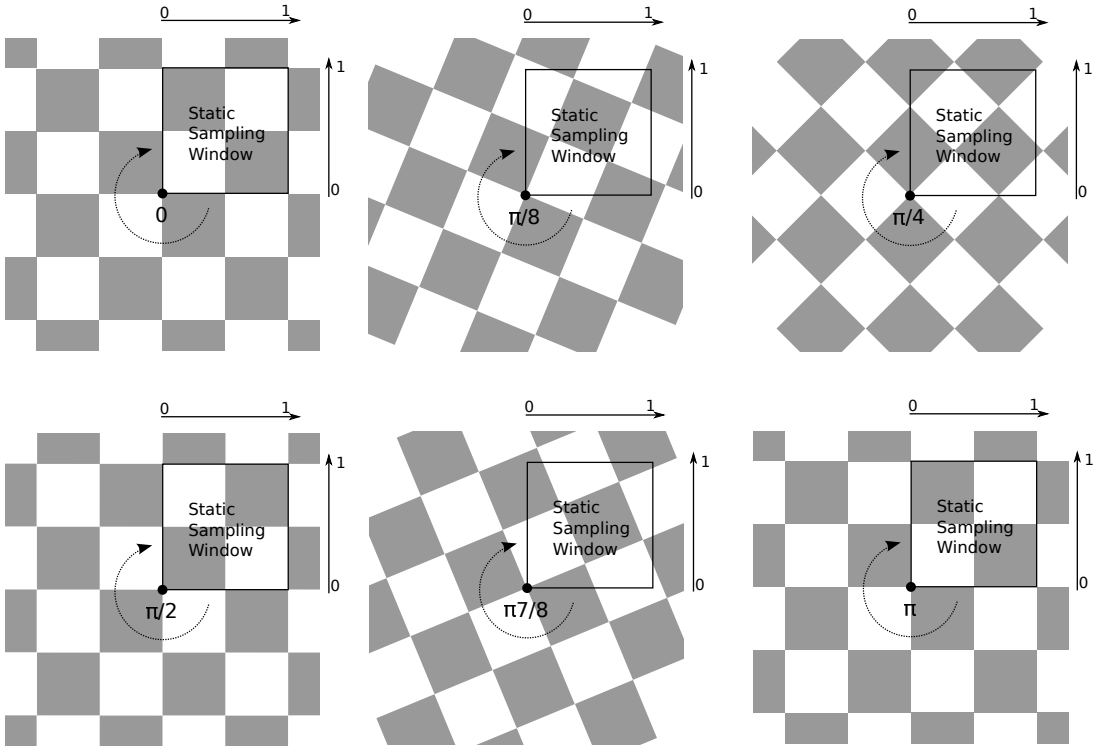


Figure 12 – Half a rotation on the checkerboard data [6]. The sampling window has 1x1 size and it is kept in a static position while the checkerboard rotates in its own axis, thus changing the objects classes in the window.

random, and the displacement distance is drawn randomly from a Gaussian distribution with standard deviation defined by the chosen centroid. Finally, the class of the centroid is assigned to the sample. This process creates a normally distributed hypersphere of samples around each centroid with varying densities. Concept drifts are introduced by moving the centroids with a constant speed.

Since the concepts are always drifting with a constant speed, this dataset can be considered as having a gradual continuous real concept drift. The concept drift severity can be trickier to define, since it depends to the speed constant value. Higher values could lead to severe drifts, since the centroids could move a large distance changing most of the target concept. Otherwise, small values could lead to a intersected drift, since most of the objects will not have their classes changed.

LED Generator: Available at the University of California, Irvine (UCI) Repository [49], this dataset was originally proposed in [50]. It contains 7 Boolean attributes representing the Light-emitting Diodes (LEDs) states of a seven-segment display, and 10 possible classes, representing all decimal digits. Each attribute has a 10% chance of being inverted (noise), thus leading to an optimal Bayes classification rate of 74%. Besides originally not having concept drifts, drifts can be introduced by swapping attributes positions [15, 5], thus generating abrupt real concept drifts. The drift severity will depend of the number of swapped attributes. Some authors also introduces a number of irrelevant

attributes to produce a more challenging classification task, like in [42, 2, 45].

Waveform Generator: Also proposed in [50] and available in the UCI Repository, this dataset defines the classification task as distinguishing between three classes of a waveform, where each waveform is generated by a combination of two or three base waves. There are two versions of the *Waveform Generator* available: the first one, called wave21, has 21 real attributes, all of them including noise, and the second one, called wave40, has the same 21 attributes and additional 19 irrelevant attributes. Similarly to the *LED Generator*, this dataset originally does not contain concept drifts, thus concept changes are introduced by swapping the attributes positions [42, 15]. Its optimal Bayes classification rate is 86% [42], and like in the *LED Generator*, this dataset can be considered to contain abrupt real concept drifts, and the drift severity will depend of the number of swapped attributes.

Gauss Datasets: this two-feature dataset consists basically of instances taken from Gaussian distributions, each centered at some point in $\mathbb{R} \times \mathbb{R}$. Each distribution represents one class, and concept drifts are introduced by changing the distributions centroids over time. In [48, 34, 51] the Gauss Dataset was employed to create two class problems, where before the drift, instances with the positive class label are normally distributed around the center $[0, 0]$ with standard deviation 1. The negative instances are normally distributed around $[2, 0]$ with standard deviation 4. After the concept drift the instances classes are swapped, thus generating an abrupt severe concept drift.

Gauss Dataset with Class Addition/Removal: Elwell & Polikar[6] created a different configuration for the Gauss Dataset, containing a total of 4 classes $\in \{C1, C2, C3, C4\}$. The problem contains 300 time steps, where the concept drift is defined as gradual. The problem begins with 3 the classes $\{C1, C2, C3\}$, and at time 120 the class $C4$ is introduced. At the time step 240 the class $C1$ is removed from the problem. At each time step 20 supervised instances are given for training, and 1024 are given for testing the method.

Sine Datasets: the sine function based datasets are composed basically of two uniformly distributed features (x, y) in the $[0 \times 1][0 \times 1]$ domain. There are two variants commonly used of this dataset: the Sine1, where before the concept drift all points bellow the curve $y = sin(x)$ belongs to the negative class, and the Sine2 that defines the negative samples as the points that satisfies $y < 0.5 + 0.3 \sin(3\pi x)$. For both cases the samples classes are swapped after the drift, thus leading to a abrupt severe concept drift [51, 48, 34].

Artificially Modified Datasets: The *LED* and *Waveform* are examples of datasets with artificially introduced concept drifts. Besides these two datasets, that are quite recurrent in the literature, this practice is common with another datasets without (known) concept drifts. Since the drifts are introduced artificially, and the moment and severity of them is known, these datasets can be fit in the Artificial Datasets category,

independently of how the data was created/collected originally. Some techniques for artificially introducing drifts includes swapping the attributes positions [42, 15, 5, 2], changing the objects classes [48, 52, 31], or even organizing the dataset in such a way that new classes will appear after a number of steps [53]. The changes introduced in the dataset will heavily depend on the characteristics of the method that will be tested using it, since most methods are created to deal with specific concept drift types.

2.7.2 Summary of Artificial Datasets

One of the first noticeable aspects of the artificial datasets studied in this work is that all of them contain real concept drifts, although virtual concept drifts could be easily introduced by picking small and possible biased portions of data for training [54] or by taking sets with different proportions of classes over time for training/testing (class priors change) [44, 38, 55]. Other properties, like the number of features and classes, and the drift speed and severity, can be seen in the Table 1, where configurable aspects of the datasets are marked as PD (Problem Dependent). Table 1 shows that most of the datasets have a low feature dimensionality, and except for the *Random RBF Generator* and the *LED Generator*, all datasets define problems with 2 to 4 classes (low class dimensionality).

Table 1 – Main Properties of the Artificial Datasets

Dataset	**Real Feat.**	**Discr. Feat.**	**Num. Class.**	**Drift Speed**	**Drift Severity**
STAGGER Concepts	0	3	2	Abrupt	Severe[1]
SEA Concepts	3	0	2	Abrupt	Intersected
Moving Hyperplane	PD	0	2	PD	PD
Rotating Checkerboard	2	0	2	PD	PD
Random RBF Generator	PD	0	PD	Gradual Continuous	PD
LED Generator	0	7 or more[2]	10	Abrupt	PD
Gauss Datasets[3]	2	0	2	Abrupt	Severe
Gauss Dataset Class Addition/Removal	2	0	4	Gradual	Intersected
Sine Datasets	2	0	2	Abrupt	Severe
Waveform Generator	21 or 40	0	3	Abrupt	PD

Table 1 also shows that most datasets have at least one configurable parameter (e.g. number of classes, drift speed, drift severity,...). This property is desirable, since the customization can expand the datasets application. However, default configurations, like

[1] The STAGGER Concepts are considered to have a severe concept drift when employing the configuration proposed in [47]

[2] Some authors include some extra irrelevant attributes in the LED Dataset, like in [2, 42]

[3] The properties were based in the configuration used in [48, 34, 51]

the ones described for the *STAGGER* and *SEA Concepts*, are necessary to make easier the comparison of different methods. The drift recurrence property was not included in Table 1, since it can be easily achieved in the datasets that does not originally contain recurrence by simply repeating old concepts. It is worth of remarking the importance of choosing the correct dataset stated in the beginning of Section 2.7, since some of them have specific properties that should be considered when picking the dataset to test a method.

2.7.3 Real Datasets

Section 2.7.1 presented the importance of employing artificial datasets to evaluate concept drift handling methods. Nevertheless, despite their importance, they may not fully represent real world environments. For this reason, testing methods in real datasets is interesting since they represent real-world challenges, where the algorithms usability can be put to test [15]. By using well established testing frameworks in these datasets (e.g. using the same number of training samples per step of other works), they may also make easier the comparison between different methods, since the data is fixed (i.e. not randomly generated like in some artificial datasets). Nevertheless, these datasets have some drawbacks, like the fact that it is not possible to know when the concept drift occurred, nor even if the dataset really contains concept drifts. It is also not possible to verify properties like the drift nature, speed and severity in these datasets.

Due to the difficulty in defining the concept drifts in these datasets, a common practice is to employ data collected in environments where unpredictable changes are expected, like the ones related to stock markets or the ones that are bounded to the user preferences or climate changes. In this subsection, the most popular datasets used to test concept drift handling methods found in literature that were collected in real world environments are presented and described.

China Stock Market & Accounting Research Database: Jian-guang et al.[41] and Sun & Li[17] employed the Chinese Stock Market information to produce a dataset with (possible) concept drifts. The task defined in the dataset is to distinguish between companies that are in financial distress from the healthy ones. Concept drifts are expected to occur in this dataset when, for instance, an enterprise evolves from one stage to other in its life cycle. For example, a starting-up enterprise can become a strong one when it grows up, even though the enterprise had shown a deficiency of liquidity or a cash flow difficulty in its first stages, which would be normal but could be misinterpreted as a financial distress.

Electricity: This time series based dataset, introduced in Harries[56], contains the data of the Australian New South Wales Electricity Market, being one of the most popular datasets when testing concept drifts dealing methods. The prices in this electricity market are not fixed and are affected by the demand and supply of the market, where these

prices are set for every five minutes. The *ELEC2* dataset, which is commonly employed for the concept drifts studies, is composed of 45,312 instances drawn from 7 May 1996 to 5 December 1998, where each instance refers to a period of 30 minutes (i.e. 48 samples per day). Each sample contains the attributes time stamp, day of the week, time (1-48 refering to the 30 minutes period of the day when the data was collected), the South Wales electricity demand, the Victorian electricity demand and the scheduled power transfer between states. Each sample also has a class, which identifies if its price is higher or lower than a moving average of the last 24 hours.

Besides the popularity of the Electricity benchmark for testing methods under concept drift scenarios, in an interesting work, Žliobaitė[57] showed that the labels of this dataset are not independent, and a naive predictor that predicts the next label to be the same as the current one (moving average of one) would achieve an accuracy higher than 85%. In this scenario, it is not possible to check if the method under test is really detecting the concept drifts or if it is just firing random change alarms (the more data it discards, the better). Thus, getting high accuracy on the Electricity dataset does not necessarily mean that the method is correctly adapting to the concept drifts. To overcome this problem, in [57] it is recommended to compare the testing accuracies with the accuracy of the moving average of size one when using the Electricity Dataset. A similar conclusion is presented in [58].

Poker-Hand dataset: In this dataset, available at the UCI Repository [49], the defined task is to predict the poker-hand in a set of five cards drawn from a standard deck of 52 cards. Each card is described according to its $suit \in \{Hearts, Spades, Diamonds, Clubs\}$ and $rank$ in the range $[1, 13]$ representing (Ace, 2, 3, ..., Queen, King), for a total of 10 predictive attributes (5 cards times 2 attributes). A class attribute in the range $[0, 9]$ informs the value of the hand, where higher values indicates better poker-hands. In the dataset the order of cards is important, which is why there are 480 possible Royal Flush hands instead of just 4. The dataset contains a total of 1,025,010 samples [42, 45].

Nebraska Weather: The U.S. National Oceanic and Atmospheric Administration has a compilation of weather measurements from over 9,000 weather stations worldwide. The data is collected since 1930, providing a wide range of samples which can include weather trends. The features present in this dataset includes temperature, wind speed, indicators for precipitation and other weather-related attributes. Elwell & Polikar[6] and Escovedo et al.[11] employed the data collected in the Offutt Air Force Base in Bellevue, Nebraska, due to its extensive range of 50 years (1949-1999) and the presence of diverse weather patterns. Both works employed only the eight features with a missing feature rate less or equal than 15%. The remaining missing values were replaced by the mean of the features in the preceding and following samples. The class labels are binary indicating the presence or not of rain in the sample. The dataset contains a total of 18,159 samples,

where 5,698 (31%) belongs to the *rain* class, and 12,461(69%) to the *no rain* class.

Forest Cover Type: Also available at UCI Repository [49], this dataset defines the classification task as identifying the forest cover type for 30×30 meters cells. Each sample is described by 10 numerical and 44 categorical attributes, which defines cartographic properties such as the area elevation, slope and soil type. The class attribute, which belongs to the range $[1, 7]$, identifies the forest cover type. The dataset is composed of 581,012 samples [2, 59]. In this work the *normalized* dataset version available at the Massive Online Analysis (MOA) [60] website[2] was used, which consists of the original dataset with the numerical attributes scaled in the range between 0 and 1 [14].

Bifet[58] presents some problems in the Forest Covert Type dataset, similar to the Electricity dataset, where the author claims that this dataset may present a high correlation between its data, thus a naive classifier, that does not detect or adapt to any concept drift (just adds new information in the model), may perform better than a concept drift detector when using the instances of this dataset in a test-then-train stream scenario.

2.7.4 Summary of Real Datasets

Table 2 shows that the real world datasets used in the literature are quite heterogeneous, showing different number of features, classes and samples. As described in the beginning of the Subsection 2.7.3, it is not possible to verify the drifts properties in real datasets, or even if they really contains concept drifts, reason why properties like the drift speed or severity are not listed in the table.

Table 2 – Main Properties of the Real Datasets

Dataset	**Real Features**	**Discrete Features**	**Classes**	**Samples**
China Stock Market & Accounting Research Database	41	-	2	-
Electricity (*ELEC2*)	4	1	2	45,312
Poker-Hand dataset	5	5	10	1,025,010
Nebraska Weather	8	0	2	18,159
Forest Cover Type	10	44	7	581,012

2.8 Dynamic Classifier Selection

Since in this work it is proposed a framework for dealing with concept drifts through the Dynamic Classifier Selection (DCS) approach, this section briefly describes the general idea behind the DCS methods under static environments (the DCS idea is extended to non-static environment in the course of this work). The DCS-based methods try to find a

[2] http://moa.cms.waikato.ac.nz/datasets/

good "custom selected" classifier or ensemble of classifiers for the unlabeled instance x, usually based on its local region on the feature space, during the classification phase.

The local region of the instance x can be defined by, for example, the neighbors of x in a validation or training set Q, where the classes of the instances in Q are known [61, 62, 63, 64]. It is worth mentioning that, as suggested in [65, 64], the Dynamic Classifier Selection (DCS) therm refers to methods that selects a single classifier [66, 67, 68], while methods that selects one or more classifiers can be define as Dynamic Ensemble Selection (DES) methods [69, 70, 62]. Nevertheless, due to the popularity of the therm and for the sake of simplicity, we will refer to Dynamic Classifier Selection based methods as DCS-based, regardless of the number of classifiers selected (i.e. a single classifier or ensemble).

Lets denote as $N_x = \{n_1, n_2, ..., n_k\}$ the set containing k neighbors of x computed in Q ($N_x \subseteq Q$). A DCS method will use the neighbors set N_x to estimate the classifiers competence and them select a custom classifier/ensemble E_x ($E_x \subseteq P$) for the test instance x [61, 62, 63, 64]. Thus, a DCS-based approach can be seen as a function $E_x = DS(N_x, P)$. The DCS basic idea is illustrated in Figure 13.

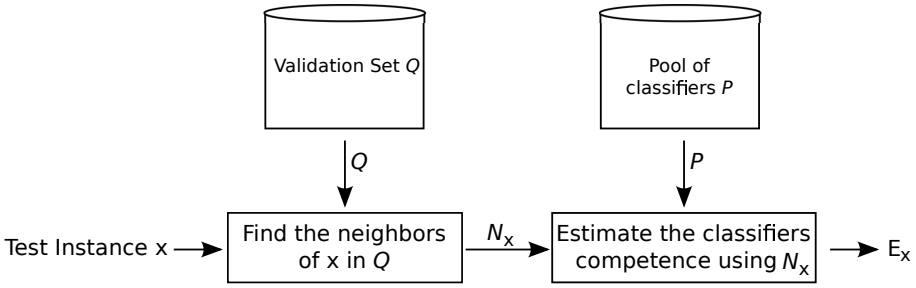


Figure 13 – Dynamic selection of classifiers basic framework.

Since under non-changing scenarios often a single training dataset is given to build a classification system, usually both the pool P and the validation dataset Q are static. Thus, since N_x is a subset of a specific region of Q, the DCS can be defined as region dependent under static scenarios (in Section 4 the DCS is modified to be also time dependent in order to deal with concept drifts). Several DCS methods that follows the general idea described in this Section where proposed over the past years. The DCS methods used in this work are next described:

Dynamic Classifier Selection by Local Accuracy (DCS-LA): Proposed by Woods et al.[68], the DCS-LA basically selects the best performing classifier in the pool for each test instance x based on its neighborhood in N_x. There are two variants of the original method [68, 61, 63]:

DCS-LA Overall Local Accuracy (DCS-LA OLA): In this variant, the most accurate classifier when considering N_x is selected to classify x. The accuracy is defined as the number of samples correctly classified in N_x.

DCS-LA Local Class Accuracy (DCS-LA LCA): Supposing that the classifier c gives the class y_c to the test instance x, the DCS-LA LCA approach computes the local accuracy as the percentage of samples in N_x correctly classified by c, where all instances in N_x must belong to the y_c class (i.e. the K nearest neighbors that belong to the class y_c are selected to be part of N_x). As in the DCS-LA OLA variant, the classifier that achieves the best accuracy is selected to classify x.

For both variants of the method, two (or more) classifiers may have the highest local accuracy estimates. In the original paper [68] tie-breaking is handled by choosing the class that is selected most often among the tied classifiers and, if a tie still exists, the classifier with the next highest local accuracy will break the tie in the same manner. Nevertheless, since the framework proposed in this work is able to handle methods that selects multiple classifiers (see Section 4.1), all tied classifiers are used in the classification in order to increase the implementation simplicity. Regarding to the neighborhood size, through a range of tests, the authors of the original work [68] concluded that a neighborhood of size 5 or 10 generally is able to generate good results for both variants of the DCS-LA approach.

***A Priori* and *A Posteriori* methods**: Both methods, proposed by Giacinto & Roli[66], follows a similar idea of the DCS-LA approach, where one classifier is selected to classify the test instance x based on its accuracy in the neighborhood N_x. Additionally, the *A Priori* and *A Posteriori* approaches weights the accuracy of the classifiers according to their *a posteriori* probabilities and the distance between each neighbor $n \in N_x$ and the test instance x. Formally, the classifier that will be selected is the one that maximizes the probability of correctly classifying the test pattern x, where the equations for calculating this probability depends on the approach variant implemented [66, 63].

Using the *A Priori* variant, the selection is performed without knowledge about the class assigned by the classifier c to the test pattern x, using the Equation 2.2, whilst the *A Posteriori* method, that uses the Equation 2.3, considers that the classifier being tested c labeled the test instance x as being from the y_c class. In both Equations 2.2 and 2.3, δ_i is equals to one divided by the Euclidean distance d_i between the neighbor n_i and the test instance x ($\delta_i = 1/d_i$) [66, 63].

$$p(correct_c) = \frac{\sum_{i=1}^{K} P(y|n_i \in y)\delta_i}{\sum_{i=1}^{K} \delta_i} \tag{2.2}$$

$$p(correct_c|c(x) = y_c) = \frac{\sum_{n_i \in y_c} P(y_c|n_i)\delta_i}{\sum_{i=1}^{K} P(y_c|n_i)\delta_i} \tag{2.3}$$

K-Nearest Oracles (KNORA): Proposed in [69, 70, 62], the KNORA works by selecting an ensemble for the instance x based on its neighborhood N_x. How the classifiers are selected to be part of the ensemble depends on the variant of the KNORA method

implemented. Bellow are described the two main variants of the KNORA method proposed in the original papers. In Section 4.7 a modification for the original K-E method for dealing with noisy environments is proposed.

KNORA-ELIMINATE (K-E): considering the set of k neighbors N_x of the unlabeled instance x, and supposing that a set of classifiers C correctly classifiers all instances in N_x, then every classifier $c_i \in C$ should submit a vote on the unlabeled instance. There is the possibility that no classifier can correctly classify all instances in N_x. In this case, the value of K should be decreased and the set N_x recomputed, until at least one classifier correctly classifies all instances in N_x.

The basic idea of the K-E is explained in Figure 14, where the instances in the validation dataset Q are represented by small circles on the left side of the image. The instance that needs to be classified x is represented by an hexagon and the K nearest neighbors of x (N_x) are shown in gray. The right side represents the classifier space, and each circle represents a set of classifiers that correct classifies a neighbor of x. The gray area represents the classifiers that correctly labels all neighbors, being the set of classifiers that should give a vote to classify x. The original K-E method is extended in Section 4.7 in order to introduce a slack variable.

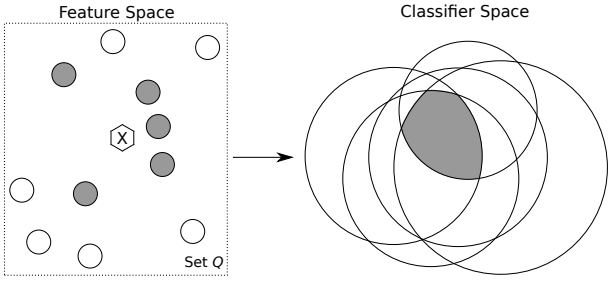


Figure 14 – The K-E scheme. On the left side the unlabeled instance x is shown as an hexagon, and the $K = 5$ nearest validation points are showed as gray circles. On the right side the intersection of all classifiers that correctly classifies all instances is painted. Figure adapted from [62].

KNORA-UNION (K-U): considering the set of neighbors N_x containing k instances and supposing that a set of classifiers C correctly classifies at least one instance in X, then every classifier $c_i \in C$ should submit v_i votes on the unlabeled instance, where v_i is the number of neighbors correctly labeled by the classifier c_i (i.e. the more neighbors a classifier labels correctly, the more votes it will give for classifying x). The K-U is represented in Figure 15, where the only difference to K-E (Figure 14) is that, instead of using the intersection of the classifiers sets, the union of all sets is employed (gray area on the right side of the figure).

In the tested scenarios discussed in the original works [69, 70, 62] the authors concluded that the KNORA algorithms family is robust to the number of neighbors selected, although a small neighborhood size generated better results in most scenarios,

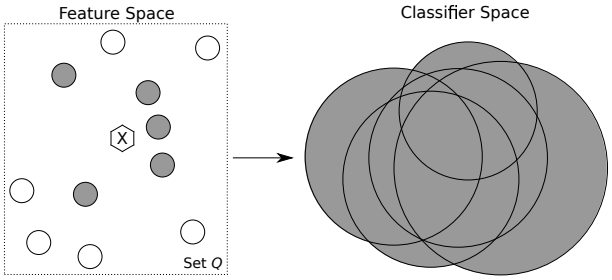


Figure 15 – The K-U scheme. On the left side the unlabeled instance x is shown as an hexagon, and the $K = 5$ nearest validation points are showed as gray circles. On the right side the union of all classifiers that correctly classifies at least one instance is painted. Figure adapted from [62].

where the $K = 7$ neighbors was the best performing configuration for the K-E approach in [70, 62].

2.9 Comparing Classifiers Over Multiple Datasets

In this work is proposed a new framework for dealing with a range of concept drift problems. Several datasets are used for testing the proposed framework performance under different concept drift scenarios, which may raise the question of how we could verify the methods performance over all datasets, since the framework may be the best performing one in a dataset A, and be the worst performing one in a dataset B.

To mitigate this problem, this section presents some non-parametric tests for statistical comparisons of classifiers over multiple datasets that will be employed in this work. For this purpose, consider a set of K classifiers tested in N datasets, where we must validate or reject the hypothesis that all classifiers have similar performances in these datasets.

In this scenario, the Friedman test [71, 72, 73] may be used. When using the Freedman test first it is necessary to rank the algorithms performance in each dataset (i.e. the best performing method receives the rank 1, the second best receives rank 2, etc.). In case of ties when assigning the ranks, the average of the ranks that would have been assigned without a tie must be assigned to each tied method (e.g. if two methods are tied in rank 5, then each method should receive the rank $(5 + 6)/2 = 5.5$). After ranking each method in each dataset, the average rank of the methods should be computed. After the methods ranking, the Friedman test can be computed according to Equation 2.4:

$$\chi_F^2 = \frac{12N}{K(K+1)} \left[\sum_{j=1}^{K} \overline{R}_j^2 - \frac{K(K+1)^2}{4} \right] \tag{2.4}$$

The Friedman test is distributed according to χ_F^2 with $K - 1$ degrees of freedom

for big enough N and k ($N > 10$ and $k > 5$) [73]. Iman & Davenport[74] showed that the Friedman test may be too conservative and derived the F_F statistic, which distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom [73]. The F_F is computed as shown in Equation 2.5:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} \tag{2.5}$$

In both Friedman and F_F tests, if the computed value is bigger than the critical values, considering their respective distributions (χ_F^2 distribution for the Friedman test and the F-Distribution for the F_F test) and desired confidence level α, the null-hypothesis is rejected, thus indicating that there are classifiers with different performances. Under this scenario, the Nemenyi post-hoc test [75] may be used to make a pairwise comparison of all classifiers in order to check if their performances are significantly different. The Nemenyi post-hoc test can be computed using the Equation 2.6, and two classifiers are considered significantly different if their average ranks differ by at least the Critical Difference (CD) computed [73].

$$CD = q_\alpha \sqrt{\frac{K(K+1)}{6N}} \tag{2.6}$$

The critical value q_α in Equation 2.6 is based on the Studentized range statistic divided by $\sqrt{2}$. When all classifiers must be compared to a control classifier (e.g. compare the proposed method with the state-of-the-art classifiers), the Bonferroni-Dunn test [76] may be a more suitable approach than the Nemenyi test, since it adjusts the critical value for making $K-1$ comparisons, whilst the Nemenyi test adjusts the critical value for $K(K-1)/2$ comparisons. The Bonferroni-Dunn test can be computed using the same equation used in the Nemenyi test (Equation 2.6), but using critical values for $\alpha(K-1)$ [73].

As an example, consider the Table 3, which contains six methods tested using five datasets. In this table, each method is first ranked according to its performance in each dataset (the numbers in parenthesis), and the average rank is computed in the last column ($\overline{R}$). The F_F statistic computed for the Table 3 is equals to 2.59, which is bigger than the critical value of 2.16, considering $K-1=5$ and $(K-1)(N-1)=20$ degrees of freedom and a confidence level of 90% ($\alpha = 0.1$). This result rejects the null hypothesis, indicating that there are classifiers with different performances.

Considering now that the Method2 is a control classifier that should be compared against all classifiers, the CD value computed through the Bonferroni-Dunn test obtained is equals to 3.05, thus indicating that the Method2 performs significantly better only when compared to the Method6 (the diference in the ranks between the Method2 and Method6 is of 3.5, which is bigger than the CD). This result is graphically represented in Figure

Table 3 – Methods ranking table example. The numbers in parenthesis indicates the ranking of the methods in each dataset, and $\overline{R}$ indicates the average rank of each method.

Method	**Dataset1**	**Dataset2**	**Dataset3**	**Dataset4**	**Dataset5**	$\overline{R}$
Method1	91%(4)	80%(2)	40%(5)	98%(1.5)	60%(4)	3.3
Method2	93%(2)	87%(1)	77%(2)	98%(1.5)	71%(1)	1.5
Method3	81%(6)	33%(6)	78%(1)	89%(5)	63%(2)	4.0
Method4	92%(3)	70%(4)	55%(4)	91%(3)	57%(6)	4.0
Method5	94%(1)	34%(5)	76%(3)	90%(4)	61%(3)	3.2
Method6	85%(5)	78%(3)	38%(6)	80%(6)	58%(5)	5.0

16, where the results achieved by the classifiers intersected by the red dashed line are not considered significantly different when compared to the Method2.

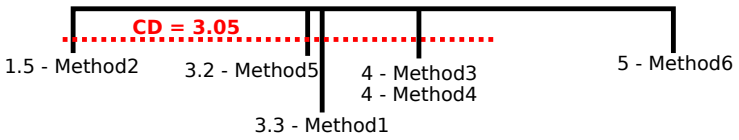


Figure 16 – Bonferroni-Dunn test for the methods in Table 3

Methods deemed as equivalent by the Bonferroni-Dunn test can be further analyzed using pairwise comparisons, considering the hypothesis of equality between each pair of algorithms, using the Bergman-Hommel procedure[3] [78, 77, 64], and the Wilcoxon Signed-Ranks test [73]. Note that when using the Wilcoxon Signed-Ranks test, the significance level should be adjusted using a correction for multiple comparison, as suggested in [73]. This should be done in order to avoid Type I Errors, where a pair of algorithms may be falsely marked as different regardless of the other $m - 2$ algorithms. As in [79], the p-value is adjusted by the Bonferroni correction [73, 79].

[3] Implementation made by [77] available at http://sci2s.ugr.es/keel/multipleTest.zip

3 State-of-the-Art of Concept Drift Handling Methods

In this chapter the state-of-the-art of methods that deal with concept drifts is presented. To make the reading easier, each method was put in one of the following categories: Windowed, Gradual Forgetting, Trigger, Ensemble, Neighborhood or Distribution Analisys based. Besides some methods may be fit in two or more categories, for the sake of simplicity, each method was put in exactly one category in this work based on the method's main characteristics.

The remainder of this Chapter is organized as follows: Section 3.1 presents the methods that use time windows to adapt to concept drifts, whilst Section 3.2 discusses the methods that employ some mechanism to gradually forget the instances. Section 3.3 shows the approaches that use some trigger as a active method to detect concept drifts, Section 3.4 presents the methods based on ensembles of classifiers, Section 3.5 discusses the methods that use the local region of the test instances to adapt to concept drifts, and Section 3.6 shows the methods that employ a distribution analysis to handle concept drift. A review containing the main properties of the methods and a discussion about them is given in Section 3.9. Finally, Section 3.7 presents the datasets employed in the works found in the literature to validate the proposed methods.

3.1 Windowed Methods

One of the classic methods to deal with the concept drift problem is to keep a window containing the M latest samples, which is employed to train/update the classifier. Using this approach, the classifier will "forget" old training instances, which could represent an old concept, thus containing conflicting information. Besides its simplicity, the window based methods raises the question of how large M should be to keep a good performance. A small window can generate a system with a fast reaction to changes, but the low number of training data may cause a loss in the classifier accuracy when the concept is stable (i.e. outside the concept change regions). An alternative would be to define a large window, which would create a stable and well trained classifier that slowly adapts when the concept changes [26, 1, 5, 80]. This compromise between the fast adaptation versus a good performance in stable regions by means of adjusting "forgetting" parameters can be viewed as the *stability-plasticity dilemma*, as discussed in Section 2.3 [36, 26, 6, 81].

Figure 17 exemplifies the basic idea of the windowed methods, where each S_i represents a training sample (or batch), and the bigger the value of i, the more recent the

instance. The window size employed in Figure 17 is equals to 5, where only the samples inside the current window are employed to build/update the classification system. At time t, samples S_6 to S_{10} are in the current window, but at time $t+1$ the instances S_{11} to S_{13} arrive (gray samples), thus moving the current window and removing the instances S_6 to S_8 from the current training set.

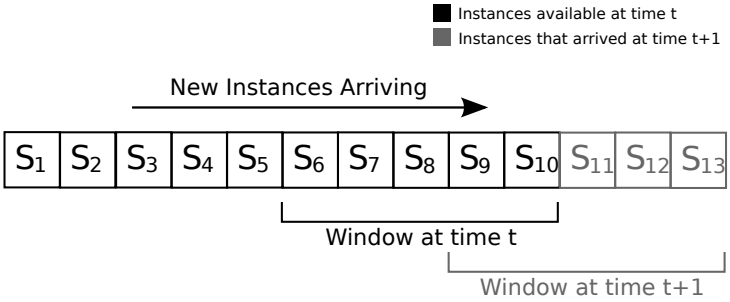


Figure 17 – Windowed methods overview.

The family of FLORA algorithms [80] is one of the first supervised methods proposed to deal with concept drifts by the use of a sliding window [15]. The original FLORA consists of a window of fixed length, where every time a new sample arrives, the oldest one is discarded and the model is updated using the current data. FLORA2 adds the capability to adapt the window size, which can be extended if the algorithm detects a stable region, or shrink in a changing region. Recurring concepts are taken into account in the FLORA3, and the ability to deal with noisy data is added in FLORA4.

In Hulten et al.[30] it is proposed the Concept-adapting Very Fast Decision Tree learner (CVFDT) algorithm, which is an extension to Very Fast Decision Tree learner (VFDT) [82], capable of keeping the tree up to date with a window containing the M latest supervised samples received. Every time a new supervised sample is available, the statistics of the tree nodes are updated considering the new sample that was added, and considering the old sample that was removed from the current window. A new tree is created every time a split that passed in the Hoeffding test [83] no long do so. When new tree becomes more accurate than the old one, it replaces the old tree.

Following the FLORA2 principle of adjusting the window size, Jian-guang et al.[41] proposed a method to adapt the current window size to deal with concept drifts when predicting companies financial distress. In their work, when a new supervised batch b_t (considering that t is the current time) is available, the method creates m training sets $K_0 = \{b_t\},\ K_1 = \{b_t, b_{t-1}\},\ K_m = \{b_t, b_{t-1}, ..., b_{t-m}\}$. For each training set, some of the most recent instances are removed and a Support Vector Machine (SVM) is trained with the remaining instances. The removed instances are then used to test the classifier, and the training set that generated the classifier with the best accuracy is elected as the best window. The instances in this window are then used to build a pool of classifiers using the

bagging method. When an instance x needs to be classified, all classifiers of the pool are employed.

The optimal window size problem is investigated in Kuncheva & Žliobaitė[81] work, where an equation is given for estimating the optimal window size for two-class abrupt concept drifts. Since the equation requires the *a priori* knowledge of some parameters, like the moment t when the concept drift occurs and the errors of the classifiers for each concept, this work consider the concept drift problem in a theoretical level. Nevertheless, by making some assumptions, like considering that the error of the classifiers will be the same in all possible concepts and that the concept drift is caused by and rotation or/and translation of a plane, the authors were able to create a system that dynamically adapts the current window size that achieved better accuracies than a fixed window size approach.

Concept drifts when predicting companies financial distress are also studied in Sun & Li[17], where a window, that keeps track of the latest batch, is employed to build a classifier. The accuracy of this classifier is verified in every former batch, and the batches where the accuracy was bigger than a threshold are inserted in the training set, since they are considered similar to the current window. The training set is then employed to build the final model that will be used to classify new instances. Although both real and virtual financial concept drifts may happen in the considered environment, the method focuses on virtual concept drifts only.

In the method proposed in Rakitianskaia & Engelbrecht[84] a sliding window defines the training dataset. Every time the window moves, a neural network is retrained using a Particle Swarm Optimisation (PSO)[85] based algorithm, namely a reinitializing PSO - which completely restarts the search after the window moves, a charged PSO - that "charges" the particles, which can repel each other, and the Quantum PSO - based on the model of an atom. Results showed that the neural networks trained with PSO outperformed the back-propagation algorithm in environments exhibiting infrequent to moderately infrequent gradual drifts. The back propagation algorithms converged faster than the PSOs, outperforming them in scenarios with frequent abrupt drifts. The algorithms that completely reinitializes the neural networks weights for retraining exhibited the worst performances, giving evidences that the previous weights were still informative when the concept changed.

3.2 Gradual Forgetting Methods

Methods that implements the gradual forgetting principle are quite similar to the windowed approach, but instead of abruptly discarding training samples (a sample is or is not present in the current window), these methods gradually decreases the importance of old instances by applying some aging factor, thus allowing a more precise control over how

instances are incorporated and removed to the model [15, 1]. Figure 18 shows the basic framework of the gradual forgetting methods, where each square represents a single sample or a training batch, and the darker the square color, the greater is its associated importance (i.e. its weight). When a new training sample/batch arrives, the gradual forgetting methods update the weights of all samples/batches by applying some fading factor. The training samples are then used to update the current model considering the weight of each train instance.

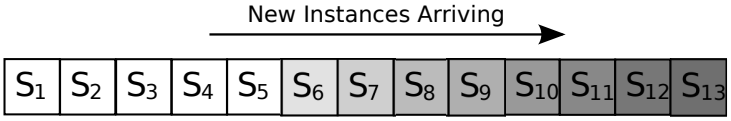


Figure 18 – Gradual Forgetting methods overview. Samples with bigger weights are represented with darker colors.

Considered one of the first concept drift handling methods, the STAGGER system [29] keeps a pair of weights for each feature that indicates its sufficiency, which approximates the degree that the presence of the feature increases the expectation of an outcome, and the necessity, which gives an approximation to the degree that the lack of the feature decreases the expectation of the outcome. To better cope concept drifts, the STAGGER system may decay its weights over time. Concepts are represented by boolean operations of the features (e.g. size = small and color = red), and the search through the space of possible representations is guided by the computed weights. When a new supervised instance is available, the method adjusts the weights in order to increase its accuracy and adjust to new concepts.

Martínez-Rego et al.[27] proposed a one-layer neural network based on a online learning algorithm to deal with concept drifts by means of a fading factor. The method consists in an objective function for training the neural network that can be adjusted to decrease the importance of old instances, where the weight decreasing method can be chosen depending of the problem been modeled (e.g. a monotonically increasing function to take into account the increment in the importance of current information in contrast with the past one).

To deal with concept drifts in one-class classification problems, Krawczyk & Woźniak[2] proposed a method that employes a Weighted One-Class Support Vector Machine [86] in order to adapt to smoothly changing environments. The method updates its model for each new batch received, where all training batches are weighted at each iteration using a fading factor, an age metric (that disconsiders the initial weight of the instances), or a sigmoidal decreasing function.

3.3 Trigger Based Methods

The methods presented in this section uses some trigger, or change detection, technique that tries to detect the exact moment when the concept drifted. Unlike the passive methods that are "always adapting" (e.g. window or gradual forgetting based methods), the trigger based methods adapts to the new concept by taking some action, like updating the models or discarding samples from the old concept, only when a change is detected. Figure 19 shows a stream of supervised samples, where at the time between the samples 7 and 8 there is a concept change. The aim of the trigger based methods is to detect this exact moment and take some measure [15, 87, 35].

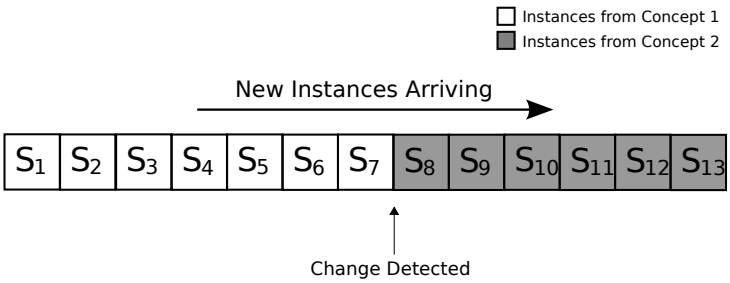


Figure 19 – Trigger methods basic idea.

Trigger based methods have the advantage that when the concept is stable, the classification system can remain unaltered, thus reducing the overhead, or every new information collected in this stable region can be aggregated in the training set to improve the current model. The fact that the moment when the drift happened is pointed by the methods also is an advantage, since the information from the old concept, which can be conflicting with the current one, can be promptly discarded. However, these methods can suffer from some problems, like false alarms and delayed or undetected drifts [5]. A false alarm case can cause an unnecessary discard of still relevant knowledge. Delayed alarms or undetected drifts will cause a performance loss, since the classification system will keep working in an old concept, which does not represent the current environment.

Instinctively, this kind of approach is more suitable to problems with severe abrupt concept drifts [88], since in this case a "frontier" between the concepts can be easier defined [89]. Gradual concept drifts or intersected ones may cause a delay in the drift detection (or even lead the method to not detect the concept drift) if, for instance, the method considers that the first batches with the new concept are just outliers.

One of the simplest trigger methods that can be employed is the checking of the system error rate, which can flag a concept drift when, for instance, the error rate or the variation of the error between two batches reaches a threshold. This kind of trigger is used by Susnjak et al.[3], where a boosted cascade system inspired by [90] is proposed for classification. The method weights every cascade layer employing the latest supervised batch as a validation set, and when the error rate is above a specified threshold, the cascade

is readjusted. Gama et al.[51] proposed a method called Drift Detection Method (DDM), where an error rate based trigger is used to implement two alarms. The first one is a warning level alarm that indicates a possibility of concept drift, and the second one represents a drift level. When the drift level alarm is triggered, the current model is rebuild using only the samples that arrived after the warning level alarm.

A variation of the DDM method, called Early Drift Detection Method (EDDM), was proposed by Baena-Garcıa et al.[91]. In their approach, instead of alarming a possible concept drift based on the error rate, the alarm is triggered based on the distance between two consecutive errors. According to the authors this modification may generate a trigger better suitable for environments containing gradual concept drifts. As in the original DDM method, the approach consists of a warning and a drift alarm to handle concept drifts.

A strategy for triggering possible changes in a window of size W, called Adaptive Windowing (ADWIN) was proposed by Bifet & Gavaldà[92]. The method works by testing several sub-windows in the window W, and whenever two "large enough" subwindows exhibit "distinct enough" means, it is concluded that the corresponding expected values are different, and the older portion of the window is discarded. A Hoeffding bound based threshold E_{cut}, defines the value when the difference between the windows is considered to be a change. Besides that this method uses a window to check the moment of change, it was considered to be trigger based since it tries to point out the moment when the concept drift occurs in the current window to resize it.

A trigger based approach that uses a dissimilarity measure in order to detect concept drifts were proposed by Pinage & Santos[93]. In their method one training cluster is generated for each possible class. The dissimilarity between the test sample and each cluster is then computed, and the class of the cluster less dissimilar to the test instance is assigned to it as a reference prediction. A trained classifier is also used to classify the test instance, and a method based on the DDM[51] or EDDM [91] triggers, is used to monitor differences between the classifier and reference predictions in order to alarm a possible concept drift.

Sakthithasan et al.[94] proposed an approach similar to the ADWIN[92] trigger called OnePassSampler (the method was renamed to SeqDrift1 in [95]), where the main difference is the use of the Bernstein inequality to define the E_{cut} threshold, whereas the Hoeffding bound (which the authors in [94] claim to be too conservative) is used in Bifet & Gavaldà[92]. The authors also concluded that the proposed method can achieve results comparable to the ADWIN trigger by using a single pass in the window W. The proposed approach was extended in Pears et al.[95], where the method called SeqDrift2 use a reservoir sampling based approach [96] to keep the repository of previous seen instances of the current concept.

Kuncheva[97] shows that trigger methods based in the Hotelling's t^2 test are

blind to changes in the distributions when they are caused by shifts in the variances or covariances between the features. In the work is also shown that the Kullback-Leibler distance criterion may be an alternative to overcome this problem, with the drawback that it lacks some fidelity when the distributions are not naturally discrete. The work shows that both the Hotelling's t^2 and the Kullback-Leibler distance can be accommodated within a common log-likelihood framework. A trigger based approach called Semiparametric Log-Likelihood (SPLL) detector is proposed in the work, where a k-means clustering algorithm is employed as a density approximation estimator, which is applied in two distinct windows in order to verify if there is a concept drift between them.

Rodríguez & Kuncheva[88] proposed an ensemble containing both window and trigger based classifiers, where the DDM [51] and the Sequential Probability Ratio Test (SPRT) [98] methods, both based in the monitoring of the error rates, were applied as change detectors in the tests. The Winnow method [99] was employed for combining the classifiers in the ensemble. Gonçalves Jr & Barros[45] implemented a method for recurring concepts that also can employ any change detection method. When a concept drift is signaled, the instances from the current concept are compared with the past ones by merging both sets that must be compared and computing the k-nearest instances of each single sample in the merged set. If the K closest instances are equally divided between the two sets that were merged, they are considered to be from the same concept, thus the stored classifier that was trained with the past dataset becomes the current one. Otherwise a new classifier is trained with the current instances.

In the same vein of methods that handle concept drifts by means of any triggering approach, Minku & Yao[32] proposed a method where before a drift, an ensemble with low and another one with high diversity are maintained and updated with every new supervised instance, but only the low diversity ensemble is used for predictions. When a drift is signaled, a new high and low diversity ensembles are created, and the old ones are kept in the system. The old high diversity ensemble then starts to learn with the new low diversity, and when an instance needs to be classified, the final result is given by the weighted majority voting of the outputs of both high and low diversity old ensembles and the new low diversity one. The old ensembles are kept in the system until the accuracies of the new ensembles are significantly better than the old ones.

In Kapp et al.[21], a method to deal with virtual concept drifts by updating dynamically SVMs hyperparameters and retrain them over time is described. The proposed method use a change detection module that compares the system error on the newest dataset to the previous one using an approximation to a binomial distribution, in order to detect if the concept is stable or not. When a concept drift is detected, an adapted grid-search is applied to find if the concept is recurrent and a previous hyperparameter configuration can be used. If the grid-search does not locate a fit configuration, a Dynamic

Particle Swarm Optimization (DPSO) is used to indicate a new optimum solution. Finally, an Incremental Support Vector Machine (ISVM) or an ensemble of ISVMs is updated with the new samples and hyperparameters. In the latter, where an ensemble is used, the fusion method uses the classifiers that minimize the generalization bound measure introduced in [100]. The method was further studied in [44].

Ross et al.[34] assume that the true label of all instances will be available some time, and consider the classifier predictions as a stream X_t, where $X_t = 0$ if the prediction of the sample t was correct, or $X_t = 1$ otherwise. Viewing X_t as a Bernoulli distribution, the Weighted Moving Average (EWMA) [101] is employed to estimate the increase in the mean of X_t, which is used to detect concept drifts. The method can limit the average run time before a false concept drift positive by means of the adjust of its parameters.

In Chen et al.[43], a Student's t-test with two significant levels α_1 and α_2 is applied as a trigger. When a new supervised batch is available, for each classifier in the pool, if the test with the significant level α_1 detects that the concept is stationary, the classifier is updated with the new batch and it is marked as useful. The test with the significant level α_2 checks if an intersected or severe concept drift occurred, where in the first case, the classifier is still considered useful, but it is not updated. In the other hand, if the test detects a severe concept drift, the classifier is not used in the classification. If none of the classifiers is marked as useful, a new one is trained with the most recent batch and added to the pool as a useful classifier. All classifiers marked as useful are employed to classify the new unsupervised instances, where the classifiers are combined by the use of weights inversely proportional to the errors of each classifier in the latest supervised batch.

Bifet & Gavaldà[102] proposed a method called Hoeffding Adaptive Trees, which basically uses estimators of frequency statistics at every node of a Hoeffding tree. These estimators are used by the tree nodes to decide which of the last instances are currently relevant for training. Between the change estimators tested by the authors, the method implemented using the ADWIN [92] achieved the best results.

The ADWIN trigger is also employed in the method proposed in Bifet et al.[103]. The method, called Leveraging Bagging, modifies the Online Bagging algorithm proposed in [104] by increasing resampling through the use of larger values to compute the Poisson distribution and by adding random output codes in order to increase the diversity of the ensemble. When the ADWIN trigger detects a change, the worst classifier in the ensemble is removed and a new one is added.

In Alippi et al.[48], a concept drift is detected by the analysis of the distribution of the input data and also through the classification estimation error. When a new supervised instance is available, it is inserted in the current concept dataset, while when just a new feature vector is available, only the feature statistics are updated. Two change-detection tests are performed to detect a concept drift. The first one inspects changes in the feature

vectors and the second one analyses the estimated classification error (by means of the supervised instances). If at least one of the tests detects a concept drift, a split method is employed to separate the current concept from the old one. If the current concept is considered recurrent, its instances are merged with the previous instances from the same concept before the model update.

A two windows approach to trigger concept changes is employed in Salperwyck et al.[105]. The method basically keeps a window W_{ref} containing the observations related to the beginning of the current concept, and another window W_{cur} which keeps track of the latest supervised data. The instances in each window are labeled according to their respective window W. The method tries to validate the hypothesis that given the instances and their classes in W_{ref} and W_{cur} it is possible to separate them according to W. If the data is considered separable, a concept drift is triggered.

In Kithulgoda & Pears[106] is proposed a method that reuses stored learners in concept stable regions in order to reduce the processing time, and employ an incremental tree in concept changing regions. The proposed method keeps a decision tree forest that is updated while the concept is considered unstable. The most accurate tree is stored in a pool, which contains a compressed version of all previous accurate trees. Under stable regions, the best performing classifier in the pool is used for classification, and no update nor training is made in order to reduce the processing time. The authors used the SeqDrift2 [95] trigger as a mechanism to infer if the region is stable or not, nevertheless the authors claim that any trigger based method may be employed.

In Chen et al.[107], a three layers trigger based concept drift detector is proposed to predict concept drifts in streams containing periodic changes. The first layer contains a drift detector that finds the drift points, whereas the second layer contains a volatility detector that locates changes in the drift intervals. Finally, the third layer contains a drift predictor, called ProSeed, that uses information from both the drift and volatility detectors to estimate the next drift point based on previous concept drifts encountered in the stream.

3.4 Ensemble Based Methods

Methods discussed in this section relies on ensemble of classifiers to deal with the concept drift problem. Although in the previous sections many of the proposed methods employed ensembles of classifiers, or could be modified to train an ensemble of classifiers instead of a single model, the works presented in this section have ensemble of classifiers as their main mechanism to deal with concept drifts, instead of using, for instance, some trigger or windowing technique.

Ensemble methods can make easier the usage of non-incremental learners in the

classification system since, for instance, a new classifier can be trained with every new supervised batch available, where each new trained classifier is added to a pool. A system that counts with a single non-incremental learner would need to discard the classifier and train a new one with a dataset created by the merge of the old training instances and the new batch. Ensembles can also deal with recurrent concepts in a simpler way, since classifiers trained with past batches that reocurred in the present concept can be simply reactivated and employed for classification [1].

One popular approach when using ensembles to deal with concept drifts is to weight the available classifiers and them classify new instances by means of the Weighted Majority method [108], where the weight of each classifier may refer to its performance in the latest supervised batch/instances received [26, 15]. This idea is illustrated in Figure 20, where it is also exemplified the approach of training and adding a new classifier to the pool every time new supervised instances become available.

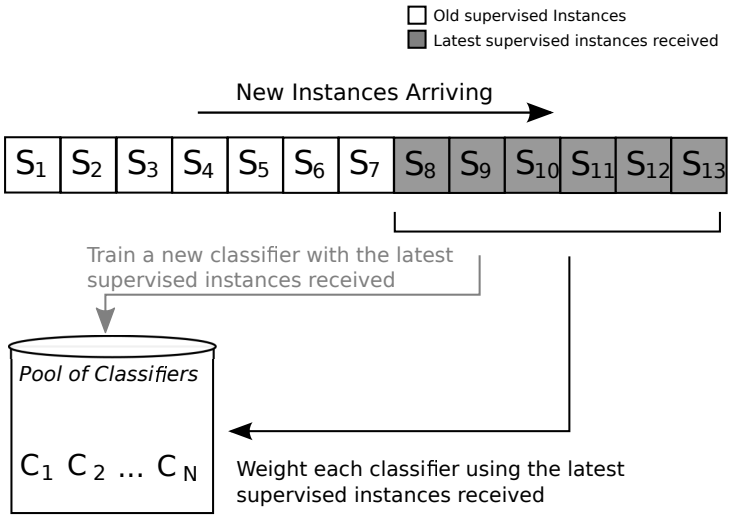


Figure 20 – Weighted pool of classifiers example.

When using a weighted ensemble, the main problem that needs to be dealt by the methods is how to determine the classifiers performance (e.g. the size of the time window with supervised samples employed to determine the classifiers weights, the metric used to define the weights, etc.). Another problem that may arise in these methods is how to prune classifiers from the pool if, for instance, the system have a limited amount of memory to keep the models, and how to define if a concept is recurrent, although one possible solution for these two questions is employing the classifiers weights (i.e. delete the classifier with the lowest weight and consider classifiers with high weights as belonging from the same concept).

Wang et al.[109] proposed a method called Accuracy-Weighted Ensembles (AWE), where a new classifier is built for each new supervised batch available, and all classifiers are weighted using the latest supervised batch. Only the K classifiers with the highest

accuracy in the latest supervised batch are kept in the pool and, when classifying an instance, the classifiers outputs are combined though a weighted averaging. In their work the authors also present some indicators that an ensemble can outperform a single classifier in the presence of concept drifts.

In Brzeziński & Stefanowski[110] is proposed a method called Accuracy Updated Ensemble (AUE), which is inspired in the original AWE method of Wang et al.[109]. The main differences of the AUE to the method proposed in Wang et al.[109] is a slightly different weighting function and the use of online base classifiers. The usage of online classifiers restriction was added since classifiers considered "accurate enough" in the latest received supervised batch are updated using this batch.

The method proposed in Kolter & Maloof[20] maintains a weighted classifiers pool, and when a new supervised sample is available, all classifiers in the pool are tested, and the classifiers that are not able to correctly classify the new supervised sample have their weights multiplied by a factor β. Classifiers with weights less than a threshold are removed from the pool. After this first phase, the global prediction for the new supervised instance is given as a weighted combination of all classifiers in the pool. If the global prediction is incorrect, a new classifier trained with the new example is created and added to the pool with a weight equal to one. Finally, all experts in the pool are trained using the new labeled sample (the method use incremental learners). The weighted majority combination of all classifiers in the pool is used to classify the unlabeled instances.

Street & Kim[46] proposed to create a new classifier for every new supervised batch available, where the new classifier is then added to a pool. If the pool exceeds the maximum number of classifiers, the one with the worst quality score is removed. The quality score is computed for every classifier int the pool using the latest supervised batch, where the classifiers weights are increased/decreased according to how hard the instance being classified was considered, when classifying it using the combination of the entire pool. The majority voting of all classifiers in the pool is employed for classification. In this work the SEA Concepts dataset was introduced, which became a common benchmark applied in many works (see Section 2.7.1).

Inspired in [111], in the work of Karnick et al.[112] is proposed the Learn++.NSE algorithm, that uses ensemble of classifiers in order to deal with concept drifts. The algorithm creates a new classifier for each new supervised batch. All classifiers are weighted using their performances in all batches, where the batch importance in the weighting method is decreased according to a sigmoidal function applied to the classifier's ages. New unsupervised instances are classified using a weighted voting of all classifiers in the pool. The work is extended in [6] and in [113] in order to adjust the weights according to the samples classification difficulty and to propose some pruning strategies for the method.

A bagging method using Hoeffding Threes [82] of different sizes to deal with data

streams containing concept drifts is proposed by Bifet et al.[42]. The method differs from the original Hoeffding Tree by having a maximum number of split nodes, and by deleting some nodes after the maximum three size exceeds a limit. The trees errors are monitored using an exponential weighted moving average, and they are weighted according to the inverse of the squared error.

Escovedo et al.[11] proposed an ensemble of neural networks trained using a Quantum-inspired Evolutionary Algorithm (QIEA) [114] for dealing with concept drifts. In the proposed method, for every new supervised batch, a new Multilayer Perceptron (MLP) with one hidden layer is trained by means of the QIEA. The new classifier is then added in a pool and if the number of classifiers is greater than a threshold, the classifier that performs the worst in the latest batch is removed. Classifiers weights are computed using the latest batch also by means of the QIEA method. In the classification phase the weighted sum of all classifiers in the pool is employed.

A method to estimate the best ensemble size at a given time t is proposed in Pietruczuk et al.[115]. The proposed method estimate the competence of the current pool of classifiers with and without the presence of a newly trained classifier considering a validation batch N. The pool accuracy is given by an approximation to a normal distribution, and the new classifier is added to the pool if the pool competence is significantly increased when adding the new classifier. A similar approach is used to ascertain if a classifier C present in the pool should be pruned, where the test is employed to check if the pool competence does not decreases significantly without the presence of C, in which case C is pruned.

Ditzler[116] proposed to weight the classifiers using the unlabeled test data by means of the spectral meta-learning method proposed in [117]. The new supervised incoming data is used to build new classifiers, that are added to a pool which keep all trained classifiers. The authors claim that the method is able to correctly estimate the weights of the classifiers under any learned concept, nevertheless it is not clear how the method could estimate the classifiers competence under real concept drifts scenarios where the *a posteriori* probabilities $P(y|\mathbf{x})$ are changing and the distribution $P(\mathbf{x})$ remain the same (the authors presented a test in the SEA concepts dataset, which contains this exact scenario).

3.5 Local Region Based Methods

Methods that use some estimation of the local competence of each classifier in the pool dynamically to deal with concept drifts are discussed in this section. When classifying an unlabeled instance x, the local competence based methods work by basically finding local region of x in the feature space (e.g. its neighborhood in a validation set), and then

the best performing classifiers in the local region are selected to classify x[13].

Since the classifiers competence are estimated in the local region of x, a labeled validation set must be incorporated, from which the local region must be extracted. Thus, when dealing with concept drifts using a local region method, one fundamental problem is how to define this validation set to keep track of the current concept. Some methods, as [37, 118, 19], use the latest supervised instances received as the validation dataset, while other methods, like [119], define the validation dataset as the training sets of the classifiers. Another important problem is how to define the local region of the test instance in the validation dataset. Some methods may use the neighbors of x in the validation dataset [118, 19], while others may define it as the nearest training sets of the classifiers to x [53, 119]. A problem inherited from the ensemble based methods that also must be solved is how to keep the pool up to date (e.g. train new classifiers over time). Figure 21 shows a schematic of a local region based method. Note that the schematic presented in Figure 21 is similar to the original DCS idea for static environments, described in Section 2.8.

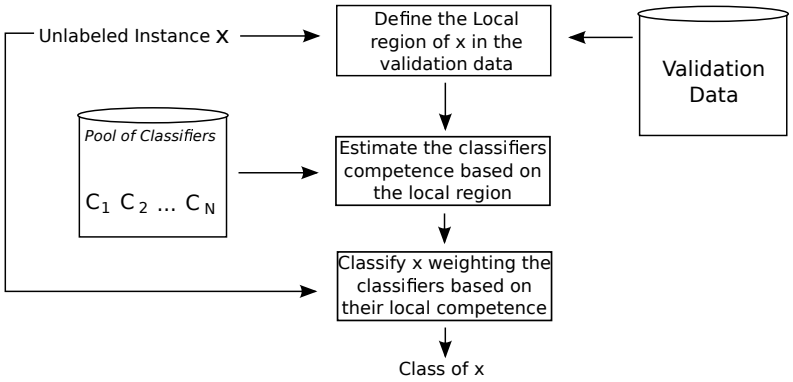


Figure 21 – Local Region based methods overview.

Following the idea of using the training datasets as validation ones, Polikar et al.[53] proposed a method that is a modification of the Learn++ [111] algorithm to handle virtual concept drifts and to accommodate new classes that could appear over time. The method iterates T times in each new supervised batch available. In each iteration, the batch is split in a training T_r and testing T_e sets. The training set is employed to create a weak classifier that is tested using both T_r and T_e. If the classifier error is greater than 50%, a new one is created and tested based on a new T_r and T_e subsets, otherwise, a weighted majority voting ensemble of all previous t generated classifiers is applied to classify the latest supervised batch. In the next iteration, the instances correctly classified by the ensemble will have their probability of being chosen into T_r reduced. The weights assigned to the classifiers are based on the Mahalanobis distance of their training sets to the instance that needs to be classified, and the final prediction of an unsupervised instance is given by the weighted majority voting of all classifiers in the pool.

Tsymbal et al.; Tsymbal et al.[12, 37] proposed to keep a window containing the

latest supervised samples received. Every time this window is moved a new classifier is trained using the latest data and added to a pool, and the local accuracy estimation of all classifiers is updated using the latest data (window). If the number of classifiers in the pool is greater than a threshold, the classifier with the worst accuracy is discarded. To integrate the classifiers in the pool, three different approaches were proposed: a Dynamic Selection, wherein the classifier with the best local accuracy is selected, Dynamic Voting, which is a Weighting Vote technique modified to apply the weights based on the local accuracy, and the Dynamic Voting with Selection, which is similar to the Dynamic Voting, but applies a threshold in the classifiers errors to discard them.

In the method proposed by Chan et al.[119], first the training set is divided in m subsets $R_1, R_2...R_m$, where each subset R_i is composed of training instances that are localized in a radius less than q of the center of the dataset. A classifier pool $C = \{c_1, c_2...c_m\}$ is then created, wherein each classifier c_i is trained with the subset R_i. In the testing phase, the method checks if the instance that must be classified x is located inside any neighborhood R_i. If x is located inside at least one R_i, then a weighted sum fusion method applied to the classifiers $C_n = \{c_i \mid c_i \textit{ is trained with } R_i\}$ is used to classify x, where the weight of each classifier is based on the distance of the unlabeled instance to the points in R_i. If the unlabeled instance is not located inside any R_i, it is classified using all classifiers combined with a majority voting method or a distance weight adjusting method, which defines the weight of a classifier based in the distance from x to the training set R_i centroid. The instance is then added to a set R_{new}, and when R_{new} reaches a predefined size, it is used to train a new classifier that is added to the pool. Note that the approach used to define local region is similar to the *clustering*-based methods described in [64].

Zhu et al.[54] proposed a method to deal with virtual concept drifts that splits each new supervised batch into small chunks $S1, S2, ..., S_i$, and train a classifier with each chunk. The classifiers are then added to a pool that holds the N most recent classifiers. The classifiers are tested in an evaluation set Z, that contains some of the most recent supervised instances. The set Z is partitioned in a series of subsets, where each subset contains the instances corresponding to one specific attribute and value, so considering that the instances contain the attributes $A_1, A2, ..., AM$ and each attribute A_i contain n_i values, Z contain $\sum_{i}^{M} n_i$ subsets. In the classification phase, the attributes of the instance that needs to be classified are used to find the K subsets that contain similar instances in the evaluation set Z. The classifier that achieved the highest accuracy when considering the K subsets is selected to classify the new instance.

The problem of uncertain data stream classification is assessed in Pan et al.[118], where it is considered that new supervised batches might not be sure of the actual classes of each instance. In this case, a supervised instance contains a probability representing how likely it belongs to each class. The method trains a new classifier for each new supervised

batch using the algorithm proposed in Jenhani et al.[120]. Each newly trained classifier is added to a pool that keeps the N most recent classifiers. The method was tested using a static and a dynamic classifier ensemble. In the static ensemble, the classifiers are weighted according to their performance in the latest supervised batch. The dynamic ensemble weights the classifiers based on the distance of the K nearest neighbors of the instance that needs to be classified considering the latest supervised batch, where the classifiers with the lowest weights are discarded. In both static and dynamic approaches, the instances are classified by means of a weighted sum of the classifiers predictions.

Pan et al.[19] proposed a method to classify positive and unlabeled text streams in concept drift scenarios, where labeled positive samples are available over time, but supervised negative ones are never fed to the system. To accomplish the task, the method uses the algorithm proposed in Fung et al.[121] to extract negative samples from each new batch. These negative samples and the supervised positive ones from the latest batch are then used to build a new classifier, that is added to a pool that keeps the N most recent classifiers. When an instance x needs to be classified, its K nearest neighbors in the most recent supervised batch are computed using a *cosine similarity* metric. Then all classifiers are weighted according to their performance in the K neighbors. The classifier weight is also adjusted according to its global performance weight, which defines the performance of the classifier in the current batch when compared with the batch that the classifier was trained. Finally, x is classified using a weighted voting scheme.

Fischer et al.[122] proposed a method for streaming data that use a pre-trained, static offline model and an online trained model. During the classification phase, the method dynamically selects between the two models based on their confidences on the test instance classification. The Generalized Learning Vector Quantization (GLVQ) [123] is employed as the base learner, and the classifiers confidences are estimated by the distance between the classifier prototypes and the test instance. The authors suggest that a virtual concept drift may generate an imprecise confidence estimation, thus a metric learning and weighting scheme is used to estimate the distances between the prototypes and instances.

3.6 Distribution Analysis Based Methods

Methods presented so far rely on supervised instances fed periodically to the system to detect and adapt to concept drifts. However, in some situations it is not possible to acquire new supervised data over time due, for instance, the high cost of labeling it. This section presents the methods found in the literature that deal with concept drifts by means of unlabeled data, which is often related to the unconditional distribution analysis.

Many of these methods basically implement some mechanism to track the appearance of outliers in the distribution in a problem called *novelty detection* [40, 26]. Methods

presented in this section also often deals with virtual concept drifts, since changes in the posterior probabilities $P(y|\mathbf{x})$ not necessarily reflects in a change in the unconditional distribution $P(\mathbf{x})$ or in the a *priori* probabilities $P(y)$, making these methods blind to some real concept drifts, as discussed in Section 2.6 (see the class swap problem) [26].

A method that deals with concept drifts by means of the distribution analysis is proposed in Kurlej & Wozniak[28]. The author argues that in some real world scenarios the categorization of all instances coming from a stream is an unfeasible task, and proposes to identify which instances characterizes a possible virtual drift, needing to be supervised and added to the classifier knowledge base. In the method, the expert is asked to supervise an unlabeled instance if its distance to the nearest point in the actual knowledge set is greater than a threshold d_e^*, or if the difference in the distance between the unlabeled instance and two points of distinct classes in the knowledge set is greater than a threshold d_d^*. When adding a new instance in the knowledge base, if the number of instances exceeds a limit, the oldest one is discarded.

A method for detecting abrupt concept drifts based on unsupervised data is proposed by Kmieciak & Stefanowski[87]. In the proposed method, firstly a decision tree is induced using the first N supervised samples from the stream. This classifier is employed to classify the incoming unsupervised samples. The probability distribution trend of changes in the leaf statistic of the tree are observed to detect possible drifts. When a trend is detected, the drift is signaled, the current classifier is discarded, a new supervised batch is requested and a new classifier is trained using it. Besides the fact that in many situations a real concept drift will not be noticed by only analyzing unlabeled data (see section 2.6), the author implies that the method is capable of dealing with real concept drifts scenarios.

The distribution analysis to deal with real concept drift is also employed in the work of Escovedo et al.[124]. The authors propose a method that detects abrupt real concept drifts by applying statistical tests in both the conditional mean vector and in the covariance matrix of the received batches of data. Since the authors considered that only the first batch of data is supervised, the subsequent bathes are clustered using the *k-means* [125] algorithm, where the number of groups (classes) and the center of each group for the clustering algorithm is set as the same present in the first (supervised) batch. In order to detect a concept drift, the method compares the conditional mean vector and the covariance matrix of the recently received batch (grouped by the *k-means*) versus the data of the first supervised batch. Besides not stated by the authors, it is clear that the abrupt real concept drift must be followed by a virtual abrupt concept drift for the proposed method detect a concept drift.

A method to cope with class prior probabilities concept drifts using the Hellinger Distance [126] is described in González-Castro et al.[38]. The authors proposed two quantification techniques to deal with the priors changes in binary problems. The first one,

called HDx, is based on generating many validation datasets with different priors, and then comparing the validation datasets with the current batch that need to be classified using the Hellinger Distance. The *a priori* probability of the new batch is the one from the validation set that minimizes the Hellinger Distance. Since the HDx method computes all metrics using only the distributions, no classifier is needed to estimate priors of the new batch. The second proposed method is the HDy, which works in a similar way to the HDx method, but it compares the distributions output generated by a classifier in the new batch and in the validation datasets. None of the methods requires supervised instances and in both cases, after the estimation of the class priors, the classifier decision threshold is adjusted to, for instance, keep a constant false positive rate. The HDy method outperformed the HDx one in many of the tested scenarios.

Radtke et al.[39] proposed a method for coping with the class priors probabilities drifts by analyzing the data in the Precision-Recall Operating Characteristic (PROC) [127] space. The method creates several imbalanced validation datasets which are employed to produce Boolean Combined curves using the method proposed in [55]. During operation, the system approximates the current class imbalance level by comparing the newest unlabeled batch to one of the unbalanced training batches using the Hellinger Distance. The imbalance level of the batch that minimizes this distance is assumed to be the current imbalance. The approximated imbalance is then used to find the most adequate set of Boolean Combinated curves in the PROC space, where the selected curves have the closest levels of imbalance when compared to the current environment.

More recently Gu et al.[128] proposed a method to deal with virtual concept drifts by the use of a trigger based on the estimation of equal density regions. A metric called *DensityScale* is used to verify the difference between two windows (e.g. the latest window of data versus the previous one). A non-parametric test method is then used to verify if the *DensityScale* change is significant enough to signal a concept drift.

Cavalcante et al.[129] proposed a method to detect concept drifts in time series based data streams, where a change in $P(\mathbf{x})$ affects $P(y|\mathbf{x})$, since $P(\mathbf{x})$ and $P(y)$ are drawn from the same distribution. The method, called Feature Extraction for Explicit Concept Drift Detection (FEDD), extracts and monitors statistical features from a reference window and compares it with the current time window of the time series in order to identify concept drifts. The Cosine or the Pearson Correlation distance [130] are used to compute the feature vectors dissimilarities. The EWMA[34] trigger is used to signal the concept drifts based on the dissimilarities computed.

Raza et al.[131] argues that when dealing with classification problems in electroencephalography based brain-computer interface, virtual concept drifts ($P(\mathbf{x})$) may appear over time. To deal with these problems, the authors proposed to built a base classifier with the available training data. In the work it is assumed that points which

are closest to each other are likely to share the same label, thus a probabilistic weighted k-nearest neighbors (KNN) algorithm, trained with the available supervised data, is used to determine labels of the test data in a semi-supervised fashion. If the confidence of the KNN algorithm is above a threshold when classifying the unsupervised instances, these instances are added to the training set of the base classifier, where their class labels are the ones given by the KNN. In the work the problem is considered binary, thus only the equations to estimate the probabilistic weighted KNN for binary problems are given.

Pérez-Gállego et al.[132] proposed a method to deal with changes in the *a priori* probabilities $P(y)$ in quantification problems by means of an ensemble of classifiers. Given a training dataset, the proposed method works by generating several training sub-datasets with different *prior* distributions, where each sub-dataset is generated by a random sampling with replacement from the original training dataset. A base quantifier algorithm (classifier) is then trained over each generated sub-dataset. The training phase is executed just once, thus no additional supervised data is necessary over time. In the testing phase, all quantifiers built in the training phase are aggregated through an arithmetic mean.

A framework to deal with concept drifts in malware detection systems is proposed by Jordaney et al.[133]. The framework, called Transcend, basically uses two quality scores based the confidence of the classifiers in each unsupervised instance received for classification. A quality score called *credibility* indicate how similar the classified instance is to the objects from the same class according to the classifier. The *confidence* quality score indicates how distinguishable the credibility is from the other classes. Both quality scores are computed using P-values. Thresholds are used to signal a possible concept drift based on the *credibility* and *confidence*, although the authors does not make clear what action should be taken when a concept drift is signaled. The authors also does not make clear if the malware detection problem involves changes in the distribution, since changes in the *a posteriori* probabilities cannot be tracked without supervised instances if there is no change in $P(\mathbf{x})$ or $P(y)$, as discussed in Sections 2.2.2 and 2.6.

3.7 Usage of the Datasets

This section presents the relationship between the reviewed methods and the datasets described in Section 2.7. As mentioned previously, not all datasets are suitable for testing all scenarios, since each dataset may contain only a specific type of concept drift. Figure 22 shows the relationship between the methods and the artificial datasets. The datasets are represented by circles and the methods by rectangles. The circle size is proportional to the number of methods that uses them. It is worth of remark that only the datasets used in two or more works in the surveyed literature are presented. To count as the same dataset, its implementation must be at least similar to the ones described in

Section 2.7.1. For instance, a Gauss distribution based dataset was used in [6], however it employed a different configuration when compared to [48, 34, 51] that contained addition and removal of classes and a gradual intersected concept drift, thus this work was not related to this specific dataset.

Figure 22 shows that the most popular artificial datasets are the SEA Concepts, STAGGER and Moving Hyperplane. Figure 22 also sustains the allegation that not all datasets can be employed for testing some methods, depending on their nature. For instance, only trigger-based methods use the Sine or the Gauss datasets, where these datasets correspond to abrupt severe concept drifts (this type of concept drift is usually easier to be detected by a trigger approach). Also note in Figure 22 that only a few local region-based methods use a common test benchmarks. According to the surveyed literature, only the the Moving Hyperplane problem was used as a common artificial benchmark when considering the local region methods. As can be seen in Figure 23, no surveyed local accuracy-based method use a real world common benchmark.

Another intriguing conclusion is that none of the distribution analysis based methods use a standard dataset for the tests. This can be attributed to the nature of the drift treated by these methods, that are mostly virtual. In this scenario the drift can be caused, for instance, by factors like sub-sampling, thus not requiring a dataset with specific properties for testing (e.g., the authors could use any common dataset and sub-sample the instances when training the classifiers, such as in [119, 54]).

Figure 23 follows the same principle of Figure 22, but it shows the relationship between the reviewed methods and the real datasets used for tests. It shows that the Electricity dataset is the most popular one, being used by sixteen of the proposed methods, where most of methods are trigger-based. As discussed in Section 2.7.3, Žliobaitė[57] and Bifet[58] demonstrate that the Electricity dataset may not be suitable as a benchmark for concept drift handling methods, since a random trigger could also achieve good results. Besides the aforementioned problems, some recent methods, such as [2, 106, 116], still use the Electricity dataset as a benchmark.

The Figures 22 and 23 analysis also indicate that most state-of-the-art methods use just a few common benchmarks for testing the methods. For instance, in the works of [43, 92, 102, 118, 11, 115] the methods are tested using only or two well known concept drift benchmarks, making it difficult to compare or reproduce the results. Also note that, a vast number of authors, such as [94, 124, 128], does not consider any well known concept drift benchmark during their tests, thus these authors works are no present in Figures 22 and 23.

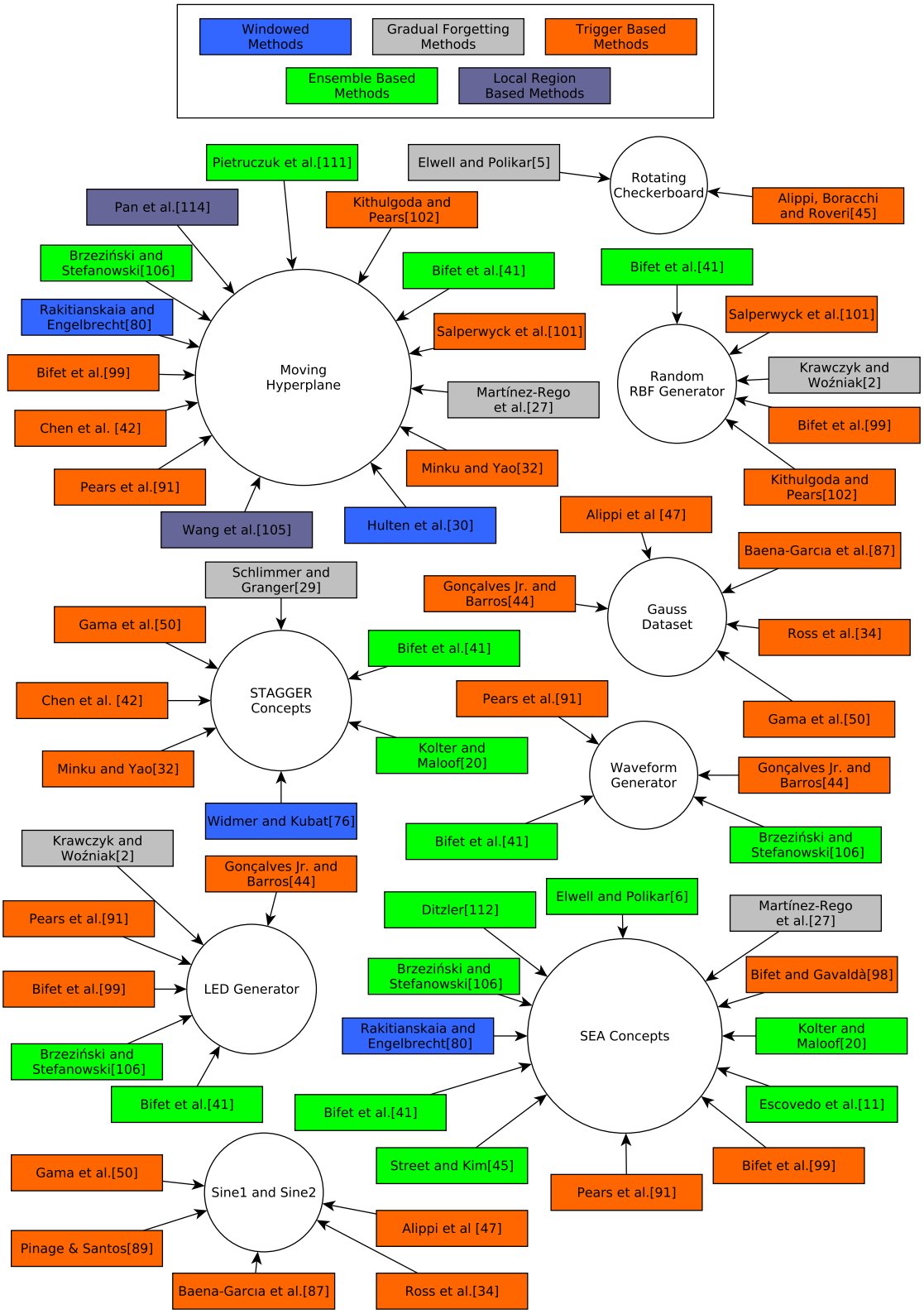


Figure 22 – Relation between the artificial datasets and the proposed methods that employed them

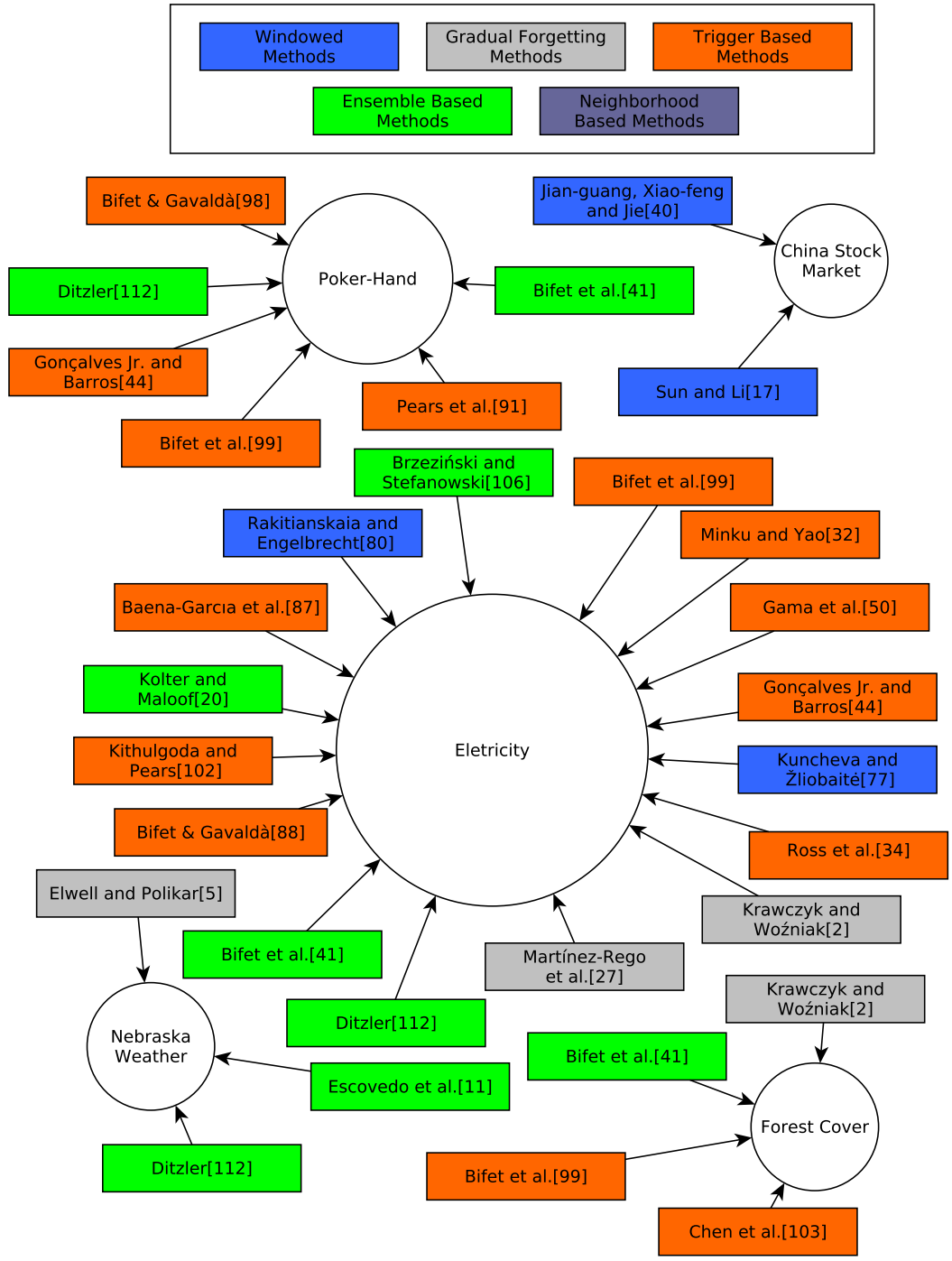


Figure 23 – Usage of real datasets.

3.8 Classifiers Pruning

When dealing with concept drift problems, one of the main challenges is to effectively and efficiently remove irrelevant or obsolete data from the current knowledge base. This data can be stored for future use (e.g. in a future concept this data can become relevant), or the irrelevant data can be discarded to spare computational resources.

Some approaches, like the window and trigger based methods (Sections 3.1 and 3.3,

respectively) strongly rely on the knowledge discard method to handle concept drifts. On the other hand, most methods that uses a pool of classifiers (e.g. Ensemble based methods - Section 3.4) could, in principle, keep classifiers trained under previous concepts, since most approaches estimate the competence of the classifiers in the latest supervised data.

Nevertheless, methods that use a pool of classifiers should employ some pruning strategy in order to keep the pool from increasing its size indefinitely. The works of Partridge & Yates; Margineantu & Dietterich[134, 135] shows that the pruning of classifiers in the pool may not significantly decrease the system performance and, in some scenarios, it can even increase the classification system performance. Nevertheless, it is important to mention that in [134, 135] the concept drift problem is not considered. Even so, a pruning strategy may still be an important tool to reduce computational resources consumption, specially in problems when the system is intended to work for a long period, where it is expected that at some point the pool of classifiers will occupy all the available memory, if the system just keeps adding classifiers to the pool without discarding any old/irrelevant classifiers.

Next, some classical pruning approaches used in methods that employ pool of classifiers to deal with concept drifts are presented. Note that, all studied methods presented in this work will always try to prune a classifier from a concept different from the current one. We present an alternative to this approach in Section 4.5, where the idea of Concept Diversity is presented.

Age Based Pruning or Replace the Oldest: This may be one of the simplest and lightweight pruning strategies used in many works [113, 44, 118, 19], where when the pool reaches its maximum size, every new trained classifier replaces the oldest one in the pool [26].

Accuracy Based Pruning or Replace the Loser: Another classical approach to prune classifiers in a environment containing concept drifts is to evaluate all classifiers using the most recent batch of data [26]. As in [113], the classifiers that exceeds an error threshold can be pruned from the pool or, as in [109, 12, 37, 42, 110, 11, 115], every time a new classifier is trained, the worst performing one can be removed from the pool to keep it with a fixed size.

Quality Score Based Pruning: Following the accuracy based pruning idea, more sophisticated "quality scores" may be developed for pruning classifiers, like in [46], where the classifier quality is defined not only by its individual accuracy, but also by its accuracy when compared with the current pool (nevertheless, in [46] only the latest supervised information received is used to compute the quality score). The quality score may also consider previous batches when estimating the importance of a classifier, where the most recent batches will affect the most the score given to the classifier [20, 113]. When the pool reaches its maximum, the classifier(s) with the worst quality scores may be then pruned.

3.9 State-of-the-Art Review

Table 4 shows the main properties of the methods discussed in this chapter. It includes aspects like the independence of classifiers of the methods, the usage of ensembles, the concept drift types that the methods are supposed to deal with, and the suitability of the methods for problems with more than two classes. This classification has been built according to the methods main characteristics, although some of them could be fit in more than one family, like the FLORA algorithms [80] that were classified as a window based method, but also uses triggers to adapt the window size (FLORA2). Table 4 also shows that some methods are designed to cope with specific concept drift scenarios (e.g. virtual concept drift), making evident the importance of knowing the properties of the concept drift present in the environment to choose the desired method correctly. However, we believe that in most real situations, it is not possible to know *a priori* all the properties of the concept drift present in the environment.

It can be observed that some methods are classifier dependent, which can be a drawback, since it forces the use of a specific algorithm that may not be suitable for the problem being modeled, even if the concept drift present in the environment is the same treated by the proposed method. Another property that could limit some methods usage is the number of classes that they can handle, since some methods are suitable for binary problems only.

Table 4 also shows that many methods, not all of them belonging to the *Ensemble Based Methods* class, use classifier ensembles to deal with both virtual and real concept drifts, indicating that the diversity can be a good tool when dealing with different concept drift scenarios. It is important to notice that only the methods that explicitly use ensembles were marked as "*uses ensembles*", although many of the *Single Classifier* based methods, like [48, 21, 51], can be easily adapted to use ensembles as the base classifier.

When multiple classifiers are considered to deal with concept drifts, some authors argue that weak classifiers should be used [3, 53, 119] since it may be simpler to train the classifiers and also to increase the diversity. Nevertheless, in [6] it is advocated that, since the environment is not stationary, the diversity is naturally provided even when employing strong classifiers. The pool may also be considered diverse due to the use of different training sets for each classifier, as stated in [64] (e.g. some methods may train new classifiers for every new supervised information received). The most common approach in the reviewed literature considers some trigger to detect changes in the concept.

Figure 24 show the main approaches used to deal with concept drifts of the surveyed works over the years, (the number in parenthesis indicate the number of works in the time period). It is worth pointing out that the survey present in this work does not contain all possible methods published over the last years, as it could be a near to impossible

task due to the vast number of published papers regarding to concept drifts. The analysis of Figure 24 indicate that the trigger based techniques is a popular approach to deal with concept drifts in all studied time periods, except before 1999, when just methods based in simple approaches (Window or Gradual Forgetting based methods) were found. Also according to Figure 24, ensemble based methods remained its popularity over the years, further indicating that a multiple classifiers pool may be a powerful tool to deal with concept drifts. The study of approaches based only on the data distribution analysis (not considering the class conditional distributions $P(y|\mathbf{x})$) became popular after 2010 according to Figure 24.

Table 4 – Main properties of some important contributions

Authors	Method Name	Year	Classifier Independent	Uses Ensembles	Concept Drift Type	Binary Only
Window Based Methods						
Hulten et al.[30]	CVFDT	2001	No	No	Real	No
Widmer & Kubat[80]	FLORA	2006	Yes	No	Real	No
Kuncheva & Žliobaitė[81][1]	-	2009	Yes	No	Real and Virtual Abrupt	Yes
Jian-guang et al.[41]	-	2010	Yes	No	Real	No
Sun & Li[17]	-	2011	Yes	No	Virtual	No
Rakitianskaia & Engelbrecht[84]	-	2012	No	No	Real	No
Gradual Forgetting Methods						
Schlimmer & Granger Jr.[29]	STAGGER	1986	No	No	Real	Yes
Martínez-Rego et al.[27]	-	2011	No	No	Real	No
Krawczyk & Woźniak[2]	-	2014	No	No	Gradual Real	Yes
Trigger Based Methods						
Gama et al.[51]	DDM	2004	Yes	No	Real	No
Baena-Garcıa et al.[91]	EDDM	2006	Yes	No	Real Abrupt and Gradual	No
Chen et al.[43]	-	2006	Yes	Yes	Real	No
Bifet & Gavaldà[92]	ADWIN	2007	Yes	No	Real and Virtual	No

Rodríguez & Kuncheva[88]	-	2008	Yes	Yes	Real	No
Bifet & Gavaldà[102]	Hoeffding Adaptive Tree	2009	No	No	Real and Virtual	No
Bifet et al.[103]	Leveraging Bagging	2010	Yes	Yes	Real and Virtual	No
Kapp et al.[21]	-	2010	No	No	Virtual	No
Susnjak et al.[3]	-	2012	Yes	Yes	-	Yes
Minku & Yao[32]	DDD	2012	No. It needs an online classifier	Yes	Real	No
Ross et al.[34]	ECDD	2012	Yes	No	Abrupt Real	Yes
Sakthithasan et al.[94]	SeqDrift1	2013	Yes	No	Real and Virtual	No
Kuncheva[97]	SPLL	2013	Yes	No	Real and Virtual	No
Gonçalves Jr & Barros[45]	RCD	2013	Yes	Yes	Real	No
Alippi et al.[48]	JIT	2013	Yes	No	Abrupt Real	Yes
Pears et al.[95]	SeqDrift2	2014	Yes	No	Real and Virtual	No
Salperwyck et al.[105]	MDD	2015	Yes	No	Virtual and Real	No
Pinage & Santos[93]	DbDDM and DbEDDM	2015	Yes	No	Real	No
Kithulgoda & Pears[106]	SOL	2016	No	Yes	Virtual and Real	No
Chen et al.[107]	ProSeed	2016	Yes	No	Real	No
Ensemble Based Methods						
Street & Kim[46]	SEA	2001	Yes	Yes	Real	No
Wang et al.[109]	-	2003	Yes	Yes	Virtual and Real	No

Kolter & Maloof[20]	DWM	2007	No. It needs online classifiers	Yes	Real	No
Karnick et al.[112]	Learn ++.NSE	2008	Yes	Yes	Virtual and Real	No
Elwell & Polikar[113]	-	2009	Yes	Yes	Virtual and Real	No
Bifet et al.[42]	ADWIN/ASHT Bagging	2009	No	Yes	Real	No
Brzeziński & Stefanowski[110]	AUE	2011	No. It needs online classifiers	Yes	Virtual and Real	No
Elwell & Polikar[6]	-	2011	Yes	Yes	Virtual and Real	No
Escovedo et al.[11]	NEVE	2013	No	Yes	Real	No
Pietruczuk et al.[115]	AASE	2016	Yes	Yes	Real and Virtual	No
Ditzler[116]	Sense	2016	Yes	Yes	Real and Virtual	Yes
Local Region Based Methods						
Polikar et al.[53]	Learn++	2003	Yes	Yes	Virtual (New classes addition)	No
Zhu et al.[54]	AO-DCS	2004	Yes	Yes	Virtual	No
Tsymbal et al.[12]	-	2006	Yes	Yes	Real	No
Pan et al.[118]	-	2010	No	Yes	Real	No
Chan et al.[119]	-	2011	Yes	Yes	Virtual	No
Pan et al.[19]	DCEPU	2012	Yes	Yes	Real Abrupt and Gradual	No
Fischer et al.[122]	-	2016	No	No	Real and Virtual	Yes

Distribution Analysis Based Methods						
Kurlej & Wozniak[28]	-	2011	Yes	No	Virtual	No
Kmieciak & Stefanowski[87]	-	2011	No	No	Abrupt Real	No
González-Castro et al.[38]	HDx and HDy	2013	Yes	No	Virtual (priors drift)	Yes
Radtke et al.[39]	-	2014	Yes	Yes	Virtual (priors drift)	Yes
Escovedo et al.[124]	A2D2	2015	Yes	No	Abrupt Real (Followed by Virtual)	No
Gu et al.[128]	-	2016	Yes	No	Virtual (distribution drift)	No
Cavalcante et al.[129]	FEDD	2016	Yes	No	Virtual (time series streams)	-
Raza et al.[131]	-	2016	Yes	No	Virtual	Yes
Pérez-Gállego et al.[132]	-	2017	Yes	Yes	Virtual (priors drift)	No
Jordaney et al.[133]	Transcend	2017	Classifier able to generate scores	No	Virtual	No

[1] The authors in Kuncheva & Žliobaitė[81] make some assumptions, like the that the classifiers accuracies will be the same in all concepts and they consider that the drift was caused by a rotation and/or translation in the dataset.

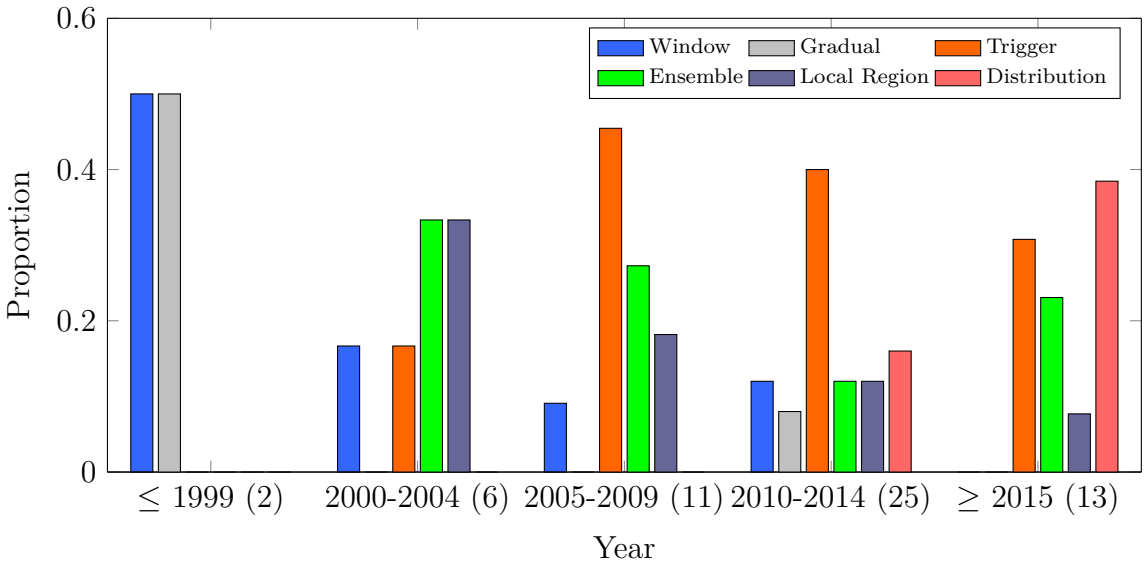


Figure 24 – Main approach used to deal with concept drifts in the surveyed works over the years. The numbers in parenthesis indicate the number of works in the time period.

Note that the Local Region based approaches are constantly used to deal with concept drifts according to the Figure 24, nevertheless these approaches are not as popular as the trigger or ensemble based techniques. Furthermore, besides some important contributions that can be fit in the Local Region category presented in this work, no surveyed work consider the DCS approach as a general solution for the concept drift problem. Although some methods may use a DCS approach to deal with a concept drift, the state-of-the-art methods lacks a study of the DCS methods in general under concept drift scenarios and the necessary adaptations that should be made in the DCS approach to deal with concept drift problems.

4 Proposed Method

This chapter describes the proposed method, which is a framework designed to cope with different concept drift scenarios. As stated in Section 1.2, the proposed framework aims to deal with concept drifts containing different complexities. The framework is designed to cope with distribution changes (virtual concept drift) and real concept drifts (Section 2.2), both of any speed and severity (Sections 2.5 and 2.3, respectively) by means of the DCS approach. The only assumption made about the data is that some supervised data (labeled according to the current concept) will be available over time to train new classifiers. The proposed framework was designed so that there are no limitations like a classifier dependency, assumption of specific concept drift type/properties or the number of classes of the problem being modeled, which are present in many of the works in the surveyed literature (see Section 3.9).

As discussed in Section 2.8, when dealing with static environments the DCS approach is region-dependent only. Nevertheless, in this Section is discussed that under a concept drift scenario, a time dependency must be incorporated in the DCS approach since the region dependency alone will not suffice, specially under real concept drift scenarios. The remainder of this Chapter is structured as follows: In Section 4.1 the Dynse framework is presented, which implements a time dependency in the DCS approach by means of the pool management (creation and pruning of classifiers over time) and by keeping a validation dataset (called accuracy estimation window) up to date with the latest information received.

In Section 4.2 it is presented a discussion about the DCS approach and its time dependency under real concept drift scenarios, while in Section 4.3 a similar discussion is presented concerning to virtual concept drifts. The neighborhood size when using a DCS based method to deal with concept drifts is discussed in Section 4.4. In Section 4.5 it is presented the Concept Diversity idea, which regards to the diversity of the pool under a concept drift scenario when using a DCS-based approach. In Section 4.6 the NNPrune algorithm is presented as a pruning strategy capable of keeping a Concept Diverse pool, while in Section 4.7 a modification in the K-E algorithm to deal with noisy environments is presented. Finally, in Section 5.1 the experimental protocol is discussed.

4.1 The Dynse Framework

In this section it is presented the Dynamic Selection Based Drift Handler (Dynse) framework, which is a tool for dealing with concept drifts that uses the local region of the test instance defined in a validation set to dynamically select a suitable ensemble

for it. Algorithm 1 depicts the basic schematics of the Dynse framework operation. The framework keeps a classifiers pool P and an accuracy estimation window W, which should contain the M latest supervised batches ($|W| = M$). This window works as the set Q described in Section 2.8, from which the local region (i.e. neighborhood) of the test instance is extracted and used to check the competence of the classifiers in P.

The size of the accuracy estimation window M is directly related to the stability-plasticity dilemma discussed in Section 2.3, since a bigger value of M could generate a more accurate system when the concept is stable, at the cost of a slower recover when a concept drift occurs. A detailed discussion about the accuracy estimation window size under a virtual and real concept drift is presented in Sections 4.2 and 4.3, respectively. Next, the framework operation when it receives new supervised data and during the classification phase are presented.

Input: *Stream* of batches $\{B_1, B_2, \ldots, B_t\}$,
Maximum pool Size (D),
Accuracy Estimation Window Size (M),
Classification Engine (CE),
Pruning Engine (PE),
Neighborhood Size (K)

```
1  W ← ∅
2  P ← ∅
3  foreach Batch B ∈ Stream do
4      if B is a Labeled Batch then
5          W ← W ∪ B
6          if |W| > M then
7              removeOldestBatch(W)
8          end
9          C ← trainNewClassifier(B)
10         P ← PE(P, W, C, D)
11     end
12     else
13         foreach test instance x ∈ B do
14             N_x ← KNearestNeighbors(x, k, W)
15             E_x ← CE(N_x, P)
16             x_class ← classify(x, E_x)
               // The result of the classification is available to the user
17             makeAvailable(x_class)
18         end
19     end
20 end
```

Algorithm 1: The Dynse Framework Algorithm

New supervised data arrival: As discussed in the beginning of this Chapter, the Dynse framework needs new supervised batches of instances over time in order to adapt to a possible concept drift. In the beginning of the algorithm, both the accuracy estimation window W and the pool P are set to empty (steps 1 and 2). For each batch available in the stream, if the next batch is supervised (labeled), the follow steps are performed:

- In steps 5 to 8, the accuracy estimation window W is updated to accommodate only the M latest supervised batches received ($|W| = M$). This window can be seen as the validation dataset Q in a DCS method, and its size should be adjusted according to the type of the concept drift to correctly estimate the classifiers' competence, as discussed in Sections 4.2 and 4.3.

- In step 9, a new classifier C is trained using the supervised batch. In this step any inducer may be used for building a new classifier (e.g. SVM, MLP, KNN, ...).

- Finally, in step 10, the current Pool P, the current accuracy estimation window W, the newly created classifier C and the Maximum pool size D are handed to the pruning engine PE, which must make a decision to maintain or to prune classifiers in P (or add C in the P), keeping the pool from increasing in size beyond the threshold D. The Pruning Engine can be seen as a function $PE(P, W, C, D) = P_p$, where P_p is the pruned pool, and $|P_p| \leq D$. Since PE is a parameter in Algorithm 1, any pruning strategy can be implemented in the framework.

On the other hand, when faced with a non-supervised batch (i.e. a test batch), the steps 13 to 18 are executed, where for each test instance x in the current batch, the follow steps are performed:

- In step 14, the k nearest instances in the accuracy estimation window W are selected to represent the local region of $\mathbf{x}$. The set containing the local region is defined as $N_x = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, $N_x \subseteq W$.

- In step 15, the Classification Engine CE uses the set of neighbors N_x to select a custom classifier/ensemble E_x to $\mathbf{x}$ using the classifiers in P. The classification engine CE is a parameter in the Dynse framework that can be seen as a function $E_x = CE(N_x, P)$, and thus any DCS method based on the neighborhood of the test instance can be used.

- In step 16, the custom classifier/ensemble E_x is used to classify $\mathbf{x}$ and, finally, in step 17, the result of the classification is made available to the user.

For the sake of simplicity, Table 5 has a summary of the main parameters and components of the Dynse framework. As one can observe, the Dynse framework is pretty general, since any classification algorithm (e.g. SVM, MLP, KNN, ...) can be employed to build the classifiers that will be added to the pool, and the pool itself can contain classifiers built with several distinct algorithms. Also no assumption is made about the number of classes of the problem, nor the number or type of the features extracted from the instances. The only constraint made by the proposed method is that new supervised

samples need to be available over time in order to adapt to new concepts (this constraint is necessary in any method that deals with real concept drifts that does not affect $P(\mathbf{x})$, as discussed in Section 2.6).

Table 5 – Summary of the components and parameters of the Dynse framework.

Parameter/ Component	**Description**
W	The Accuracy estimation window containing the latest supervised instances received
M	The Accuracy estimation window size (i.e. $\|W\| = M$)
k	The number of neighbors of the test instance x selected from W
P	The pool of classifiers, which may contain classifiers trained using any algorithm
D	The pool maximum size (i.e. $\|P\| \leq D$)
CE	The DCS-based classification engine used to dynamically select the classifiers
PE	The pruning engine used to keep the pool from increasing its size above D

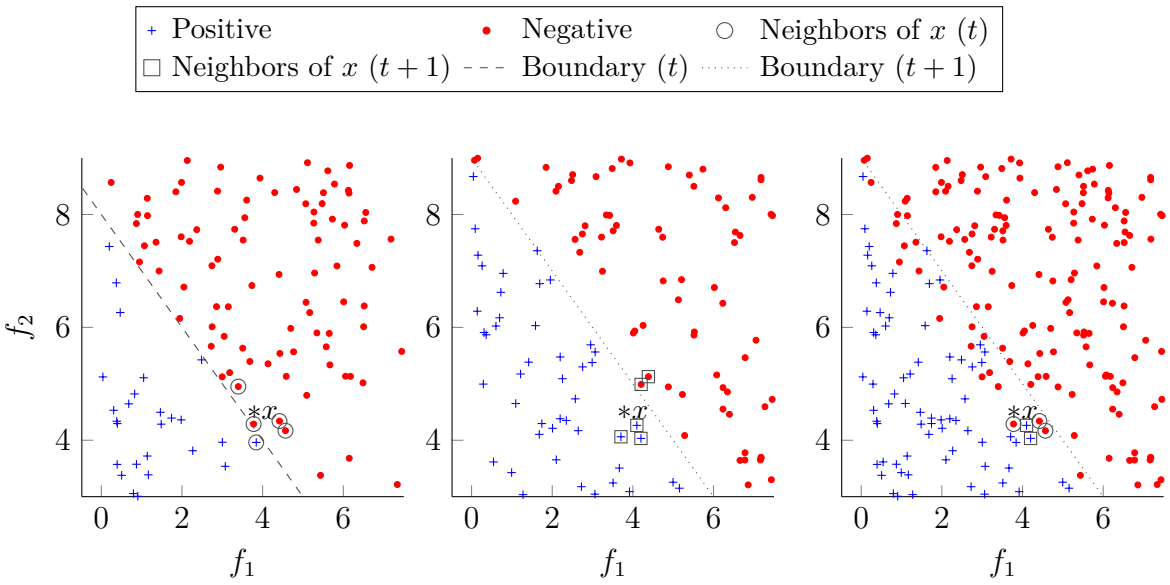
It is important to notice that the accuracy estimation window may contain instances used to train some classifiers in the pool (specially the latest created one). Under stationary environments, this scenario is avoided by considering a separate validation dataset in order to find the local region, where this validation dataset is not used to train any classifier. Nevertheless, under concept drifting environments the supervised instances may be scarce, and the latest supervised instances are the best representation of the current concept. Thus, in this work it is recommended to build new classifiers using the new supervised batches as soon as possible in order to have classifiers representing the current concept in the pool and, in order to estimate the classifiers' competences using the most adequate instances for the current concept, also consider the latest supervised batches to estimate the compentece of the classifiers. In order words, since the supervised data from the current concept may be scarce, the same instances in the supervised batches should be used as both as the accuracy estimation window and training sets.

Finally, since the proposed framework considers the neighborhood of the instance that needs to be classified, in this work it will be considered as a Local Region based method according to the state-of-the-art discussion presented in Chapter 3. However, it could also be fit as an Ensemble based method, since it contains a pool of classifiers, or a Windowed one, since the validation dataset only considers the latest M supervised batches. The proposed framework is designed to be able to deal with real concept drifts, and virtual concept drifts that affects only $P(\mathbf{x})$, as described in Subsections 4.3 and 4.2, respectively.

4.2 Dealing with real concept drifts using a DCS-based approach

Under a static environment a DCS-based method selects a classifier/ensemble based on the classifiers' competence with respect to the neighborhood N_x of the test instance x in a validation dataset Q (e.g., select the classifier with the highest accuracy in N_x). Seeing that the classifiers' competence is estimated using a subset of Q ($N_x \subseteq Q$), Q must have a good representation of the feature space and the *a posteriori* probabilities of the problem. Since under a real concept drift, the *a posteriori* probabilities of the instances may change over time (i.e., $P_t(y|\mathbf{x}) \neq P_{t+1}(y|\mathbf{x})$), it is imperative to keep Q up to date with the current *a posteriori* probabilities.

To illustrate this idea, consider the two discriminant features f_1 and f_2 of the SEA Concepts benchmark (See Section 2.7.1). Also consider that between the times t and $t+1$ there is a concept drift from Concept 1, where $\theta = 8$, to Concept 2, where $\theta = 9$ (in the SEA Concepts problem, if $f_1 + f_2 \leq \theta$ the instance belongs to the positive class, or to the negative class otherwise). Figure 25a shows a validation dataset (i.e., containing labeled samples) Q_t containing instances collected at time t, while in Figure 25b, the validation dataset Q_{t+1} contains instances collected at $t+1$ only. Circles are used to denote the $k = 5$ nearest neighbors of a test instance x in Figure 25a (time t) and squares are used to identify the $k = 5$ nearest neighbors of x in Figure 25b (time $t+1$).



(a) Validation dataset Q_t (b) Validation dataset Q_{t+1} (c) $Q_t \bigcup Q_{t+1}$

Figure 25 – Neighborhood N_x of a test instance x in a validation dataset collected at t (25a) and $t+1$ (25b). In 25c N_x was computed using a merged validation dataset containing both t and $t+1$ instances. In all figures x is placed at $(4, 4.5)$.

Considering $t+1$ as the current time, and a pool P containing classifiers trained under Concept 1 (t) and classifiers trained under Concept 2 ($t+1$), the neighbors N_x

of x in Q_{t+1} may lead to a good competence estimation of the classifiers in P, since N_x represents the local region of x, and N_x is a subset of Q_{t+1}, which represents the current *a posteriori* probabilities (see Figure 25b). To better understand the time dependency of the validation dataset when dealing with real concept drift problems using a DCS approach, consider Figure 25c, which contains a merge of the validation samples collected at both t and $t+1$.

Under a static environment, this bigger validation dataset could improve the DCS performance, since it would possibly have a better coverage of the feature space. Nevertheless, since the boundary changed between t and $t+1$, this dataset may have some conflicting information. As can be seen in Figure 25c, only two neighbors of the test instance x come from the current concept $t+1$ (i.e., the current *a posteriori* probabilities), while 3 neighbors come from the old concept t. Thus, the neighbors depicted in Figure 25c may lead to a poor estimation of the classifiers' competence.

This time dependence of the validation dataset Q raises the question of how to keep Q always up to date with the current concept or, in other words, we may ask "How do we keep track of changes?". Since often when dealing with real concept drifts, some supervised samples are needed time to time (see Section 2.6), a possible solution is to keep only the latest M supervised samples/batches received in Q. This idea is implemented in the Dynse framework as the *accuracy estimation window* W, where $|W| = M$.

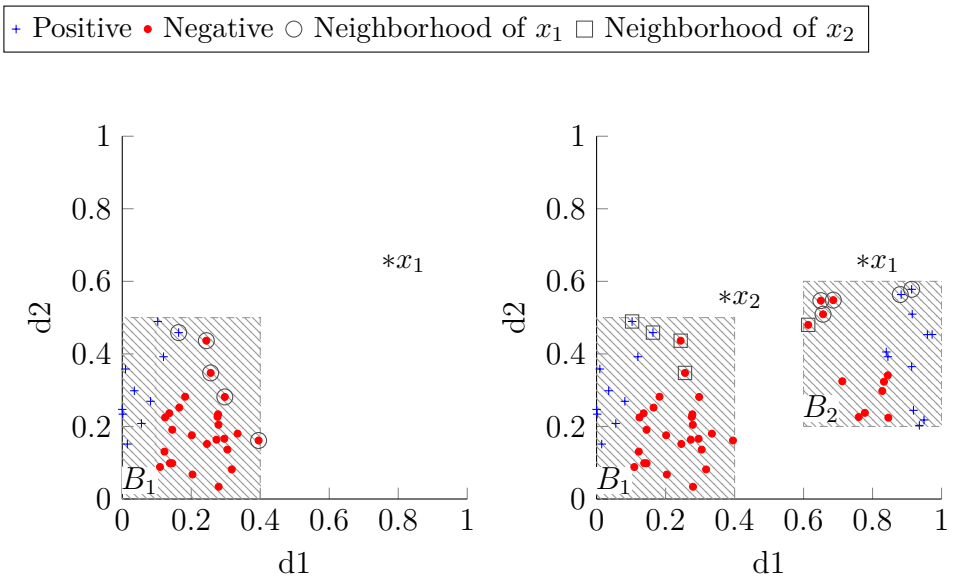
This approach can be seen as a windowing strategy in the validation dataset, and it can be related to the stability-plasticity dilemma as follows: A bigger value for M could lead to a better competence estimation of the classifiers under regions where the concept is stable (more instances available and possibly a better coverage of the feature space). Nevertheless, the instances belonging to an old concept (i.e. old *a posteriori* information) would take more time to be pruned in the presence of a concept change, and thus, a bigger window could lead to a poor estimation of the classifiers' competence under concept changing regions (i.e., some neighbors may belong to the old concept).

4.3 Dealing with virtual concept drifts using a DCS-based Approach

In this Section some insights about the DCS approach under changes in $P(\mathbf{x})$ are given (changes in $P(y)$ are not within the scope of this work). Differently from a real concept drift, under a virtual concept drift scenario, the *a posteriori* probabilities do not change over time. Thus, data acquired in the past can be accumulated in the validation dataset Q, since the classifiers' competence estimation will not be negatively affected as only the neighborhood of the test instance x in Q is considered to estimate the classifiers competence. Differently from a real concept drift scenario, keeping as much data as possible

in the validation set Q may be beneficial due to the better coverage of the feature space. In order to demonstrate this, consider an artificial two-feature $d_1 \in [0, 1]$ and $d_2 \in [0, 1]$ binary problem, where the instances belong to the *positive* class if $\sin(\pi d_1) \times (\frac{7}{9}) > d_2$, or to the *negative* class otherwise.

Also consider that at the beginning of the system run time, at time t, only supervised samples located in a region B_1 are available, and so the pool P contains only classifiers trained using instances located in B_1. If we consider a validation dataset Q_t at time t, composed of some samples from B_1, and a test instance x_1, the neighborhood of x_1 in Q_t may be considered as the "closest known local region" of x_1. Thus, these instances can be used to estimate the classifiers' competence. This scenario is depicted in Figure 26a, where the 5 nearest neighbors of x in Q_t are considered.



(a) Validation dataset Q_t (b) Validation dataset Q_{t+1}

Figure 26 – Virtual concept drift caused by a change in $P(\mathbf{x})$. In 26a are shown the neighbors of a test instance x_1 in the validation dataset at t, when just the B_1 region was known. In 26b, are shown the neighbors of the instances x_1 and x_2 at $t + 1$, when B_1 and B_2 regions were known.

If new data belonging to a region B_2 become available at $t + 1$, this new supervised data can be used to train new classifiers and to estimate the classifiers' competence. Since the concept drift here is only virtual, the supervised samples collected in t are still relevant for estimating the classifiers' competence in $t + 1$. This idea is presented in Figure 26b, where instances collected at both t and $t + 1$ are used in the validation dataset Q_{t+1}. As can be observed in Figure 26b, the $k = 5$ validation samples collected at $t + 1$ (i.e., in the B_2 region) are the closest to x_1, and thus, these instances may be used to estimate the classifiers' competence with respect to x_1. Still in Figure 26b, a test instance x_2 is introduced, and as can be observed, the neighborhood of x_2 is divided between the

instances received at t (B_1) and $t+1$ (B_2), indicating that instances in both regions B_1 and B_2 may contribute to estimating the classifiers' competence with respect to x_2.

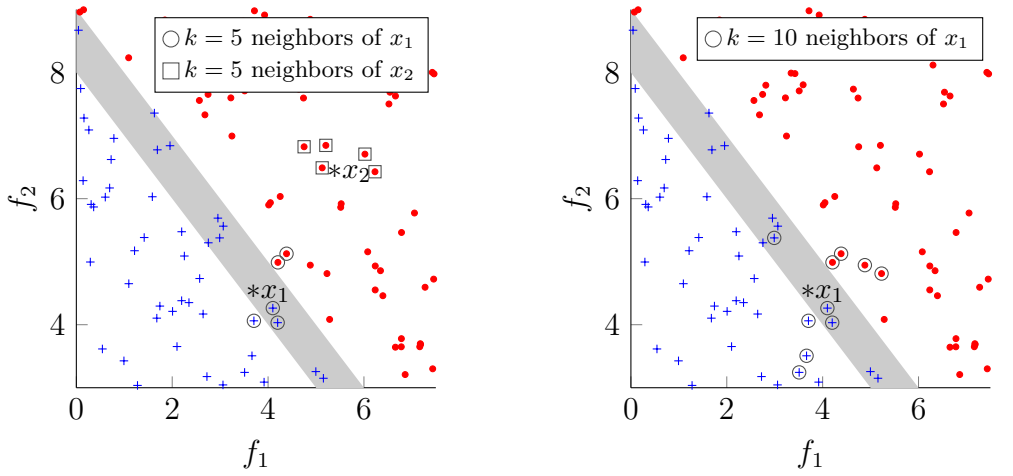
Note that under a scenario where the test instances have drifted to another region in the feature space before new training data arrived with respect to this feature region, a DCS-based approach is still able to estimate the classifiers competence based on the closest "known" regions of the feature space. In summary, considering M as the number of the latest supervised batches/instances accumulated in the validation dataset Q (e.g., the *accuracy estimation window* W in the Dynse framework), we may say that a good policy would be to set M to be as big as possible in order to deal with a virtual concept drift, whilst a small value for M should be used in a real concept drift scenario in order to procure a faster adaptation to changes.

4.4 The Local Region of Competence

By taking into account the neighborhood of the test instance x under the current validation set Q by means of a DCS method, we are assuming that some information can be shared between concepts, and a classifier trained with an old concept may be still suitable under the current concept in some regions of the feature space [12, 37]. To illustrate the rationale behind this thinking, see the example in Figure 27a that shows the changed region between Concepts 1 and 2 in the SEA Concepts problem (without noise and not considering the irrelevant feature f_3). Consider the instances x_1 and x_2 in Figure 27a, as well as Concept 2 ($\theta = 9$) as the current one, and that Q only contains instances labeled according to the current concept.

If we use the entire current validation dataset Q to estimate the classifiers' competence for both instances, then the very same set of classifiers will be selected to classify both x_1 and x_2. However, if we take into account only the local region of the test instances (see the neighbors of x_1 and x_2 in Figure 27a), it becomes clear that classifiers trained under Concept 1 or 2 can classify x_2, since the *a posteriori* probabilities did not change in the region where x_2 was placed. On the other hand, it will be possible to verify that only the classifiers trained under the presence of Concept 2 will be able to classify x_1, since it is expected that at least part of the neighborhood of x_1 changed its *a posteriori* probabilities when the Concept changed from 1 to 2 (i.e., some instances are in the changed area).

As we estimate the classifiers' competence using the neighbors of the test instance, one may wonder how big k should be to define the local region. This is a fundamental problem with any DCS-based approach, regardless of whether or not the environment is static. Under a concept drift scenario, it is possible to relate the neighborhood size to the DCS plasticity, especially when the concept drift is intersected (as in the SEA Concepts problem). To demonstrate this, consider the neighborhood of x_1 in Figures 27a and 27b.



(a) $K = 5$ nearest neighbors (b) $K = 10$ nearest neighbors

Figure 27 – Neighbors of test instances in a validation dataset Q that contains only instances with respect to the concept 2 in the SEA Concepts problem. The gray area depicts the region that has a change between Concepts 1 ($\theta = 8$) and 2 ($\theta = 9$).

As the neighborhood size is increased, as in Figure 27b, there is a greater probability for some validation samples to be taken from regions that did not change; consequently, classifiers that are oblivious to this change may be wrongly selected to classify the test instance, depending on the DCS strategy implemented (e.g., a classifier trained in the old concept may be selected, since it would have a high accuracy as most neighbors are outside the changed region). On the other hand, a smaller neighborhood (Figure 27a) could better represent the local region of the test instance in the current concept, thus giving a better competence estimation for the classifiers with respect to the test instance (i.e., a bigger proportion of the neighborhood is inside or near to the region that changed between concepts).

It is important to note, though, that even a smaller neighborhood may contain validation instances from areas where the concept did not change, specially when the test instance is close to the boundary that changed between concepts. Therefore, under a real concept drift scenario, the DCS method may take into consideration the fact that if a classifier is able to correctly classify only a small portion of the local region of the test instance x, it may be unsuitable for classifying x.

Estimating the optimal local region size may be a challenging task, but as discussed in this section, some DCS-based methods may decrease their performances when using larger local regions. Based on the fact that under static environments, DCS methods provide good results with small neighborhoods (e.g., 5 nearest neighbors for the DCS-LA methods [68] - see Section 2.8), it is reasonable to use the same values proposed in static environment as a first guess when modeling a DCS approach for a concept drift scenario.

Note that in the Dynse framework, the neighborhood size can be specified in the *Find the k neighbors of x* module.

4.5 Concept Diversity

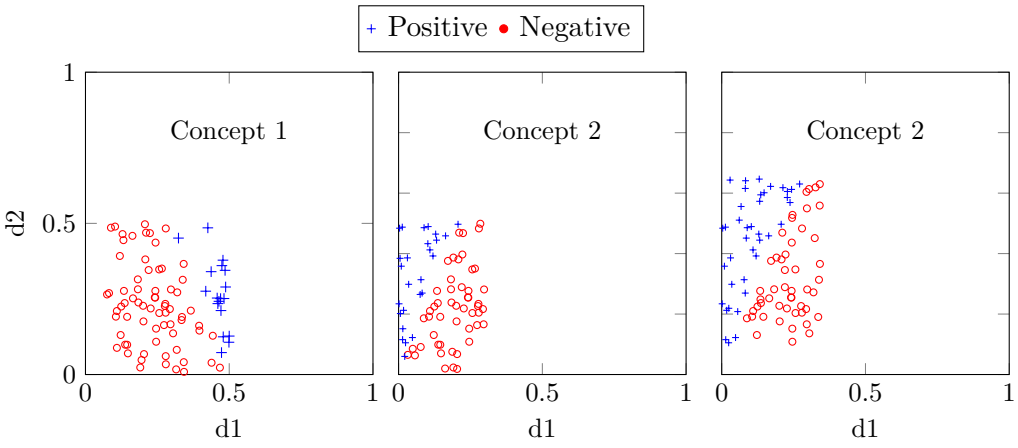
Section 3.8 briefly presented a discussion about some of the classical pool pruning approaches used in the state-of-the-art. All studied methods present in the literature will always try to remove classifiers that belong to a concept that is different from the current one (i.e. different $P(\mathbf{x})$ or $P(y|\mathbf{x})$). Using this kind of approach may increase the method accuracy when the concept is stable. Nevertheless, under a recurrent concept scenario, where the classifiers from the old concept could be simply reactivated, approaches that keep the pool updated considering only the current concept may lead to a suboptimal performance.

In order to make the most from the Dynse framework, we should keep a diverse pool. The diversity is important for the Dynse due to the use of a DCS based approach to select the most promising ensemble for the test instances (see the Classification Engine in Section 4.1). Under a static environment, a DCS based method may benefit from a pool that contains classifiers specialized in different regions of feature space [136], thus the diversity is region dependent. The region dependency alone may also suffice in scenarios that suffers from virtual concept drifts only. Nevertheless, the region dependency may not be sufficient in a real concept drift scenario, where it is also necessary to add a time dependency in order to keep classifiers trained under different concepts (here, a different concept refers to a different *a posteriori* probability). In this work a pool that contains classifiers specialized in different regions of the feature space (i.e. region dependency), and trained under different concepts (i.e. time dependency) is defined as a *Concept Diverse* pool.

The Dynse framework may benefit from a concept diverse pool in two folds: first, when a concept reoccurs, the method can just reactivate the classifiers trained under the previous concept to classify the incoming test instances. Second, under intersected concept drift scenarios (see Section 2.5), some regions of the feature space may share the same *a posteriori* probabilities between different concepts [12, 37]. Since the classification engine selects the best ensemble based on the local region of the test instance, a classifier trained under a previous concept may still be selected by the classification engine to classify a test instance, if the classifier is still competent in the test instance region.

To better understand the concept diversity idea, consider the example in Figure 28, where 3 supervised batches arrived at different times. Also consider that a classifier was trained for each supervised batch. As can be observed in Figure 28, all received supervised batches reffer to a similar region of the feature space, and there is a real concept drift

between times $t-2$ and $t-1$. Considering *Concept 2* as the current concept, if one of the classifiers trained with these batches need to be removed, some authors may argue that the information from the previous concept should be removed (e.g. remove the worst classifier according to the latest supervised batch received) [12, 37, 11, 42, 110, 115].



(a) time: $t-2$ (b) time: $t-1$ (c) time: t

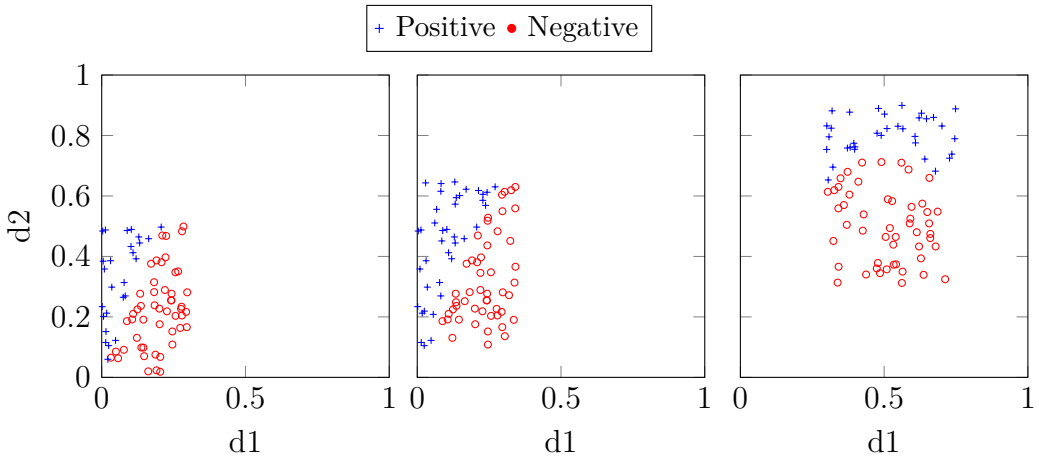
Figure 28 – Supervised batches received at different times. The Concepts marked in the figures refers to a real concept drift.

Nevertheless, note that that both batches that arrived in $t-1$ and t (Figures 28b and 28c, respectively) share the same *a posteriori* probabilities and belong to a similar region in the feature space, thus it is expected that the classifiers generated by these datasets will be quite similar. In order to keep a concept diverse pool, containing classifiers that may be specialist in different regions of the feature space and trained at different concepts, a better solution may be to keep the classifier trained at Concept 1, and prune one of the classifiers trained at Concept 2.

In this scenario, the Dynse framework should be able to select the classifiers trained under the presence of Concept 2 to classify the test instance in the current concept (Concept 2), and select the classifier trained under the presence of Concept 1 if this concept reoccurs. Note that the classifier trained under the Concept 1 could also be selected to classify test instances in the presence of Concept 2 (and vice versa), since some regions of the feature space did not change its *a posteiriori* probabilities between the concept drifts.

Under the presence of changes in the unconditional distribution only (i.e. virtual concept drift) it is possible to apply a similar reasoning when a classifier must be removed from the pool. Consider the example in Figure 29, where we received 3 supervised batches that share the same *a posteriori* distributions, but that may represent different regions of the feature space. In this scenario, a reasonable approach should remove the classifier trained with the dataset that arrived at time $t-2$ or $t-1$, since these datasets represent similar information.

In both examples of a real concept drift and virtual concept drift (Figures 28 and



(a) time: $t-2$ (b) time: $t-1$ (c) time: t

Figure 29 – Supervised batches received at different times. Only the unconditional distribution is changed between batches (virtual concept drift).

29, respectively), a similar decision was made, nevertheless with different results. In the real concept drift example, the diversity was kept in therms of the *a posteriori* information, or in other words, we may have classifiers specialist in the same region, but trained with different *a posteriori* probabilities. In the virtual concept drift example, the pool was kept diverse in therms of the region of the training data.

4.6 The NNPrune Pruning Algorithm

In this Section the Nearest Neighbors Based Pruning (NNPrune) algorithm is presented as a pruning strategy capable to keep a concept diverse pool (See Section 4.5). The NNPrune algorithm takes into consideration both the features location of the train set of the classifiers (virtual concept drift), and also the *a posteriori* probabilities of the train set (real concept drift) in order to keep a fixed concept diverse size pool.

The algorithm works by comparing the training sets of each pair of classifiers in the pool, where the removed classifier (or classifiers if more than on classifier must be removed) is the one that has its training set "best represented" by another classifier that is present in the pool. The basic algorithm is depicted in the Algorithm 2, where the current pool of classifiers, the most recent classifier created and the maximum pool size are passed as parameters. We take all possible pairs of classifiers Ci and Cj that can be made considering the current pool (except for the newest created classifier). We take the training set T_i of the classifier Ci to build a One-nearest neighbor (1NN) based classifier, which is tested in the training set T_j of the classifier Cj.

The 1NN classifier was chosen to estimate similarity between batches in the algorithm due to its high dependency on the feature region ($P(\mathbf{x})$) and class-conditional

Input: $currentPool, newestClassifier, maxSize$
Output: the pruned pool PP

$M \leftarrow \varnothing$
$PP \leftarrow \varnothing$
foreach *Classifier Ci in CurrentPool* **do**
  foreach *Classifier Cj in CurrentPool* **do**
    if *Ci != Cj and Cj != newestClassifier* **then**
      $OneNNClassifier \leftarrow Ci.trainSet$
      $acc \leftarrow test(OneNNClassifier, Cj.trainSet)$
      $M \leftarrow M \cup \{acc, Ci, Cj\}$
    end
  end
end
$t \leftarrow currentPool.size - maxSize$
while $t > 0$ **do**
  $\{acc, Ci, Cj\} \leftarrow highestAccuracy(M)$
  $M \leftarrow M \oplus \{acc, Ci, Cj\}$
  if *Cj not in PP* **then**
    if *Ci not in PP* **then**
      $PP \leftarrow PP \cup Cj$
      $t \leftarrow t - 1$
    end
  end
end
$PP \leftarrow P \oplus PP$

Algorithm 2: NNPrune Algorithm

probabilities ($P(\mathbf{x}|y)$) of its training set (actually, the training set itself is the classifier in this algorithm), thus the 1NN can be considered a good representation of its training set, differently from classifiers that may extrapolate the knowledge acquired during its training as, for instance, a MLP classifier. Possibly for a similar reason, the 1NN is used for some data complexity measures (see the N1,N2,N3 and N4 measures in [137]), although the authors does not make this clear[137].

It is expected that when using T_i as a 1NN to classify T_j, a high accuracy will be achieved only if $P(\mathbf{x})$ and $P(\mathbf{x}|y)$ of T_i is similar, or more general than T_j (e.g. T_i may cover a wider region of the feature space, or the instances in T_i may better represent the boundary when compared to T_j). In this scenario, the classifier C_i, which was trained with T_i, can be considered a good replacement to C_j (that was trained with T_j) . Note that since T_i can be more general than T_j, a high accuracy when testing the train set T_i against T_j does not necessarily reflects in a high accuracy when T_j is tested against T_i. Thus, C_i may be a good replacement to C_j, nevertheless C_j may not be a good replacement to C_i (i.e., it is not a reasonable to select arbitrarily between C_i and C_j to be removed from the pool when one of the classifiers achieve an high accuracy when tested against the other).

The result of each test between pairs of classifiers is stored in a set M in tuples

that follows the format $\{accuracy, trainclassifier, testclassifier\}$, where M is sorted by the accuracy in a descending order. The top t test classifiers that appears in M are stored in PP, where t is the number of classifiers that must be removed, and PP is the return set containing the classifiers to prune. As an example of the set M, consider Table 6, where if just one classifier should be removed, the best choice should be to prune classifier $C3$ (the $C2$ classifier already has a good representation of $C3$).

Table 6 – Tuples containing the accuracy and the classifiers from which the datasets were used as the train and test sets.

Acc	Train	Test
99%	$C2$	$C3$
98%	$C3$	$C2$
97%	$C1$	$C3$
85%	$C1$	$C4$
⋮	⋮	⋮

Note the conditions in the lines 16 and 17 in the Algorithm 2. These conditions are necessary in scenarios where more than one classifier should be pruned in a single call of the NNPrune. The condition in the line 16 checks if the classifier to be pruned is already in the prune list PP (e.g. the classifier $C3$ would be removed twice according to Table 6 if 3 or more classifiers were pruned). We remove the classifier Cj assuming that the classifier Ci already has a good representation of the same knowledge of Cj, nevertheless, there is the possibility that Ci is already marked to be removed. For instance, considering that we should remove 2 classifiers in the scenario depicted in Table 6, we would remove the classifier $C3$, since $C2$ is similar to $C3$, and then we would remove $C2$, thus both $C2$ and $C3$ are marked to removal, and we could end without a good classifier to represent the knowledge of $C2/C3$. The condition in line 17 was created to avoid this scenario.

Since under a concept drift scenario the latest supervised batch of information received is the one that best represents the current scenario, the newest classifier created may replace another one in the pool, nevertheless no classifier is allowed to replace the newest classifier (see the condition in line 5). Note that, by using the NNPrune algorithm, it is necessary to keep the training set of each classifier in the pool. This can be considered a drawback in systems where the available memory is scarce. Thus, in this work, the NNPrune strategy will be considered as a preliminary study, thus this pruning approach will not be used in the Dynse framework when comparing the results of the framework with the state-of-the-art methods.

4.7 The KNORA-Eliminate Method For Noisy Environments

The original K-E method presents a drawback when dealing with noisy environments, as some neighbors in N_x computed in the validation dataset Q may be wrongly labeled, thus it is likely that no classifier will be able to correctly classify the entire neighborhood N_x (the K-E method selects all classifiers that correctly classifies the entire neighborhood N_x). Originally, when no classifier in the pool is able to classify all k neighbors in N_x, a new set, containing $k-1$ neighbors is generated, and the classifiers are tested again. This process is repeated until at least one classifier correctly classifiers the entire neighborhood.

Besides being valid, this approach does not mitigate the problem caused by noise in the validation dataset, since a noisy sample may be the closest one to the test instance x. Consider the scenario depicted in Figure 30, where it is expected that no classifier will be able to correctly classify all instances in N_x due to the noisy sample. The size of N_x will be decreased iteratively until only the $k = 1$ neighbors are selected to be part of N_x, which will contain only the noisy instance (it is worth noticing this will not happen in the unlikely scenario where some classifier in the pool is able to classify all neighbors, including the noisy instance according to the labels given in Q). Finally, the classifiers will be tested using a single wrongly labeled sample and, in this worst case scenario, classifiers that wrongly classify the neighborhood will be selected to be part of the ensemble.

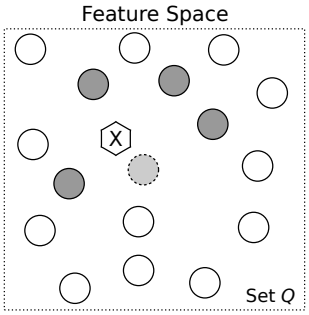


Figure 30 – The $k = 5$ nearest neighbors of the test instance x in a validation dataset Q (gray instances). The closest instance to x (lighter gray dashed) is noise.

The problem faced in the presence of noise may affect the K-E DCS method when it is employed as a classification engine in the Dynse framework, since some samples containing conflicting *a posteriori* probabilities may be found in the accuracy estimation window W when the concept drifts. The instances that does not meet with the current *a posteriori* probabilities may be considered noise (besides the targed classes of these instances being correct in the past concept, they will have the same effect of noise in W in the presence of the current concept), and it may take some time steps to completely prune these instances from W depending on its size (i.e. considering M as the size of W, it will be necessary M time steps to completely prune the instances from the past concept

when the concept drifts).

In order to make the K-E more suitable to noisy environments, in this work is proposed a modification in the original K-E algorithm in order to introduce a slack variable l, where $0 \leq l < k$. This variable controls the maximum number of neighbors that can be incorrectly labeled by the classifiers that will be selected to be part of the ensemble. In other words, considering that N_x contains k neighbors, all classifiers that correctly labels at least $k - l$ instances in N_x will be selected to be part of the ensemble.

When no classifier is able to correctly classify at least $k - l$ instances, instead of decreasing the N_x size as in the original method (i.e. decrease the value of k), we propose to increase the slack variable l by one iteratively, until at least one classifier is selected to be part of the ensemble. This approach make easier to a classifier that is able to classify just a small portion of the neighborhood to be part of the ensemble (a behavior that is similar to the K-U approach), nevertheless it may help to mitigate the problem generated when the noise is too close to the test instance, as discussed in this Section (by decreasing the neighborhood size, it may be possible to select a neighborhood set that contains only noise).

5 Experiments

This chapter focuses on a series of experiments carried out in order to assess the proposed framework and its different configurations under several artificial and real world scenarios. In Section 5.1 the experimental protocol used in this work is discussed. In Section 5.2 the configurations of the proposed and state-of-the-art methods used during the experiments are briefly described, while in Section 5.3 the used benchmarks configurations are described. In Section 5.4 the slack variable introduced in the K-E DCS method is put to test. In Section 5.5, the tests regarding to the impact of the main parameters of the Dynse framework are presented. In Section 5.6 different classification engines are tested using several well known benchmarks, where a default classification engine is also defined for the Dynse framework. In Section 5.7 the impact of the classifiers pruning is studied. The insights presented in Sections 5.4, 5.5, 5.6 and 5.7 are used to define a default configuration of the Dynse framework, which is presented in Section 5.8. The default configuration of the proposed method is tested against the state-of-the-art methods using a series of common benchmarks in Section 5.9. Finally, in Section 5.8 tests in the real world PKLot benchmark are presented.

5.1 Method Analysis

The proposed framework applicability will be analyzed using a standardized experimental protocol (Subsection 5.1.1) under several real and virtual concept drift scenarios (Subsections 5.1.3 and 5.1.2, respectively) by the use of both artificial and real datasets. This analysis aims to identify not only the efficacy of the proposed framework under different scenarios, but also the impact of diverse configurations of the approach by means of the analysis of several validation window sizes, classification engines (also with different configurations), and pruning strategies. How the method behaves under the presence of recurring concepts is also studied by means of different datasets configurations (Subsection 5.1.5). The set of metrics that will be used to asses the framework performance is presented in Subsection 5.1.6.

5.1.1 Baseline Methods for Comparison

In this Chapter several configurations of the proposed framework are tested and compared with each other in order to detect which design generated the best results for each concept drift scenario and to verify the behavior of the proposed method under different scenarios. The different configurations that are tested include the implementations of the proposed framework considering several classification engines (e.g. DCS-LA OLA, K-U,

K-UW, etc.), different accuracy estimation window sizes and different classifier pruning strategies.

The proposed framework is also tested against the state-of-the-art concept drift handling methods. These methods may include some of the latest breakthroughs in the concept drift detection field and also some of the classical methods found in the surveyed literature. Both the proposed method and the state-of-the-art approaches were implement using the MOA framework [60], which is a tool specifically designed to build, beside others, data stream mining and concept drift dealing methods. The MOA framework is based on the Waikato Environment for Knowledge Analysis (WEKA) workbench [138], and it already has implementations of some concept drift handling methods and data generators, like the SEA and STAGGER Concepts.

Besides the different configurations of the proposed method and the other state-of-the-art concept drift handling methods, we propose to use other some classical strategies to make a more accurate analysis of the results in the experiments, and also propose the *Oracle* classifier selector as a upper bound for Neighborhood based methods for dealing with concept drift scenarios. Each of these strategies has a different purpose in the experiments, as described bellow.

***Single Classifier**:* in this approach a single, and possible strong, classifier is trained and used to classify all unsupervised instances. This classifier is never updated or discarded. This approach can be used to check how a classical classification method behaves in the benchmark employed in the tests. It may give an indication of the presence of concept drifts in the dataset, since it is expected that if the dataset really contains a concept drift, a statically trained classifier will suffer one or more performance drops over time.

***Naive Combination**:* in this method a new classifier is trained for each new supervised batch and added to a pool of infinite size. When classifying an unlabeled instance, the method just combines all classifiers in the pool by means of the majority voting. The Naive combination was not designed to cope with real concept drift scenarios, since no knowledge discard or selection method is implemented. This approach may be included in some tests to check how a method without any explicit mechanism to deal with concept drifts behave in the tested datasets.

This approach can also indicate if the concept drift is generated by changes in $P(\mathbf{x})$ only (virtual concept drift), since it is expected that in this scenario a method that is always aggregating data in its knowledge base may improve its performance (i.e. at each time step an unknown part of the distribution may be discovered by the method). Nevertheless, as the *Single Classifier* method, the *Naive Combination* may suffer from severe accuracy drops under scenarios containing real concept drifts, since under these scenarios the *Naive Combination* will keep using old classifiers that may be inaccurate

under the current concept.

Oracle: A common approach employed when testing DCS methods is to compare the performance of the proposed method with an oracle, which can be considered a theoretical upper limit of the performance of the pool from where the classifiers are selected [70, 61, 139]. Given a pool of classifiers P, the basic idea of the Oracle is to indicate if it is possible to build an ensemble of classifiers using a subset of P that is capable of correctly classify the test instance x.

The Oracle performance can be estimated by checking if there is at least one classifier in P capable of correctly classify x, where in this scenario x is marked as correctly classified by the Oracle. The Oracle idea can be easily extended to verify the performance of a DCS-based method designed to handle concept drifts. Considering x_t the test instance that arrived at moment t, and P_t the pool of classifiers at t, the instance x_t can be marked as correctly classified by the Oracle if there is at least one classifier in the pool P_t that can correctly classify x_t.

The Oracle can be seen as any other classification method (i.e. as any other Classification Engine in the Dynse framework), thus a metric like the *Accuracy versus Time* or the *Average Accuracy* (See Section 5.1.6) may applied in the Oracle in order to compare the theoretical upper bound of those metrics achieved by the Oracle versus the one achieved by the method being tested. It is important to observe that it only make sense to compare a method with the Oracle if both of them have the same classifiers in the pool at any given time t, or at least the classifiers are similar (e.g. the same classifier using a different training subset taken randomly from a bigger set). An example of a *Accuracy Versus Time* plot containing a comparison between the Oracle and the Dynse framework (CE =K-E, $M = 4$, $k = 5$, no pruning engine and Hoeffding Trees as the base learner) in the SEA Concepts problem is showed in Figure 31, where the accuracy achieved by the Oracle in each batch can be seen as a upper limit. In other words, the plot in Figure 31 can give an indication of the limit of improvement that is possible by tuning the DCS considering the same pool of classifiers.

Besides its popularity as a upper bound for DCS-based methods under static environments, as far as studied in this work, the *Oracle* based upper bound was not employed in any work that deal with concept drift scenarios. It is important to mention that, besides being valid as an upper limit for the DCS methods, the Oracle may be considered an overly optimistic accuracy limit [139, 136].

5.1.2 Method Evaluation Under Real Concept Drifts

In order to check the behavior of the proposed method under different concept drift scenarios, both artificial and real datasets will be used as benchmarks. As stated in

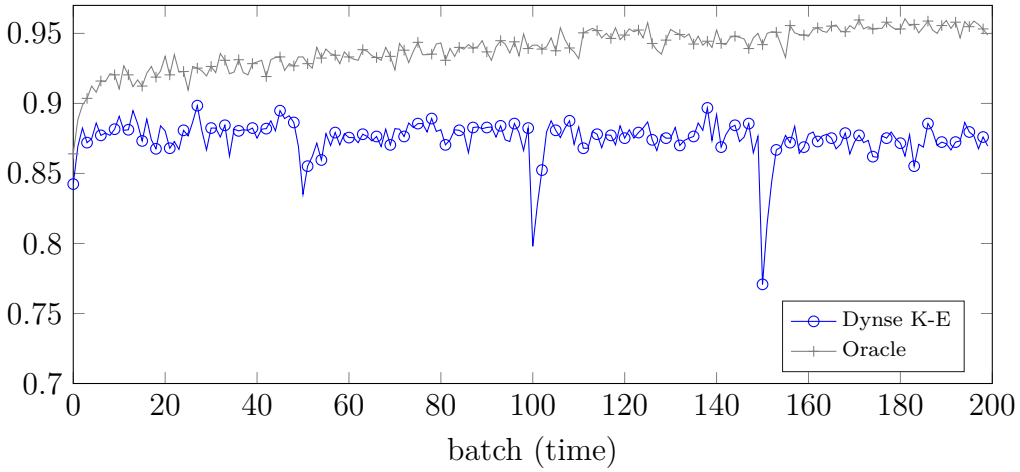


Figure 31 – Oracle versus the Dynse Framework accuracy over time plot example in the SEA Concepts problem.

the objectives of this work (Section 1.2), the proposed framework should be flexible and adapt to concept drifts with different properties, thus requiring several benchmarks, each containing and specific concept drift scenario, to evaluate the proposed framework.

The first test battery will include some common artificial datasets found in literature. As mentioned in Subsection 2.7.1, artificial datasets are ideal for analyzing a method behavior in a concept drift scenario, since both the nature and the moment when the concept drift happens are known, and the use of common datasets can make easier the results comparison with others authors works. To verify the framework performance under abrupt concept drift scenarios, the STAGGER and SEA Concepts will be used as benchmarks, which contains severe and intersected concept drifts, respectively. The Moving Checkerboard datasets family and the Gauss dataset configurations proposed in [6] are employed to check the proposed framework under Gradual concept drifts scenarios, where the Gauss benchmark is also be used to test the proposed approach in a problem where classes are added/removed over time.

The proposed framework is also be put to test in some real world datasets in order to check its behavior in a real environment. As one can see in Subsections 2.7.3 and 2.7.4, the real world datasets found in literature usually have more features, thus leading to a bigger classification challenge. It is important to reinforce that, despite the fact that these datasets should contain real world classification tasks, they lack information about their concept drift properties (e.g. when the concept drift happened, its speed, severity, ...), and it is not even possible to know if a concept drift really happened in these datasets.

The real world benchmarks commonly found in the literature that will be employed to assess the proposed approach performance are the Nebraska Weather and Forest datasets. Besides its popularity, the Electricity dataset was not considered in the tests due to its high correlation between test samples and lack of evidence of any concept drift, as discussed

in Section 2.7.3. In this work the PKLot is proposed as a real world benchmark to verify concept drift dealing methods performance and it is employed to asses the proposed framework. In Section 5.1.4 the PKLot dataset is briefly described, and an experimental protocol is defined for the PKLot benchmark.

5.1.3 Method Evaluation Under Virtual Concept Drifts

As stated in Section 1.2, the proposed framework should be able to deal with virtual concept drifts, specifically, distribution changes concept drifts. To asses the framework under virtual concept drift environments, small and possibly biased training batches will be taken from well known datasets (e.g. the datasets available at the UCI Repository [49]) that are not considered to have real concept drifts. This strategy were used in several works, like in [21, 119, 54, 39].

In order to generate the biased training sets to simulate a virtual concept drift, the following approach will be used: given a dataset that does not contain any concept drift, at each time step, a random sample and its $G - 1$ nearest neighbors are taken from the dataset and then used in the training phase (i.e. G training samples are used). The training samples are removed from the dataset, and then G samples are randomly taken for the testing phase (differently from the training phase, all G samples are taken randomly during the testing phase). The testing samples are also removed from the dataset, and a new step begins. This process is repeated until the dataset is empty.

Note that this approach may generate a problem that is specially difficult in the first time steps, since just some small regions of the feature space are known by the classifiers, and the testing instances may be generated in any region of the feature space. Figure 32 exemplifies this approach in a two-dimensional problem containing three classes, where the circles $D1, D2$ and $D3$ represent portions of training data taken at times $t1, t2$ and $t3$, respectively. As one can see, each training data chunk covers a small problem area, thus leading to a virtual concept drift (i.e. the distribution changes over time). It is important to notice that there is no change in the *a posterior* probabilities, therefore the problem does not contain real concept drifts.

In Figure 32 it is also exemplified a scenario where new classes could appear over time, since the classification system would receive a training dataset containing samples from the class *Class 3* only at time $t3$. In this scenario the class addition could also be considered a virtual concept drift, since there is no posterior probabilities changes, but the models should be updated considering that the problem frontiers from a new area are now known. This approach, where is considered that new classes will appear over time, is applied in [53].

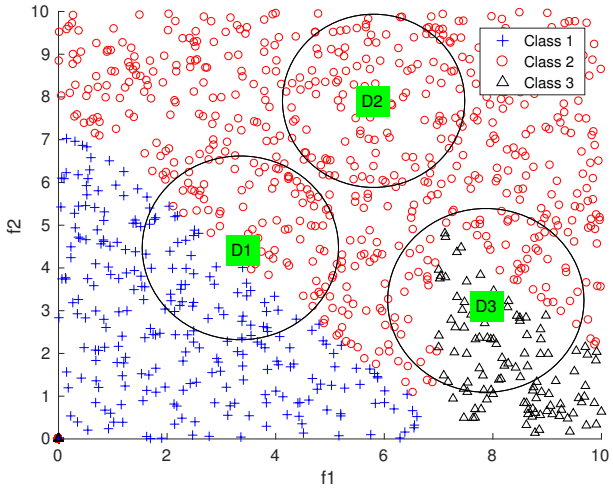


Figure 32 – Training data in a two dimensional problem containing three classes. Each circle marked as $D1$, $D2$ and $D3$ represents a portion of the data taken for training at times $t1, t2$ and $t3$, respectively.

5.1.4 Concept Drifts in the PKLot Dataset

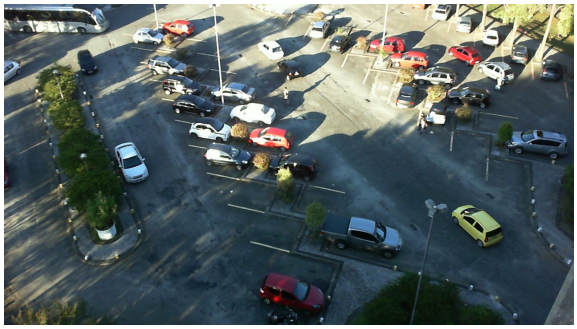
In this work the PKLot dataset is proposed as a real world benchmark to check the performance of concept drift handling methods. This dataset was introduced in Almeida et al.[140] and it was further extended in Almeida et al.[8]. It contains 12,417 images, collected from two different parking lots (UFPR and PUCPR) under several weather and light conditions. In the UFPR parking lot the images were also collected in two different angles and heights (UFPR04 for the imagens collected from the 4th floor of a near building and UFPR05 for the imagens collected in the 5th floor).

Each image present in the PKLot dataset has an associated XML file which contains the image information, such as the position and class (occupied or empty) of its parking spaces. Figure 33 shows some image examples from the PKLot dataset. As one can see, due to the nature of the images, which were collected under different weather and light conditions, and due to the presence of more than one parking lot and capture angle, this dataset has the potential of containing several concept drifts.

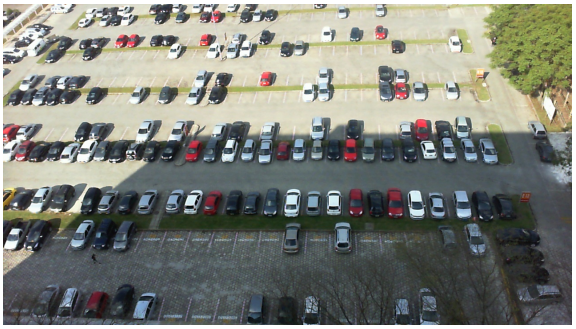
The results achieved in [140, 8] demonstrated that a *strong classifier*, created using 50% of the samples present in the dataset is able to achieve an accuracy close to 100%. Nevertheles, in the same works it is demonstrated that the change between different parking lots may cause a severe drop in the accuracy (e.g. a classifier trained with the UFPR04 may not perform well when tested in the PUCPR images). Thus, in this work this dataset will be used as a real world benchmark to test methods to deal with concept drifts.

In order to increase the test difficulty and possibly add a virtual concept drift in the problem, just a small amount of supervised samples will be given for training for every day present in the dataset. All experiments that will be conducted in this dataset will

(a) UFPR04 image example



(b) UFPR05 image example



(c) PUCPR image example

Figure 33 – Image samples from each parking lot/angle of the PKLot dataset containing several lighting conditions

use the Local Binary Patterns (LBP) with uniform patterns [141] as the feature set, due to the good results of these features in [140, 8] and simplicity. Next follows the complete experimental protocol for the PKLot dataset:

- The problem is defined as classifying each parking individual space as vacant or occupied, as in the original work [140, 8].
- The LBP with uniform patterns must be extracted from each individual parking space to be used as the feature set (59 features).
- Days containing less than 50 samples for each class (vacant or occupied) will not be considered.
- The parking lots are presented in the order UFPR04, UFPR05 and PUCPR. The images collected in each parking lot will be ordered in the chronological order. Each day will represent a time step (i.e. the days will be fed neatly to the classification method). Thus, the order will be: Day1_UFPR04, Day2_UFPR04, ..., Last_UFPR04, Day1_UFPR05, ..., Last_UFPR05, Day1_PUCPR, ..., Last_PUCPR.
- For each time step, 50 supervised samples of each class (i.e. 100 samples) referring to the current day will selected randomly and given for training. The remaining instances of the day will be used for testing (the training instances are not used in the test).

This protocol was designed in order to simulate a real world scenario, since a human supervisor can easily label 100 instances every day in a few minutes using a tool specifically designed for this purpose (e.g. a graphical tool where the human supervisor just clicks in the occupied spaces of some images collected in the day). The parking lots order was defined in order to first introduce an angle of view change (i.e. UFPR04 to UFPR05), and then introduce an angle of view change accompanied of an environment change (i.e. UFPR05 to PUCPR). The tests using the defined protocol in the PKLot dataset are presented in Section 5.10, altogether with tests designed specifically to demonstrate that this dataset may be used as a benchmark for testing methods that deal with concept drift problems.

5.1.5 Concept Recurrence

The proposed framework should deal with concepts recurrence (Section 2.4), where it is expected that previous acquired knowledge (i.e. classifiers in the pool) should be reactivated when necessary. The datasets considered for testing recurrent concepts will be the same specified in Subsection 5.1.2, since a recurrence can be easily simulated through the repetition of some previous concepts (e.g. in Chen et al.[43] a configuration of the STAGGER Concepts where the first concept is repeated is used).

In the concept recurrence tests the main objective will be to verify if classifiers related to the old concept that repeated are reactivated, and if it leads to a better performance by, for instance, comparing the accuracy of the proposed framework when the concept first appeared versus its performance when it repeated. To this end, some of the tested scenarios will not include a pruning approach in order to check if the classifiers from the same concept created in the past are reactivated.

5.1.6 Metrics

When evaluating the performance of classification methods in static environments, some common approaches may include classic metrics like the accuracy, false/true positive/negative rates, and the Receiver Operating Characteristic (ROC) curve analysis [142]. Besides their importance, these classical metrics may not fully represent the quality of methods that deal with concept drift problems, since the performance of these methods varies over the time. As an example, consider a problem with abrupt severe concept drifts containing several long stable regions. A method that can keep a good performance in stable regions may achieve a high average accuracy when put to test, even if the method did not performed well under the concept changing regions, since in most of the time the method will be dealing with stable regions. This example shows that just the classical performance metrics are not sufficient to define how good a concept drift dealing method is. In this Section the metrics of performance used in this work are presented, which are

inspired in the metrics used in the state-of-the-art works, specially the Ensemble and Local Region-based methods (Sections 3.4 and 3.5, respectively).

Average Accuracy: this classical metric is often used in works that deals with the concept drift problems. The basic strategy is to compute the final accuracy achieved when considering all presented test instances/batches [12, 42, 103, 6, 119, 17, 34]. As discussed in the beginning of this Section, this metric may be a good indicator of performance, nevertheless it may not suffice under a concept drift scenario when used alone. However, some works in the state-of-the-art, such as [53, 54, 115, 122], use the average accuracy as the only metric when measuring the quality of different methods. In this work the average accuracy is one of the metrics considered and, in order to increase the discriminative power of the metric, the accuracy standard deviation between test batches is also considered. The accuracy standard deviation may be used as an indicator of the stability of the method being tested.

Accuracy over Time: A common approach used in most of the works [29, 46, 43, 12, 20, 112, 42, 6, 119, 11, 2] is to plot the accuracy over the time, where the time can be represented, for instance, by the testing batch number. An example of this representation is given in Figure 34, where the concept changing areas are marked as vertical dashed lines (considering that the moment when the concept drift happened is known). As one can observe, this representation is ideal to verify the method behavior over time and, when the moment of the concept drift is known, it can be used to check how well the method performed in the concept changing area. This metric is used in this work as the main approach to verify the behavior of the methods over time.

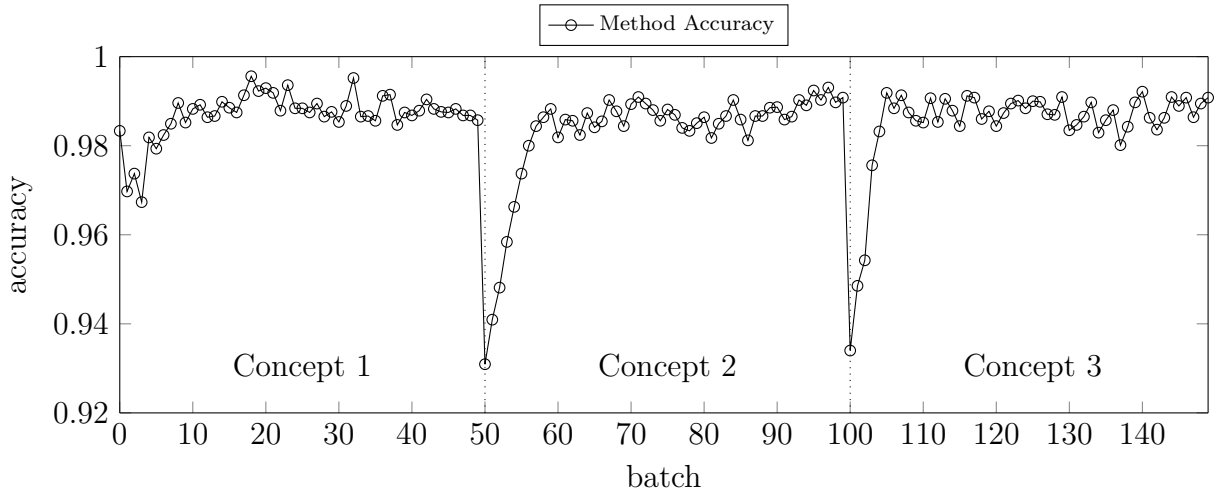


Figure 34 – Accuracy versus Time (batch) example

Memory Usage: This metric can be used to evaluate the amount of resources taken by the method being tested. Some authors define the memory usage in therms of the RAM-Hours metric, where one RAM-Hour is defined as 1GB of RAM used in a period of 1 hour. This metric is based on cloud computing services, where the resources can be

deployed and charged by hour [103, 5, 15]. In this work a different strategy is employed, where the memory consumption will be represented by a plot similar to the accuracy over time, where at each time step the amount of memory used by the tested method will be plotted [103, 33]. The final amount of memory needed by the method will be defined as the maximum amount of memory consumed by the method, considering all time steps of the problem. In [110] a similar approach is used, although in [110] the final memory used is defined as the average memory consumed.

The *Average Accuracy*, *Accuracy over Time* and *Memory Usage* are the main metrics used in this work, since they are the largely used to verify the performance of methods that deals with concept drift by means of ensembles or local regions [46, 12, 20, 112, 119, 110, 11]. When suitable, a Bonferroni-Dunn test (see Section 2.9) with a 95% of confidence will be used to compare a single configuration of the proposed Dynse framework with the state-of-the-art methods. The Wilcoxon Signed-Ranks test [143, 73] is used to make a pairwise comparison of different configurations of the Dynse framework.

5.2 Tested Methods

In this Section both the proposed framework configurations and the state-of-the-art methods used in the experiments are presented. All experiments were designed using the MOA [60] framework as a tool for implementing the proposed method. Most approaches in the state-of-the-art employed in this work are already part of the MOA framework, thus giving a reliable and well tested set of methods for comparison. Due to the vast number of methods tested, each of them possibly containing several parameters that could be tuned for optimization, in this work all comparisons between different methods use the methods default configurations (available at the MOA framework). By following this guideline we aim to provide a fair comparison without the need to deal with an explosive number of parameters combinations during the tests.

Therefore, only the default configuration of the proposed Dynse framework is tested against the state-of-the-art (although different classification and pruning engines are tested in order to demonstrate that several DCS approaches may be used to deal with concept drifts). Table 7 contain the main acronym of the state-of-the-art methods used in this work, and the main techniques used to deal with concept drifts used by the method (Window based, Trigger based, etc.). For a more detailed description about the state-of-the-art methods consult Chapter 3. The methods discussed Subsection 5.1.1 are also present in the Table 7, since they were used as baselines for several tests.

All methods, including the Dynse framework, was configured to use Hoeffding Trees [82] as base learners. This base learner was chosen since some of the tested methods require an online learner. Note that the DDM and EDDM methods are just triggers. To make the

comparisons fair, these triggers uses a pool of Hoeffding Trees as the base learner, where each new supervised batch received is used to create a new classifier, and the classifiers in the pool are combined using the plurality vote.

Table 7 – Acronyms of the State-of-the-art tested methods and main types.

Acronym	Description	Type
AUE	Accuracy Updated Ensemble (AUE) method proposed in [110].	Ensemble
AWE	Accuracy-Weighted Ensembles (AWE) method proposed in [109].	Ensemble
DDM	The Drift Detection Method (DDM) method proposed in [51].	Trigger
EDDM	The Early Drift Detection Method (EDDM) method proposed in [91].	Trigger
HAT	The Hoeffding Adaptive Tree proposed in [102] using the ADWIN [92] method as a trigger.	Trigger
LevBag	The Leveraging Bagging method proposed in [103] using the ADWIN [92] trigger to detect changes.	Trigger / Ensemble
OzaAD	The method proposed in [42] using the ADWIN [42] trigger.	Trigger / Ensemble
OzaAS	The method proposed in [42] using Adaptive-Size Hoeffding Trees.	Trigger / Ensemble
NaiveC	The Naive Combination method discussed in Subsection 5.1.1.	-
SingleTrain	The Single Static Classifier approach discussed in Subsection 5.1.1.	-

Several configurations of the proposed framework are tested in this work. The configurations include the use of different classification engines, pruning engines, number of neighbors considered for the DCS and the accuracy estimation window size. The main variables that may be changed in the Dynse framework are presented in Table 8, altogether with the possible values that can be used in each variable during the tests. Note that the Oracle theoretical DCS method, discussed in Subsection 5.1.1, is used in the tests as a "perfect classification engine" in order to give an upper bound for the classification engines. All results presented in this Chapter are an average of 10 executions.

5.3 Benchmarks Configuration

In Section 2.7 there is a description of some of the most common benchmarks used in the literature. Despite being valid, this description may be too general since distinct authors may employ a different strategy to use a benchmark in order to test a concept drift handling method. To avoid any misinterpretation, in this Section the protocol employed

Table 8 – Summary of the components and parameters of the Dynse framework.

Param.	**Description**	
M	The *accuracy estimation window size* used	
K	This parameter defines the number of neighbors of the test instance x selected from the *accuracy estimation window*	
D	The pool maximum size	
l	The Slack Variable for the K-E method (See Section 4.7)	
CE	The classification engine used by the Dynse to dynamically select the classifiers. The dynamic selection based methods described in Section 2.8 were used in the tests with the acronyms bellow:	
	Acronym	**Dynamic Classifier Selection (DCS)**
	OLA	The DCS-LA OLA
	LCA	The DCS-LA LCA
	Priori	The A Priori method
	Post	The A Posteriori method
	K-U	The K-U method
	K-UW	The K-UW method
	K-E	The K-E method
	Oracle	The The Oracle hypothetical dynamic selection method discussed in Section 5.1.1, considered as an upper bound
PE	The pruning engine used to prune classifiers from the pool. The pruning approaches discussed in Section 3.8 were used in the tests with the acronyms bellow:	
	Acronym	**Pruning Engine (Method)**
	Inf	No prunnig engine (the pool may increase its size indefinitely)
	NN	The NNPrune pruning algorithm (See Section 4.6)
	Age	Age based pruning (replace the oldest)
	Acc	Accuracy based pruning (replace the worst performing classifier considering the current accuracy estimation window)

for each benchmark used is briefly described. Unless explicitly stated, the tests in this Chapter will follow this protocol to use the benchmarks.

The main properties of each benchmark are showed in Table 9. Observe that depending on the benchmark used in the tests, a holdout or a test-then-train approach is used. In the holdout strategy, for each time step a train set is given, followed by an independent test set. In the test-then-train approach, at each time step the system must classify the received batch and, after the classification, the true labels of the batch are given and used for training [16, 89]. The acronyms and a brief discussion about the datasets used in the tests are follow given.

STAGGER: The tree different concepts defined in the original work [29] are used, plus a fourth concept, with the same boundaries of the first one, is generated to simulate a recurring scenario. During the experiments, there is a concept change for every 10 steps. A holdout evaluation is made [42, 15, 16], where at each time step, 20 supervised

samples are given for training, and then 200 unsupervised samples are given for testing.

SEA: This benchmark refers to the SEA Concepts problem. The testing procedure is inspired in [6], where for each time step, a supervised batch containing 250 samples is given for training, and another batch containing 250 samples from the same concept is generated for testing. The four original concepts of the problem are used [46], and the concept is changed for each 50 steps, thus generating a test with 200 steps in total. A holdout approach is used, thus the testing samples are never used as training ones. It is worth mentioning that in [6] only the training instances contain noise, while in the tests of this work both training and testing instances have 10% of noise.

SEARec: This benchmark follows a similar specification of the SEA benchmark. Nevertheless, in this configuration there is a concept change for every 25 steps (in the SEA benchmark the concept changes for every 50 steps). As in the SEA benchmark, there is a total of 200 steps in the problem, thus all concepts are repeated once in the order $\theta = 8, \theta = 9, \theta = 7, \theta = 9.5, \theta = 8, \theta = 9, \theta = 7, \theta = 9.5$.

CkrE, CkrP and CkrS: These configurations refer to the Rotating Checkerboard problem considering a *exponential*, *Gaussian pulse* and *sinusoidal change*, respectively. There is a total of 400 time steps, where at each time step 25 samples are given for training, and 1024 are given for testing (Holdout test). This version of the benchmark is proposed in [6], where the authors also provide a link to download the datasets.

Gauss: The Gauss Dataset with Class Addition/Removal used in [6]. This problem contains 300 time steps, where at each time step 20 samples are given for training, and 1024 samples are used for testing. All classes are constantly drifting. At the time step 120 a new class is introduced, and at time 240 one of the classes is removed. The authors in [6] made this dataset available for download, nevertheless only the class priors of the testing batches are available (i.e. it is possible to know how many instances of each class are available in the testing batch, but it is not possible to know the true label of each individual instance), thus the Oracle accuracy was not computed for this dataset.

Nebr: The Nebraska Weather dataset using the same configuration employed in [6] and [11], where only the eight features with a missing feature rate less or equal than 15% were used, and the remaining missing values are replaced by the mean of the features in the preceding and following samples. The dataset is ordered in a chronological order and an interleaved batches approach is used. At each time step, 30 samples are used for training, and the next 30 samples are given for testing. In the next time step, the testing samples are used as a train batch (the real labels are given), and the subsequent 30 samples are given for testing. This procedure is repeated until all samples are used. There is a total of 604 time steps in this problem.

For: This benchmark refers to the Forest Cover Type problem available at the

UCI repository [49]. Besides its popularity as a test-then-train dataset (i.e. a instance is used for testing, and then for training in a stream fashion) [42, 103, 2, 107], in Bifet[58] it is argued that this dataset may present a high correlation between its data, making it a trivial problem to a classifier that just adds new information in its knowledge base without detecting any concept drift. Thus, in order to increase the difficulty of the problem, the instances are presented in the same order present in the original dataset, and at each time step, a batch containing 200 samples is given for training, and a batch containing the next 2,000 samples is given for testing. The dataset is used in a holdout form, thus, the testing samples are never used for training and, conversely, the training samples are never used for testing. This process is repeated until all samples are used, hence generating a problem with 264 time steps.

Dig and Let: The Digit [144] and Letters [145] problems available at the UCI [49] repository with artificially introduced virtual concept drifts. To introduce a virtual concept drift, the methodology described in Section 5.1.3 is employed. The parameter G, that defines the batch size used for train/test, is set to 50 ($G = 50$). The Digit (Dig) dataset generated 56 time steps, while the Letters (Let) generated a problem with 200 time steps.

PKLot: The PKLot dataset [140, 8] using the protocol specified in Section 5.1.4 (days ordered in the chronological order and for each day 100 samples are given for training, and the remaining instances are used for testing).

Table 9 – Main properties of each benchmark used. The train/test sizes refer to the batch size given for training/testing at each time step.

Benchmark	Drift Type	Test Type	Feat.	Classes	Steps	Train Size	Test Size
Real Concept Drift Benchmarks							
STAGGER [29]	Abrupt Real	Holdout	3	2	40	20	200
SEA [46, 6]	Abrupt Real	Holdout	3	2	200	250	250
SEARec[1] [46]	Abrupt Real	Holdout	3	2	200	250	250
Ckr[2] [6]	Gradual Real	Holdout	2	2	400	25	1024
Gauss [6]	Gradual Real	Holdout	4	2	300	20	1024
Real World Benchmarks							
Nebr [6, 11]	-	Test-Train	8	2	604	30	30
For [49]	-	Holdout	54	7	264	200	2000
PKLot [140, 8]	-	Holdout	59	2	82	100	-
Virtual Concept Drift Benchmarks							
Dig [49]	Virtual	Holdout	64	10	56	50	50
Let [49]	Virtual	Holdout	16	26	200	50	50

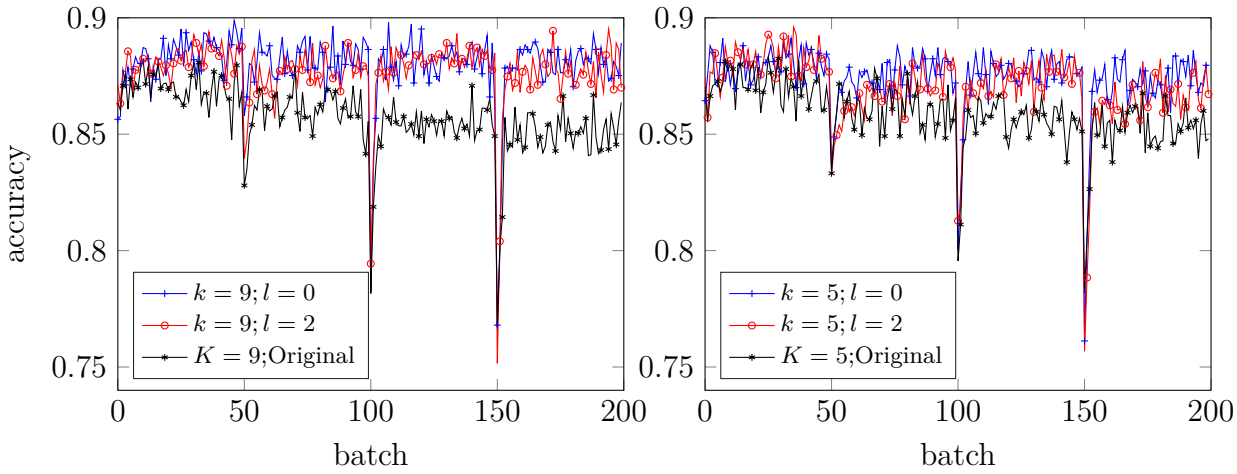
[1] The SEARec is a variation of the original SEA benchmark containing recurrences
[2] Configuration valid for all checkerboard benchmark variants (CkrE, CkrP, CkrS)

5.4 KNORA-Eliminate Slack Variable Tests

In this Section the original configuration of the SEA Concepts benchmark, which contains 10% of noise, is used to test the behavior of the slack variable defined for the K-E method in Section 4.7. The Dynse framework was configured with an accuracy estimation

window size equals to 4 without any pruning strategy. The tested scenarios include the original version of the K-E and the new proposed version including the slack variable l, as classification engines.

Figure 35 shows the accuracy over time plots for the original version of the K-E method, and the proposed version configured with a slack variable of 0 ($l = 0$) and 2 ($l = 2$). The tested scenarios in Figure 35 includes a neighborhood of size 9 and 5 (Figures 35a and 35b, respectively). Note that, at first sight, the $l = 0$ configuration may seems to be equal to the original method. Nevertheless, one may observe that in the original version the neighborhood N_x size is decremented when no classifier is able to classify all instances in N_x, while in the proposed modification the slack variable l value is incremented in this same scenario. The results in Figure 35 indicates that the introduction of the slack variable, and the behavior change when no classifier is able to classify the entire neighborhood (increase of the slack variable value), generated better results in the presented problem for both k values tested.



(a) K-E using $k = 9$ (b) K-E using $k = 5$

Figure 35 – Original and modified version of the K-E method as the classification engine in the SEA Concepts benchmark.

To better understand the behavior presented in Figure 35, see Table 10, where the average number of neighbors in N_x correctly classified by the ensemble, the average ensemble size and average accuracy for the original and modified version of the K-E are presented. Note in Table 10 that the ensemble size is similar when comparing the original and the version with a slack of size 0 of the K-E, nevertheless the proposed modification generate a better result. This behavior may be explained by the average number of neighbors correctly classified by the ensemble. For instance, in the original version this number is 5.88 for the neighborhood of size 9 ($k = 9$), indicating that often it is necessary to consider a much smaller neighborhood in order to make possible to at least one classifier correctly classify the entire N_x. This may be caused by noise too close to the test instance, as discussed in Section 4.7. On the other hand, the modified version

keeps the neighborhood size, but starts to accept some errors by the increase of the slack variable size. Thus, even if the noise is close to the test instance, a bigger neighborhood is correctly classified by the modified version when compared with the original one.

Table 10 – Averages of the number of correctly classified neighbors, ensemble size and accuracy in the SEA Concepts problem (with noise) using the original K-E and the proposed modification.

k	Configuration	Avg. Neighbors Correct	Avg. Ensemble Size	Avg. Accuracy
$k = 5$	Original	3.89	74.33	85.90%
	$l = 0$	4.56	75.49	87.44%
	$l = 1$	3.93	82.86	87.39%
	$l = 2$	2.99	90.33	87.02%
	$l = 3$	2.00	95.38	85.84%
	$l = 4$	1.00	98.79	84.60%
$k = 9$	Original	5.88	68.94	85.73%
	$l = 0$	8.18	69.08	**88.00**%
	$l = 1$	7.76	73.78	87.87%
	$l = 2$	6.95	80.92	87.73%
	$l = 3$	5.99	86.82	87.36%
	$l = 4$	5.00	91.66	86.67%

As the modified version selects classifiers that correctly classifies more instances in the neighborhood, it is likely that the generated ensemble will be more competent than the one generated by the original version. This also may explain the accuracy decrease when the slack variable is started with values bigger than 0. When the value is started with 0, only classifiers that correctly classifies the entire neighborhood will be selected when possible (i.e. the slack is increased only if necessary), nevertheless, when the value is started with, for instance, 3, all classifiers that correctly classifies the $k, k-1, k-2$ and $k-3$ neighbors will be selected, even if there is a subset of classifiers able to correctly classify the entire neighborhood. Thus, by choosing bigger values of l, a more permissive configuration of the K-E will be generated, where classifiers that commit more errors in the neighborhood are accepted to be part of the ensemble. Due to the good results achieved, the proposed modified version of the K-E with a slack of 0 ($l = 0$) will be used in this work when defining the K-E as the classification engine.

5.5 Validation of the DCS approach under concept drift scenarios

It was argued in the course of this work that by incorporating a time dependency in the DCS approach, it is possible to use the DCS methods in order to deal with the concept drift problem. As a tool capable of incorporating the time dependency in any DCS-based method in this work it is proposed the Dynse framework, which is tested under

several artificially generated scenarios in this Section to validate the discussions presented in Sections 4.2, 4.3 and 4.4. It is worth mentioning that the tests in this Section were not designed to make any comparison with the state-of-the-art, thus the tests may involve non-standard benchmark configurations (different configurations of the Dynse framework and methods in the state-of-the-art are tested in standart benchmarks in Sections 5.6, 5.7 and 5.9).

All tests in this Section consider a pool of infinite size (i.e. no pruning engine). Tests using different pruning approaches are presented in Section 5.7. First, the DCS approach is tested in an intersected concept drift scenario, where it should be able to select a good custom classifier/ensemble for the test instance x, regardless of the train concept of each selected classifier. This assertion is justified since classifiers trained under previous concepts may still be relevant in some regions of the feature space under an intersected concept drift. To validate this, the Dynse framework configured with the K-E method as the classification engine, and $k = 5; l = 0; M = 4;$ was tested in the SEA Concepts benchmark problem without considering noise [1].

Since the SEA Concepts benchmark represents an intersected concept drift problem, it is expected that under the presence of concept j, some classifiers from a previous concept i ($i \neq j$) should still be selected by the Dynse framework. Table 11 shows the average proportion of classifiers trained with the concept i (lines) selected to be part of the ensemble when classifying a test instance in which j is the current concept (columns) - e.g. in the presence of Concept 2, 46.3% of the classifiers selected to be part of the ensembles were trained under the presence of Concept 1. The numbers under parenthesis represents the hypothetical perfect proportion that should be selected. The last line in Table 11 contains the average accuracy achieved under the presence of each concept.

Table 11 – Average accuracy and proportion of classifiers trained in each concept selected to classify the instances in the SEA Concepts problem without noise.

		Test Concept				
		Concept 1	Concept 2	Concept 3	Concept 4	Concept 1
Train Concept	1	100% (100%)	46.5% (47.8%)	34.5% (33.5%)	23.3% (24.0%)	26.3% (27%)
	2	0.0% (0.0%)	53.5% (52.2%)	30.8% (30.4%)	26.7% (26.4%)	25.9% (24.7%)
	3	0.0% (0.0%)	0.0% (0.0%)	34.7% (36.2%)	21.3% (21.9%)	23.8% (24.9%)
	4	0.0% (0.0%)	0.0% (0.0%)	0.0% (0.0%)	28.7% (27.7%)	23.9% (23.4%)
Accuracy		97.2%	97.6%	97.7%	97.4%	97.9%

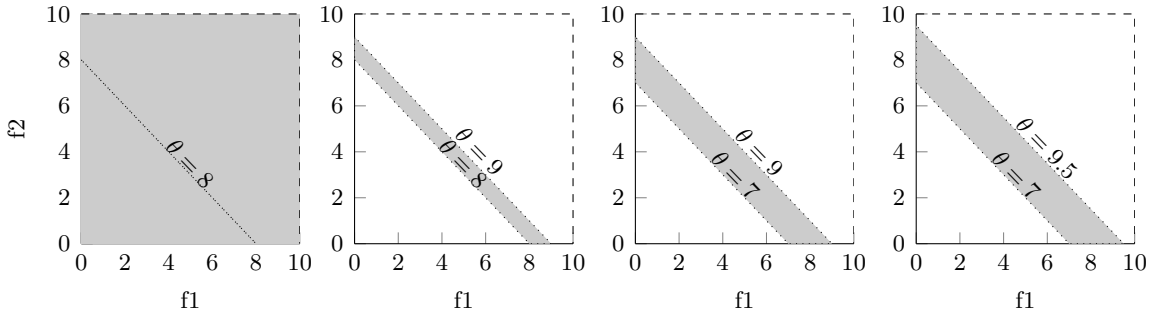
The proportions presented in Table 4 are close to the theoretical perfect values, indicating that with the Dynse framework, it is possible to create a DCS-based method

[1] The noise was removed in order to better visualize the behavior of the method in the Sea Concepts problem. Experiments including noise in the Sea Concepts benchmark are shown in Sections 5.6, 5.7 and 5.9.

capable of correctly select classifiers trained under the current concept and also capable of reusing classifiers trained under old concepts in regions where they are still useful.

Note that the Concept 1 was repeated as the last concept in Table 11 in order to simulate a recurrent concept scenario. The proportion of selected classifiers close to the theoretical perfect proportion, and the accuracy increase in the last concept compared when it first appeared (97.9% versus 97.2%, respectively), further indicates the ability of the Dynse framework to correctly select a promising ensemble when the pool contains classifiers trained under several concepts, and its ability to reactivate classifiers under a recurrent concept scenario.

In order to verify the impact of the local region size (the neighborhood size k in the Dynse framework) discussed in Section 4.4, again the SEA Concepts benchmark without noise is used. Nevertheless, in this test only the area that changed its *a posteriori* probabilities between the concept change was used to generate the test instances. In the experiment, at each time step 250 samples regarding to the entire feature space are given for training, and 250 samples drawn from the changed area only are handed for testing. The concept changes for every 50 steps, and there is a total os 200 steps. Note that in the first 50 steps there is no concept drift, thus test instances are drawn from the entire feature space in these steps. The gray regions in Figures 36a to 36d represent the areas where the testing samples were generated for each concept.



(a) Concept 1 (b) Concept 2 (c) Concept 3 (d) Concept 4

Figure 36 – The gray areas correspond to the regions in the feature space containing *a posteriori* changes between concepts, which were used to generate the test instances for the results presented in Table 12.

Since the test instances are drawn only from regions where the *a posteriori* probabilities changed, the Dynse framework must select mostly classifiers trained under the current concept to classify the incoming test instances. Note that the instances in the changed areas are the closest ones to the boundary of the problem, making it even more difficult. In this test the K-E and K-U were tested as classification engines, where the number of neighbors varied from 3 to 9, and the slack variable l(see Section 4.7) varied from 0 to 2 for the K-E. The remaining of the configuration is the same from the previous test (i.e. $M = 4$ and no pruning engine). The results are shown in Table 12.

Table 12 – K-E and K-U as classification engines in the SEA Concepts problem. The test instances were taken from the changed *a posteriori* areas only. The best results are shown in bold.

Engine	k	Con. 1	Con. 2	Con. 3	Con. 4	Avg.
K-E $l = 0$	3	97.0%	71.0%	91.2%	84.8%	86.0%
	5	97.3%	74.9%	92.0%	87.2%	87.9%
	7	97.3%	78.4%	92.4%	88.1%	89.1%
	9	97.4%	78.9%	92.6%	89.2%	**89.5**%
K-E $l = 1$	3	96.4%	50.1%	87.5%	73.9%	77.0%
	5	97.0%	61.8%	89.8%	79.6%	82.0%
	7	97.2%	67.6%	91.2%	82.4%	84.6%
	9	97.2%	72.2%	91.6%	84.3%	**86.4**%
K-E $l = 2$	3	95.3%	23.5%	82.1%	57.4%	64.6%
	5	96.1%	41.2%	86.5%	69.5%	73.4%
	7	96.7%	52.7%	89.0%	75.2%	78.4%
	9	96.9%	59.7%	90.0%	78.3%	**81.2**%
K-U	3	96.0%	35.0%	85.5%	63.6%	**70.0**%
	5	95.6%	24.6%	84.4%	56.8%	65.4%
	7	95.4%	20.7%	83.7%	53.1%	63.2%
	9	95.2%	17.4%	83.1%	50.6%	61.6%

As can be observed in Table 12, larger values of k may increase the accuracy when considering the K-E as the classification engine. At first sight, this result seems to contradict the discussion presented in Section 4.4. However, the K-E works by selecting the classifiers that correctly classify all $(k - l)$ instances in the neighborhood. Thus, for larger values of k, even if some neighbors may belong to areas that did not drift, these added neighbors will just make more difficult for a classifier to be part of the custom ensemble.

This also explains why larger values defined in the slack variable l decreased the accuracy in Table 12. Since l controls the maximum number of neighbors that can be misclassified for a classifier to still be selected to be part of the ensemble. One may argue that the slack variable should not be used in the K-E, as its use increased the error rate in all scenarios presented in Table 12. Nevertheless, we should remember that the experimental results in Table 12 represent a rather simple problem without any noise (the only "noise" is the one caused by a concept change, which will take at most 4 time steps to be removed every time a change occurs), and the presence of noise would make it impossible to correctly classify all k neighbors, as discussed in Section 4.7. More detailed experiments regarding the modifications made in the K-E method are presented in Section 5.4.

The results of the K-U as a classification engine in Table 12 show an interesting behavior, since larger values of k decreased the accuracy of the method. These results were somehow expected, since a larger local region increases the possibility of taking

neighbors in regions that did not change their *a posteriori* probabilities (i.e., outside the areas presented in Figure 36). Since the K-U selects all classifiers that correctly classified at least one neighbor, taking neighbors from outside the region that drifted may lead to a poor classifier selection.

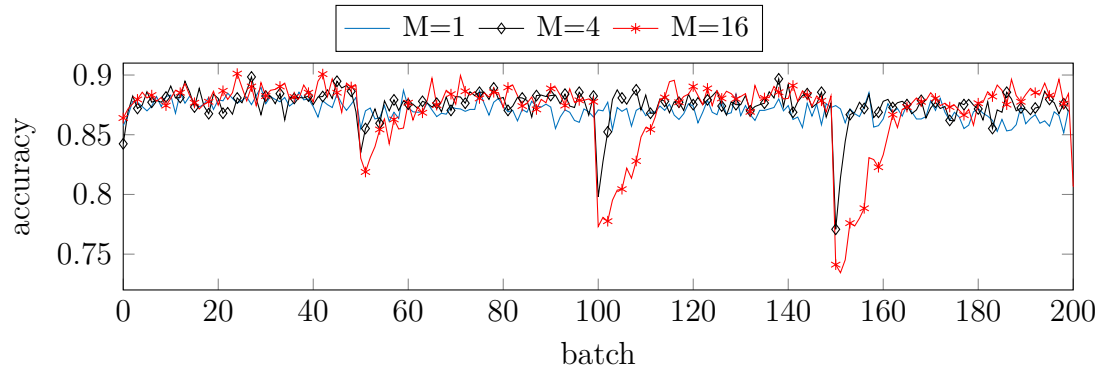
Although the K-E method performed better for bigger neighborhoods, it also presented good results for smaller values of k. Thus, it is possible to conclude that a small neighborhood size can be a good starting point for most DCS-based methods when dealing with concept drift problems, specially under the presence of local changes (i.e. intersected concept drift). This is an interesting result, since for non changing environments, the authors of the DCS methods reported in Section 2.8 often define a small neighborhood for their methods (usually a value between 5 and 10) [68, 70, 62], thus the very same values defined for the non changing environments may be used as a good starting point when adapting the methods for concept drift problems. Note that this result corroborates with the discussion presented in Section 4.4.

In order to verify the impact of the accuracy estimation window size M in a real concept drift scenario, the configuration of the SEA Concepts benchmark discussed in Section 5.3 (containing noise) is used. The average accuracy achieved for different M sizes considering the K-E and K-U as classification engines, and for the Naive Combination and the Oracle methods, are presented in Table 13. The complete plot containing the accuracy over time for some of the M sizes are presented in Figure 37. No pruning strategy was used in any scenario.

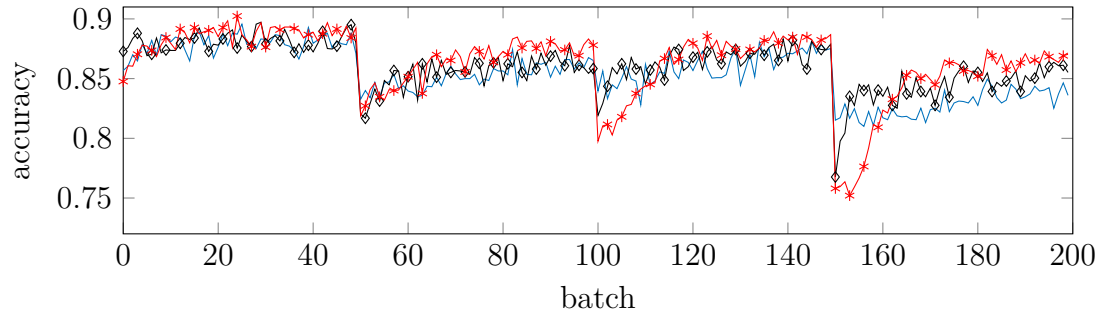
Table 13 – Average accuracy of the Dynse framework in the original SEA Concepts Benchmark (including noise), considering different accuracy estimation window sizes.

Classification Eng.	Accuracy Estimation Window Size (M)					
	$M = 1$	$M = 4$	$M = 8$	$M = 16$	$M = 32$	$M = \infty$
K-E $k = 5; l = 0$	87.07%	**87.48%**	87.38%	86.98%	85.95%	84.22%
K-U $k = 5$	85.48%	86.09%	86.46%	86.33%	85.54%	84.09%
NaiveC						83.71%
Oracle						93.90%

Table 13 and Figure 37 shows that by incorporating more data in the accuracy estimation window (i.e. bigger M values) causes a slower adaptation when the concept changes (i.e. more time is needed in order to recover from a concept drift). Notwithstanding, note that when the value of M is increased from 1 to 4 (and also from 4 to 8 for the K-U), a better average accuracy is achieved. By analyzing the plots in Figure 37, it becomes clear that, besides the larger value of M generated a slower recover under concept changes, the average accuracy was improved by a more accurate classification when the concept is stable, since for larger values of M the accuracy estimation window will take more time to effectively discard information in the presence of a change, but it will have more data,



(a) K-E Classification Engine



(b) K-U Classification Engine

Figure 37 – Average accuracy achieved in each test batch in the SEA Concepts problem described for different M values. In 37a the K-E with $k = 0; l = 2$ was used, while in 37b the K-U considering $k = 5$ was employed.

possibly covering a wider region of the feature space, in order to define the local region of the test instance, which is beneficial when the concept is stable. These results meet with the discussion about the stability-plasticity dilemma in the accuracy estimation window presented in Section 4.2.

Note that the K-U classification engine, which selects all classifiers that correctly classifies any instance in the local region, did not performed as well as the K-E classification engine in this test, indicating that a classification engine that may select classifiers that correctly classifies only a small portion of the local region may lead to suboptimal results under real concept drift scenarios. By considering the results presented in this Section, and also the discussion in Section 4.2, in this work an accuracy estimation window of size 4 ($M = 4$) will be considered as a good starting point for real concept drift scenarios, thus this value will be defined as the default value for the variable M. This value was chosen since it represents a good trade off between a fast reaction and a good performance in stable regions.

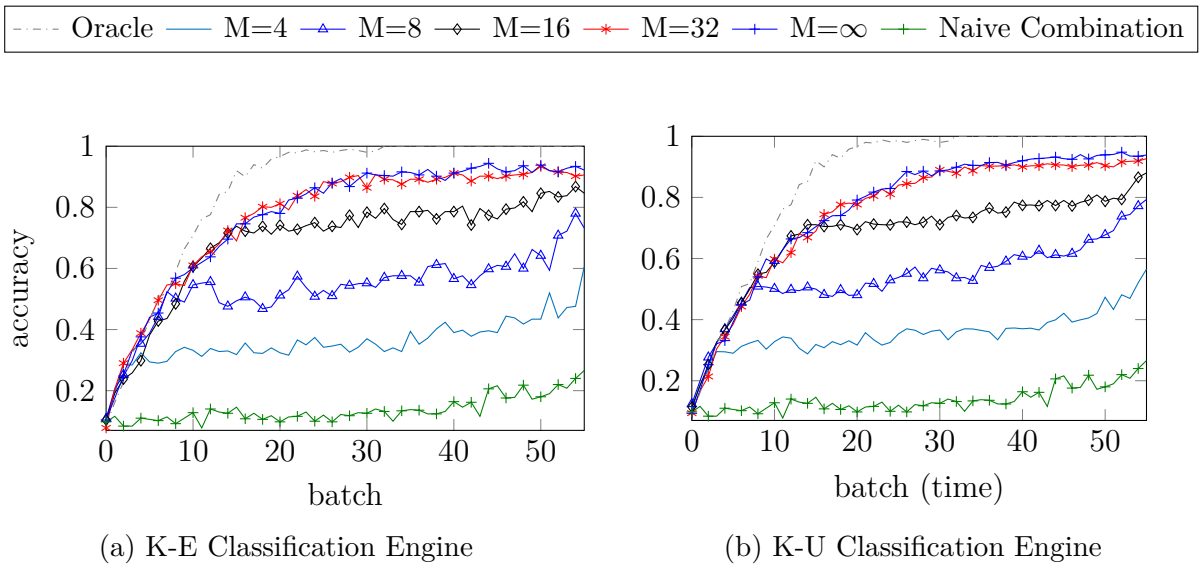
To test a virtual concept drift scenario, the digit recognition problem described in [144] with a virtual concept drift artificially introduced, as described in Section 5.1.3, was used. Table 14 contains the results achieved by varying the accuracy estimation window size for the Dynse framework using the K-E ($k = 5; l = 0$) and K-U ($k = 5$) classification

engines, without any pruning. For comparison purposes, Table 14 also shows the results achieved by the *Naive Combination* and the *Oracle* methods.

Table 14 – Average accuracy of the Dynse framework in a virtual concept drift scenario in the Digit dataset[144], considering different accuracy estimation window sizes

Classification Eng.	Accuracy Estimation Window Size (M)					
	$M = 1$	$M = 4$	$M = 8$	$M = 16$	$M = 32$	$M = \infty$
K-E $k = 5; l = 0$	13.78%	36.02%	53.46%	68.56%	76.79%	**77.17%**
K-U $k = 5$	15.02%	38.90%	54.00%	68.70%	74.4%	75.80%
NaiveComb						12.89%
Oracle						86.15%

Some interesting insights can be inferred from Table 14. As can be observed, the K-U achieved better results than the K-E-based classification engine for smaller values of M. This indicates that differently from a real concept drift scenario, a classification engine that selects a classifier to be part of the ensemble, even if it was able to correctly classify just a small portion of the neighborhood, may be beneficial in a virtual concept drift scenario. Clearly, it is possible to change the values of k and l for the classification engines in order to create configurations where the K-E performed better than the K-U for most M values. However, the point is to show that different classification engines may be more suitable to a problem, depending on the problem characteristics. Figure 38 contains a plot of the accuracy over time for the different accuracy estimation window sizes (M) tested in this problem considering the K-E (Figure 38a) and K-U (Figure 38b) as classification engines.



(a) K-E Classification Engine

(b) K-U Classification Engine

Figure 38 – Average accuracy achieved in each test batch for a virtual concept drift artificially introduced in the problem described in [144] for different M values.

It can be observed from Table 14 and Figure 38 that bigger values for M lead to better results in the virtual concept drift problem. These results corroborate the discussion

presented in Section 4.3, where is stated that the aggregation of previous data in the accuracy estimation window is beneficial, when no change in the *a posteriori* probabilities is expected (i.e. virtual concept drift only). Thus, in this work the accuracy estimation window size will be set to 32 ($M = 32$) when dealing to a virtual concept drift that affects $P(\mathbf{x})$. Since the dataset used for the virtual concept drift test is relatively small, it was possible to configure the Accuracy Estimation Window to keep all supervised instances received. Of course, this may be impossible in most problems in the real world, and thus, an interesting challenge for future works could be a method to maintain this window with a fixed size and covering the biggest possible feature space in order to better adapt to a virtual concept drift (see Chapter 6).

Observe in Table 14 and Figure 38 that even if the concept drift is only virtual, some classifiers may be highly specialized in a region, and thus, it may be suboptimal to use these classifiers to classify test instances that are too distant from the regions where the classifiers are specialized. This behavior can be demonstrated by the poor performance achieved by the *Naive Combination* method, since the method combines all classifiers, regardless of their regions of expertise. Also observe that in the beginning of the test the Dynse framework configured with large values for M generated results close to the *Oracle* upper bound, however after about 10 time steps the *Oracle* become the leading method, indicating that there is room for improvement for the methods implementation and configuration.

Concluding this Section, the presented results in the SEA Concepts indicate that a DCS-based approach is able to detect regions in the feature space that did not changed between different concepts, and use classifiers from past concepts in order to classify test instances located in these regions. These results meet with the discussion presented in Section 4.4 and the hypothesis that a DCS-based approach can be a natural answer to intersected concept drift problems. Results achieved under real and virtual concept drifts indicate that, as hypothesized in this work, the accuracy estimation window size must be related to the concept drift nature (time dependency in the accuracy estimation window size). An increase in the accuracy under the presence of a recurrent concept indicates that the DCS approach may benefit from classifiers trained under previous concepts by reactivating them when the concept reoccurs. The results also indicate that, as discussed in Section 4.4, a small neighborhood may be a good starting point when adapting the DCS approach to concept drift scenarios for most DCS methods.

5.6 Classification Engines Tests using Common Benchmarks

In this Section the DCS methods listed in Section 5.3 (and described in Section 2.8) are used as classification engines for the Dynse framework. During the tests no

pruning approach was implemented. As estimated in Section 5.5, the accuracy estimation window size will be set to 4 ($M = 4$) for real concept drift problems, and to 32 for virtual concept drift problems ($M = 32$). Since the authors of the DCS methods reported in Section 2.8 often defined a small neighborhood to be a good starting point (usually a value between 5 and 10), a neighborhood of size 5 ($k = 5$) was defined in the tests for all DCS-based methods implemented as classification engines. As discussed in Section 4.4, and demonstrated empirically in Section 5.5, this small neighborhood size can mitigate a suboptimal estimation of the classifiers competences under a intersected real concept drift.

The results of the experiments using all benchmarks listed in Section 5.3 can be seen in Table 15, where the average accuracy achieved in each dataset and the accuracy standard deviation between test batches is shown. The results achieved by the *Oracle* classification engine and by the Naive Combination methods are also presented in Table 15, as upper and lower bounds, respectively. As can observed, the DCS-LA LCA classification engine presented the worst results in the Dynse framework when considering the datasets in Table 15, indicating that using the *a posteriori* information of the classifiers in order to select the neighbors may lead to a poor estimation of the classifier competence under a concept drift scenario, especially when new classes may appear over time (see the Digit and Letters datasets results). This behavior may be explained by the *a posteriori* change over time, which may lead to a suboptimal neighborhood selection when the neighbors are chosen according to the class given to the test instance. This conclusion is reinforced by the results of the *A Posteriori* method, which also consider the *a posteriori* information of the classifier in order to compute the classifiers competences. Besides being able to adapt to the concept drifts, the *A Posteriori* method was only the 5th best performing method from the 7 classification engines tested.

Table 15 – Artificial and real world benchmarks average accuracies (%), using different classification engines in the Dynse framework. Best results are in bold. The average Rank ($R.$) is also shown.

CE	STG	SEA	SEARec	CkrE	CkrP	CkrS	Gauss[1]	Nebr	For	Dig	Let	R.
	$M = 4$									$M = 32$		
K-E ($l = 0$)	92.0(16.6)	**87.5**(1.4)	**87.1**(1.8)	85.3(3.3)	86.5(4.9)	86.2(5.1)	89.5(5.7)	74.5(1.0)	**78.3**(10.6)	**76.8**(20.5)	66.5(14.0)	2.5
Priori	93.6(12.0)	86.5(1.4)	86.3(1.6)	**86.1**(3.5)	**86.8**(4.8)	**86.7**(4.6)	**90.1**(5.4)	73.7(1.0)	77.5(10.5)	75.4(20.1)	64.6(14.3)	2.9
OLA	91.9(17.7)	87.4(1.3)	**87.1**(1.8)	85.3(3.3)	86.5(4.9)	86.2(5.1)	89.5(5.7)	74.5(1.0)	**78.3**(10.6)	74.9(20.6)	66.2(14.9)	3.0
K-UW	91.8(14.3)	86.3(1.9)	86.3(1.9)	85.8(3.5)	86.7(4.7)	86.3(5.0)	89.5(7.0)	**74.7**(1.0)	77.8(11.0)	74.5(21.3)	**70.0**(16.1)	3.2
Post	**93.9**(12.3)	86.9(1.5)	86.6(1.7)	83.9(3.4)	85.2(5.0)	84.2(4.7)	89.9(6.6)	71.9(1.1)	76.3(11.0)	72.6(19.8)	58.5(11.6)	4.4
K-U	86.1(17.9)	86.1(1.9)	86.3(2.0)	83.2(3.5)	85.1(5.1)	84.0(5.3)	89.4(7.1)	74.4(1.1)	77.5(11.1)	74.9(22.1)	67.1(15.7)	5.0
LCA	75.2(13.9)	83.5(3.1)	83.5(3.0)	64.0(6.1)	68.4(6.9)	70.4(9.8)	85.9(8.1)	70.0(1.1)	50.2(19.8)	11.2(2.4)	23.1(8.4)	7.5
NaiveC	66.3(17.6)	83.7(3.0)	83.8(3.2)	51.7(10.2)	49.2(8.8)	63.7(18.9)	78.4(13.3)	70.4(1.1)	68.8(11.9)	12.9(3.7)	41.9(16.0)	7.5
Oracle	99.7(1.2)	93.9(1.4)	94.1(1.4)	99.6(3.8)	99.8(2.8)	99.6(3.6)	-	97.2(4.3)	98.6(7.4)	86.2(24.4)	93.2(18.7)	-

[1] Only the class priors of the test batches are available in [6], thus it was not possible to compute the Oracle accuracy.

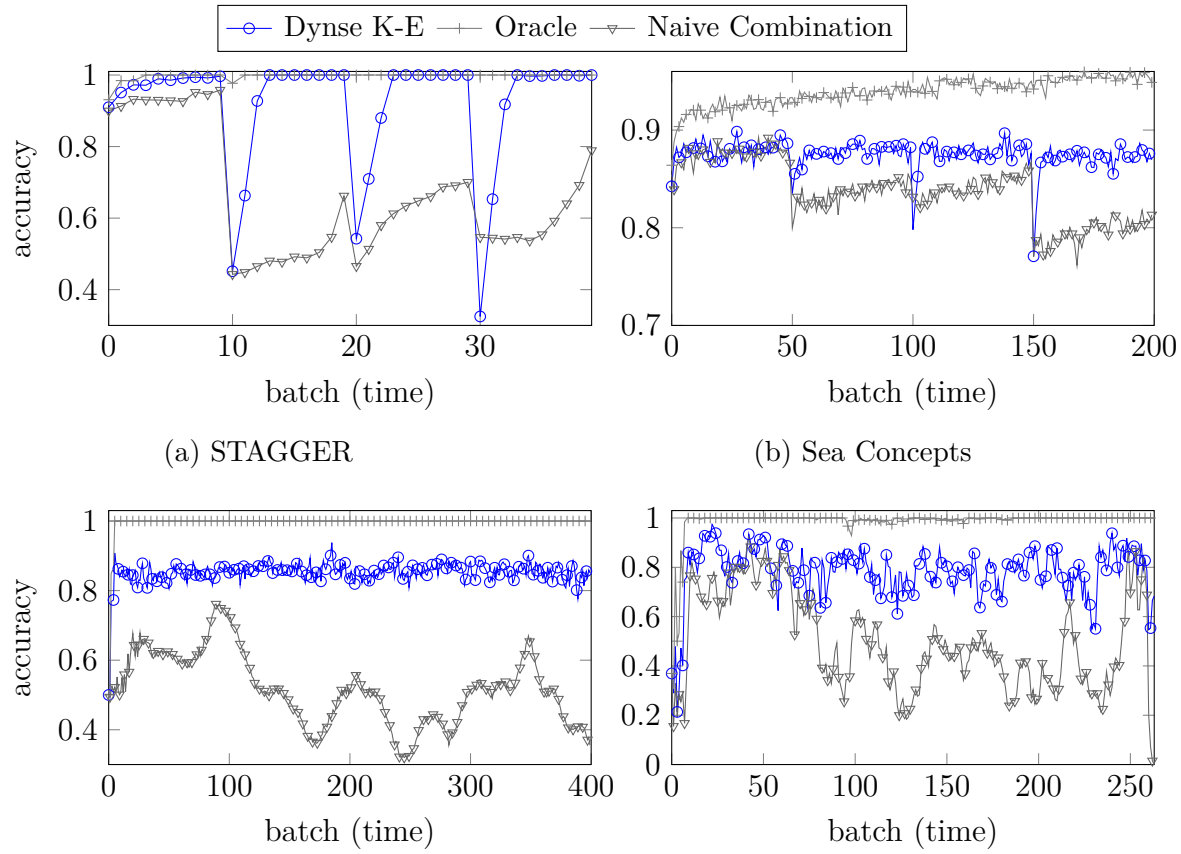
The results of the K-U method in Table 15 corroborates with the results presented

in Section 5.5, since this classification engine showed a good performance in the virtual concept drift problems, and achieved suboptimal results in some of the real concept drift scenarios (e.g. see the STAGGER benchmark result in Table 15). Observe that, in average, the weighted version of the Knora-Union method (K-UW) achieved better results than the unweighted (K-U) version. This result indicates that DCS methods that take into consideration the distance between the neighbors and the test instance when estimating the pool competence are more suitable for concept drift scenarios. In the K-UW, even thought a classifier that correctly classifies only a small portion of the neighborhood is still selected to be part of the ensemble, this classifier has its weigh adjusted according to the distance of these instances from the test instance.

This result is reinforced by the good performance of the A Priori method (which also uses weights based on the neighborhood distance to define the pool competence). The results presented by Tsymbal et al [37], which also use weights, corroborate these findings. Note that the A Priori method is "less permissive" than the K-UW approach, since only the best classifier is selected. This behavior seems to be beneficial in a gradual always changing real concept drift environments, due to the good results of the A Priori approach in all Checkerboard benchmark variants.

The results in Table 15 also demonstrate that, in general, the DCS approach adapted using the Dynse framework, is able to handle the concept drifts in all tested datasets with almost no parameter tuning (except for the accuracy estimation window size, the same parameter values were used for all experiments). This can be concluded by the good results achieved by the different DCS methods used as classification engines when compared with the Naive Combination method, since the former does not implement any approach to explicitly adapt to a concept drift (besides the addition of information in the pool over time). Nevertheless, the results achieved by the *Oracle* classification engine are far better than the results generated by any other classification engine, thus indicating that there is room for improvement when considering a DCS-based approach to deal with the presented concept drifts.

When comparing DCS methods that select a single classifier versus an ensemble of classifiers as classification engines, the results does not indicate a clear best approach. However, when considering the best average rank, the K-E approach, which selects an ensemble of classifiers for each test instance, is the best performing approach. Some accuracy over time plots with the K-E as the classification engine of the Dynse framework, the Oracle and Naive Combination methods can be seen in Figure 39. Note that when the change is abrupt, as in the STAGGER and SEA benchmarks (Figures 39a and 39b, respectively), the K-E can quickly recover from the concept drift. Considering all plots in Figure 39, it is clear that the Dynse framework is able to adapt to the concept drifts, although the Oracle demonstrates that some improvement can still be made.



(a) STAGGER (b) Sea Concepts

(c) Checkerboard Exponential (d) Forest

Figure 39 – Accuracy over time plots considering the K-E classification engine, the Oracle and the Naive Combination Methods.

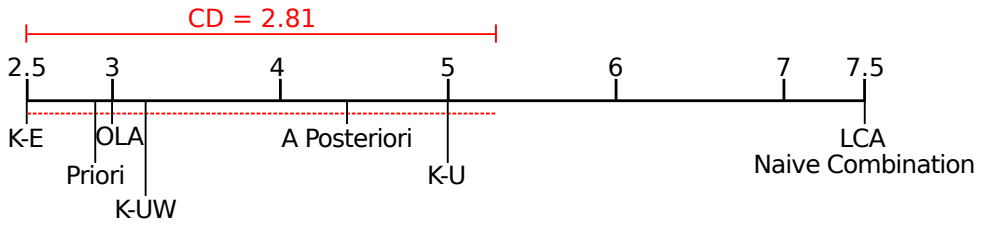


Figure 40 – Bonferroni-Dunn test with 95% confidence showing the methods that are not significantly different from the K-E Classification Engine (i.e., the connected methods).

In Figure 40 the classification engines that are not significantly better than the K-E classification engine according to a Bonferroni-Dunn test for $\alpha = 0.05$ are shown, where it is possible to verify that most classification engines have no significant difference to the K-E. However, in Figure 40 it is demonstrated that the Dynse framework using the K-E classification engine is significantly more competent that a method that just accumulates the new information without any adaptation (i.e. the Naive Combination method) and than the DCS-LA LCA method. The six best ranked classification engines deemed as equivalent by the Bonferroni-Dunn test are further analyzed using pairwise comparisons, considering the hypothesis of equality between each pair of algorithms, using

the Bergman-Hommel procedure [78, 77, 64], and the Wilcoxon Signed-Ranks test [79, 73], considering the hypothesis of equality between each pair of algorithms. The comparison results are presented in Table 16, where as in [64], hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a $\bullet$, $\bullet\bullet$, and $\bullet\bullet\bullet$, respectively.

Table 16 – Pairwise comparisons of the top 6 Classification Engines. (a) Comparison with the adjusted p-values using the Bergmann-Hommel procedure. (b) Comparison using the Wilcoxon Signed-Ranks test. The hypothesis are ordered according to the p-value. Hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a $\bullet$, $\bullet\bullet$, and $\bullet\bullet\bullet$, respectively.

(a)

Hypothesis	p_{Berg}
K-E vs K-U	$\bullet\bullet$ 0.0314
Priori vs K-U	$\bullet$ 0.0740
OLA vs K-U	$\bullet$ 0.1000
K-E vs Post	0.2265
K-UW vs K-U	0.2265
Priori vs Post	0.3604
OLA vs Post	0.3938
Post vs K-UW	0.5539
K-E vs K-UW	1.0000
Post vs K-U	1.0000
K-E vs OLA	1.0000
Priori vs K-UW	1.0000
K-E vs Priori	1.0000
OLA vs Priori	1.0000
OLA vs K-UW	1.0000

(b)

Hypothesis	$p_{Wilcoxon}$
K-E vs K-U	0.0049
K-E vs Post	0.0176
K-UW vs K-U	0.0176
OLA vs K-U	0.0215
OLA vs Post	0.0234
Priori vs Post	0.0303
Post vs K-UW	0.1035
K-E vs OLA	0.1250
Priori vs K-U	0.1367
K-E vs Priori	0.3203
Priori vs K-UW	0.5742
K-E vs K-UW	0.6426
OLA vs Priori	0.6504
OLA vs K-UW	0.8242
Post vs K-U	0.8984

As one can observe in Table 16, the K-E classification engine is significantly better than the K-U for $\alpha = 0.05$ according to the Bergmann-Hommel test (Table 16a). The A Priori and the OLA methods are also significantly better than the K-U according to the Bergmann-Hommel procedure, but for $\alpha = 0.10$. According to the Wilcoxon Signed-Ranks test (Table 16b) no pairwise test resulted in a significant difference (i.e. no hypothesis was rejected). Besides most classification engines being deemed as equivalent by the statistical tests, the K-E will be used as the default classification engine for the Dynse framework, since it was the best ranked method and showed a good performance in all tested benchmarks.

5.7 The Pruning Impact

In this work it is hypothesized that a pool that contains as many classifiers as possible, where these classifiers may be specialist in different concepts and feature space regions, may be beneficial when using a DCS-based approach to deal with concept drift scenarios. In Sections 5.5 and 5.6 it is demonstrated that the DCS-based approach can adapt to concept drifts using a pool of "infinite size". Nevertheless, an infinite size pool is unfeasible under a real world scenario, thus, in this Section, the classical Age (remove the oldest classifier), Accuracy (remove the less accurate classifier considering the current

accuracy estimation window), and also the NNPrune strategies are used as pruning engines in the Dynse framework. The pruning strategies were configured to keep at most 25 classifiers in the pool during the tests (note that this pool size is compatible with the values used in state-of-the-art methods [103, 110, 109]).

The tests results are compared with the "infinite pool" of classifiers (i.e. no pruning strategy). Note that the infinite pool can be considered the most Concept Diverse (See Section 4.5) technique in the tested scenarios, since classifiers trained under all past concepts are kept. In the tests the Dynse framework configured with the K-E considering $k = 5$ and $l = 0$ as classification engine is used. The accuracy estimation window size is set to 4 for the real concept drift scenarios ($M = 4$) and to 32 for the virtual concept drift tests ($M = 32$).

Table 17 contains the average accuracies achieved considering the discussed pruning strategies. Except for the Nebraska dataset[2], all benchmarks in Section 5.3 were used in the tests present in Table 17. As it can be observed in Table 17, except for the SEA and Gauss benchmarks, the infinite size approach generated the best results in all considered scenarios. This is an interesting result that helps to validate the hypothesis that a pool containing as many classifiers as possible, including classifiers trained under different concepts, is beneficial when using a DCS-based approach to deal with concept drifts, where this pool can be considered a Concept Diverse pool. The Concept Diversity importance is also reinforced by the good performance of the NNPrune pruning engine, since it performed better that the age and accuracy based pruning strategies in the majority of the tested scenarios.

Table 17 – Average accuracy achieved for different pruning strategies. The numbers in parenthesis indicate the accuracy standard deviation between testing batches (time steps).

Benchmark	Pruning Engine			
	Infinite	NNPrune	Age	Acc
STAGGER	**92.0% (16.6)**	91.8% (17.3)	91.4% (17.4)	91.7% (17.2)
SEA	87.5% (1.4)	87.4% (1.4)	87.6% (1.5)	**87.8% (1.6)**
SEARec	87.1% (1.8)	87.0% (1.8)	**87.2% (2.2)**	87.1% (2.2)
CkrE	**85.3% (3.3)**	83.6% (3.5)	83.4% (3.5)	83.5% (3.6)
CkrP	**86.5% (4.9)**	84.6% (4.9)	84.6% (4.8)	84.9% (5.0)
CkrS	**86.2% (5.1)**	84.8% (5.2)	84.3% (4.7)	84.3% (4.8)
Gauss	89.5% (5.7)	**90.1% (5.7)**	89.6% (5.8)	89.7% (5.7)
For	**78.3% (10.6)**	77.6% (10.8)	78.2% (10.8)	78.2% (10.8)
Dig	**76.8% (20.5)**	74.6% (20.4)	74.3% (20.7)	68.0% (17.3)
Let	**66.5% (14.0)**	62.2% (13.4)	60.3% (13.5)	62.0% (12.7)

The pruning strategies are analyzed using pairwise comparisons in Table 18, considering the hypothesis of equality between each pair of algorithms, using the Bergman-

[2] The Nebraska was not considered in the pruning tests since it appears to have only minor drifts.

Table 18 – Pairwise comparisons of the pruning strategies. (a) Comparison with the adjusted p-values using the Bergmann-Hommel procedure. (b) Comparison using the Wilcoxon Signed-Ranks test. The hypothesis are ordered according to the p-value. Hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a $\bullet$, $\bullet\bullet$, and $\bullet\bullet\bullet$, respectively.

(a)

Hypothesis	p_{Berg}
Infinite vs Age	$\bullet$ 0.0919
Infinite vs NNPrune	0.2498
Infinite vs Acc	0.2498
NNPrune vs Age	1.0
NNPrune vs Acc	1.0
Age vs Acc	1.0

(b)

Hypothesis	$p_{Wilcoxon}$
Infinite vs NNPrune	0.0117
Infinite vs Acc	0.0430
Infinite vs Age	0.0449
NNPrune vs Age	0.2266
Age vs Acc	0.3125
NNPrune vs Acc	0.6406

Hommel procedure [78, 77, 64], and the Wilcoxon Signed-Ranks test [79, 73], considering the hypothesis of equality between each pair of algorithms. Hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a $\bullet$, $\bullet\bullet$, and $\bullet\bullet\bullet$, respectively. Note in Table 18 that the only significant difference found regards to the test of the infinite size pool versus the age based pruning, considering the Bergmann-Hommel procedure (Table 18a). Thus, in order to better visualize the approaches behavior, consider the plot in Figure 41, where the average accuracy difference between each pruning strategy and the infinite size pool is shown, where smaller values refers to better performances (results closer to the infinite size pool). As one can observe in Figure 41, the NNPrune and Accuracy pruning strategies achieved similar accuracies, although the NNPrune results are slightly better. The Accuracy based pruning generated the worst results when considering the average performance loss in Figure 41.

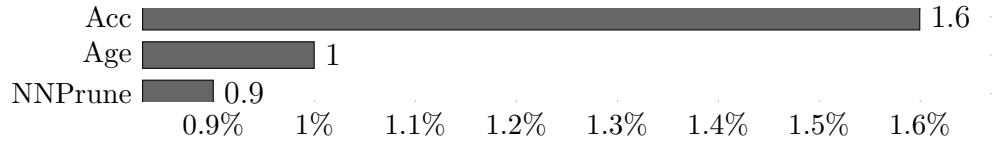
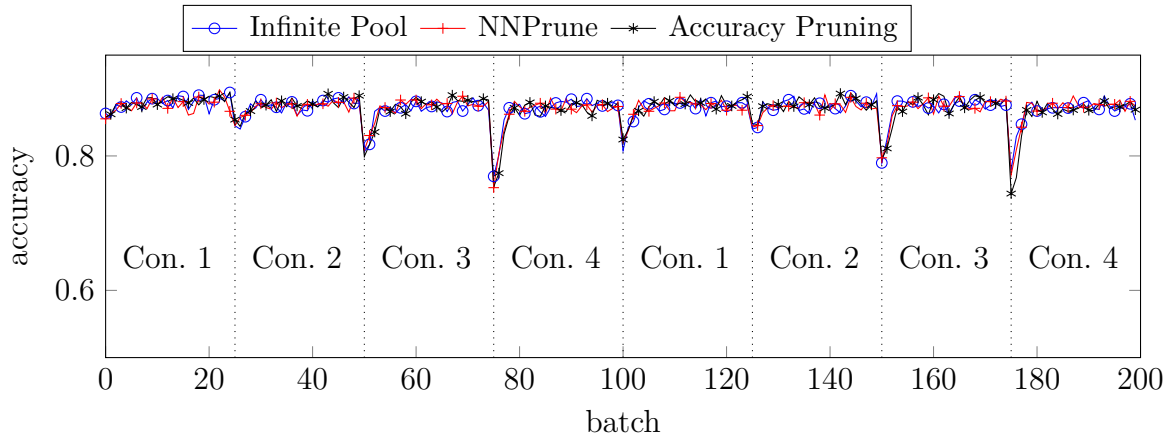


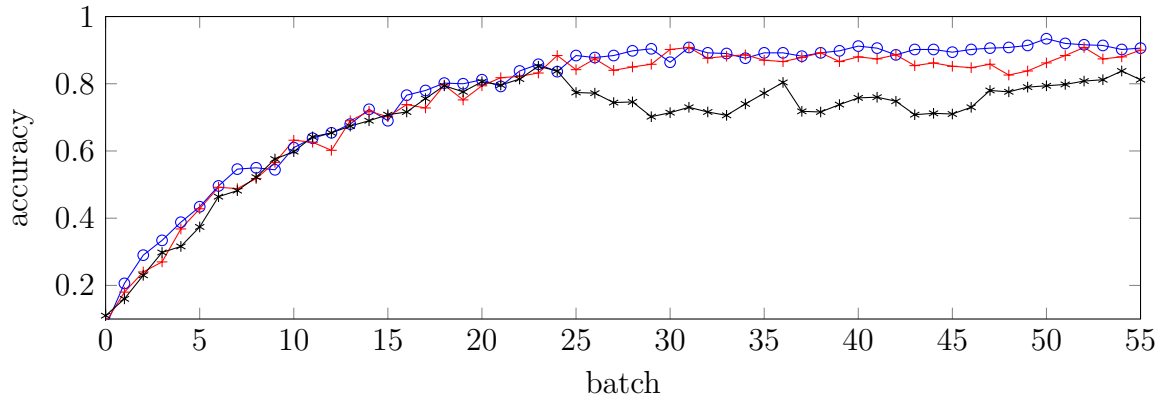
Figure 41 – Average accuracy difference between each pruning strategy versus the infinite size pool (smaller values are bettter).

The behavior of the Dynse framework when using the Infinite pool, the NNPrune and the Accuracy based pruning strategies is showed in Figure 42, where the accuracy over time plots for two of the tested benchmarks are shown. Figure 42 exemplifies two different scenarios, where in Figure 42a, which represents a relativelly simple problem in stable regions, all pruning strategies performed equally well. On the other hand, in Figure 42b, it is demonstrated that the accuracy based pruning engine may remove some relevant information, which can lead to a poor performance under a more complex problem. The age based pruning engine, that is not shown in Figure 42 in order to make the visualization cleaner, generates curves similar to the NNPrune approach for both scenarios.

As stated before, the results in Figure 41 indicate that the NNPrune and Accuracy



(a) SEARec

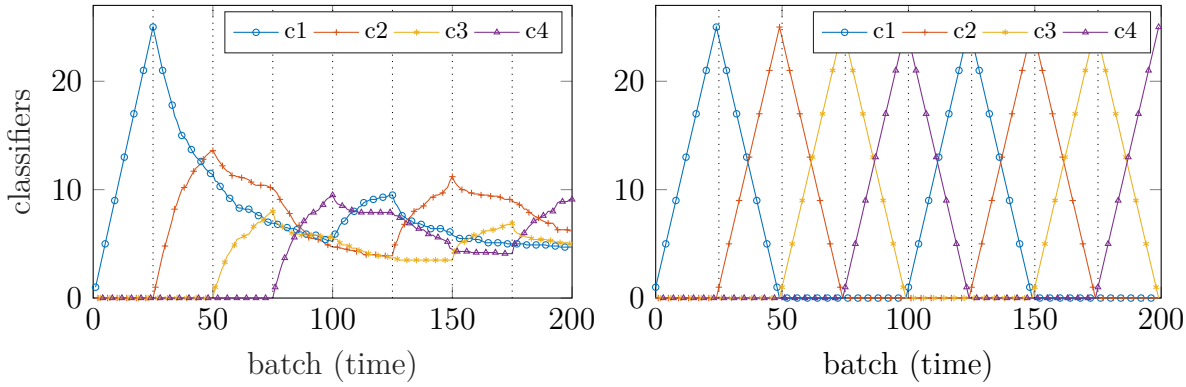


(b) Digit

Figure 42 – Accuracy over time plots considering different pruning strategies.

pruning strategies can generate similar results. Nevertheless, the pools generated by these two strategies are completely different. In order to demonstrate this, consider the plot in Figure 43, where the number of classifiers trained under each concept in the pool over time for the NNPrune and Age based pruning strategies are compared. The SeaRec was used as a benchmark. As can be observed, when using the Age based pruning, classifiers trained under a past concept will be completely pruned from the pool in at most 25 steps (i.e. the maximum pool size in the test), while the NNPrune keep classifiers trained under different concepts, regardless of the current concept (nevertheless, usually the majority of the classifiers present in the pool are trained in the current concept).

The Concept Diversity created by the NNPrune approach may impact differently on the accuracy of the Dynse framework, depending on the classification engine used. To demonstrate this, consider Figure 44, where the K-UW is used as the classification engine in the SEARec problem. As can be observed in Figure 44, the concept diverse pool generated by the NNPrune can mitigate the accuracy drops under the concept change regions when using the K-UW classification engine. In the same test discussed earlier, the K-E classification engine achieved similar results for all pruning strategies (See Figure



(a) NNPrune (b) Age prune

Figure 43 – Number of classifiers trained with each concept over time in a pool that supports at most 25 classifiers in the SEARec problem, using the NNPrune algorithm (a) and an Age based Pruning (b). Dashed lines were put in the concept change areas.

43a).

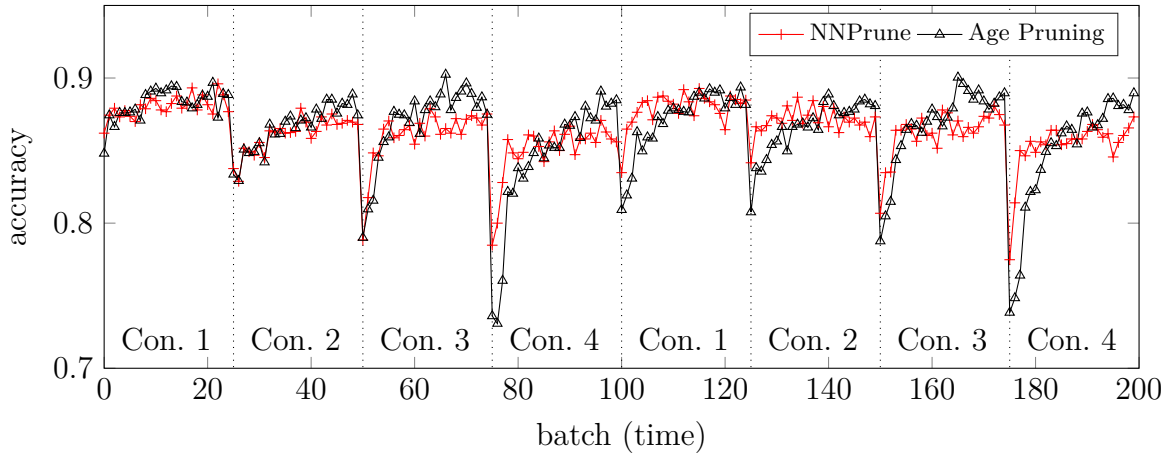


Figure 44 – Accuracy over time plot considering the K-UW classification engine and different pruning strategies in the SEARec benchmark.

In Figure 45 is showed the accuracy over time plot for different pruning approaches implemented in the AWE [109] method. Although the AWE method may not be able to detect regions where classifiers from past concepts are still relevant (the weight is given according to the entire feature space available in the latest supervised batch), the AWE is capable of benefiting from the reactivation of classifiers when a concept reoccurs. The results achieved by the Infinite Pool and by the NNPrune approaches in Figure 45 demonstrates that not only the DCS-based methods can benefit from a concept diverse pool.

Note that, when using the NNPrune strategy, the maximum pool size must be "big enough" in order to accommodate the classifiers specialist in different regions and in different concepts. Otherwise, the pruning engine will have no option besides prune

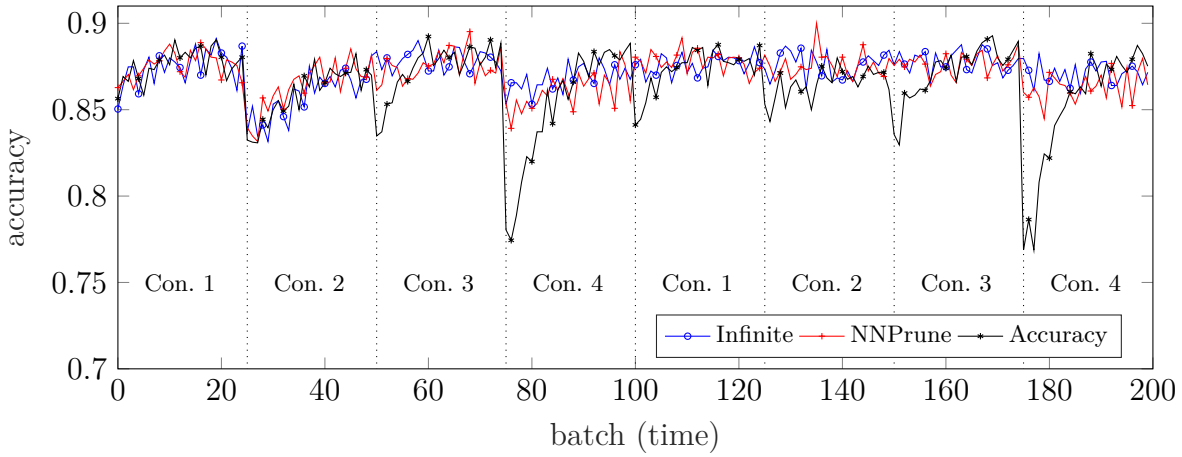


Figure 45 – The AWE method accuracy over time plot in the SeaRec problem, considering an infinite size pool, the NNPrune (NN) and an accuracy based pruning.

classifiers that represent relevant information that no other classifier does. A possible approach that may be used to mitigate this problem is to implement a flexible sized pool using the NNPrune approach. The idea is to insert a classifier in the pool only when it represents a region and a *a posteriori* information that no other does, regardless of the current number of classifiers. The implementation of this approach is proposed as a future work (See Section 6). Although, note that in the experiments in this work the size of 25 leaded to good results for the "fixed pool size" implementation of the NNPrune pruning strategy.

As stated in this work, the primary function of the pruning engine in the Dynse framework is to save computational resources, by keeping the pool from increasing its size indefinitely. To this end, consider Table 19, where the amount of memory necessary by the Dynse framework using each pruning engine is showed. The results in Table 19 demonstrates that, as expected, the Age and Accuracy-based pruning engines generated a similar memory consumption for all considered benchmarks. The NNPrune approach, on the other hand, consumed more memory than the infinite size pool in some tests. To better hunderstand the NNPrune and the other pruning strategies behavior, consider the memory over time plots in Figure 46 (the age based plot, which generate a curve similar to the Accuracy Pruning, is not plotted). A Dashed line represents the instant when the pool reaches its maximum (except for the infinite size pool), and the pruning strategies starts to take action.

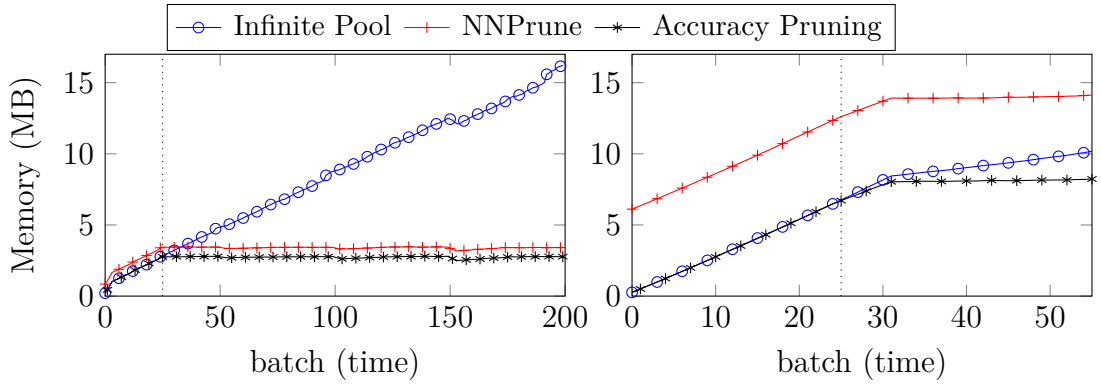
As can be observed, in both scenarios depicted in Figure 46 the pruning strategies stabilizes the amount of memory consumed when the pruning starts, while the infinite size pool continues to consume more memory linearly as time passes. Note that in the Digit benchmark (Figure 46b) all tested approaches continues to consume more memory until the batch 32, since new supervised data is still aggregated in the accuracy estimation window, which contains the 32 latest supervised batches received in this virtual concept

drift scenario.

Table 19 – Memory (MB) used by the Dynse framework considering each pruning engine, for each tested benchmark.

Benchmark	Pruning Engine			
	Infinite	NNPrune	Age	Acc
STAGGER	0.24	0.78	0.18	0.18
SEA	16.23	3.46	2.77	2.80
SEARec	16.17	3.46	2.77	2.77
CkrE	3.24	0.92	0.30	0.31
CkrP	3.23	0.92	0.29	0.31
CkrS	3.20	0.92	0.29	0.30
Gauss	1.77	0.98	0.34	0.35
For	20.67	7.50	4.89	4.96
Dig	10.18	14.13	8.21	8.23
Let	15.85	7.85	4.62	5.24

Besides the memory consumption stabilization, the NNprune strategy consumes more memory than the accuracy pruning method, since it is necessary to maintain the training sets of the classifiers in the pool in order to choose which classifier should be removed. In the SeaRec problem (Figure 46a) this can be considered a little price to pay, nevertheless in the Digit problem (Figure 46a), the NNPrune uses about 70% more memory than the Accuracy/Age based prunings and, due to the small number of steps in the problem, the infinite size pool does not have steps enough to surpass the amount of memory consumed by the NNPRune. Nevertheless, it is worth pointing out that under a real world scenario, a method that linearly increases the amount of memory consumed over time is unfeasible. Thus, the NNPrune approach can be considered for real world applications since a fixed amount of memory is necessary.



(a) SEARec (b) Digit

Figure 46 – Memory (MB) over time plots considering different pruning strategies.

As demonstrated in this Section, a DCS-based approach can benefit from a pool containing classifiers trained under past concepts, since the “infinite size” pool generated

better results than a pruned pool. The NNPrune strategy, which tries to keep a Concept Diverse pool, was the best performing pruning strategy, although the results are close to the Age based pruning approach. Besides the NNPrune strategy being able to keep the Dynse framework from indefinitely increasing the amount of memory necessary over time, one may argue that keeping the training sets of each classifier in the pool can be a suboptimal solution.

Thus, despite the results in this Section demonstrated that the amount of memory necessary by the NNPrune algorithm is relatively small, the Age pruning approach will be defined as the default pruning engine for the Dynse framework in this work, since it achieved results close to the NNPrune (not significantly different than the NNPrune according to the statistical tests), and does not require the store of training information. This may lead to a more fair comparison with the state-of-the-art methods, since most authors claim that their methods operate with "restricted" computational resources. The NNPrune will be considered a preliminary study in this work, and strategies to avoid storing the complete training sets for the classifiers in the pool will be studied in the future (See Chapter 6).

5.8 The Dynse Default Configuration

Based on the tests presented in Sections 5.4, 5.5, 5.7 and 5.9, and also considering the discussions presented in the course of this work, in Table 20 is presented a default configuration for the Dynse framework. Obviously, a parameter optimization may be performed, and any component of the Dynse framework may be tuned/changed in order to generate a more suitable configuration for a specific problem. Nevertheless, as it may be difficult to acquire enough relevant data to fine tune a system under a concept drift scenario, the proposed framework default configuration can be considered as a good starting point. The proposed default configuration is used in this work in order to compare the results with the state-of-the-art methods.

5.9 Tests Comparing to the State-Of-The-Art Results

In this Section the experiments performed on several real world and artificial well-known benchmarks are presented. The benchmarks are used as defined in Section 5.3 to guarantee the experiments reproducibility. In this Section some important state-of-the-art methods are also presented and its performances are compared with the proposed Dynse framework. The state-of-the-art methods were tested using their default configuration, available in the MOA framework. The default configuration of the Dynse framework used in the tests is the one discussed in Section 5.8, where the maximum pool size was set to 25 classifiers.

Table 20 – The proposed default configuraion of the Dynse framework.

Description	**Var.**	Proposed Configuration/values
The *accuracy estimation window size*.	M	$M = 4$ for real concept drift scenarios and $M = 32$ for virtual concept drift scenarios.
The number of neighbors of the test instance x selected from the *accuracy estimation window*	K	$k = 5$
The classification engine used by the Dynse framework to dynamically select the classifiers.	CE	The K-E method considering $l = 0$.
The pool maximum size	D	$D = 25$.
The pruning engine used to prune classifiers from the pool.	PE	The Age Pruning engine (remove the oldest).

As discussed in Section 5.2, all methods use Hoeffding Trees [82] as base learners and, except for the *Hoeffding Adaptive Tree* (HAT) state-of-the-art method, all tested methods use some sort of pool of classifiers (the DDM and EDDM triggers are also configured to use a pool of Hoeffding Trees, as discussed in Section 5.2). Table 21 summarizes the results of the experiments, where the ranks presented refers to the average ranks considering all datasets. As can be observed in Table 21, the default configuration of the Dynse framework generated the best results in the majority of the real concept drift and real world scenarios, although closely followed by the Leveraging Bagging method. In the virtual concept drift scenarios, the proposed framework is the best performing method for both tested benchmarks. It is worth remembering that the accuracy estimation window size was set to 32 in the virtual concept drift scenario, while the parameters in the state-of-the-art methods was kept the same. Albeit it can be considered a tuning, a wider accuracy estimation window size is a specification for the Dynse framework when facing a virtual concept drift problem (See Section 5.8), and the tested methods in the state-of-the-art does not specify any modification for this specific scenario.

In Table 21 it is also possible to verify that some state-of-the-art methods performs worse than the *Naive Combination* (NaiveC) baseline method, which does not take any action to adapt to a concept drift (besides the add of new classifiers in its pool over time). For instance, the DDM trigger performed worse than the NaiveC method in all Checkerboard benchmark variants. The results in Table 21 also demonstrate that, in spite of the fact that some state-of-the-art methods surpassed the Dynse framework in some benchmarks, these methods may be more sensitive to the parameters tuning or to the concept drift properties featured in the datasets. This conclusion becomes clear when the average rank is considered in Table 21, since the default Dynse framework configuration is the best ranked method. In Figure 47 is showed a Bonferroni-Dunn test for $\alpha = 0.05$, where it is demonstrated that the default configuration of the Dynse framework performs

significantly better than 5 of the state-of-the-art methods (and also better than the NaiveC method).

Table 21 – Artificial and real world benchmarks average accuracies (%), the default configuration of the Dynse framework and some state-of-the-art methods. Best results are in bold (the Oracle was not considered).

Method	STG	SEA	SEARec	CkrE	CkrP	CkrS	Gauss	Nebr	For	Dig	Let	R.
					$M = 4$					$M = 32$		
Oracle	99.4(1.8)	92.1(1.4)	92.6(1.3)	98.4(4.1)	99.7(2.9)	99.4(3.8)	-	94.8(3.7)	95.4(8.2)	82.7(21.1)	87.9(16.5)	
Dynse	91.4(17.4)	87.6(1.5)	**87.2**(2.2)	**83.4**(3.5)	**84.6**(4.8)	**84.3**(4.7)	89.6(5.8)	74.1(1.1)	**78.2**(10.8)	**74.3**(20.7)	**60.3**(13.5)	**1.7**
LevBag	85.4(20.3)	**88.1**(1.8)	87.1(2.4)	77.0(7.9)	82.4(11.8)	79.4(10.6)	**90.8**(5.5)	**77.0**(1.3)	75.1(13.6)	61.4(17.3)	56.8(12.2)	2.5
AUE	93.7(10.3)	86.4(1.1)	86.3(1.5)	58.6(7.8)	54.3(7.2)	59.2(9.4)	90.5(5.2)	73.7(0.6)	73.7(11.9)	43.4(22.5)	58.4(12.1)	4.0
OzaAD	79.8(20.8)	86.4(2.2)	85.7(2.2)	59.0(7.8)	59.5(12.4)	50.7(8.6)	90.2(5.3)	73.4(1.5)	49.7(19.8)	62.9(19.3)	58.3(11.5)	5.0
AWE	**97.0**(3.0)	87.1(2.0)	86.7(1.8)	57.8(7.6)	54.2(7.9)	56.2(7.6)	90.4(5.5)	73.8(0.8)	66.4(15.8)	14.6(2.5)	23.2(10.1)	5.4
OzaAS	67.2(19.1)	85.2(2.5)	85.1(2.2)	58.4(10.4)	63.7(13.6)	57.1(10.9)	86.8(9.0)	73.5(1.6)	74.2(11.5)	65.0(19.0)	57.2(11.1)	5.5
HAT	66.3(20.1)	86.1(2.4)	84.9(2.6)	59.0(8.7)	56.4(9.4)	58.4(11.6)	61.3(23.6)	72.3(2.2)	67.2(12.5)	62.7(18.2)	56.7(11.3)	6.5
DDM	75.1(9.9)	86.3(3.0)	85.4(2.6)	49.3(5.5)	47.2(7.6)	47.9(5.9)	71.1(13.7)	70.1(1.4)	51.9(24.9)	22.0(7.6)	58.1(18.6)	7.7
NaiveC	66.3(17.6)	83.7(3.0)	83.8(3.2)	51.7(10.2)	49.2(8.8)	63.7(18.9)	78.4(13.3)	70.4(1.1)	68.8(11.9)	12.9(3.7)	41.9(16.0)	7.8
EDDM	69.0(8.5)	83.4(5.6)	82.9(4.1)	51.5(7.6)	49.0(7.3)	50.4(5.8)	86.9(13.2)	65.2(2.0)	46.2(21.9)	12.5(2.4)	56.5(19.7)	8.9

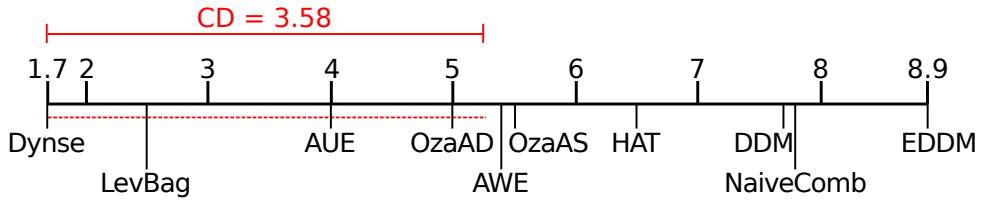


Figure 47 – Bonferroni-Dunn test with 95% confidence showing the methods that are not significantly different from the default configuration of the Dynse framework (i.e., the connected methods).

The Dynse, Leveraging Bagging, AUE and OzaAD methods, deemed as equivalent by the Bonferroni-Dunn test, are further analyzed using pairwise comparisons, considering the hypothesis of equality between each pair of algorithms. The pairwise tests results can be seen in Table 22, where the Bergman-Hommel procedure (Table 22a), and the Wilcoxon Signed-Ranks test (Table 22b) were performed, considering the hypothesis of equality between each pair of algorithms. Hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a $\bullet$, $\bullet\bullet$, and $\bullet\bullet\bullet$, respectively.

As one can observe, the Dynse framework was considered significantly better than the OzaAD and AUE methods for $\alpha = 0.01$ and $\alpha = 0.05$, according to the Bergman-Hommel test (Table 22a). Thus, according to the Bergman-Hommel test only the Leveraging Bagging is not significantly different from the Dynse framework when considering the significance levels tested. Note that the Wilcoxon Signed-Ranks leaded to a more conservative result, where only the OzaAD method was deemed significantly worse than the Dynse framework for $\alpha = 0.05$.

Table 22 – Pairwise comparisons of the top 4 methods. (a) Comparison with the adjusted p-values using the Bergmann-Hommel procedure. (b) Comparison using the Wilcoxon Signed-Ranks test. The hypothesis are ordered according to the p-value. Hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a •, ••, and •••, respectively.

(a)

Hypothesis	p_{Berg}
Dynse vs OzaAD	•••0.0043
Dynse vs AUE	••0.0314
LevBag vs OzaAD	••0.0499
LevBag vs AUE	0.1167
Dynse vs LevBag	0.6435
AUE vs OzaAD	0.6435

(b)

Hypothesis	$p_{Wilcoxon}$
Dynse vs OzaAD	••0.0020
LevBag vs OzaAD	0.0186
Dynse vs AUE	0.0264
Dynse vs LevBag	0.0420
LevBag vs AUE	0.0537
AUE vs OzaAD	0.4180

A compelling characteristic of the Dynse framework is its stability or, in other words, if the method is not the best, it is close to the best. This is exemplified in Figure 48 where it is showed the average accuracy difference between each method presented in Table 21 and the default configuration of the Dynse framework. The plot in Figure 48 indicates, for instance, that on average the default configuration of the Leveraging Bagging method (the best performing state-of-the-art method) generates an accuracy about 3.1% worse than the Dynse on average. The biggest average accuracy difference refers to the EDDM method (22.0%), which not surprisingly was the worst ranked method in the tests present in Table 21. Note that in Figure 48 no method performed better than the Dynse framework considering the average accuracy on all datasets (i.e. no negative average accuracy difference).

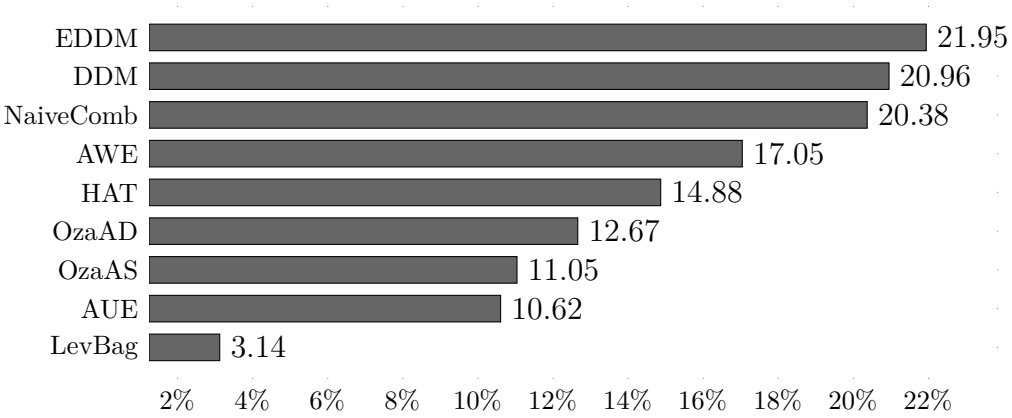
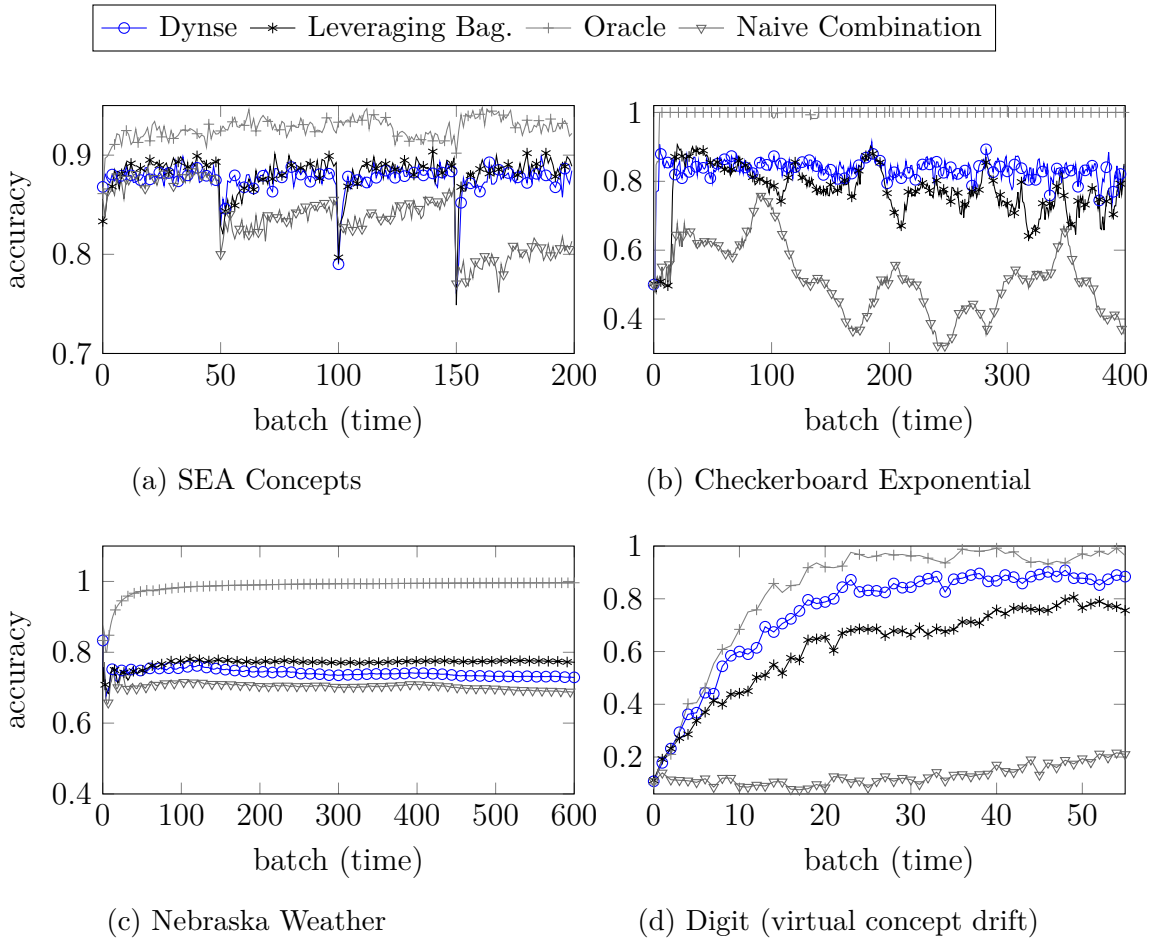


Figure 48 – Average accuracy difference between each tested method versus the Dynse framework, considered as a control.

Since in most real world scenarios it may be difficult to know *a priori* the exact properties of the concept drift, or to collect a relevant amount of data in order to fine tune the methods, the results in Table 21 and Figure 48 indicate that a DCS-based approach should be considered in these scenarios due to its good performance without any fine tuning. The method does need to know, however, if the concept drift is real or virtual in

order to define a big or small accuracy estimation window, as discussed in this work. To better visualize the behavior of the methods under the benchmarks tests presented in this Section, the accuracy over time plots for some benchmarks are shown in Figure 49. Only the default configuration of Dynse framework, the Leveraging Bagging, the Oracle and the Naive Combination methods are present in the plots of the Figure 49.



(a) SEA Concepts

(b) Checkerboard Exponential

(c) Nebraska Weather

(d) Digit (virtual concept drift)

Figure 49 – The accuracy over time plot considering the default Dynse framework configuration, the Leveraging Bagging, the Oracle and Naive Combination methods under some benchmarks.

As one can observe in Figure 49a, the Dynse framework was able to recover as fast as the Leveraging bagging method under the concept drift regions of the SEA Concepts problem (the Dynse framework was even faster in the first concept drift that happened). Under the stable regions, the results were similar for both the Dynse and Leveraging Bagging methods, although the Leveraging Bagging performed slightly better. In the Checkerboard problem (Figure 49b), which represents a "always drifting" scenario, the Dynse framework was able to keep its accuracy during the entire test, while the Leveraging Bagging presented several accuracy drops. The Nebraska Weather problem (Figure 49c), on the other hand, represents a scenario where the Leveraging Bagging was the best performing method. Finally, in Figure 49d, the Digit virtual concept drift is showed,

where the Dynse method was the best performing method, although both the Dynse and Leveraging Bagging were able to increase their accuracies over time.

Note that, as expected, the Naive Combination method showed difficulties to recover from the concept drift scenarios presented in the plots of Figure 49. Nevertheless, in the Nebraska Weather problem (Figure 49c), the Naive Combination method did not suffered any sudden accuracy drop after the first steps, and its accuracy dropped relatively slowly over time. This behavior may indicate that this benchmark suffers from minor changes over time, and even a method that is not able to deal with concept drifts can keep a reasonable accuracy over all time steps. It is also possible to observe in all plots in Figure 49, and in the results in Table 21, that the Oracle upper bound is far more accurate than any other method, indicating that plenty of improvements can be made in the DCS approach in order to try to reach the Oracle accuracy.

One may argue that the classification engine K-E was chosen specifically to get the best results in the tests presented in Table 21 and in Figure 49, since the same datasets are present in the tests of Section 5.6. Nevertheless, as showed in Section 5.6, no significant difference between most classification engines was fount and, by switching the K-E by the K-U classification engine (which is significantly worse than the K-E), the Dynse framework would tie with the Leveraging Bagging method in Table 21, with an average rank of 1.5. In this scenario, both the Dynse and Leveraging Bagging methods would share the best rank.

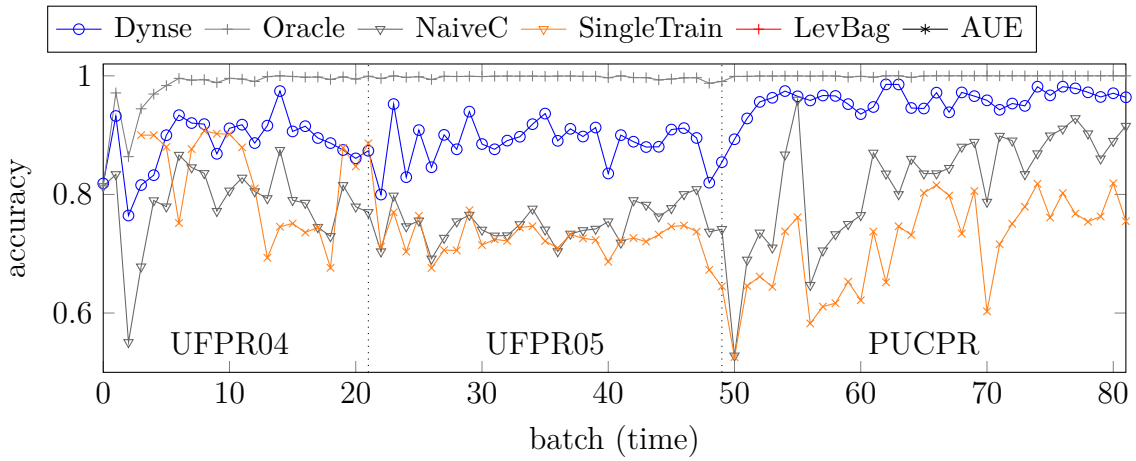
Table 23 – Amount of memory (MB) used by the Dynse framework and the Leveraging Bagging methods.

	Benchmark										
	STAGGER	SEA	SEARec	CkrE	CkrP	CkrS	Gauss	Nebr	For	Dig	Let
Dynse	0.18	2.80	2.77	**0.31**	**0.31**	**0.30**	0.35	**0.53**	4.96	8.23	5.24
LevBag	**0.15**	**1.13**	**0.90**	0.42	0.40	0.42	**0.22**	1.40	**1.44**	**1.02**	**0.42**

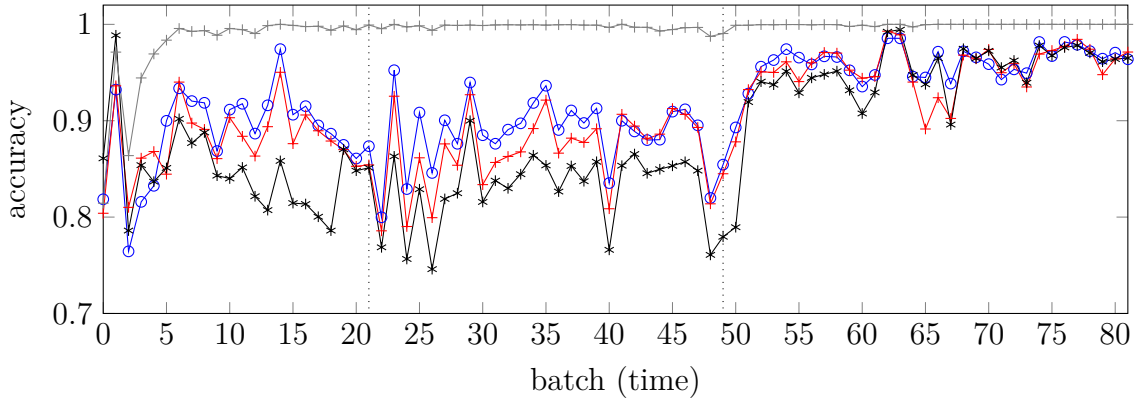
As a reference, the amount of memory spent by the default Dynse configuration and by the Leveraging Bagging method is showed in Table 23, where it can be observed that, often, the Leveraing Bagging uses less memory than the Dynse framework. Nevertheless, as discussed in Section 5.7, in the worst case scenario the Dynse framework consumed only 8.23MB of memory, which is a feasible amount of memory for most systems. Finally, it is worth reminding that no parameter optimization was performed for any test in the experiments of this Section. This means that it is possible to increase the performance and reduce the amount of computational resources spent (e.g. reduce the number of classifiers in the pool to reduce the amount of memory consumed) of all of the presented approaches. Nevertheless, the focus of this Section is to demonstrate that the Dynse framework may achieve results comparable to the state-of-the-art methods by means of its default configurations.

5.10 Tests in the PKLot Dataset

In this Section the PKLot datset is used as a real world benchmark to evaluate the proposed method and the methods in the state-of-the-art. The experimental protocol used is the same defined in Section 5.1.4. The default configuration of the Dynse framework for real concept drift environments defined in Section 5.8 is used (considering the maximum size pool $D = 25$ and $M = 4$). In the first experiment the Dynse framework accuracy is compared with the *Naive Combination* method that keeps all trained classifiers in the pool, and with a *Single Train* based classifier trained with 2,000 samples from each class (i.e. 4,000 samples in total) collected int the first 3 days of the UFPR04 dataset. The accuracy over time plot of this test can be seen in Figure 50a.



(a) The Dynse framework versus methods not designed to deal with concept drifts and the Oracle.



(b) The Dynse framework and the two best performing methods in the state-of-the-art.

Figure 50 – Accuracy over time plots int the PKlot Dataset.

As can be observed in Figure 50a, the *Single Train* classifier quickly decreases its performance in the dataset, indicating that a classical classifier, trained with a considerable number of samples and never updated, may not be suitable considering the proposed experimental protocol for the PKLot dataset (nevertheless, a static classifier trained with

a large number of samples randomly taken from possibly all days can generate a highly accurate classifier under all parking lots, as demonstrated in [8]). The *Naive Combination* method also does not perform well in this scenario, demonstrating that a method that just adds new information in its knowledge base may be a suboptimal solution for this problem. Note a severe accuracy drop for the *Naive Combination* and *Single Train* methods in the change between UFPR05 and PUCPR parking lots. The presence of concept drifts when changing the parking lot area or camera capture angle is reinforced in Almeida et al.[8], where the authors verified that when the training and testing sets were collected in the same parking lot and capture angle, the classifiers performed better than when the training and testing sets were collected under different parking areas or when the camera capture angle changed between them.

In Figure 50b the default configuration of the Dynse framework is compared with the two best performing methods in the state-of-the-art according to Section 5.9. As one can observe, the Dynse framework achieved better or similar results when compared to the state-of-the-art methods in most of the time steps. The average accuracy[3] considering all test batches for the default version of the Dynse framework and for the state-of-the-art methods is available in Table 24, where it is possible to verify that the default configuration of the Dynse framework method was the best performing approach when considering the average accuracy. The results in Figure 50b and in Table 24 reinforces the assumption that a DCS-based method can be a good general solution for the concept drift problem, since the Dynse framework performed better than the state-of-the-art methods in this scenario.

Table 24 – Average accuracies in the PKlot benchmark considering different methods.

Method	**Accuracy**
Oracle	99.72%
Default Dynse	**93.5**%
Leveraging Bagging	92.5%
AUE	91.0%
AWE	90.3%
Oza Bagging Adwin Tree	90.0%
Oza Bagging ASHT Tree	89.3%
Hoeffding Adaptive Tree	87.1%
Naive Combination	80.8%
DDM	80.6%
EDDM	80.0%
Single Train	73.9%

The Leveraging Bagging was the best performing state-of-ther-art method in this test considering the average accuracy. The Leveraging Bagging method is able to quickly create a diverse ensemble of classifiers using a modified version of the Bagging algorithm

[3] Since each batch may contain a different number of test samples, the accuracy standard deviation between batches was not included, as it could mislead the reader.

when a concept drift is signaled by its trigger. This behavior may explain the good performance of the Leveraging Bagging in the PKLot dataset, and also in the results presented in Section 5.9. As a future work, the same pool generation strategy may be implemented in the Dynse framework in order to improve its performance (See Chapter 6).

6 Conclusions

In this work it is demonstrated that through some modifications, the DCS approach can be a powerful tool to deal with the concept drift phenomenon, especially under scenarios where some areas do not change between concepts (intersected concept drifts). Although under a static environment a DCS approach may be only region dependent, in this work it is showed that under a concept drift scenario, this dependency alone is not sufficient, thus a time dependency must also be considered. This time dependency was modeled as a time window that keeps the latest supervised data received, called the *accuracy estimation window*, which is used to track the current environment. When a test instance x needs to be classified, the neighborhood of x in the accuracy estimation window is computed and used to estimate the classifiers competence. The pool of classifiers must also be time-dependent, where for every new supervised information received, a new classifier is trained and added to the pool.

By taking into consideration the necessary modifications in the DCS approach to deal with concept drift scenarios, the Dynse framework is proposed in this work as a modular tool, capable to adapt the DCS approach to concept drift scenarios. The framework deals with the time dependency through the accuracy estimation window and by creating new classifiers when new supervised information is received. Any neighborhood-based DCS method can be used to select the most promising classifier/ensemble from the accuracy estimation window, where the DCS method is defined in the *Classification Engine* module of the Dynse framework. Classifiers are pruned according to a component called *Pruning Engine*, where any pruning strategy may be implemented. Another components of the Dynse framework that may be tuned are the accuracy estimation window size and the number of neighbors used to define the local region. The Dynse framework is classifier independent, and its implementation was made publicly available in this work[1].

Through the presented discussions and tests it was demonstrated that the size of the accuracy estimation window must be defined according to the nature of the concept drift, where a small window should be used in a real concept drift scenario, and a window containing as much supervised data as possible should be defined in a concept drift that affects only $P(\mathbf{x})$. A discussion about the neighborhood size is also presented in this work, where it is argued that bigger sets of neighbors may impact the classifier competence estimation under a intersected real concept drift negatively. Tests using different neighborhood sizes confirm that smaller neighborhood sizes may lead to better results, except for approaches like the K-E method, that make it more difficult for a classifier to be part of the ensemble as the neighborhood size increase.

[1] The framework is fully available at https://web.inf.ufpr.br/vri/software/dynse/

It is also argued that, differently from most approaches used to deal with concept drifts, a DCS-based method may benefit from a pool containing classifiers trained under different concepts, since classifiers trained under previous concepts may still be relevant in some regions of the feature space. Ideally, the pool of classifiers should increase its size indefinitely (i.e. an infinite size pool) in order to contain classifiers trained under different concepts (time diversity) and specialists in different regions of the feature space (region diversity). We defined such a pool, that is time and local diverse, as a *Concept Diverse* pool. The experiment results demonstrate that a DCS-based approach does benefit from this hypothetical infinite size pool. Nevertheless, since a infinite size pool is unfeasible in the real world, a pool that keeps as many classifiers as possible, and that is kept as Concept Diverse as possible should be considered. A method based on the 1NN algorithm, called NNPrune, is presented as an alternative pruning strategy, which is able to keep a Concept Diverse pool with a fixed size, however this method was not considered as the default classification engine for the Dynse framework due to the necessity of keeping the training sets of all classifiers in the pool.

Tests under intersected concept drift scenarios demonstrated that a DCS-based approach is able to use classifiers trained under previous concepts in the regions of the feature space that did not changed before the concept drift. Tests with recurrent concepts also demonstrate that the classifiers from a concept that reoccurs may be reactivated, thus leading to an increase in the accuracy of the method.

A default configuration of the Dynse framework is also proposed in this work, where the K-E DCS method is used as the classification engine. This default configuration may be used as a starting point configuration when using the Dynse framework to deal with concept drift problems. Through an extensive range of experiments, it is demonstrated that a DCS-based approach (the Dynse framework) is able to achieve results that are better or similar to the state-of-the-art results in a range of problems without much tuning of its parameters (it is only necessary to know if the concept drift is real or virtual), which is advantageous in many concept drift scenarios since it may be impossible to know *a priori* all the concept drift properties present in the environment, or to collect relevant data in order to fine tune the method before deploying it.

The main drawback of the proposed framework is that it does need to know if the concept drift is real or virtual in order to define accuracy estimation window size. The tested methods in the state-of-the-art does not explicitly suggests any adjustment for virtual concept drift scenarios, nevertheless the test results demonstrated that most of these methods were not able to deal with this type of concept drift when considering its default configurations.

The results presented in this work indicate that taking into consideration the distance of each neighbor when selecting the custom ensemble for the test instance may

lead to better results in some scenarios, as observed in the results achieved by the A Priori and K-UW methods. As expected, most DCS methods used as classification engines in the Dynse framework were capable to deal with the concept drifts present in the benchmark problems used in this work. However, the DCS-LA LCA method achieved a poor performance when compared to the other DCS methods, indicating that taking into consideration the *a posteriori* information of the classifiers in order to select the neighbors may be a suboptimal solution for dealing with the concept drift problem through a DCS-based approach. Other contributions of this work include the definition of a slack variable for the K-E DCS method, which may be used under noisy/concept drifting environments, and a definition of an experimental protocol to use the PKLot dataset as a real world benchmark for methods designed to deal with concept drift problems.

In short, the results indicate that the hypothesis of this Thesis that any neighborhood DCS-based method can be adapted to deal with concept drift problems is valid, although the test results indicate that methods that use the *a posteriori* information in order to compute the neighborhood should be avoided. The DCS approach is able to reuse classifiers from old concepts in regions not affected by concept drifts, making it an appealing approach for scenarios containing intersected concept drifts, or even recurrences. This also indicate that a pool containing as many classifiers as possible can be beneficial when using a DCS approach. As hypothesized in this work, the results show that the time dependency for the DCS-based methods must be modeled according to the concept drift nature (real or virtual).

The discussions and tests presented in this work lead to several possible improvements that are proposed as future works. Since the accuracy estimation window should be set to be as big as possible under a virtual concept drift, one challenge for a future work is to keep only the most representative instances in this window to spare computational resources. Methods to avoid storing the entire training sets of each classifier when considering the NNPrune pruning approach should be considered, and the NNPrune algorithm could be used to define the pool size dynamically, where a new classifier could be added to the pool only if it represents a new information (i.e. a region in the feature space combined with an *a posteriori* information that is not represented by any other classifier), regardless to the current pool size. By adding only classifiers that represents new information to the pool, it would not be necessary to configure the variable D that defines the pool maximum size.

Another future work is to explore alternatives to maintaining the accuracy estimation window up to date with the current concept under a concept drift scenario, such as replacing the sliding window approach by a trigger-based one, which could even remove the need to know the type of concept drift (real or virtual) a priori, in order to define this window size. This trigger may be configured to signal a concept drift when any change is

noticed, without concerning to false alarms, since the trigger could be used just to adjust the accuracy estimation window to be as small as possible when the concept drift signal is fired (e.g. put only the latest supervised batch in the accuracy estimation window). In other words, the pool of classifiers, which should be as concept diverse as possible, must not be updated according to this trigger.

Since in this work concept drifts that affects $P(y)$ are not studied, classification engines capable to deal with this type of virtual concept drift can be explored in future works. Finally, alternatives to generate new classifiers for the pool could be explored in future works, such as generating multiple classifiers every time a new supervised batch arrives. By generating multiple classifiers, each of them trained with a different number of supervised batches, the DCS strategy may be able to select the classifiers trained with smaller sets under the presence of a concept drift (e.g. the classifiers trained with smaller sets are less likelly to have conflicting information), and the classifiers trained with more instances when the concept is stable. Another option is to train the pool of classifiers using a boosting or bagging approach, as done in the Leveraging Baging state-of-the-art method.

Bibliography

1 HOENS, T.; POLIKAR, R.; CHAWLA, N. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, Springer-Verlag, v. 1, n. 1, p. 89–101, 2012. ISSN 2192-6352.

2 KRAWCZYK, B.; WOźNIAK, M. One-class classifiers with incremental learning and forgetting for data streams with concept drift. *Soft Computing*, Springer Berlin Heidelberg, p. 1–14, 2014. ISSN 1432-7643.

3 SUSNJAK, T.; BARCZAK, A. L. C.; HAWICK, K. A. Adaptive cascade of boosted ensembles for face detection in concept drift. *Neural Computing and Applications*, Springer-Verlag, v. 21, n. 4, p. 671–682, 2012. ISSN 0941-0643.

4 MORENO-TORRES, J. G.; RAEDER, T.; ALAIZ-RODRÍGUEZ, R.; CHAWLA, N. V.; HERRERA, F. A unifying view on dataset shift in classification. *Pattern Recognition*, v. 45, n. 1, p. 521 – 530, 2012. ISSN 0031-3203.

5 GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. *Machine Learning*, Springer US, v. 90, n. 3, p. 317–346, 2013. ISSN 0885-6125.

6 ELWELL, R.; POLIKAR, R. Incremental learning of concept drift in nonstationary environments. *Neural Networks, IEEE Transactions on*, v. 22, n. 10, p. 1517–1531, Oct 2011. ISSN 1045-9227.

7 FREE SOFTWARE FOUNDATION. *GNU General Public License*. 2007. Available at: <http://www.gnu.org/licenses/gpl.html>.

8 ALMEIDA, P. R. de; OLIVEIRA, L. S.; JR., A. S. B.; JR., E. J. S.; KOERICH, A. L. {PKLot} – a robust dataset for parking lot classification. *Expert Systems with Applications*, v. 42, n. 11, p. 4937 – 4949, 2015. ISSN 0957-4174.

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

10 WANG, S.; SCHLOBACH, S.; KLEIN, M. Concept drift and how to identify it. *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 9, n. 3, p. 247 – 265, 2011. ISSN 1570-8268. Semantic Web Dynamics Semantic Web Challenge, 2010.

11 ESCOVEDO, T.; CRUZ, A. V. A. D.; VELLASCO, M. M. B. R.; KOSHIYAMA, A. S. Learning under concept drift using a neuro-evolutionary ensemble. *International Journal of Computational Intelligence and Applications*, v. 12, n. 04, p. 1340002, 2013.

12 TSYMBAL, A.; PECHENIZKIY, M.; CUNNINGHAM, P.; PUURONEN, S. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In: *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on*. [S.l.: s.n.], 2006. p. 679–684. ISSN 1063-7125.

13 KUNCHEVA, L. *Combining Pattern Classifiers: Methods and Algorithms*. [S.l.]: Wiley, 2014. ISBN 9781118914540.

14 THEODORIDIS, S.; KOUTROUMBAS, K. *Pattern Recognition*. [S.l.]: Elsevier Science, 2008. ISBN 9780080949123.

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

16 KRAWCZYK, B.; MINKU, L. L.; GAMA, J.; STEFANOWSKI, J.; WOŹNIAK, M. Ensemble learning for data stream analysis: A survey. *Information Fusion*, v. 37, p. 132 – 156, 2017. ISSN 1566-2535.

17 SUN, J.; LI, H. Dynamic financial distress prediction using instance selection for the disposal of concept drift. *Expert Systems with Applications*, v. 38, n. 3, p. 2566 – 2576, 2011. ISSN 0957-4174.

18 KELLY, M. G.; HAND, D. J.; ADAMS, N. M. The impact of changing populations on classifier performance. In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 1999. (KDD '99), p. 367–371. ISBN 1-58113-143-7.

19 PAN, S.; ZHANG, Y.; LI, X. Dynamic classifier ensemble for positive unlabeled text stream classification. *Knowledge and Information Systems*, Springer-Verlag, v. 33, n. 2, p. 267–287, 2012. ISSN 0219-1377.

20 KOLTER, J. Z.; MALOOF, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, v. 8, p. 2755–2790, Dec 2007.

21 KAPP, M.; SABOURIN, R.; MAUPIN, P. Adaptive incremental learning with an ensemble of support vector machines. In: *Pattern Recognition (ICPR), 2010 20th International Conference on*. [S.l.: s.n.], 2010. p. 4048–4051. ISSN 1051-4651.

22 SHIMODAIRA, H. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, v. 90, n. 2, p. 227 – 244, 2000. ISSN 0378-3758.

23 LAZARESCU, M. M.; VENKATESH, S.; BUI, H. H. Using multiple windows to track concept drift. *Intell. Data Anal.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 8, n. 1, p. 29–59, jan. 2004. ISSN 1088-467X.

24 SALGANICOFF, M. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, Kluwer Academic Publishers, v. 11, n. 1-5, p. 133–155, 1997. ISSN 0269-2821.

25 GAO, J.; FAN, W.; HAN, J.; YU, P. S. A general framework for mining concept-drifting data streams with skewed distributions. In: *In Proc. SDM'07*. [S.l.: s.n.], 2007.

26 KUNCHEVA, L. Classifier ensembles for changing environments. In: ROLI, F.; KITTLER, J.; WINDEATT, T. (Ed.). *Multiple Classifier Systems*. [S.l.]: Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 3077). p. 1–15. ISBN 978-3-540-22144-9.

27 MARTÍNEZ-REGO, D.; PÉREZ-SÁNCHEZ, B.; FONTENLA-ROMERO, O.; ALONSO-BETANZOS, A. A robust incremental learning method for non-stationary environments. *Neurocomputing*, v. 74, n. 11, p. 1800 – 1808, 2011. ISSN 0925-2312. ICONIP 2009.

28 KURLEJ, B.; WOZNIAK, M. Active learning approach to concept drift problem. *Logic Journal of IGPL*, 2011.

29 SCHLIMMER, J.; GRANGER JR., R. Incremental learning from noisy data. *Machine Learning*, Kluwer Academic Publishers, v. 1, n. 3, p. 317–354, 1986. ISSN 0885-6125.

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

31 MINKU, L.; WHITE, A.; YAO, X. The impact of diversity on online ensemble learning in the presence of concept drift. *Knowledge and Data Engineering, IEEE Transactions on*, v. 22, n. 5, p. 730–742, May 2010. ISSN 1041-4347.

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

33 SIDHU, P.; BHATIA, M. Empirical support for concept drifting approaches: Results based on new performance metrics. *International Journal of Intelligent Systems and Applications*, 2015.

34 ROSS, G. J.; ADAMS, N. M.; TASOULIS, D. K.; HAND, D. J. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, v. 33, n. 2, p. 191 – 198, 2012. ISSN 0167-8655.

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

36 GROSSBERG, S. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, v. 1, n. 1, p. 17 – 61, 1988. ISSN 0893-6080.

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

38 GONZÁLEZ-CASTRO, V.; ALAIZ-RODRÍGUEZ, R.; ALEGRE, E. Class distribution estimation based on the hellinger distance. *Information Sciences*, v. 218, n. 0, p. 146 – 164, 2013. ISSN 0020-0255.

39 RADTKE, P. V.; GRANGER, E.; SABOURIN, R.; GORODNICHY, D. O. Skew-sensitive boolean combination for adaptive ensembles – an application to face recognition in video surveillance. *Information Fusion*, v. 20, n. 0, p. 31–48, 2014. ISSN 1566-2535.

40 MARKOU, M.; SINGH, S. Novelty detection: a review—part 1: statistical approaches. *Signal Processing*, v. 83, n. 12, p. 2481 – 2497, 2003. ISSN 0165-1684.

41 JIAN-GUANG, H.; XIAO-FENG, H.; JIE, S. Dynamic financial distress prediction modeling based on slip time window and multiple classifiers. In: *Management Science and Engineering (ICMSE), 2010 International Conference on*. [S.l.: s.n.], 2010. p. 148–155. ISSN 2155-1847.

42 BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KIRKBY, R.; GAVALDà, R. New ensemble methods for evolving data streams. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2009. (KDD '09), p. 139–148. ISBN 978-1-60558-495-9.

43 CHEN, H.; YUAN, S.; JIANG, K. Adaptive classifier selection based on two level hypothesis tests for incremental learning. In: YEUNG, D.-Y.; KWOK, J.; FRED, A.; ROLI, F.; RIDDER, D. de (Ed.). *Structural, Syntactic, and Statistical Pattern Recognition*. [S.l.]: Springer Berlin Heidelberg, 2006, (Lecture Notes in Computer Science, v. 4109). p. 687–695. ISBN 978-3-540-37236-3.

44 KAPP, M. N.; SABOURIN, R.; MAUPIN, P. A dynamic optimization approach for adaptive incremental learning. *International Journal of Intelligent Systems*, Wiley Subscription Services, Inc., A Wiley Company, v. 26, n. 11, p. 1101–1124, 2011. ISSN 1098-111X.

45 GONÇALVES JR, P. M.; BARROS, R. S. M. de. Rcd: A recurring concept drift framework. *Pattern Recognition Letters*, v. 34, n. 9, p. 1018 – 1025, 2013. ISSN 0167-8655.

46 STREET, W. N.; KIM, Y. A streaming ensemble algorithm (sea) for large-scale classification. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2001. (KDD '01), p. 377–382. ISBN 1-58113-391-X.

47 MALOOF, M.; MICHALSKI, R. Selecting examples for partial memory learning. *Machine Learning*, Kluwer Academic Publishers, v. 41, n. 1, p. 27–52, 2000. ISSN 0885-6125.

48 ALIPPI, C.; BORACCHI, G.; ROVERI, M. Just-in-time classifiers for recurrent concepts. *Neural Networks and Learning Systems, IEEE Transactions on*, v. 24, n. 4, p. 620–634, April 2013. ISSN 2162-237X.

49 LICHMAN, M. *UCI Machine Learning Repository*. 2013. Available at: <http://archive.ics.uci.edu/ml>.

50 BREIMAN, L.; FRIEDMAN, J.; STONE, C.; OLSHEN, R. *Classification and Regression Trees*. [S.l.]: Taylor & Francis, 1984. (The Wadsworth and Brooks-Cole statistics-probability series). ISBN 9780412048418.

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

52 SCHOLZ, M.; KLINKENBERG, R. An ensemble classifier for drifting concepts. In: *In Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*. [S.l.: s.n.], 2005. p. 53–64.

53 POLIKAR, R.; KRAUSE, S.; BURD, L. Ensemble of classifiers based incremental learning with dynamic voting weight update. In: *Neural Networks, 2003. Proceedings of the International Joint Conference on*. [S.l.: s.n.], 2003. v. 4, p. 2770–2775 vol.4. ISSN 1098-7576.

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

55 KHREICH, W.; GRANGER, E.; MIRI, A.; SABOURIN, R. Iterative boolean combination of classifiers in the {ROC} space: An application to anomaly detection with {HMMs}. *Pattern Recognition*, v. 43, n. 8, p. 2732–2752, 2010. ISSN 0031-3203.

56 HARRIES, M. *SPLICE-2 Comparative Evaluation: Electricity Pricing*. [S.l.], 1999.

57 ŽLIOBAITĖ, I. *How good is the Electricity benchmark for evaluating concept drift adaptation*. 2013.

58 BIFET, A. Classifier concept drift detection and the illusion of progress. In: _____. *Artificial Intelligence and Soft Computing: 16th International Conference, ICAISC 2017, Zakopane, Poland, June 11-15, 2017, Proceedings, Part II*. Cham: Springer International Publishing, 2017. p. 715–725. ISBN 978-3-319-59060-8.

59 GAMA, J. a.; ROCHA, R.; MEDAS, P. Accurate decision trees for mining high-speed data streams. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 523–528. ISBN 1-58113-737-0.

60 BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KRANEN, P.; KREMER, H.; JANSEN, T.; SEIDL, T. Moa: Massive online analysis, a framework for stream classification and clustering. In: *Journal of Machine Learning Research (JMLR) Workshop and Conference Proceedings, Volume 11: Workshop on Applications of Pattern Analysis*. [S.l.]: Journal of Machine Learning Research, 2010. p. 44–50.

61 WOODS, K.; KEGELMEYER W.P., J.; BOWYER, K. Combination of multiple classifiers using local accuracy estimates. *Pattern Analysis and Machine Intel., IEEE Trans. on*, v. 19, n. 4, p. 405–410, Apr 1997. ISSN 0162-8828.

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

63 BRITTO JR., A. S.; SABOURIN, R.; OLIVEIRA, L. E. Dynamic selection of classifiers - A comprehensive review. *Pattern Recognition*, v. 47, n. 11, p. 3665 – 3680, 2014. ISSN 0031-3203.

64 CRUZ, R. M.; SABOURIN, R.; CAVALCANTI, G. D. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, v. 41, n. Supplement C, p. 195 – 216, 2018. ISSN 1566-2535.

65 OLIVEIRA, D. V.; CAVALCANTI, G. D.; SABOURIN, R. Online pruning of base classifiers for dynamic ensemble selection. *Pattern Recognition*, v. 72, p. 44 – 58, 2017. ISSN 0031-3203.

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

67 GIACINTO, G.; ROLI, F. Dynamic classifier selection. In: *Proceedings of the First International Workshop on Multiple Classifier Systems*. London, UK, UK: Springer-Verlag, 2000. (MCS '00), p. 177–189. ISBN 3-540-67704-6.

68 WOODS, K.; BOWYER, K.; KEGELMEYER W.P., J. Combination of multiple classifiers using local accuracy estimates. In: *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*. [S.l.: s.n.], 1996. p. 391–396. ISSN 1063-6919.

69 A new dynamic ensemble selection method for numeral recognition. In: HAINDL, M.; KITTLER, J.; ROLI, F. (Ed.). *Multiple Classifier Systems*. [S.l.]: Springer Berlin Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4472). p. 431–439. ISBN 978-3-540-72481-0.

70 KO, A. H.-R.; SABOURIN, R.; BRITTO, A. de S. K-nearest oracle for dynamic ensemble selection. In: *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*. [S.l.: s.n.], 2007. v. 1, p. 422–426. ISSN 1520-5363.

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

72 FRIEDMAN, M. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 11, n. 1, p. 86–92, 1940. ISSN 00034851.

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

74 IMAN, R. L.; DAVENPORT, J. M. Approximations of the critical region of the Friedman statistic. *Communications in Statistics - Theory and Methods*, Taylor & Francis, v. 9, n. 6, p. 571–595, jan. 1980.

75 NEMENYI, P. *Distribution-free Multiple Comparisons*. Tese (Doutorado) — Princeton University, 1963.

76 DUNN, O. J. Multiple comparisons among means. *American Statistical Association*, p. 52–64, 1961.

77 GARCIA, S.; HERRERA, F. An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of Machine Learning Research*, v. 9, n. Dec, p. 2677–2694, 2008.

78 BERGMANN, B.; HOMMEL, G. Improvements of general multiple test procedures for redundant systems of hypotheses. In: _____. *Multiple Hypothesenprüfung / Multiple Hypotheses Testing: Symposium, 6. und 7. November 1987*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988. p. 100–115. ISBN 978-3-642-52307-6.

79 BENAVOLI, A.; CORANI, G.; MANGILI, F. Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, v. 17, n. 5, p. 1–10, 2016.

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

81 KUNCHEVA, L. I.; ŽLIOBAITĖ, I. On the window size for classification in changing environments. *Intelligent Data Analysis*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 13, n. 6, p. 861–872, dez. 2009. ISSN 1088-467X.

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

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

84 RAKITIANSKAIA, A.; ENGELBRECHT, A. Training feedforward neural networks with dynamic particle swarm optimisation. *Swarm Intelligence*, Springer US, v. 6, n. 3, p. 233–270, 2012. ISSN 1935-3812.

85 KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

86 BICEGO, M.; FIGUEIREDO, M. A. Soft clustering using weighted one-class support vector machines. *Pattern Recognition*, v. 42, n. 1, p. 27 – 32, 2009. ISSN 0031-3203.

87 KMIECIAK, M. R.; STEFANOWSKI, J. Semi-supervised approach to handle sudden concept drift in enron data. *Control and Cybernetics*, Vol. 40, no 3, p. 667–695, 2011.

88 RODRÍGUEZ, J. J.; KUNCHEVA, L. I. Combining online classification approaches for changing environments. In: LOBO, N. da V.; KASPARIS, T.; ROLI, F.; KWOK, J.; GEORGIOPOULOS, M.; ANAGNOSTOPOULOS, G.; LOOG, M. (Ed.). *Structural, Syntactic, and Statistical Pattern Recognition*. [S.l.]: Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5342). p. 520–529. ISBN 978-3-540-89688-3.

89 DITZLER, G.; ROVERI, M.; ALIPPI, C.; POLIKAR, R. Learning in nonstationary environments: A survey. *Computational Intelligence Magazine, IEEE*, v. 10, n. 4, p. 12–25, Nov 2015. ISSN 1556-603X.

90 VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.: s.n.], 2001. v. 1, p. I–511–I–518 vol.1. ISSN 1063-6919.

91 BAENA-GARCIA, M.; CAMPO-ÁVILA, J. del; FIDALGO, R.; BIFET, A.; GAVALDA, R.; MORALES-BUENO, R. Early drift detection method. In: *Fourth international workshop on knowledge discovery from data streams*. [S.l.: s.n.], 2006. v. 6, p. 77–86.

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

93 PINAGE, F. A.; SANTOS, E. M. d. A dissimilarity-based drift detection method. In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2015. p. 1069–1076. ISSN 1082-3409.

94 SAKTHITHASAN, S.; PEARS, R.; KOH, Y. S. One pass concept change detection for data streams. In: _____. *Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 461–472. ISBN 978-3-642-37456-2.

95 PEARS, R.; SAKTHITHASAN, S.; KOH, Y. S. Detecting concept change in dynamic data streams. *Machine Learning*, v. 97, n. 3, p. 259–293, 2014. ISSN 1573-0565.

96 VITTER, J. S. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, ACM, New York, NY, USA, v. 11, n. 1, p. 37–57, mar. 1985. ISSN 0098-3500.

97 KUNCHEVA, L. I. Change detection in streaming multivariate data using likelihood detectors. *IEEE Transactions on Knowledge and Data Engineering*, v. 25, n. 5, p. 1175–1180, May 2013. ISSN 1041-4347.

98 REYNOLDS, M.; STOUMBOS, Z. The sprt chart for monitoring a proportion. *IIE Transactions*, Kluwer Academic Publishers, v. 30, n. 6, p. 545–561, 1998. ISSN 0740-817X.

99 LITTLESTONE, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, Kluwer Academic Publishers-Plenum Publishers, v. 2, n. 4, p. 285–318, 1988. ISSN 0885-6125.

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

101 YEH, A. B.; MCGRATH, R. N.; SEMBOWER, M. A.; SHEN, Q. Ewma control charts for monitoring high-yield processes based on non-transformed observations. *International Journal of Production Research*, v. 46, n. 20, p. 5679–5699, 2008.

102 BIFET, A.; GAVALDÀ, R. Adaptive learning from evolving data streams. In: ADAMS, N.; ROBARDET, C.; SIEBES, A.; BOULICAUT, J.-F. (Ed.). *Advances in Intelligent Data Analysis VIII*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5772). p. 249–260. ISBN 978-3-642-03914-0.

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

104 OZA, N. C.; RUSSELL, S. Online bagging and boosting. In: *In Artificial Intelligence and Statistics 2001*. [S.l.]: Morgan Kaufmann, 2001. p. 105–112.

105 SALPERWYCK, C.; BOULLE, M.; LEMAIRE, V. Concept drift detection using supervised bivariate grids. In: *Neural Networks (IJCNN), 2015 International Joint Conference on*. [S.l.: s.n.], 2015. p. 1–9.

106 KITHULGODA, C. I.; PEARS, R. Staged online learning: A new approach to classification in high speed data streams. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 1–8.

107 CHEN, K.; KOH, Y. S.; RIDDLE, P. Proactive drift detection: Predicting concept drifts in data streams using probabilistic networks. In: *International Joint Conference on Neural Networks*. [S.l.: s.n.], 2016. p. 780–787.

108 LITTLESTONE, N.; WARMUTH, M. The weighted majority algorithm. In: *Foundations of Computer Science, 1989., 30th Annual Symposium on*. [S.l.: s.n.], 1989. p. 256–261.

109 WANG, H.; FAN, W.; YU, P. S.; HAN, J. Mining concept-drifting data streams using ensemble classifiers. In: *Proceedings of the Ninth ACM SIGKDD International Conference on knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 226–235. ISBN 1-58113-737-0.

110 BRZEZIŃSKI, D.; STEFANOWSKI, J. Accuracy updated ensemble for data streams with concept drift. In: CORCHADO, E.; KURZYŃSKI, M.; WOŹNIAK, M. (Ed.). *Hybrid Artificial Intelligent Systems*. [S.l.]: Springer Berlin Heidelberg, 2011, (Lecture Notes in Computer Science, v. 6679). p. 155–163. ISBN 978-3-642-21221-5.

111 POLIKAR, R.; UPDA, L.; UPDA, S.; HONAVAR, V. Learn++: an incremental learning algorithm for supervised neural networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, v. 31, n. 4, p. 497–508, Nov 2001. ISSN 1094-6977.

112 KARNICK, M.; AHISKALI, M.; MUHLBAIER, M. D.; POLIKAR, R. Learning concept drift in nonstationary environments using an ensemble of classifiers based approach. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. [S.l.: s.n.], 2008. p. 3455–3462. ISSN 2161-4393.

113 ELWELL, R.; POLIKAR, R. Incremental learning in nonstationary environments with controlled forgetting. In: *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*. [S.l.: s.n.], 2009. p. 771–778. ISSN 1098-7576.

114 CRUZ, A. da; VELLASCO, M.; PACHECO, M. Quantum-inspired evolutionary algorithm for numerical optimization. In: ABRAHAM, A.; GROSAN, C.; ISHIBUCHI, H. (Ed.). *Hybrid Evolutionary Algorithms*. [S.l.]: Springer Berlin Heidelberg, 2007, (Studies in Computational Intelligence, v. 75). p. 19–37. ISBN 978-3-540-73296-9.

115 PIETRUCZUK, L.; RUTKOWSKI, L.; JAWORSKI, M.; DUDA, P. A method for automatic adjustment of ensemble size in stream data mining. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 9–15.

116 DITZLER, G. A study of an incremental spectral meta-learner for nonstationary environments. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 38–44.

117 PARISI, F.; STRINO, F.; NADLER, B.; KLUGER, Y. Ranking and combining multiple predictors without labeled data. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 111, n. 4, p. 1253–1258, 2014.

118 PAN, S.; WU, K.; ZHANG, Y.; LI, X. Classifier ensemble for uncertain data stream classification. In: ZAKI, M.; YU, J.; RAVINDRAN, B.; PUDI, V. (Ed.). *Advances in Knowledge Discovery and Data Mining*. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6118). p. 488–495. ISBN 978-3-642-13656-6.

119 CHAN, P.; ZHANG, Q.-Q.; NG, W.; YEUNG, D. Dynamic base classifier pool for classifier selection in multiple classifier systems. In: *Machine Learning and Cybernetics (ICMLC), 2011 International Conference on*. [S.l.: s.n.], 2011. v. 3, p. 1093–1096. ISSN 2160-133X.

120 JENHANI, I.; AMOR, N. B.; ELOUEDI, Z. Decision trees as possibilistic classifiers. *International Journal of Approximate Reasoning*, v. 48, n. 3, p. 784 – 807, 2008. ISSN 0888-613X. Special Section on Choquet Integration in honor of Gustave Choquet (1915–2006) and Special Section on Nonmonotonic and Uncertain Reasoning.

121 FUNG, G. P. C.; YU, J.; LU, H.; YU, P. Text classification without negative examples revisit. *Knowledge and Data Engineering, IEEE Transactions on*, v. 18, n. 1, p. 6–20, Jan 2006. ISSN 1041-4347.

122 FISCHER, L.; HAMMER, B.; WERSING, H. Online metric learning for an adaptation to confidence drift. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 748–755.

123 SATO, A.; YAMADA, K. Generalized learning vector quantization. *Advances in neural information processing systems*, MORGAN KAUFMANN PUBLISHERS, p. 423–429, 1996.

124 ESCOVEDO, T.; KOSHIYAMA, A.; VELLASCO, M.; MELO, R.; CRUZ, A. A. da. A2d2: A pre-event abrupt drift detection. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–8. ISSN 2161-4393.

125 BISHOP, C. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. (Information Science and Statistics). ISBN 9780387310732.

126 CSISZÁR, I.; SHIELDS, P. *Information Theory and Statistics: A Tutorial*. [S.l.]: Now Publishers, 2004. (Foundations and trends in communications and information theory). ISBN 9781933019055.

127 LANDGREBE, T.; PACLIK, P.; DUIN, R.; BRADLEY, A. Precision-recall operating characteristic (p-roc) curves in imprecise environments. In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. [S.l.: s.n.], 2006. v. 4, p. 123–127. ISSN 1051-4651.

128 GU, F.; ZHANG, G.; LU, J.; LIN, C.-T. Concept drift detection based on equal density estimation. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 24–30.

129 CAVALCANTE, R. C.; MINKU, L. L.; OLIVEIRA, A. L. I. Fedd: Feature extraction for explicit concept drift detection in time series. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 740–747.

130 STREHL, A.; GHOSH, J.; MOONEY, R. Impact of similarity measures on web-page clustering. In: *In Workshop on Artificial Intelligence for Web Search (AAAI 2000*. [S.l.]: AAAI, 2000. p. 58–64.

131 RAZA, H.; CECOTTI, H.; PRASAD, G. A combination of transductive and inductive learning for handling non-stationarities in motor imagery classification. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 763–770.

132 PÉREZ-GÁLLEGO, P.; QUEVEDO, J. R.; COZ, J. J. del. Using ensembles for problems with characterizable changes in data distribution: A case study on quantification. *Information Fusion*, v. 34, p. 87 – 100, 2017. ISSN 1566-2535.

133 JORDANEY, R.; SHARAD, K.; DASH, S. K.; WANG, Z.; PAPINI, D.; NOURETDINOV, I.; CAVALLARO, L. Transcend: Detecting concept drift in malware classification models. 2017.

134 PARTRIDGE, D.; YATES, W. Engineering multiversion neural-net systems. *Neural Computation*, v. 8, n. 4, p. 869–893, May 1996. ISSN 0899-7667.

135 MARGINEANTU, D. D.; DIETTERICH, T. G. Pruning adaptive boosting. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. (ICML '97), p. 211–218. ISBN 1-55860-486-3.

136 SOUZA, M. A.; CAVALCANTI, G. D. C.; CRUZ, R. M. O.; SABOURIN, R. On the characterization of the oracle for dynamic classifier selection. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2017. p. 332–339.

137 HO, T. K.; BASU, M. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 24, n. 3, p. 289–300, Mar 2002. ISSN 0162-8828.

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

139 DIDACI, L.; GIORGIO; ROLI, F.; MARCIALIS, G. L. A study on the performances of dynamic classifier selection based on local accuracy estimation. *Pattern Recognition*, v. 38, n. 11, p. 2188 – 2191, 2005. ISSN 0031-3203.

140 ALMEIDA, P.; OLIVEIRA, L.; SILVA, E.; BRITTO JR., A.; KOERICH, A. Parking space detection using textural descriptors. In: *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*. [S.l.: s.n.], 2013. p. 3603–3608.

141 OJALA, T.; PIETIKäINEN, M.; MÄENPÄÄ, T. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, Washington, DC, USA, v. 24, n. 7, p. 971–987, jul. 2002. ISSN 0162-8828.

142 BROWN, C. D.; DAVIS, H. T. Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, v. 80, n. 1, p. 24 – 38, 2006. ISSN 0169-7439.

143 WILCOXON, F. Individual comparisons by ranking methods. *Biometrics bulletin*, JSTOR, v. 1, n. 6, p. 80–83, 1945.

144 ALPAYDIN, E.; KAYNAK, C. Cascading classifiers. *Kybernetika*, v. 34, p. 369–374, 1998.

145 FREY, P. W.; SLATE, D. J. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, v. 6, n. 2, p. 161–182, 1991. ISSN 1573-0565.