UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LEARNING CONCEPT DRIFT WITH OPTIMUM-PATH FOREST

ADRIANA SAYURI IWASHITA

ORIENTADOR: PROF. DR. JOÃO PAULO PAPA

São Carlos – SP

Dezembro/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LEARNING CONCEPT DRIFT WITH OPTIMUM-PATH FOREST

ADRIANA SAYURI IWASHITA

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Processamento de imagens e sinais
Orientador: Prof. Dr. João Paulo Papa

São Carlos – SP

Dezembro/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Tese de Doutorado da candidata Adriana Sayuri Iwashita, realizada em 17/12/2020.

Comissão Julgadora:

Prof. Dr. João Paulo Papa (UFSCar)

Prof. Dr. Alexandre Luis Magalhães Levada (UFSCar)

Prof. Dr. Diego Furtado Silva (UFSCar)

Prof. Dr. Kelton Augusto Pontara da Costa (Fatec)

Prof. Dr. Fabricio Aparecido Breve (UNESP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Aos meus pais.

AGRADECIMENTOS

Agradeço a Deus por ser o apoio em todos os momentos da vida, me dando força, coragem e saúde para enfrentar as dificuldades.

Agradeço aos meus pais que tanto amo. Ao meu pai *Kussuo Iwashita*, que me ensinou que sempre há um lado bom nas coisas, e que mesmo nos momentos difíceis não devo me deixar abater; e à minha mãe *Regina Iwashita*, que através de suas ações me ensina como ter um bom caráter.

Agradeço ao meu orientador Prof. *João Paulo Papa*, por todo o apoio e incentivo que sempre me proporcionou durante todos esses anos, por ser esse professor que é também um amigo e uma pessoa de muito bom coração. Obrigada pela paciência nos momentos difíceis, pela disponibilidade nos momentos de dúvida e pelas orientações nos momentos de insegurança.

Agradeço aos professores e funcionários do DC/UFSCar - São Carlos, que auxiliaram na realização deste trabalho direta ou indiretamente. Agradeço também à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) pelo auxílio financeiro concedido.

Agradeço ao *Bruno Kochimizu*, que esteve sempre ao meu lado me apoiando em tudo, em todos os momentos, nos altos e baixos.

Agradeço a todos os membros do grupo RECOGNA, em especial aos amigos *Rafael*, *Luis*, *Leandro*, *Clayton* e *Douglas* pelo companheirismo e por toda a disposição em ajudar.

Enfim, agradeço aos meus amigos, familiares e a todos que me ajudaram e que torceram por mim.

Muito obrigada!

If I have seen further it is by standing on the shoulders of Giants.

Isaac Newton

Resumo

Algoritmos de classificação baseiam sua decisão de acordo com o seu aprendizado na base de dados sobre um conjunto de treinamento. Logo, os dados a serem classificados no conjunto de teste devem possuir distribuição igual do conjunto de treinamento para que sejam corretamente identificados. Atualmente, as aplicações industriais e de empresas geram uma enorme quantidade de fluxo de dados, tais como os dados de uma rede de sensores, registros de chamadas, entre outros. Ainda, com as novas tecnologias sendo desenvolvidas em serviços de internet, surgem fluxo de dados dos mais diversos domínios, incluindo transações de compras na internet e pesquisas na web. Esses fluxos de dados apresentam características que os métodos tradicionais em mineração de dados agora precisam lidar, que são bases de dados com grande volume e que estão sujeitas à mudança de conceito, a qual refere-se a um problema de aprendizagem não estacionário ao longo do tempo, ou seja, o classificador de determinado problema pode não ser mais útil após decorrido algum tempo, por estar "desatualizado". Isso ocorre pois um conceito pode sofrer modificações com o tempo. Por exemplo, um leitor pode gostar de artigos com notícias relativas à "esportes"; mas com o passar do tempo sua preferência de leitura pode mudar para "economia" e o tópico anterior se tornar irrelevante para ele, ou seja, o conceito de artigo relevante para este leitor foi alterado. O presente trabalho de pesquisa propõe o estudo do classificador *Optimum-Path Forest* (OPF) em ambientes com mudança de conceito, tanto na abordagem supervisionada (utilizando alguns métodos para lidar com mudança de conceito como o uso de janelas nos dados e comitês de decisão) como na abordagem não supervisionada, e realizamos experimentos em bases de dados encontrados na literatura.

Palavras-chave: OPF, Mudança de conceito

ABSTRACT

Classification algorithms take their decisions according to a learning process on the training set. Therefore, the data to be classified in the test set must have the same distribution as the training set to be correctly identified. Nowadays, industrial and enterprise applications generate a huge amount of data streams, such as sensor network data, and call records, among others. Also, with the new technologies being developed in internet services, data can stream from diverse domains, including internet transactions and web searches. These data streams present characteristics that traditional data mining methods have to deal with, which are databases with high volume and susceptible to concept drift, which refers to a non-stationary learning problem over time, i.e., the classifier of a certain problem may not be suitable as time goes by for being “outdated.” This occurs because a concept may change over time. For example, a reader might like news articles on “sports”; but over time your reading preference may change to “economy” and the previous topic becomes irrelevant, i.e., the concept of an article relevant to this reader has changed. The present research proposes the study of Optimum-Path Forest (OPF) classifier in dynamic environments, both in supervised approach (using some methods to deal with concept drift as data windows and decision committees) as in the unsupervised approach, and we conducted experiments on databases observed in the literature.

Keywords: OPF, Concept drift

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Chapter 1
INTRODUCTION

"Concept drift" refers to a non-stationary learning problem over time (ZLIOBAITE, 2010), unlike traditional learning techniques where the static concept learned can be used to classify future occurrences indefinitely. In the real world, training and application data can present incompatible problems as their behavior can change over time. Sequential data distribution can also change, becoming the model built on the previous data inconsistent with the new one. Thus, a periodic update of the model becomes necessary (TSYMBAL, 2004). For example, personal preferences for products may change depending on the economy, the day of the week, and inflation, among others. When the conditions of an industry change, the validation process of a product also varies. Commonly, the cause of a change is not straightforward, leaving it to be deduced by the classifiers. Algorithms to deal with concept drift must be able to identify a change in the target concept without knowledge of the change in the subsequent distribution. If the concept changes over time, the classification problem becomes harder. Learning must continue as new instances arrive for the concept drift be traced (ZLIOBAITE, 2010).

Several works about concept drift were proposed in the literature. Widmer and Kubat (WIDMER; KUBAT, 1996), for example, proposed a framework (FLOating Rough Approximation - FLORA) that consists of a family of algorithms with the following characteristics: FLORA1 is the simplest one, and the representation of the hypothesis is in the form of three sets of descriptors that summarize both positive and negative information; FLORA2 uses a flexible "forgetting" operator controlled by a time window in the input data; FLORA3 can deal with environments where a concept can reappear; while FLORA4 is concerned with noise. Klinkenberg and Joachims (KLINKENBERG; JOACHIMS, 2000) proposed a supervised method to recognize and treat concept drift using Support Vector Machines (SVMs), whose main idea is to automatically adjust the size of the training time window to minimize the estimated generalization error. A large window provides lots of data, usually providing good generalization in environments where the concept does not change. However, it may contain old data that are no longer relevant in changing environments. Stanley (STANLEY, 2003) employs a weighted voting instead of the window method, named Concept Drift Committee (CDC). In this method, there is a committee composed of members whose vote is weighted according to their recent history. When a committee member's performance is considerably affected, a new member replaces it. Each committee member has a decision tree as a concept learning algorithm, and the author states the possibility of using any other supervised learning algorithm.

Harel et al. (HAREL et al., 2014) proposed a model to detect or predict concept drift by resampling, which investigates and identifies changes that occur along the data stream more robust to noise and invariant to the learning problem, assuming temporal independence of the samples. Masud et al. (MASUD et al., 2011b) proposed a work to deal with Concept Evolution in concept drift environments. Concept Evolution is the appearance of a new class in the data. The authors used the k-nearest neighbors and decision trees as classification algorithms and defined a new-class appearance when none of the classification models could classify an instance, named Foutlier. They also presented a coefficient to measure the cohesion and the separation of Foutliers: when this coefficient has a positive value for a q number of Foutliers, it indicates that these form a new class. Later on, Masud et al. (MASUD et al., 2011a) proposed a semi-supervised technique to deal with recurrent classes, i.e., instances that appear in the dataset and later reappear. The algorithm groups the training data using k-means, builds a decision boundary, stores the summary of each cluster, and comprises information such as centroid and radius. When an instance in the input data is considered as an outlier (if the instance is outside the decision boundary), it is called P-outlier (primary outlier). It is then verified if this instance is an outlier of all previous models. In the negative case, it is classified as an instance of a recurring class. In a positive case (S-outlier - second outlier), this instance is stored in a repository that will be analyzed after predefined stored instances. Using a neighborhood coefficient, we can declare that these instances compose a new class. Klinkenberg (KLINKENBERG, 2001) further proposed a method for recognizing and dealing with concept drift using SVM and unlabeled data to reduce the need for labeled data, which is an extension of Klinkenberg and Joachims (KLINKENBERG; JOACHIMS, 2000).

Other authors proposed different alternatives to deal with concept drift. Liu et al. (LIU; LU; ZHANG, 2020) presented a cluster-based histogram named equal intensity k-means space partitioning (EI-kMeans) with a heuristic method to improve the sensitivity of drift detection. The authors used Pearson's chi-square test as the statistical hypothesis test and implemented greedy centroids initialization algorithm, a cluster amplify-shrink algorithm, and a drift detection algorithm. Heusinger et al. (HEUSINGER; RAAB; SCHLEIF, 2020) proposed adaptive versions of Robust Soft Learning Vector Quantization (RSLVQ) and Generalized Learning Vector Quantization (GLVQ) using Adadelta and Adamax and applying momentum-based stochastic gradient descent techniques to tackle concept drift passively. Sethi and Kantardzic (SETHI; KANTARDZIC, 2017) proposed an unsupervised and incremental algorithm named Margin Density Drift Detection (MD3), which tracks the number of instances in the uncertainty region of a classifier (margin) as a drift detection metric. Gao et al. (GAO et al., 2020) proposed a graph-based clustering method to identify instances from novel classes in a semi-supervised manner named Semi-supervised Adaptive ClassifiCation Over data Stream (SACCOS). They perform an online normalization on samples along the stream to unify their scale and use an ensemble of clusters to detect novel classes. The supervised training phase obtains a C classifier from normalized data then a set of clusters is generated and added to the initial cluster ensemble M (the ensemble size is finite with a maximum of T_M sets of clusters). For every new data instance, the method saves it in a temporary data buffer S, whose maximum size is given by C. Whenever S is full, SACCOS updates the normalization parameters, normalizes those instances in S, sends it to other modules for further processing, and cleans S. The clustering method aims to help to discover significant clusters of correlated samples in the feature space. They propose a Mutual Graph Clustering (MGC) algorithm based on mutual k_{NN} for identifying significant clusters.

These techniques aim at solving the classification problem when the trained model does not handle current data well, as there has been a concept drift over time. Examples of environments where such changes can occur: personal interest of a user for subjects of newsgroups (e.g., medicine, space, baseball); defining types of cover forest in a region; recognizing whether the price of electricity will increase or decrease according to demand and supply of products; cases of Non-technical losses (also known as commercial losses) which stand primarily for energy theft in power distribution systems but not limited to it; among others. Dealing with this type of situation is an important task.

Papa et al. (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012) proposed a methodology for pattern classifiers based on the Optimum-Path Forest (OPF) classifiers. OPF-based classifiers reduce the pattern recognition problem to a graph partitioning in optimum path trees, which are rooted by prototype samples, and an element belonging to a given tree is more strongly connected to its root than any other. Such connectivity strength is established by a path-cost function. Currently, two variants were proposed: (i) the unsupervised version (ROCHA; CAPPABIANCO; FALCÃO, 2009) and (ii) the supervised variant, which is divided in the OPF with complete graph (PAPA; FALCÃO; SUZUKI, 2009) and OPF with k-nearest neighbors graph (PAPA; FALCÃO, 2008). Also, an optimization of the OPF training algorithm was proposed by Iwashita et al. (IWASHITA et al., 2014) based on the theoretical relationship between minimum spanning forests and optimum path forests. As optimum paths follow the form of the Minimum Spanning Tree (MST), an optimum path forest can be built by removing arcs between prototypes, edge redirection, and label and path cost propagation using the f_{max} function, thus building the optimum-path forest at the same time as the MST is computed.

1.1 Hypotheses

As far as we are concerned, OPF-based classifiers have never been evaluated in concept drift problems or handling situations with non-stationary environments. OPF classifier behavior is also unknown in such conditions. Thus, the *hypotheses of this Ph.D. thesis are: i) OPF can be efficient to handle non-stationary environments, ii) the OPF classifier can be robust enough to address concept drift problem, iii) we can provide an effective OPF implementation to handle dynamic environments.*

1.2 Objectives

In this context, we proposed to apply supervised and unsupervised approaches based on the Optimum Path Forest algorithm in dynamic environments to deal with concept drift. Also, as main contributions, it will be conducted a study in incremental learning concerning the supervised OPF.

The next chapters present a literature review, theoretical reference on concept drift and the OPF classifier, as well as the conducted experiments:

- Chapter 2 presents a bibliographical review of concept drift proposed by Iwashita and Papa.
- Chapter 3 presents the Optimum Path Forest classifier in its supervised versions (with complete graph and k-nn graph), unsupervised version, and the accuracy measure employed in the experiments.
- Chapter 4 presents an initial evaluation of the OPF classifier in a concept drift environment.
- Chapter 5 evaluates an ensemble of OPF classifiers in the concept drift problem and also assesses OPF classifier with a drift detection method based on accuracy.

- Chapter 6 presents an OPF$_{knn}$ approach with incremental learning to add new instances to the optimum path forest without re-training the entire database (Incremental OPF$_{knn}$). This Chapter also presents the performed experiments.

- Chapter 7 presents some preliminary results concerning drift detection with unsupervised OPF.

- Chapter 8 presents conclusions and future works.

Appendix A extends some experiments presented in Chapter 5.

Chapter 2

Literature review

This chapter presents a literature review about techniques to deal with concept drift presented by Iwashita and Papa (IWASHITA; PAPA, 2019).

2.1 Introduction

Since concept drift is an important area that has gained the attention in the last years, this work presents a compilation of several works to foster the research in the area of knowledge. The main contribution of this survey is to study different techniques to detect and deal with concept drift, as well as to study public synthetic and real datasets in this area. The remainder of this work is organized as follows. Section 2.2 presents the main theoretical background regarding concept drift, types of algorithms to deal with (Subsection 2.2.1), and techniques in the literature that handles the concept drift issue (Section 2.3). Section 2.4 presents a summary of the articles studied, and Section 2.5 states conclusions and future directions of the area.

2.2 Concept Drift

Concept drift happens when the target concept changes in a non-stationary environment. Let $C1$ and $C2$ be two target concepts, and $\mathscr{I} = \{i_1, i_2, \ldots, i_n\}$ an ordered instance sequence. Instances prior to i_d have a stable concept $C1$ and does not change. After Δx instances, the concept stabilizes once more, but in another target concept $C2$. The concept among instances i_{d+1} and $i_{d+\Delta x}$ is *drifting* between $C1$ and $C2$ (STANLEY, 2003). According to Δx length, the drift can be called gradual or abrupt. In gradual drift, the two concepts slowly swap; whereas in abrupt drift it sudden occurs.

Wadewale and Desai (WADEWALE; DESAI, 2015) classify the variations of the target concept drift in sudden, incremental, gradual, recurring, blip and noise drifts. Fig. 2.1 contains this different types of drifts. Fig. 2.1a shows a sudden drift that the data changes instantly and without alternation. In incremental and gradual drifts the changes occur slowly. Incremental drift (Fig. 2.1b) happens when the data values gradually change over time, whereas gradual drift (Fig. 2.1c) also includes changing in class distribution. Recurring drifts (Fig. 2.1d) occurs when instances of a concept temporary disappear and return after a while. Fig. 2.1e shows a rare event which in a static distribution can be considered as an outlier, and Fig. 2.1f shows random changes in instances (noise) that have to be filtered out (WADEWALE; DESAI, 2015). Kuncheva (KUNCHEVA, 2008) says that blip events represent a "rare event" and finding it in streaming data can indicate the beginning of a concept drift. Hence the methods for online detection of rare events can be a component of the novelty detection paradigm. Some authors point out that there is no universal definition of "outlier" and alternative terminologies permeate the literature, e.g., novelty detection, anomaly detection, noise detection, deviation detection, or exception mining (KUNCHEVA, 2008).

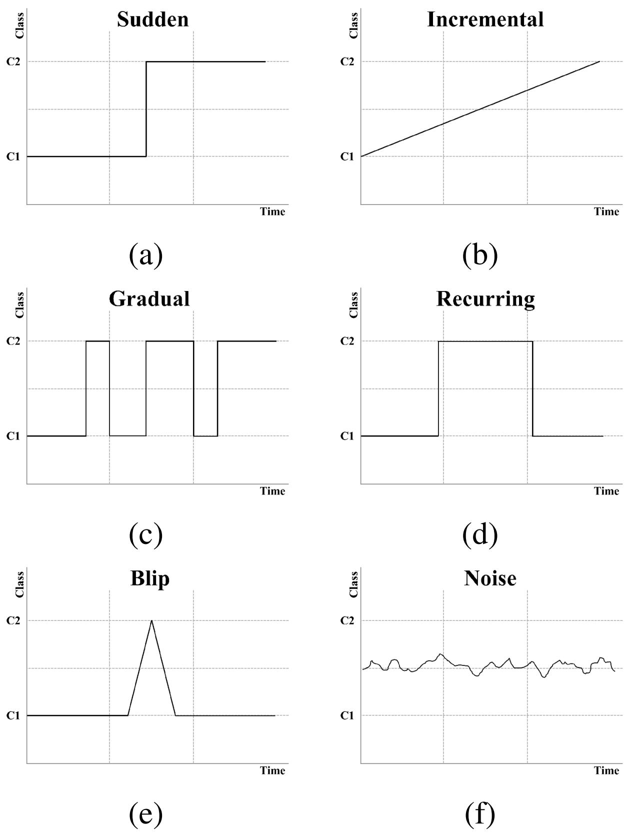

Figure 2.1: Types of concept drift, extracted from (WADEWALE; DESAI, 2015): (a) sudden drift, (b) incremental drift, (c) gradual drift, (d) recurring drift, (e) outlier, and (f) noise.

Some authors (GAMA et al., 2014) define the drifts in two types: real concept drift and virtual drift. Considering concept drift as changes in data distribution, the real concept drift occurs when the conditional distribution changes on the output whereas the input distribution remains unchanged. Virtual drift has different interpretations in the literature, as the changes in the distribution of incoming data, among others (GAMA et al., 2014).

2.2.1 Types of Algorithms

The most usual ways to handle concept drift are the following three (FARID et al., 2013): (a) instance selection or window-based approach, (b) weight-based approach, and (c) ensemble of classifiers.

Instance selection or window-based approaches select an appropriate set of previous data to train a classifier (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010). It selects instances inside of a fixed or dynamic sliding window (FARID et al., 2013). These approaches assume older examples are incompatible with new data classification, so it has to forget old instances which are considered useless to handle concept drift (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010). Therefore, the training employs the last batch of information with the last training instances. The window of fixed size approach is the simplest rule and the window size is usually determined by the user. By having information on the time-scale of change, a window of fixed size approach is a good choice. Yet, the user is often caught in a trade-off: choosing a small window size (reflects the current distribution with fast adaptivity) or a large window size (having more instances in periods of stability with no concept drift may increase accuracy and have better generalization) (BIFET; GAVALDÀ, 2007). The adaptive window approach adjusts the window size to the length of the drift (KLINKENBERG; JOACHIMS, 2000). It commonly maintains the examples up until the concept drift: in this case, it is not necessary to choose a priori and unknown parameter (BIFET; GAVALDÀ, 2007). Other approaches maintain some of the examples in memory but select a training set of it for classification (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010).

A weight-based approach weights instances and deletes the outdated training ones based on their weights (FARID et al., 2013). As time passes, it considers old information increasingly irrelevant. Despite considering all instances of learning, new instances have more relevance. Within this framework, an updateable classifier capable of weighted learning must be chosen (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010).

The ensemble of classifier combines several outputs from different learners to define a final classification (FARID et al., 2013). With a dynamic set of classifiers, performance (or another metric) is observed. If performance decreases, new classifiers replace the aged and poor performing classifiers on the ensemble. Outputs from learners are combined to classify instances on classification phase, commonly with a weighted-vote mechanism (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010). Benefits of weighted ensembles have been studied empirically and theoretically (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010).

Data streams can be divided into batches in window-based approaches; thus, it continuously receives batches of data. Some examples are described below (KLINKENBERG; JOACHIMS, 2000):

- Full-memory: the learner employs all instances seen so far (batches), i.e., it can not “forget” old data (Figure 2.2).
- No-memory: the learner employs a single batch on training, i.e., the most recent of the stream (Figure 2.3).
- Window of fixed size *n*: the learner employs *n* batches on training, e.g., a sliding window of size $n = 3$ is used with the most recent instances (Figure 2.4).

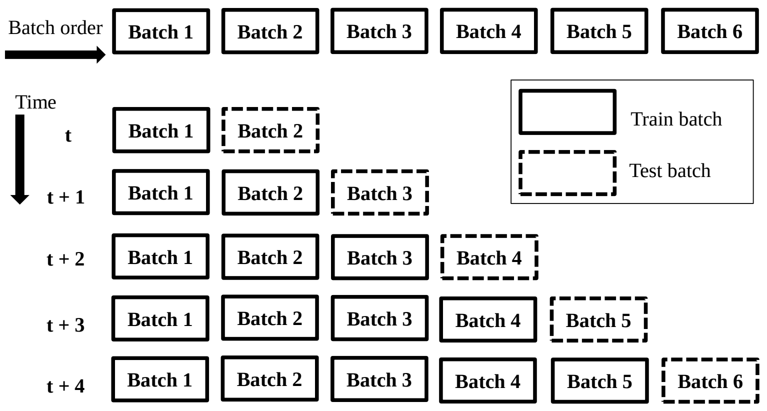

Figure 2.2: *Full-memory* approach.

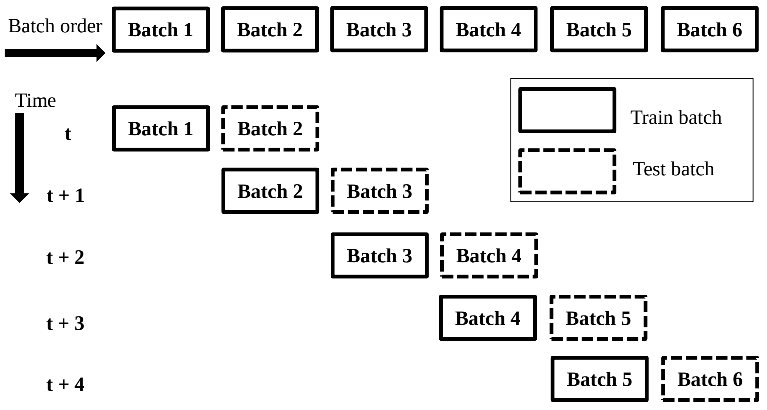

Figure 2.3: *No-memory* approach.

Ditzler and Polikar (DITZLER; POLIKAR, 2013) characterized various forms of concept drift handling algorithms:

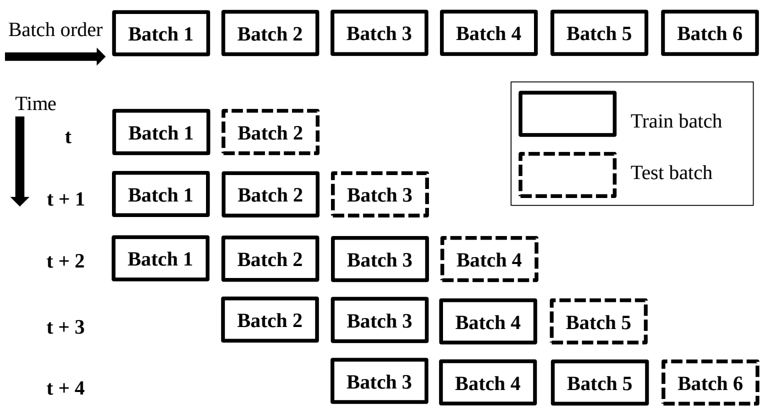

Figure 2.4: ***Window*-of-size-3 approach.**

- Online or Batch approaches: determined by the amount of instances considered at training phase;
- Single classifier or Ensemble-based approaches: determined by the number of learners employed to decide a classification;
- Incremental or Non-incremental approaches: determined by the reusing a data or not; and
- Active or Passive approaches: determined by the use of a drift detection mechanism or not.

After receiving an instance, the online approach updates the classifier; whereas the batch approach waits to receive plenty of instances to start learning (TSYMBAL, 2004). Single classifiers use one learner to decide on classification phase; whereas ensemble learning combines the results of a set of concept learners with a single or weighted vote, or selecting the most significant result (TSYMBAL, 2004). Incremental learning behaves like online learning with the model update as instances arrive; whereas non-incremental reuses data on learning phase. Active drift detection observes the stream to search for changes and determine whether and when a drift occurs: after a drift, it warns the learner to take the correct action. Passive drift detection considers drift may occur constantly or occasionally, therefore continually updates the learner as data arrive (DITZLER; POLIKAR, 2013).

Kuncheva (KUNCHEVA, 2008) considered that change detection from unlabeled data in classification problems could be useful in three situations: i) be more sensitive to error, ii) missed opportunities, and iii) label delay. About being sensitive to error, it would be the case of detecting a change in the distribution to anticipate an error change; and regarding missed opportunities, if a change is not detected, we are missing the opportunity to improve the classifier training a new one with a smaller error. Concerning label delay or label latency, occurs when the labels come with a considerable delay and are not available at the time of classification, e.g., applications where the true class label becomes known after some time (as an "intermediate latency"), after very little time ("zero latency") or in unlabelled data where it is expensive or impossible to obtain the class labels ("infinite latency").

2.3 Literature Review

In 1986, Schlimmer and Granger (SCHLIMMER; GRANGER, 1986) proposed the first technique to handle concept drift, a supervised learning method named STAGGER (KOLTER; MALOOF, 2007), which trains Boolean characterization via formula based on Bayesian statistics. It keeps a set of concept descriptions (produced by using feature construction and features from the method itself), and select those with better relevance to the present data (TSYMBAL, 2004). It starts with its own features and trains new characterizations by middle-out beam search through the space of possible conjunctive, disjunctive, and negated characterizations. STAGGER is a binary classification method which detects concept drift using backtracking and Bayesian weighting measures to discriminate concept drift from noise (SCHLIMMER; GRANGER, 1986).

Salganicoff (SALGANICOFF, 1993), in 1993, proposed the DARLING method (Density-Adaptive Reinforcement Learning), a supervised density-adaptive forgetting technique that uses exponential weight-decay based on nearest neighbor criterium to exclude instances considered obsolete. The method excludes an example if its weight drops below a threshold and new examples take its place. In the space of attributes, the number of samples per unit volume is the local "density." DARLING builds a *kd-tree* structure for the nearest-neighbor forgetting method in a binary classification way (SALGANICOFF, 1993).

Kubat and Widmer (KUBAT; WIDMER, 1995) use Radial Basis Functions (RBF) to learn in non-stationary numeric domains. FRANN algorithm (Floating Rough Approximation in Neural Networks) uses hill-climbing to search for the best "sliding window," and build an RBF network from it. This dynamic sliding window uses heuristics to delete older instances and to determine its window size (KUBAT; WIDMER, 1995).

Widmer and Kubat (WIDMER; KUBAT, 1996) proposed a supervised framework named FLORA (FLOating Rough Approximation), which consists of a family of algorithms where each version is an anterior extension. In the FLORA framework, three description sets represent a concept: the Accepted DEScriptors (ADES) used to classify new samples, represent the present (positive) hypothesis; the Negative DEScriptors (NDES) represent negative samples and employed to avoid over-generalization of ADES; and the Potential DEScriptors (PDES) which is a set that might become pertinent in the future, used as a reservoir of general hypotheses. FLORA keeps the relevant descriptions items of the most recent instances in the window. FLORA2 has a “forgetting” operator, which dynamically adjust the window size in the training phase. Two heuristics are used to detect drift: the predictive performance of old classifications, and some syntactic properties of the hypotheses. If performance drops significantly or a substantial difference appears in the number of items in ADES, the method signals a warning of concept drift. The window size decreases and the algorithm “forget” old instances if drift occurs. Otherwise, the window size enlarges to produce a stable concept. FLORA3 can deal with recurring contexts, storing old concepts for later reuse. When a stable concept reached, it stores the current hypothesis. When concept drifts, the algorithm verifies if there is any old descriptor that can describe the present instances. FLORA4 was proposed to deal with noise. Each description item has a classification record, and it builds statistical confidence intervals around these measures (WIDMER; KUBAT, 1996).

Klinkenberg and Joachims (KLINKENBERG; JOACHIMS, 2000) use Support Vector Machines to deal with concept drift. The method keeps a window of examples, trying to minimize its generalization error discarding irrelevant data. The authors used the full-memory, no-memory, window of fixed and adaptive size data management approaches (KLINKENBERG; JOACHIMS, 2000).

Street and Kim (STREET; KIM, 2001) proposed the SEA algorithm (Streaming Ensemble Algorithm), an ensemble of decision trees and each one created by one batch. It uses unweighted majority-vote, similar to bagging, to classify an instance. The ensemble has a maximum number of classifiers, and once this number achieved, those new classifiers that reach certain criteria replace old classifiers. Performance estimates are computed on the next batch using the new tree, and the ensembles are built with Quinlan’s C4.5 (STREET; KIM, 2001).

Stanley (STANLEY, 2003) proposed the CDC (Concept Drift Committee) algorithm, a supervised method that employs a weighted committee of hypotheses. All committee members can access all features and when an older committee member’s voting value falls below a threshold, a new member replaces it. Each member gives a (weighted) vote and keeps a hypothesis based on instances seen in its lifetime, each other having a different amount of instances considered. An implicit window considers only the latest examples, and each committee member uses a decision tree (but according to authors any supervised learning algorithm can be used (STANLEY, 2003)).

Scholz and Klinkenberg (SCHOLZ; KLINKENBERG, 2005) proposed the KBS (Knowledge-Based Sampling) algorithm, a boosting-like ensemble method. This supervised algorithm employs KBS and SVM with the linear kernel or Decision Tree in a binary classification way ("relevant" or "irrelevant" class). It considers the latest instances batch, inducing and reweighting base models continuously (SCHOLZ; KLINKENBERG, 2005).

Bifet and Gavalda (BIFET; GAVALDÀ, 2007) presented the ADWIN2 (ADaptive WINdowing) algorithm, an improved version of ADWIN algorithm. ADWIN2 has a variable sized window: it grows or shrinks when no change or concept drift is detected, respectively. This supervised method detects drifts using the average of the elements in the window (BIFET; GAVALDÀ, 2007).

Abdulsalam et al. (ABDULSALAM; SKILLICORN; MARTIN, 2011) combine the ideas of streaming decision trees and Random Forests in an incremental multiclass algorithm. It can adjust its parameters to handle drifts and uses the difference in entropy between two-windows – the current and the reference window – to detect drift. Each attribute has counters and the probabilities of occurrences are calculated, being the differences averaged and used to calculate entropy changes. When the method builds a new tree; it adds or replaces an old one depending on the number of existing trees in the forest (ABDULSALAM; SKILLICORN; MARTIN, 2011).

With minimal distance classifier and a sliding window, Kurlej and Wozniak (KURLEJ; WOZNIAK, 2012) proposed an active learning approach that ponders if an outside expert has to label a new example as training instance or not. A heuristic algorithm defines if a new example is a good reference using two values obtained in this set: the distance to the closest point and the difference in the distance to two closest points belonging to distinct classes. If a new example achieves certain conditions, it replaces the oldest example of the reference set (KURLEJ; WOZNIAK, 2012).

Vivekanandan and Nedunchezhian (VIVEKANANDAN; NEDUNCHEZHIAN, 2011) proposed an online Genetic Algorithm (GA) that takes small snapshots of the training sample and creates rules for all classes separately. For each class, it uses multiple windows of fixed size to keep the examples. The window size of each class varies depends on his overall distribution. The oldest examples will be replaced by new ones if the window of the class becomes full (VIVEKANANDAN; NEDUNCHEZHIAN, 2011).

Sun and Li (SUN; LI, 2011) proposed the first study on Financial Distress Concept Drift (FDCD): if there is FDCD and in what way to discard it. To discard FDCD, they build a dynamic FDP model, which embraces instance selection, FDP modeling, and future prediction. To deal with FDCD, the algorithm uses methods like the ones based on windows (full-memory, no-memory, fixed and adaptative size), and batch selection. The authors also integrate Mahalanobis distance for feature selection and employed Fisher discriminant for classification (SUN;

LI, 2011).

Hegedus et al. (HEGEDUS; ORMÁNDI; JELASITY, 2013) presented adaptative versions of GoLF (Gossip Learning Framework): ADAGoLF (with age-based drift handling) and CD-DGoLF (with concept drift detection), to handle concept drift in large networks. Adaptivity in AdaGoLF uses the models in the network, employing age distribution manipulation. It offers both young and old models at any moment, providing diversity of varied ages. CDDGoLF is employed to detect concept drift. It can estimate and monitor performance drifts to forget data with a performance decay (HEGEDUS; ORMÁNDI; JELASITY, 2013).

Bertini et al. (BERTINI; NICOLETTI; ZHAO, 2013) proposed the Ensemble of CPp-AbDG (Complete P-partite Attribute-based Decision Graph), being the data representation based on graph structures. It can handle missing attribute values. CPp-AbDG extends other data graph constructor: AbDG (Attribute-based Decision Graph). The AbDG can be theoretically designed to a graph in which one vertice represents a subrange of the range value of one attribute. Graph-based model is employed as classifiers in the ensemble, since the authors justify the use by the advantages of representing data topologically, with arbitrary shapes, and hierarchically (BERTINI; NICOLETTI; ZHAO, 2013).

Escovedo et al. (ESCOVEDO et al., 2013) presented the NEVE (Neuro-EVolutionary Ensemble) algorithm, a supervised ensemble of weighted classifiers (neural networks), which employs QIEA (Quantum-Inspired Evolutionary Algorithm) to train. QIEAs can estimates class distribution, providing good performance. It also determines weights for each ensemble's classifier when a new batch arrives. A new classifier is added to the ensemble once a new batch arrives, and all weights are updated to improve the performance (ESCOVEDO et al., 2013).

Li et al. (LI et al., 2015) proposed the EDTC (Ensemble Decision Trees for Concept drifting data streams), an incremental method that defines cut-points in the growing tree with three different random feature selection. When an instance arrives, each growing node split-features randomly to avoid producing unnecessary branches. EDTC employs two thresholds and uses local data distributions to detect drift (LI et al., 2015).

Loeffel et al. (LOEFFEL; MARSALA; DETYNIECKI, 2015) proposed an online algorithm (the Droplets algorithm) that can opt to abstain from predicting to handling drifts. Each instance is considered as a droplet falling on a plan (feature space), i.e., the "Droplets' map." Classes can be considered as a "chemical" composition of the droplets which are reciprocally repellent. Thus, two droplets from different classes will not mutual coverage parts of the map. The method does not employ a fixed threshold. According to authors, it is the first "algorithm to handle concept drift" proposed to abstaining from prediction (LOEFFEL; MARSALA; DETYNIECKI, 2015).

Chen et al. (CHEN et al., 2015) proposed a Genetic Algorithm to obtain fuzzy concept drift patterns. CDGFM (Concept Drift Genetic-Fuzzy Mining) handles drift with instance selection in a multiclass classification way. Membership functions of items are transformed into chromosomes and obtained on GA process. Each example has a fitness value assessed by the amount of concept drift patterns (new concept, drift and added/expired rules) and the membership function. The membership functions search for satisfying sets of fuzzy association rules to define drift (CHEN et al., 2015).

ZareMoodi et al. (ZAREMOODI; BEIGY; SIAHROUDI, 2015) proposed LOCE (LOcal Classifier Ensemble), an algorithm capable to detect the appearance of new classes in streams. Each class has an ensemble of classifiers, which are updated by its own metrics and by a pruning phase (cutting classifiers to system update). If one class no longer exists, all classifiers belonging to the ensemble will be eliminated. It classifies existing classes' examples and discerns between the novel and existing classes with local patterns. Using a neighborhood graph to detect new classes, it stores new class candidates to its nodes using cohesion and separation to determine the components of it (ZAREMOODI; BEIGY; SIAHROUDI, 2015).

Diaz et al. (DÍAZ et al., 2015) proposed the FAE (Fast Adapting Ensemble) algorithm, a multiclass ensemble algorithm which can deal with recurring concepts. It employs a batch scheme, but it does not need to wait for the entire batch to start classification. A drift detector (often Drift Detection Method - DDM) decides when to increase the number of classifiers to the ensemble. To deal with recurring concepts, it keeps several aged classifiers (former concepts) which are awakened if these concepts reoccur, avoiding unnecessarily inclusion of new classifiers. To make a global decision, it employs a weighted majority ensemble vote. It dynamically adjusts the base classifiers weights, which permits classifiers to stay longer on the ensemble (DÍAZ et al., 2015).

Mirza et al. (MIRZA; LIN; LIU, 2015) proposed the ESOS-ELM (Ensemble of Subset Online Sequential Extreme Learning Machine), a drift detector which can deal with class imbalance on stream of data. The main ensemble represents the short-term memory, in which each classifier trains with a balanced selection of the original imbalanced data. It also has a concept drift detector, and the long-term memory keeps information. m classifiers process a minority class instance, being m the imbalance ratio; while a single classifier is employed to a majority class instance, which is processed in a round-robin fashion. According to the authors, the method can do both online and batch learning (MIRZA; LIN; LIU, 2015).

ZareMoodi et al. (ZAREMOODI; SIAHROUDI; BEIGY, 2016) proposed the SVSCLASS (Support Vector-based Stream CLASSifier), a support vector-based method that can deal with the appearance of new classes. Using the support vector domain description, it maintains classes' boundaries with spheres in the kernel space, which is used to label instances of the incoming batches. Instances located outside of the spheres may stand for an emergence of a new class. To detect a new class, it builds a neighborhood graph to analyze the cohesion together and separated from existing classes. Using support vector clustering, the instances will be partitioned into clusters and analyzed to be labeled (the method acquires the true labels and the sphere is updated). To handle concept drift, the spheres boundaries shrink, enlarge and merge during learning (ZAREMOODI; SIAHROUDI; BEIGY, 2016).

Klinkenberg and Renz (KLINKENBERG; RENZ, 1998) presented a method to detected concept drift using windows approach with fixed or adaptive size. The subject addressed in this paper was information filtering, having two concepts to deal with: relevant or irrelevant. Using some indicators (recall, precision, and accuracy, being the latter considered less important to detect drift according to authors), it detects concept drift and calculates heuristics to adapt the window size. If an abrupt change occurs, the window reaches its minimal size (one batch); whereas if a gradual change occurs, its size is decreased according to a reduction rate defined by the user. If no drift is detected, instances are stored until a maximum training set size to build a stable learner (KLINKENBERG; RENZ, 1998).

Gama et al. (GAMA et al., 2004) proposed the DDM (Drift Detection Method), a drift detector that works with probability distribution using the online error-rate. It defines two levels: the warning and the drift level. If the error reaches respectively the warning level at instance i_w and the drift level at instance i_d, it is considered the occurrence of a drift, and the method starts the training process with data from i_w. According to the authors, it can operate with online and incremental methods, and also as a wrapper to batch classifiers (GAMA et al., 2004).

Yang et al. (YANG; WU; ZHU, 2005) proposed a RePro system (REactive plus PROactive). RePro can conduct reactive prediction: it detects the concept drift and learns with new data; and also can be proactive: predict the next concept given the present one. They generate a concept history from the stream (which is more compact than raw data) to learn patterns of concept transitions. RePro employs a sliding window structure which starts with a misclassified example. Once the window is filled and its error rate stands above a threshold, the first example is considered a trigger; otherwise, the window pass to the next misclassified example, excluding previous examples. The proactive mode works after trigger detection and does not employ window warning. When a new trigger is identified, the predicted concept takes the lead. The reactive mode builds a model only after drift detection, using trigger examples. According to the authors, RePro can detect the emergence of a new concept and the reappearance of an old one (YANG; WU; ZHU, 2005).

García et al. (GARCÍA et al., 2006) presented EDDM (Early Drift Detection Method), a drift detector that works with distance-error-rate (estimated distribution of the distance among classification errors) rather than errors classifications. EDDM has two thresholds: the Warning level – exceeded this level, instances are kept; and the Drift level – the method considers concept has drifted and builds a new model with data from warning level. EDDM starts to search for a concept drift after 30 errors occurred. Otherwise, if the system has a rise of similarity after warning threshold, the instances are deleted and the system reverts to an "in-control level" (GARCÍA et al., 2006).

Da Silva et al. (SILVA et al., 2006) proposed the RL-CD (Reinforcement Learning with Context Detection), which evaluates the prediction quality of partial model. It manages several partial models, creating, updating and selecting them. The method only select the partial model with the highest quality, each one specialized in a particular environment dynamics. A model replacement means a drift detection in the environment. A new model is created if the best model quality is below a threshold. The new model produced will consider dynamics predictor and the corresponding optimal policy (SILVA et al., 2006).

Kolter and Maloof (KOLTER; MALOOF, 2007) proposed the DWM (Dynamic Weighted Majority), an ensemble algorithm to deal with concept drift that creates and removes weighted experts according to global performance. If the main procedure gives a wrong weighted majority answer, the method includes an expert to the ensemble. Otherwise, if an expert gives a wrong answer, its weight is decreased. If an expert gets his weight greatly decreased, it is deleted from the ensemble (KOLTER; MALOOF, 2007).

Katakis et al. (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010) proposed the CCP (Conceptual Clustering & Prediction) framework, an incremental ensemble to deal with recurring contexts which uses clusters and a transformation function to map batches into conceptual representation models. Each classifier in the ensemble represents a concept, and when a new batch comes from the stream, the clustering algorithm identifies its concept and CCP employs the corresponding classifier to label the instances (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010).

On the area of face detection, Susnjak et al. (SUSNJAK; BARCZAK; HAWICK, 2012) proposed an adaptive learning method for cascades of boosted ensembles. The algorithm employs a hybrid of active and passive updating approach. During training, classifiers belonging to the same layer are combined into ensemble-clusters in which its results are weighted based on their performance. During classification, the algorithm uses the deliberations of ensemble-clusters per layer to calculate a collective value, which is compared with the learned layer thresholds to give a decisive answer. To detect drift it uses trigger mechanism with classification error rate (SUSNJAK; BARCZAK; HAWICK, 2012).

Minku and Yao (MINKU; YAO, 2012) proposed the DDD (Diversity for Dealing with Drifts) online ensemble algorithm. The authors also analyzed the combination of low and high diversity ensembles, concluding that each diversity level reaches better prequential accuracy according to the drift type. The paper further reveals the possibility of old concept information benefit the training process of a new concept by using different levels of diversity on ensemble learning, which is a measure that minimizes the error (i.e., discordance) among classifiers. So, DDD keeps ensembles with different diversity levels to deal with drifts, taking old concept information to assist the learning (MINKU; YAO, 2012).

Farid et al. (FARID et al., 2013) proposed an adaptive Ensemble Model (EM) able to detect new class. EM uses automatic decision trees with clustering, which classify an instance by majority weighted voting. Some instances are labeled to train a new classifier, and when it becomes competitive, it can replace an older one with the smallest weight (which means it has the minimum classification accuracy rate). To detect new class, the assumption is that instances should be closer to each other if they belong to the same class, otherwise should be distant from instances of other classes. If an instance is distant from the present clusters, it is considered as a new class example (FARID et al., 2013).

Harel et al. (HAREL et al., 2014) proposed a method that analyzes the empirical loss distribution, whose statistics are acquired by reusing the data multiple times via resampling. The method employs random permutations to detect drift, creating multiple train-test data from the stream. If no drift occurred, the ordered data prediction should not quite differ from the shuffled data when the algorithm achieves stability. Receiving time indices when concept drift, the method uses it to modify the windows size and initiate a training phase, and also providing information to ensemble learners. Instances suppose to have temporal independence, but according to authors, it can apply tactics to maintain exchangeability (HAREL et al., 2014).

Raza et al. (RAZA; PRASAD; YUHUA, 2014) proposed the ALCSD (Adaptive Learning with Covariate Shift-Detection), an adaptive algorithm that employs dataset shift-detection with exponential-weighted moving average (EWMA) model. An EWMA model-based shift-detection test supervises the covariate shift and initiates adaptation with the shift-detection point: it retrains classifiers with the updated knowledge base and different adaptation methods (RAZA; PRASAD; YUHUA, 2014).

Lu et al. (LU; ZHANG; LU, 2014) proposed a method to detect drift in case-based reasoning (CBR) environment. They presented a competence model to detect drifts (using two sliding windows, it compares the data distribution through competence instead of the feature space). Competence measures the success of CBR system to accomplish its goals. Prior knowledge of case distribution is not necessary. It presents secure statistical of the drift detected and also maintains records and drift quantifications (LU; ZHANG; LU, 2014).

Wang and Abraham (WANG; ABRAHAM, 2015) proposed the LFR (Linear Four Rates) framework, a drift detector that can deal with batch and stream approaches, binary class problems and imbalanced data. LFR requires user-specific parameters but does not depend on the underlying statistical-model, and the drift detector is not dependent on the classifier (WANG; ABRAHAM, 2015).

Khamassi et al. (KHAMASSI et al., 2015) proposed the EDIST2, a drift detector with self-adaptive windowing to deal with different types of drift. EDIST2 supervises performance and detects drift with two sliding windows: a global and a current one. The global window (GW) increases in stable environments and shrinks if a drift is detected. The current window (CW) comprises only the current batch instances. EDIST2 computes the error distance distribution of GW and CW, and compares the difference between their error distance averages. It has three thresholds: In-Control level – assumes that there are no changes, therefore CW's instances are added to GW; Warning level – starts storing instances in an auxiliary window for a potential change; and Drift level – drift is detected and only instances kept from Warning level remains in GW. Drift threshold is automatically calculated with statistical proofs (KHAMASSI et al., 2015).

Gao et al. (GAO et al., 2007) argued that descriptive model similar to posterior probability is preferable for real-stream classification. They proposed the UCB (Uncorrelated Bagging), a framework to deal with imbalanced data that apply sampling and ensemble methods to skewed stream mining problem. It makes a balanced training data by maintaining all minority instances and under sampled-majority instances (minority class is assumed as stationary) (DITZLER; POLIKAR, 2013). The averaged probability calculated on application data by several models are the final outputs (GAO et al., 2007).

Yalcin et al. (YALCIN; ERDEM; GURGEN, 2007) employed Support Vector Machines in an ensemble-based incremental learning algorithm. In previous works, they integrate SVM and an ensemble framework with Learn++ to create an incremental learning algorithm (SVMLearn++). A forgetting approach is employed on SVMLearn++ to eliminate the effects of redundant data, and in this work SVMLearn++ was assessed in the following learning approaches: Learn++ without pruning, with top N highest performance classifiers, and with replacing the looser device. The authors use SVM with RBF kernel in two-class problems (YALCIN; ERDEM; GURGEN, 2007).

Chen and He (CHEN; HE, 2009) proposed the SERA (SElectively Recursive Approach) algorithm to handle imbalanced data. It selectively retains minority instances in the current batch and it assigns the sampling probabilities proportionally to the majority and minority instances to increase minority instances performance. The amount of minority instances is limited by the authors to be proportional to the size of majority data, and a Mahalanobis distance decides the priority order of acceptance. They also elaborated a biased bagging approach (BBagging) to boost the performance of a single classifier focusing on minority instances on imbalanced datasets (CHEN; HE, 2009).

Elwell and Polikar (ELWELL; POLIKAR, 2011) presented the Learn++.NSE algorithm, an incremental ensemble algorithm depicted by NonStationary Environments (NSEs). Learn++.NSE receives the batches and incrementally learn from them without requesting access to previous data and handles concept drift with a passive approach. One classifier is trained at each batch and the results are obtained by combining the ensemble dynamically-weighted-majority vote based on their time-adjusted errors. Learn++.NSE can handle the appearance of a new class and the deletion of an old one (ELWELL; POLIKAR, 2011).

Ditzler et al. (DITZLER; POLIKAR; CHAWLA, 2010) proposed the Learn++.SMOTE, a hybrid algorithm containing Learn++.NSE and SMOTE approaches to deal with class imbalance. This hybrid algorithm can boost the recall of the minority class, and any supervised learning algorithm can be used as base classifier (DITZLER; POLIKAR; CHAWLA, 2010).

Ditzler and Polikar (DITZLER; POLIKAR, 2010) proposed a method based on Learn++.NSE algorithm: the Learn++.NIE (Nonstationary and Imbalanced Environments). Differences in NSE and NIE are i) for each batch NIE generates a sub-ensemble (rather than an individual classifier), and ii) another metric (not a classification error) is employed as an evaluation measure. Compared to Learn++.NSE, Learn++.NIE has slow recovery but can boost minority class performance. It can separately weight the average error of the majority and minority class recall and can also reward classifiers with good performance not only in the majority class but both minority and majority classes (DITZLER; POLIKAR, 2010).

In (DITZLER; POLIKAR, 2013), Ditzler and Polikar proposed two incremental ensemble algorithm to deal with imbalanced data. The first method (Learn++.CDS) is a logical union of Learn++.NSE algorithm and the Synthetic Minority class Oversampling TEchnique (SMOTE) to learn in imbalanced environments. The second method is Learn++.NIE. Learn++.CDS employs SMOTE to readjust the class balance with synthetic minority class examples, then uses Learn++.NSE on this balanced data. Both Learn++.CDS and Learn++.NIE reveal to perform better on fixed ensemble size (DITZLER; POLIKAR, 2013).

Klinkenberg (KLINKENBERG, 2001) employed Support Vector Machines to handle concept drift. Exploiting the work in (KLINKENBERG; JOACHIMS, 2000), they reduce the need for labeled data using unlabeled instances in a transductive way. It discards irrelevant data minimizing the generalization error with the use of automatically adjusted size windows. The method employs some properties of SVMs, adapting $\varepsilon\alpha$-estimates (a particular process to assess SVM performance based on the idea of leave-one-out estimation) to select the window size (KLINKENBERG, 2001).

Widyantoro (WIDYANTORO, 2007) proposed a framework that employs unlabeled data on information-filtering domains. A concept hierarchy is incrementally built in an unsupervised way to be used on classification. To deduce concepts is employed a persistence assumption in temporal reasoning. The method permits classifier to be tailored to target applications. The performance depends on the method for identifying classes, the method to build the hierarchy, and the chosen classifier (WIDYANTORO, 2007).

Spinosa et al. (SPINOSA et al., 2008) proposed the OLINDDA (OnLIne Novelty and Drift Detection Algorithm), a cluster-based algorithm that can detect new concepts and assessed in intrusion detection domain. Initially, it creates a normal profile of a single class, and to identify a new concept it uses cohesive sets of clusters, merging similar concepts during learning. OLINDDA stores information in three hypersphere-based models, which is about normal profile, extended concepts of normal profile and new concepts. The normal model is static and used as a reference. The extension and new concepts are continuously created and updated. It has both supervised and unsupervised learning, where the first occurs building the normal model and the latter treating unlabeled instances and detecting novel concepts (SPINOSA et al., 2008).

Masud et al. (MASUD et al., 2011b) proposed the ECSMiner (Enhanced Classifier for data Streams with novel class Miner), a method to detect the appearance of a new class. To handle drift, it uses M classifiers in an ensemble and the majority vote, which is continuously updated: when a new model is trained, it can replace an older one with the highest error. Each classifier has a class detector and if all of them declare a novel class, a new class is determined. To do such thing, it verifies cohesion and separation, and if an instance is isolated from training examples, it is recognized as an $Foutlier$. If an instance is not an $Foutlier$, ECSMiner uses ensemble voting to classify it. If a number of cohesive $Foutlier$ is reached, a new class is determined (MASUD et al., 2011b).

Masud et al. (MASUD et al., 2011a) proposed the SCANR (Stream Classifier And Novel and Recurring class detector), a method to deal with recurring class and to detect new ones in multi-class problems. It maintains a primary and auxiliary ensembles to store old classification models. After building a new primary model, it substitutes the model in primary ensemble with the worst prediction error. The primary ensemble analyses instances to detect outliers and starts classification if the instance is not an outlier. If it is classified as an outlier (primary outlier), the auxiliary ensemble verifies it again. If it is not classified as an outlier, it is assumed to be a recurring class and classified by the auxiliary ensemble. Otherwise, it is named as secondary outlier, and it is provisionally added into a buffer. The new class detection module starts if buffer instances reach a certain number (MASUD et al., 2011a).

Li et al. (LI; WU; HU, 2012) proposed the SUN method, a Semi-supervised classification algorithm for data streams with concept drifts and UNlabeled data. It creates a decision tree generating concept clusters at leaves with k-Modes. It detects drifts from noise with a bottom-up search and the deviation among history concepts and new ones. The concept clusters classify instances, and the labeled examples are used again trying to reduce the drift rate (LI; WU; HU, 2012).

Katakis et al. (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008b) proposed the CCP (Conceptual Clustering and Prediction) framework, a probabilistic representation model for stream learning employing incremental cluster algorithms. It maps batches into "Conceptual Vectors" (CV) containing conceptual information, and those vectors geometrically close do always belong to the same conceptual theme. Clustering works in the stream of CV, summarizing batches into concepts. For each batch, CCP assigns the concept (cluster) and employs the specific classifier. One advantage of CPP is having to store only the clusters' centers and the classifiers for every cluster, with no need to store old batches or CV (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008b).

Sethi and Kantardzic (SETHI; KANTARDZIC, 2017) proposed the MD3 (Margin Density Drift Detection), a drift detector algorithm for unlabeled stream of data. The number of instances in the area of uncertainty of the classifier (margin) is used as a metric to detect drift. If a variation in margin density occurs, the classifier needs labeled instances to be retrained.

Tennant et al. (TENNANT et al., 2017) proposed the MC-NN (Micro-Cluster Nearest Neighbour) method that calculates statistics from the stream and employs nearest neighbour algorithm, which can characterize MC-NN in parallel. The serial MC-NN data has no need to be in memory and is incrementally processed. A statistical summary is built as a set of variance based Micro-Clusters (MC). MC handles concept drift through statistics update of new instances. The parallel MC-NN distribute MC to computational nodes in a computer cluster. Each node of parallel version adapts to drift likewise serial version with a voting mechanism to classify instances (TENNANT et al., 2017).

Liu et al. (LIU et al., 2017) proposed a drift detector in sensor network domain based on angle optimized global embedding (AOGE) and principal component analysis (PCA). The drift detector decreases time processing due to the compatibility between the detector and data processing and also improves performance through dimension reduction (with PCA). PCA and AOGE examine the projection variance and angle, respectively. Combined, they are used to identify changes on objective function. The authors used Extreme Learning Machine (ELM) and SVM as classifiers (LIU et al., 2017).

Silva et al. (SILVA; HRUSCHKA; GAMA, 2017) proposed the FEAC-Stream (Fast Evolutionary Algorithm for Clustering data Streams) algorithm, which uses k-means clustering with k automatically estimated from the stream. FEAC-Stream uses the Page-Hinkley Test to identify decreases in clusters quality to start the re-estimation of k with an evolutionary algorithm. It considers that partially unknown data can afford valid stream knowledge (SILVA; HRUSCHKA; GAMA, 2017).

Xu and Wang (XU; WANG, 2017) proposed the DELM (Dynamic Extreme Learning Machine) to classify online data stream employing ELM as classifier. With the use of thresholds to detect drift, it employs a double hidden layer structure to train and improve the performance: when an alert of drift is issued, additional hidden layer nodes are included on neural network; once the drift is detected, a new classifier replaces an older classifier with low performance (XU; WANG, 2017).

Arabmakki and Kantardzic (ARABMAKKI; KANTARDZIC, 2017) proposed the RLS-SOM (Reduced labeled Samples-Self Organizing Map) framework for imbalanced stream. An ensemble classifies with DWM and retrains a new model using partial labeled samples when a drift is detected. The method uses the global answer and each one of individual answers: if an individual model has higher performance than the ensemble's performance, it is chosen instead of the others. After drift detection, the method selects majority and minority class instances to train a new model. If it is a conditional drift, SVM is employed to choose the closest instances of decision boundary (margin). However, if no minority instance is found on decision boundary, SOM algorithm maps the batch to search for minority instances in the whole feature space (ARABMAKKI; KANTARDZIC, 2017).

Zhang et al. (ZHANG et al., 2017) proposed a three-layered drift detection technique in text data streams domain, where each layer denotes, respectively: the layer of label space, the layer of feature space, and the layer of the mapping relationships between labels and features. According to authors, it can employ any classifier and can measure changes in each layer to detect different types of drift (ZHANG et al., 2017).

The methods are generally evaluated with metrics as accuracy, precision, recall, and error.

The metrics less used include time, little detection delay, detection delay, false alarm, prequential error, the total number of changes detected, prequential accuracy, Monte Carlo error, predictive accuracy, prediction errors, Gmean, among others.

Table 2.1 categorizes the techniques according to the drift detector mechanism, and Table 2.2 presents the classifiers employed in the articles considered in this work. Table 2.3 depicts the type of concept drift handling used: instance selection (i.e., a window of fixed size, window with automatically adjustment size, etc), instance weighting, ensemble learning, clustering or sampling. Table 2.4 categorizes the techniques according to the learning approach, i.e., supervised, unsupervised or semi-supervised.

Table 2.1: Drift detector mechanism.

Drift detector	(SCHLIMMER; GRANGER, 1986; WIDMER; KUBAT, 1996; BIFET; GAVALDÀ, 2007; ABDULSALAM; SKILLICORN; MARTIN, 2011; HEGEDUS; ORMÁNDI; JELASITY, 2013; LI et al., 2015; DÍAZ et al., 2015; MIRZA; LIN; LIU, 2015; KLINKENBERG; RENZ, 1998; GAMA et al., 2004; YANG; WU; ZHU, 2005; GARCÍA et al., 2006; SILVA et al., 2006; SUSNJAK; BARCZAK; HAWICK, 2012; MINKU; YAO, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2010; HAREL et al., 2014; RAZA; PRASAD; YUHUA, 2014; LU; ZHANG; LU, 2014; WANG; ABRAHAM, 2015; KHAMASSI et al., 2015; LI; WU; HU, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2008b; SETHI; KANTARDZIC, 2017; LIU et al., 2017; SILVA; HRUSCHKA; GAMA, 2017; XU; WANG, 2017; ARABMAKKI; KANTARDZIC, 2017; ZHANG et al., 2017)
No drift detector	(SALGANICOFF, 1993; KUBAT; WIDMER, 1995; KLINKENBERG; JOACHIMS, 2000; STREET; KIM, 2001; STANLEY, 2003; SCHOLZ; KLINKENBERG, 2005; KURLEJ; WOZNIAK, 2012; VIVEKANANDAN; NEDUNCHEZHIAN, 2011; SUN; LI, 2011; HEGEDUS; ORMÁNDI; JELASITY, 2013; BERTINI; NICOLETTI; ZHAO, 2013; ESCOVEDO et al., 2013; LOEFFEL; MARSALA; DETYNIECKI, 2015; CHEN et al., 2015; ZAREMOODI; BEIGY; SIAHROUDI, 2015; ZAREMOODI; SIAHROUDI; BEIGY, 2016; KOLTER; MALOOF, 2007; FARID et al., 2013; GAO et al., 2007; YALCIN; ERDEM; GURGEN, 2007; CHEN; HE, 2009; DITZLER; POLIKAR, 2010; DITZLER; POLIKAR; CHAWLA, 2010; DITZLER; POLIKAR, 2013; ELWELL; POLIKAR, 2011; KLINKENBERG, 2001; WIDYANTORO, 2007; SPINOSA et al., 2008; MASUD et al., 2011b, 2011a; TENNANT et al., 2017)

2.3.1 Datasets used in the literature

In this subsection, we summarize some real-world and synthetic datasets used in the literature to test and simulate concept drift environments. Synthetic datasets are very significant as we can affirm that concept drift really exist and specify which type of change is (i.e., gradual or abrupt).

Table 2.2: Classifiers.

SVM	(KLINKENBERG; JOACHIMS, 2000; SCHOLZ; KLINKENBERG, 2005; ZAREMOODI; SIAHROUDI; BEIGY, 2016; KLINKENBERG; RENZ, 1998; RAZA; PRASAD; YUHUA, 2014; WANG; ABRAHAM, 2015; YALCIN; ERDEM; GURGEN, 2007; ELWELL; POLIKAR, 2011; SETHI; KANTARDZIC, 2017; LIU et al., 2017; ARABMAKKI; KANTARDZIC, 2017; ZHANG et al., 2017; KLINKENBERG, 2001)
Decision trees	(STREET; KIM, 2001; STANLEY, 2003; SCHOLZ; KLINKENBERG, 2005; ABDULSALAM; SKILLICORN; MARTIN, 2011; DÍAZ et al., 2015; LI et al., 2015; MINKU; YAO, 2012; FARID et al., 2013; GAO et al., 2007; DITZLER; POLIKAR, 2010; MASUD et al., 2011b; LI; WU; HU, 2012; SETHI; KANTARDZIC, 2017; ARABMAKKI; KANTARDZIC, 2017; ZHANG et al., 2017; KLINKENBERG; RENZ, 1998; GAMA et al., 2004; GARCÍA et al., 2006; YANG; WU; ZHU, 2005; KHAMASSI et al., 2015)
Naive Bayes	(BIFET; GAVALDÀ, 2007; LI et al., 2015; KLINKENBERG; RENZ, 1998; KOLTER; MALOOF, 2007; MINKU; YAO, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2010; FARID et al., 2013; RAZA; PRASAD; YUHUA, 2014; GAO et al., 2007; ELWELL; POLIKAR, 2011; KATAKIS; TSOUMAKAS; VLAHAVAS, 2008b; ZHANG et al., 2017)
k-means clusterer	(BIFET; GAVALDÀ, 2007; SPINOSA et al., 2008; MASUD et al., 2011a; SILVA; HRUSCHKA; GAMA, 2017; ZAREMOODI; BEIGY; SIAHROUDI, 2015)
knn	(KLINKENBERG; RENZ, 1998; HAREL et al., 2014; RAZA; PRASAD; YUHUA, 2014; LU; ZHANG; LU, 2014; DITZLER; POLIKAR; CHAWLA, 2010; MASUD et al., 2011b)
Logistic Regression	(HEGEDUS; ORMÁNDI; JELASITY, 2013; GAO et al., 2007)
Neural Networks	(ESCOVEDO et al., 2013; CHEN; HE, 2009; DITZLER; POLIKAR, 2010; MINKU; YAO, 2012; GAMA et al., 2004; TENNANT et al., 2017; DITZLER; POLIKAR; CHAWLA, 2010)
Regression tree	(DITZLER; POLIKAR, 2013; ELWELL; POLIKAR, 2011)
ELM	(LIU et al., 2017; XU; WANG, 2017; MIRZA; LIN; LIU, 2015)
Others	(SCHLIMMER; GRANGER, 1986; SALGANICOFF, 1993; KUBAT; WIDMER, 1995; WIDMER; KUBAT, 1996; KURLEJ; WOZNIAK, 2012; VIVEKANANDAN; NEDUNCHEZHIAN, 2011; SUN; LI, 2011; BERTINI; NICOLETTI; ZHAO, 2013; LOEFFEL; MARSALA; DETYNIECKI, 2015; CHEN et al., 2015; KLINKENBERG; RENZ, 1998; GARCÍA et al., 2006; SILVA et al., 2006; KOLTER; MALOOF, 2007; SUSNJAK; BARCZAK; HAWICK, 2012; WIDYANTORO, 2007; ARABMAKKI; KANTARDZIC, 2017)

2.3.1.1 Synthetic Datasets

Below, some synthetic datasets used in literature:

- SINE1: abrupt concept drift, noise-free instances. Two relevant attributes, each one with a uniformly distributed value in $[0, 1]$. In the first concept, it classifies as positive if a

Table 2.3: Concept drift handling.

Instance selection	(KUBAT; WIDMER, 1995; WIDMER; KUBAT, 1996; KLINKENBERG; JOACHIMS, 2000; STANLEY, 2003; BIFET; GAVALDÀ, 2007; KURLEJ; WOZNIAK, 2012; VIVEKANANDAN; NEDUNCHEZHIAN, 2011; SUN; LI, 2011; HEGEDUS; ORMÁNDI; JELASITY, 2013; LOEFFEL; MARSALA; DETYNIECKI, 2015; CHEN et al., 2015; ZAREMOODI; SIAHROUDI; BEIGY, 2016; KLINKENBERG; RENZ, 1998; GAMA et al., 2004; YANG; WU; ZHU, 2005; GARCÍA et al., 2006; SILVA et al., 2006; HAREL et al., 2014; RAZA; PRASAD; YUHUA, 2014; LU; ZHANG; LU, 2014; WANG; ABRAHAM, 2015; KHAMASSI et al., 2015; KLINKENBERG, 2001; SETHI; KANTARDZIC, 2017; TENNANT et al., 2017; LIU et al., 2017; XU; WANG, 2017)
Instance weighting	(SCHLIMMER; GRANGER, 1986; SALGANICOFF, 1993)
Ensemble learning	(STREET; KIM, 2001; STANLEY, 2003; SCHOLZ; KLINKENBERG, 2005; ABDULSALAM; SKILLICORN; MARTIN, 2011; BERTINI; NICOLETTI; ZHAO, 2013; ESCOVEDO et al., 2013; LI et al., 2015; ZAREMOODI; BEIGY; SIAHROUDI, 2015; DÍAZ et al., 2015; MIRZA; LIN; LIU, 2015; KOLTER; MALOOF, 2007; SUSNJAK; BARCZAK; HAWICK, 2012; MINKU; YAO, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2010; FARID et al., 2013; GAO et al., 2007; YALCIN; ERDEM; GURGEN, 2007; DITZLER; POLIKAR, 2010; DITZLER; POLIKAR; CHAWLA, 2010; DITZLER; POLIKAR, 2013; ELWELL; POLIKAR, 2011; MASUD et al., 2011b, 2011a; ARABMAKKI; KANTARDZIC, 2017; ZHANG et al., 2017)
Clustering	(WIDYANTORO, 2007; SPINOSA et al., 2008; LI; WU; HU, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2008b; SILVA; HRUSCHKA; GAMA, 2017)
Sampling	(GAO et al., 2007; CHEN; HE, 2009)

value stands below the curve $y = sin(x)$; is classified as negative otherwise. After drift, the classification is reversed (GAMA et al., 2004).

- SINE2: has two relevant attributes like SINE1. Classification function is $y < 0.5 + 0.3 \times sin(3\pi x)$. After drift, the concept is inverted (GAMA et al., 2004).
- SINIRREL1: has classification function as SINE1, the instances, however, present two irrelevant attributes (GAMA et al., 2004).
- SINIRREL2: has classification function as SINE2, but like SINIRREL1, instances present two irrelevant attributes (GAMA et al., 2004).
- CIRCLES: presents gradual drift and instances without noise. It has four classification function determined by four circles (four concepts). Instances are classified according to its location: if it is inside the circle defined by circular function, it is classified as positive; otherwise is negative. The gradual change occurs by modifying the center and radius size

Table 2.4: Learning approaches.

Supervised	(SCHLIMMER; GRANGER, 1986; SALGANICOFF, 1993; KUBAT; WIDMER, 1995; WIDMER; KUBAT, 1996; KLINKENBERG; JOACHIMS, 2000; STREET; KIM, 2001; STANLEY, 2003; SCHOLZ; KLINKENBERG, 2005; BIFET; GAVALDÀ, 2007; ABDULSALAM; SKILLICORN; MARTIN, 2011; KURLEJ; WOZNIAK, 2012; VIVEKANANDAN; NEDUNCHEZHIAN, 2011; SUN; LI, 2011; HEGEDUS; ORMÁNDI; JELASITY, 2013; BERTINI; NICOLETTI; ZHAO, 2013; ESCOVEDO et al., 2013; LI et al., 2015; LOEFFEL; MARSALA; DETYNIECKI, 2015; CHEN et al., 2015; ZAREMOODI; BEIGY; SIAHROUDI, 2015; DÍAZ et al., 2015; MIRZA; LIN; LIU, 2015; ZAREMOODI; SIAHROUDI; BEIGY, 2016; KLINKENBERG; RENZ, 1998; GAMA et al., 2004; YANG; WU; ZHU, 2005; GARCÍA et al., 2006; SILVA et al., 2006; KOLTER; MALOOF, 2007; SUSNJAK; BARCZAK; HAWICK, 2012; MINKU; YAO, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2010; FARID et al., 2013; HAREL et al., 2014; RAZA; PRASAD; YUHUA, 2014; LU; ZHANG; LU, 2014; WANG; ABRAHAM, 2015; KHAMASSI et al., 2015; GAO et al., 2007; YALCIN; ERDEM; GURGEN, 2007; CHEN; HE, 2009; DITZLER; POLIKAR, 2010; DITZLER; POLIKAR; CHAWLA, 2010; DITZLER; POLIKAR, 2013; ELWELL; POLIKAR, 2011; TENNANT et al., 2017; LIU et al., 2017; XU; WANG, 2017; ARABMAKKI; KANTARDZIC, 2017; ZHANG et al., 2017)
Unsupervised	(SETHI; KANTARDZIC, 2017; SILVA; HRUSCHKA; GAMA, 2017)
Semisupervised	(KLINKENBERG, 2001; WIDYANTORO, 2007; SPINOSA et al., 2008; MASUD et al., 2011b, 2011a; LI; WU; HU, 2012; KATAKIS; TSOUMAKAS; VLAHAVAS, 2008b)

of the circle (GAMA et al., 2004).

- GAUSS: has abrupt drift and instances with noise. Domain $R \times R$ with two relevant attributes. Positive instances are located as a normal distribution with center $[0,0]$ and standard deviation as of 1. Negative instances have center $[2,0]$ and standard deviation as of 4. After drift, the classification is reversed (GAMA et al., 2004).

- SINE1G: presents very slow gradual drift and instances without noise. Similar to Sine1, but the gradual drift is reached by selecting instances from the past and the current concept (transition time among concepts). To select an instance from the past concept and the new one has, respectively, gradually lower probability and gradually higher probability as time passes (GARCÍA et al., 2006).

- STAGGER: has abrupt drift and instances without noise. Instances have three symbolic attributes – *size* (*small*, *medium*, *large*), *color* (*red*, *green*), and *shape* (*circular*, *non-circular*). In the first concept, instances are positive if $size = small \wedge color = red$. In second concept the instances are defined by $color = green \vee shape = circular$. In third concept the instances are defined by $size = medium \vee size = large$ (GAMA et al., 2004).

- MIXED: has abrupt drift and instances without noise. Instances have four attributes: two boolean v, w and two numeric between $[0,1]$. If an instance has the following two of three conditions satisfied, it is classified as positive: $v, w, y < 0.5 + 0.3 \times sin(3\pi x)$. After drift, classification is reversed (GAMA et al., 2004).

- Rotating Hyperplane Dataset: designed by (FAN, 2004). The (k,t) pairs details of each concept and all files are available in the Internet [1] (WANG; ABRAHAM, 2015).

- Usenet1 and Usenet2: used in (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008a), are available in the Internet [2]. They collected reports from several newsgroups (e.g., medicine, space, baseball) of a user. The distinction among datasets is the drift dimension: the Usenet1 dataset has a sharper topic drift (WANG; ABRAHAM, 2015).

- Usenet: text dataset inspired by Katakis et al. (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010), available in the Internet [3]. Usenet simulates a news filtering from 20 Newsgroups with change of interest of a user (concept drift). There are six topics, the user is interested in two, but is subscribed in four. It also simulates recurring concepts repeating topics of interest (three concepts in training data and three recurring concepts in testing data). The dataset has 5,931 instances and 659 attributes, which are binary values (presence or absence of the respective word).

- SEA Concepts Dataset: proposed by Street and Kim (STREET; KIM, 2001), this dataset has 60,000 instances, 3 attributes, and 3 classes; with 10% of noise. The numeric attributes are between 0 and 10 with two relevant attributes. Instances are divided into groups of 15,000 into four concepts. Each concept has different thresholds values (8, 9, 7, and 9.5), and the concept function to determine 0 to a class instance is $relevant\_feature1 + relevant\_feature2 > Threshold$. Dataset is available in the Internet [3], and is quite used by concept drift handling algorithms (WANG; ABRAHAM, 2015).

2.3.1.2 Real-world Datasets

In regard to the real-world datasets, UCI machine learning repository (LICHMAN, 2013) are also cited in concept drift literature:

- KDD Cup 1999: the Knowledge Discovery and Data mining 1999 (KDD99) competition data contains simulated invasions in a military network domain. The complete dataset

[1] `http://www.win.tue.nl/~mpechen/data/DriftSets/`
[2] `http://mlkd.csd.auth.gr/concept_drift.html`
[3] `http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift`

has 5,000,000 instances, and the dataset available in the Internet [3] contains only 10% of the size. The original dataset has 24 attack types, and to simplify it into a binary-class problem, it changes the labels of attack types to abnormal and normal. To detect intrusion, it has to differentiate between attacks and normal connections. It includes a wide type of intrusions, so the attack is not a minority class. The dataset has 494,020 instances, where each one stands for a connection with 41 attributes (i.e., connection length, protocol type, network service on the destination, etc). This dataset is tested in many concept drift handling algorithm (MINKU; YAO, 2012).

- NSL-KDD Database: it is a KDD99 dataset version that solves issues of the anterior one: it does not include redundant and duplicate instances. Each instance has 41 attributes, and the 25,192 instances are distributed into 23 classes, consisting of normal classes (13,449 instances) and intrusion classes, which can be types of: Denial of service - DoS (9,234), Remote to user - R2U (209), User to root - U2R (11) or Probing (2,289) (FARID et al., 2013).

- Large Soybean Database: there are 683 instances and 19 classes, each instance consisting of 35 attributes, some nominal and some ordered (LICHMAN, 2013) (some authors uses all attributes nominalized, e.g., employing string rather numerical values (FARID et al., 2013)).

- Image Segmentation Database: this dataset emphasizes image segmentation and boundary detection domain. There are 2,310 instances with 19 attributes and 7 classes (outdoor images) including brickface, sky, foliage, etc (FARID et al., 2013). It creates a classification for every pixel with image handsegmention, and each instance is a 3x3 region (LICHMAN, 2013).

- Adult: extracted from U.S. Census Bureau with the aim to predict if a person achieves an amount of around $50,000 per year by using 14 demographic features (i.e., age, level of education, marital status, occupation, gender). This dataset has 44,848 people and 29.3% of them belongs to "over 50k" class (STREET; KIM, 2001).

- SEER Breast Cancer: used in (STREET; KIM, 2001), it contains 44,000 breast cancer patients accompaniment from the Surveillance, Epidemiology, and End Results (SEER) program of the National Institutes of Health. They consider that patients of class 1 died of breast cancer in between five years of surgery and patients of class 2 have lived at least five years. The resulting dataset has 37,715 instances, 25.7% of it classified as class 1 (STREET; KIM, 2001).

- Covertype: comprises types of cover forest from US Forest Service. It has 581,000 instances, 54 attributes, and 7 classes. The authors in (MASUD et al., 2011b) and (ZAREMOODI; BEIGY; SIAHROUDI, 2015) normalized the data to have two or three classes in each batch, with the appearance of new random classes in some of them.

- Poker: each instance represents a hand, which is five cards from a deck of 52. An instance has 10 attributes, in which each card having two attributes (suit and rank). "Poker Hand" is a class attribute. The order of cards matters and it provides 480 Royal Flush hands in contrast to 4 (one for each suit) (LICHMAN, 2013).

Another real-world dataset frequently tested in literature is Electricity Market Dataset (ELEC2), first described by Harries (HARRIES, 1999). The goal of this dataset is to recognize if the electricity price will increase or decrease (KOLTER; MALOOF, 2007). Data was collected from TransGrid, an Australian New South Wales Electricity Market, in which the demand and supply of products affect its prices. Harries (HARRIES, 1999) presents the seasonality and the sensitivity of the price and short-term events (like weather variations), respectively. Electricity market was extended to nearby areas: the excess production of one area can be sold in the adjacent one, which can dampener the extreme prices. The ELEC2 dataset comprises 45,312 instances from 7 May 1996 to 5 December 1998. Each instance assigns to a 30 minutes duration, and has 5 attributes: the weekday (an integer between $[1,7]$); the time stamp (a day period, a number between $[1,48]$); the New South Wales electricity demand (numeric attribute); the Victoria electricity demand (numeric attribute); the programmed electricity transfer between states (numeric attribute) and the class label (a binary value between up or down that recognizes price changes of the last 24 hours). The dataset attractive is the real-world data characteristics: not knowing if there is a drift and when it occurs (GARCÍA et al., 2006).

Real-world datasets less used can be referred as well:

- Calendar Apprentice (CAP): dataset used to predict user preferences for scheduling appointment in an academic institution (MITCHELL et al., 1994). The users preference to be predicted is the local of the appointment, duration, starting time, and weekday. An instance has 34 features – such as the type and scope of the meeting, kind of participants, and if it happens during lunchtime – with combinations of these features. There are 12 features for places, 11 for duration, 15 for start time, and 16 for the day of week (KOLTER; MALOOF, 2007).

- PAKDD 2009: consist of data from private label credit card operation on stable inflation condition of a major Brazilian retail chain. It has 50,000 instances of a one-year period, in which each instance represents a client by the use of 27 attributes, such as sex, age, marital status, profession, income, etc. Class identifies if the client is a “good” or a “bad” one, being the last a minority class composed by 19.5% of the data (MINKU; YAO, 2012).

The next two high dimensional datasets were from e-mail filtering domain. The former depicts sudden drift and recurring contexts, and the latter depicts gradual drift. Both datasets are accessible in Weka (ARFF) format in Boolean bag-of-words vector representation at `http://mlkd.csd.auth.gr/conceptdrift.html` (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010):

- Emailing List (elist) Dataset: consists of e-mail messages simulating some topics, labeled as interesting or junk depending on the user interest: the goal is to train and classify messages with user feedback. It collects messages from usenet posts of 20 Newsgroup collection (KLINKENBERG; JOACHIMS, 2000). The selected topics are: science/medicine, science/space, and recreation/sports/baseball. The dataset contains 1,500 instances and 913 attributes with words found at least 10 times on the message (boolean bag-of-words). 300 instances are assigned in five time periods: In the first period, medicine is the interesting topic, and the topic of interest at the end of each period changes to simulate concept drift (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010).
- Spam Filtering Dataset: consists of e-mail messages from the Spam Assassin Collection. It has four segments: spam, spam2, ham (legitimate), and easy ham, which is quickly identified legitimate messages. Spam ratio of the original set is nearly 20%, and to transform it into a longitudinal data, the email sent date and time is extracted and converted into the format *yyyyMMddhhmmss* (*yyyy*: year, *MM*: month, *dd*: day, *hh*: hours, *mm*: minutes, *ss*: seconds). It maintains all copies, even if the user has more than one of the same e-mail, but the attachments are removed. It employs the boolean bag-of-words approach to represent e-mails. The dataset contains 9,324 instances and 500 attributes (words acquired by employing feature selection with the X^2 measure). This dataset has gradual concept drift (KATAKIS; TSOUMAKAS; VLAHAVAS, 2010). In the Internet [3] is available a spam dataset consisting of 9,324 instances and 40,000 attributes with gradual drift and binary class (legitimate and spam) with nearly 20% spam ratio.

2.4 Discussion

In this section, a summarization concerning all works are presented and further discussed. Figure 2.5 shows the percentage of each learning method used in the articles separated by supervised, unsupervised or semi-supervised learning. Clearly, one can observe that supervised learning is by far the widest methodology employed in the context of concept drift. Such numbers can be explained by the preference for using supervised classifiers in most articles, and by the advantage of detecting changes in data distribution when one has the class information of a sample.

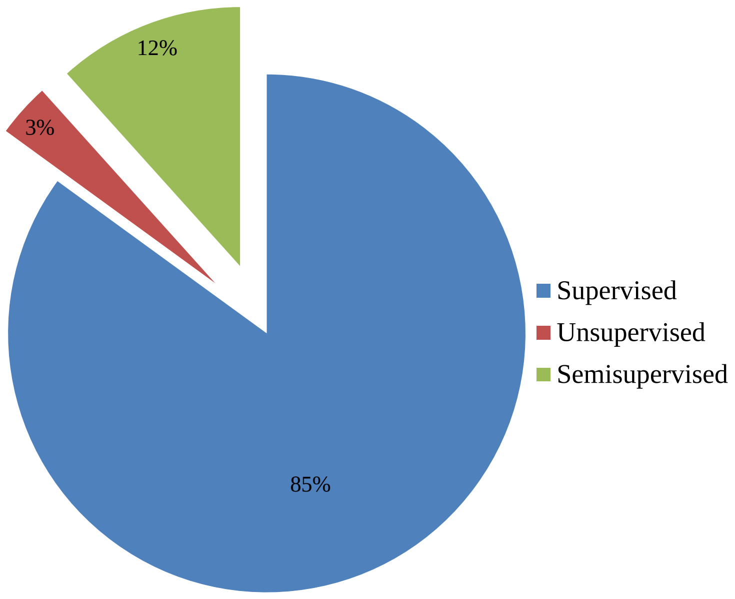

Figure 2.5: Percentage concerning different learning approaches used in concept drift.

Figure 2.6 shows the percentage of drift detection mechanism usage, i.e, whether the method is an active or a passive approach. If it is an active approach, it has a drift detection method that informs whether and when a drift occurs. If it is a passive approach, it does not have a drift detection mechanism and the algorithm assumes the drift may occur at any time and updates the model independently. The use of a drift detection mechanism requires a metric to determine that there is a drift, which can influence the performance. Therefore, the lack of adoption of a drift detection mechanism can influence the training time, i.e., the method has to determine when to update its model, which can be costly to do regularly. It is not clear to assume which is the best one since these methods were implemented in almost equivalent quantities, being the passive approach a little more used.

Figure 2.7 shows the percentage of classifiers the methods can handle, i.e., whether the method is specific for one classifier or it can be used with any other classifier. Notice the methods use a specific classifier to handle concept drift mostly, which means the techniques are usually designed with a specific technique in mind, and in some cases taking advantage of the characteristics of the classifier on the method.

Figure 2.8 shows the percentage of classifiers used in the articles, being the most used

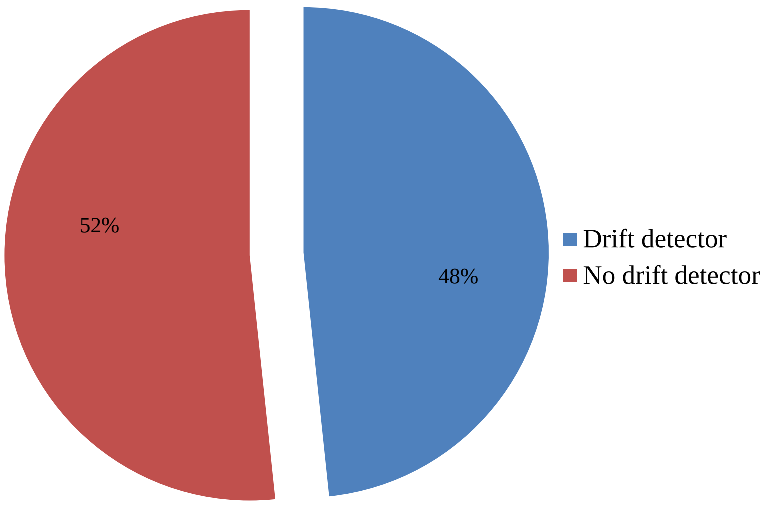

Figure 2.6: Percentage of drift detector mechanisms.

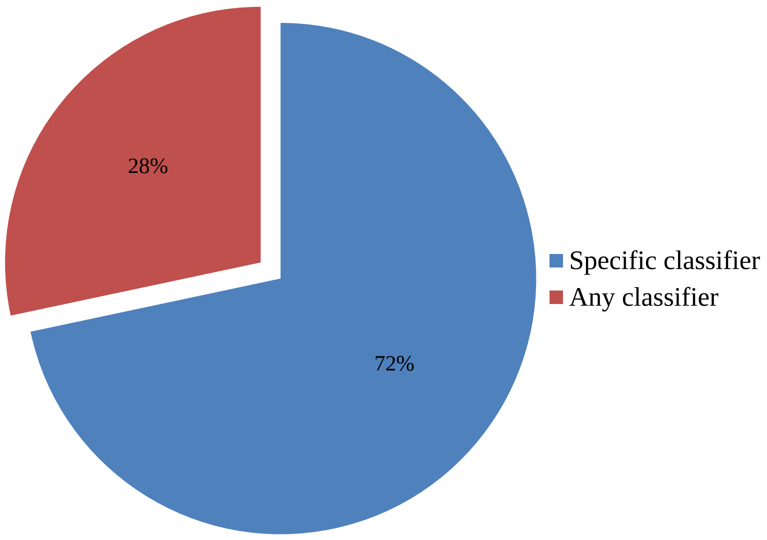

Figure 2.7: Percentage of specific or any-classifier approach that can be used.

one the decision trees (DTs), followed by SVM and naïve Bayes. DTs are commonly used in ensemble learning due to their efficiency, thus turning out to be pretty much suitable for handling data streams efficiently. Since SVM and naïve Bayes are very popular classifiers in the community, it is expected they have been employed more regularly.

Figure 2.9 shows the percentage of types of concept drift handling used in the articles. The most used approaches are the instance selection and ensemble learning. Instance selection can be easier to implement, i.e., it selects instances within a fixed or dynamic sliding window considering recent samples that are more significant. Ensemble updating is easier to perform as well since it maintains a dynamic set of classifiers that are updated according to some criteria.

Figure 2.10 shows the percentage of methods using online learning (which evolves and updates a model as instances are processed) and batch learning (which learns by examining a

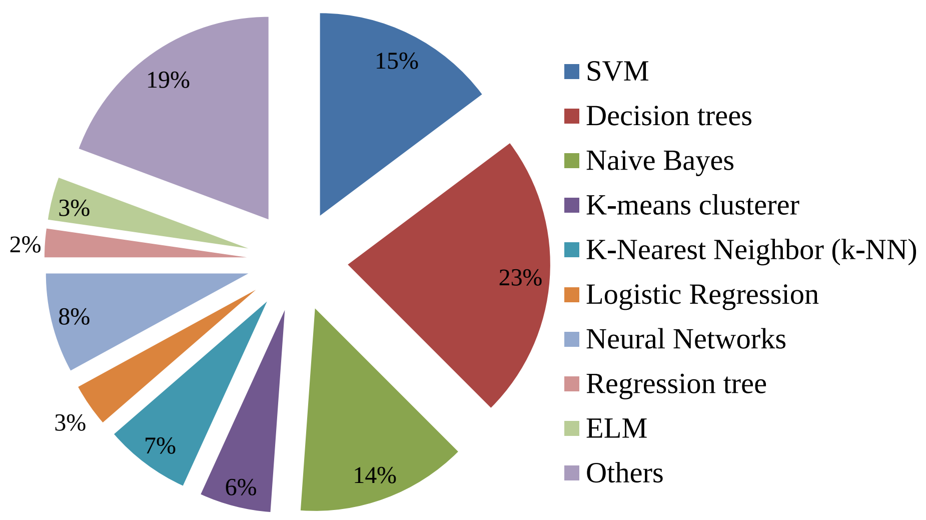

Figure 2.8: Percentage of the classifiers used in the works considered in this survey.

collection of instances at once), being the batch learning approaches employed a little more in the evaluated methods. The approach selection depends on the classifier and the method implementation, among other details.

Figure 2.11 shows the percentage of other issues addressed in articles in addition to concept drift handling, being the most addressed drawbacks the imbalanced data (when the class distribution is imbalanced, i.e., having minority and majority classes), followed by new concept handling (or concept evolution – emergence of a new concept in the environment mainly in unsupervised learning), and reoccurring concept (when old concepts may re-appear in the future).

Figure 2.12 shows the percentage of binary classifiers (when the problem has only two classes, i.e., relevant or irrelevant class, positive or negative class, among others) and multiclass classifiers used in the articles, being both nearly equally used (multiclass classifiers are a little more used). Figure 2.13 shows the percentage of each dataset used in the articles, being the most used dataset the "KDD Cup 1999"; followed by "STAGGER," "Electricity," "Hyperplane" and "SEA" datasets; and then "Gauss" and "Forest Cover type" datasets.

Figure 2.14 shows the percentage of methods compared in the articles, being the most used methods the Window-of-fixed-size, DDM, Learn++ family, and methods that do not handle concept drift; followed by Full-memory methods, EDDM, CVFDT, and DWM. The Window-of-fixed-size is widely used due to its implementation simplicity for new classifiers in nonstationary environments; the DDM method is a popular drift detector method for active approaches; and the Learn++ family has different techniques to deal with concept drift.

Finally, Figure 2.15 shows the percentage of articles published by year. Notice that 2015

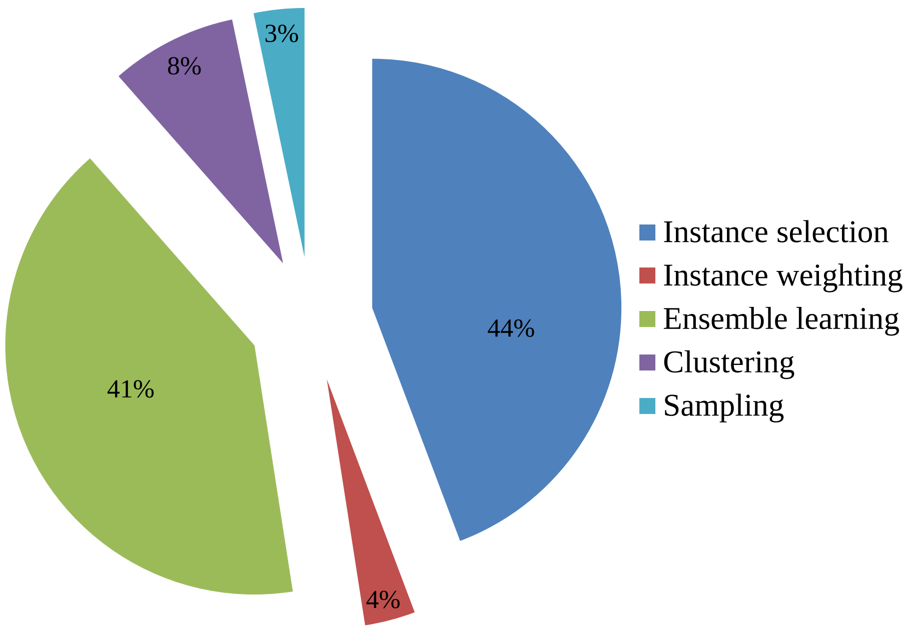

Figure 2.9: Concept drift handling.

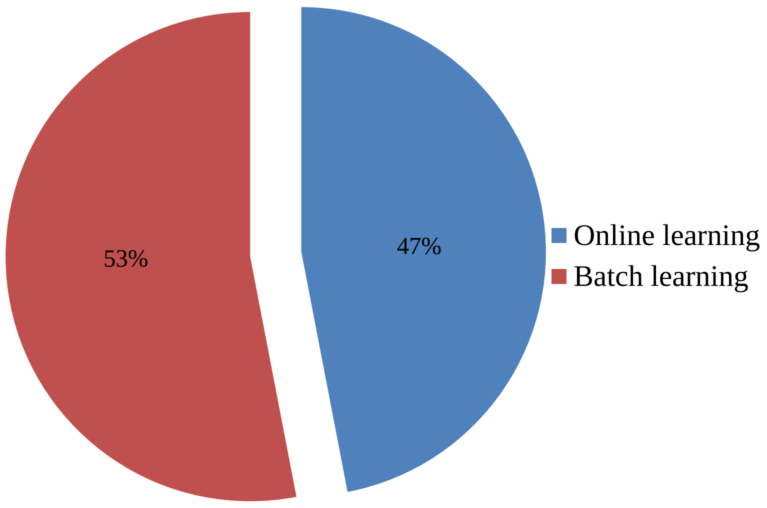

Figure 2.10: Online or Batch approaches.

had more articles published, followed by the year of 2017. However, one can observe that such area of research has been focused even more yearly, thus showing the increasing interest by the scientific community. Some observations to make is that the analyzed articles were collected until the year 2017, the review was written in 2018 and published between the end of that year and the beginning of 2019.

2.5 Conclusions

In this work, we presented the concept drift problem, classifying the variants of target concept in different forms. We also named some types of algorithms to deal with concept drift,

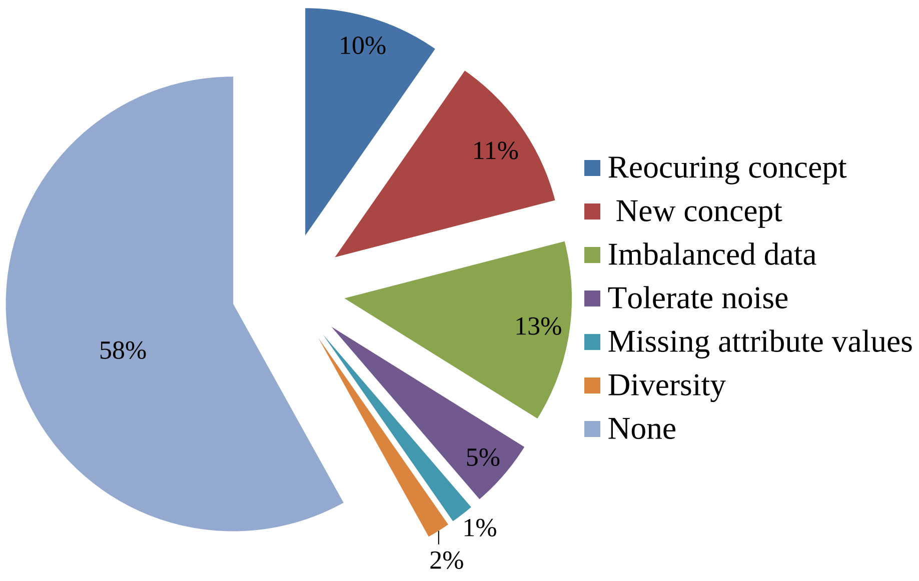

Figure 2.11: Other issues addressed in the articles.

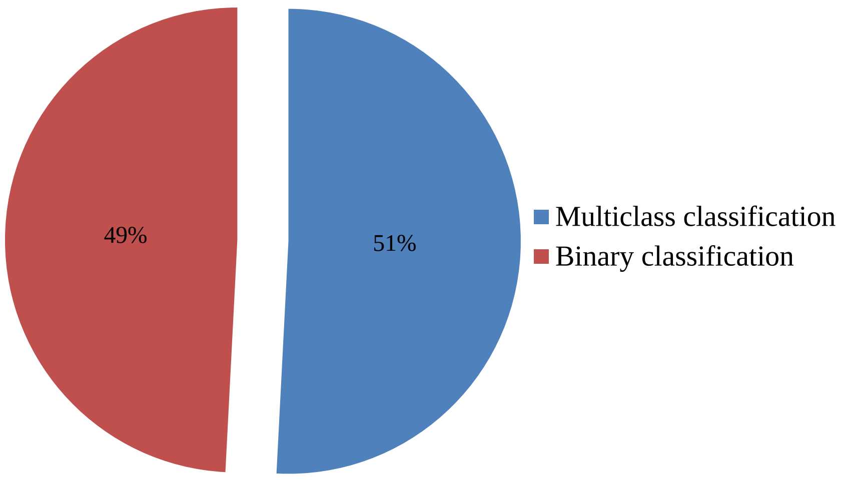

Figure 2.12: Number of classes recognized by the methods.

including instance selection or window-based approaches, weight-based approaches, and ensemble of classifiers. The window-based approaches can be full-memory, no-memory, window-of-fixed-size, or window-of-adapting-size, depending on the treatment of the batches. In (DITZLER; POLIKAR, 2013), they characterized concept drift algorithms in others ways such as online vs. batch approaches; single classifier vs. ensemble-based approaches; incremental vs. non-incremental approaches; and active vs. passive approaches.

We also summarized some techniques available in the literature that detect or deal with concept drift, like Learn++ family, DDM, DWM; in addition to some real-world and synthetics datasets used in the literature to test and simulate concept drift environment like Electricity, KDD Cup 1999, STAGGER, Hyperplane and SEA datasets. Finally, we summarized in percentage charts the articles considered in this overview.

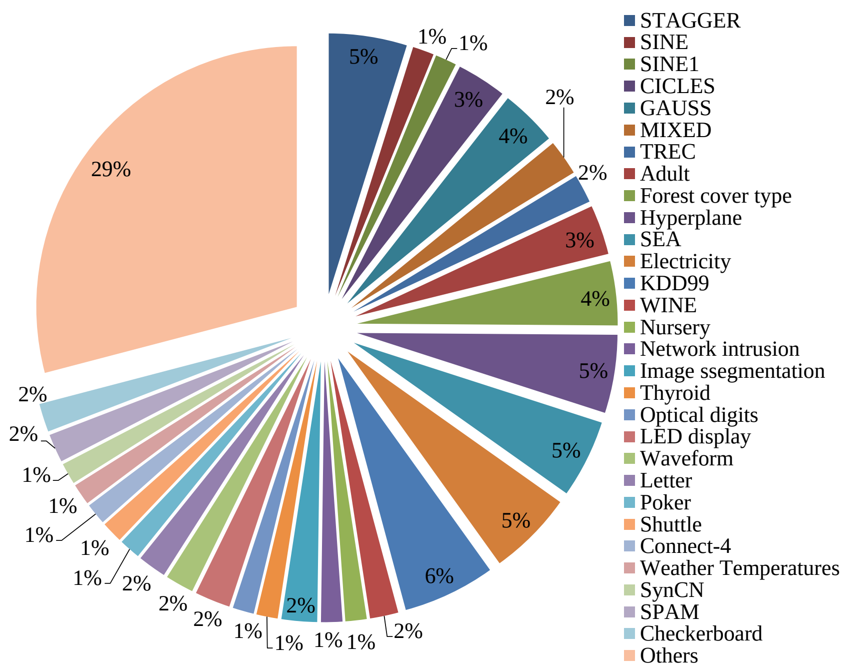

Figure 2.13: Datasets used in the articles.

Future directions concerning the area may be related to other issues in addition to concept drift handling like imbalanced data, new concept handling and reoccurring concept, as well as more studies in unsupervised environments.

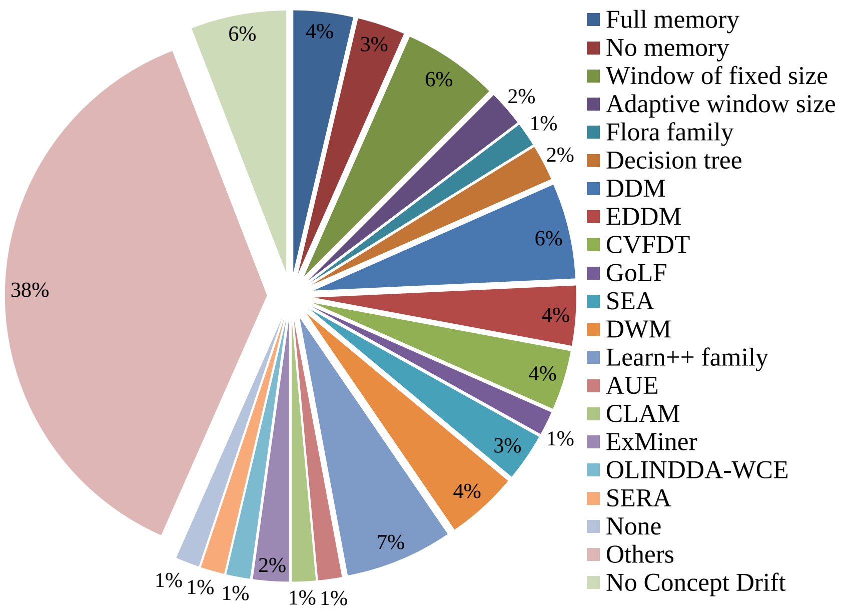

Figure 2.14: Methods compared in the articles.

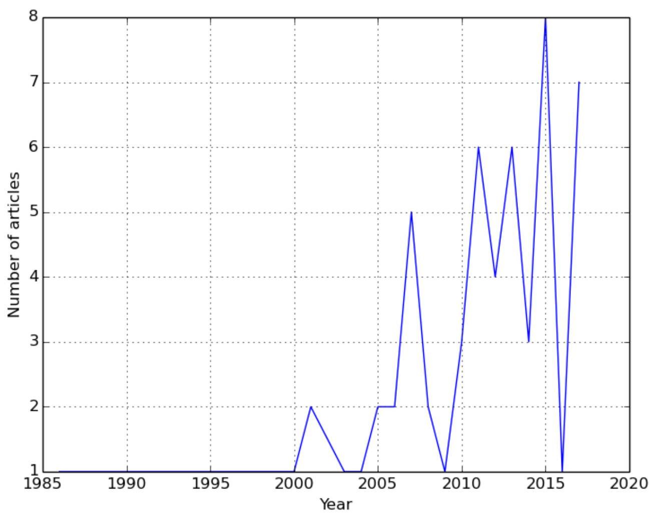

Figure 2.15: Articles separated by years.

Chapter 3

OPTIMUM PATH FOREST

In this chapter, we present the theoretical background regarding the OPF classifier, which comprises two different learning procedures: a supervised and an unsupervised one.

3.1 Supervised Classification

The supervised OPF has two approaches: one that uses a complete graph (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012) and another that uses a k-nn graph (PAPA; FALCÃO, 2008, 2009; PAPA; FERNANDES; FALCÃO, 2017), being the former the most used and widespread. We first present the supervised OPF with complete graph and further the supervised OPF with k-nn graph.

3.1.1 Complete Graph

Regarding to OPF with complete graph, let $\mathscr{Z} = \mathscr{Z}_1 \cup \mathscr{Z}_2$ be a labeled dataset, where $\mathscr{Z}_1$ and $\mathscr{Z}_2$ stand for the training and test sets, respectively. Let $\lambda(\mathbf{s})$ be the function that associates the correct label to any sample $\mathbf{s} \in \mathscr{Z}_1 \cup \mathscr{Z}_2$, such that $\lambda(\mathbf{s}) \in \{1, 2, \ldots, c\}$ and $\mathbf{s} \in \Re^n$. Let $\mathscr{S} \in \mathscr{Z}_1$ be the set of prototype samples (i.e., the most important samples that better represent the classes), and $d(\mathbf{s}, \mathbf{t})$ the distance between two samples $\mathbf{s}$ and $\mathbf{t}$. The problem consists in using S, d and $\mathscr{Z}_1$ to project an optimal classifier that can predict the correct label $\lambda(\mathbf{s})$ of any sample $\mathbf{s} \in \mathscr{Z}_2$.

Roughly speaking, the OPF classifier models the problem of pattern recognition as a graph partition task, where each node is encoded by a dataset sample that is connected to others by means of a predefined adjacency relation. The partition process is ruled by a competition process among prototype samples, which try to conquer the remaining ones offering to them optimum-path costs. When a sample is conquered, it receives the label of its conqueror together with an updated cost. At the final of the process, we have a collection of optimum-path trees (OPTs), which are rooted at each prototype.

Let $(\mathscr{Z}_1, \mathscr{A})$ be a complete graph in which nodes are samples in $\mathscr{Z}_1$, and any pair of samples defines an edge in $\mathscr{A}$ (i.e., a complete graph). Also, the edges are weighted by the distance d among their corresponding nodes (Figure 3.1a). A path in $\mathscr{Z}_1$ is a sequence of samples $\pi = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle$, where $(\mathbf{s}_i, \mathbf{s}_{i+1}) \in A$ for $1 \leq i \leq k-1$. Additionally, a path is said to be trivial when $\pi = \langle \mathbf{s}_1 \rangle$.

A given path-cost function $f(\bullet)$ is associated to each path π, being denoted by $f(\pi)$, and a given path π' is said to be optimum if $f(\pi') \leq f(\tau)$ for any path τ, where π' and τ end at the same sample $\mathbf{s}$, regardless of their origin. We also denote $\pi \cdot (\mathbf{s}, \mathbf{t})$ the concatenation of the path π ending at $\mathbf{s}$ and the edge $(\mathbf{s}, \mathbf{t}) \in \mathscr{A}$.

The OPF algorithm can be used with any smooth path-cost function, which can combine samples with similar properties (FALCÃO; STOLFI; LOTUFO, 2004). Papa et al. (PAPA; FALCÃO; SUZUKI, 2009) opted to use the f_{max} cost function due to its theoretical properties to estimate optimum prototypes, which can be defined as follows:

$$\begin{aligned} f_{max}(\langle \mathbf{s} \rangle) &= \begin{cases} 0 & \text{if } \mathbf{s} \in \mathscr{S}, \\ +\infty & \text{otherwise} \end{cases} \\ f_{max}(\pi \cdot (\mathbf{s}, \mathbf{t})) &= \max\{f_{max}(\pi), d(\mathbf{s}, \mathbf{t})\}. \end{aligned} \tag{3.1}$$

Notice $f_{max}(\pi)$ computes the maximum distance among adjacent samples in π, when π is not a trivial path.

The OPF comprises a training and a classification step, being the former in charge of building the optimum-path forest over the training set using f_{max} and $\mathscr{S}$, and the classification phase simply computes the training sample that will conquer that specific testing node (details below). Papa et al. (PAPA; FALCÃO; SUZUKI, 2009) proposed to compose $\mathscr{S}$ with the nearest nodes from different classes in $\mathscr{Z}_1$, since such samples fall in the boundary of the classes, thus being informative enough to the learning process. In order to find them, one just needs to compute a minimum spanning tree in $\mathscr{Z}_1$ (Figure 3.1b), and then select the connected samples with different labels as the prototype nodes (Figure 3.1c).

Roughly speaking, the OPF training algorithm associates an optimal path $P^*(\mathbf{s})$ from $\mathscr{S}$ to all samples $\mathbf{s} \in \mathscr{Z}_1$, thus building an optimum path forest P (a function without cycles which associates to all $\mathbf{s} \in \mathscr{Z}_1$ its predecessor $P(\mathbf{s})$ in $P^*(\mathbf{s})$, or assigns *nil* when $\mathbf{s} \in \mathscr{S}$). Let $R(\mathbf{s}) \in \mathscr{S}$ be the root of $P^*(\mathbf{s})$ that can be achieved using $P(\mathbf{s})$. The OPF algorithm computes, for each $\mathbf{s} \in \mathscr{Z}_1$, the cost $C(\mathbf{s})$ of $P^*(\mathbf{s})$, the label $L(\mathbf{s}) = \lambda(R(\mathbf{s}))$, and its predecessor $P(\mathbf{s})$.

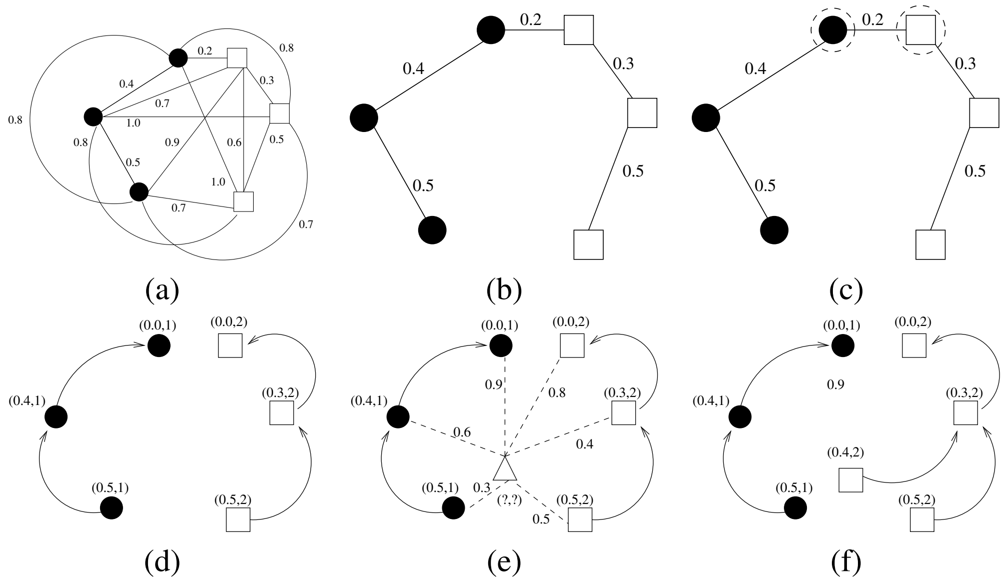

Figure 3.1: OPF training step: (a) Complete and weighted graph representing the training set and its (b) MST, (c) prototypes highlighted and (d) resulting optimum-path forest. OPF classification step: (e) a testing sample (triangle) is added into the graph and connected to all training samples, and (f) the testing sample is conquered by the sample that offered the lowest cost.

The classification step is straightforward, i.e., given a test sample $\mathbf{t} \in \mathscr{Z}_2$, we connect it to all training nodes (Figure 3.1e) of the optimum-path forest generated in the training phase (Figure 3.1d), and we evaluate which node $\mathbf{p} \in \mathscr{Z}_1$ minimizes the equation:

$$C(\mathbf{t}) = \min\{\max\{C(\mathbf{p}), d(\mathbf{p}, \mathbf{t})\}\}, \forall \mathbf{p} \in \mathscr{Z}_1. \tag{3.2}$$

Thus, the node $\mathbf{p} \in \mathscr{Z}_1$ that minimizes $C(\mathbf{t})$ will be the one that conquer $\mathbf{t}$ (Figure 3.1f).

3.1.2 K-nn Graph

The OPF with k-neighborhood (OPF$_{knn}$) differs in some points with respect to the OPF with complete graph variant (PAPA; FALCÃO; SUZUKI, 2009; PAPA; FALCÃO, 2009; PAPA; FERNANDES; FALCÃO, 2017). The adjacency relation (A_k), in this case, connects each sample to its k-nearest neighbors; the prototypes are now estimated as the nodes located at the highest density regions; and the path-cost function aims at maximizing the cost of every sample. Roughly speaking, OPF$_{knn}$ comprises two phases: training and classification. The former is responsible for computing the density of each training node using A_{k^*}, being the $k^* \in [1, k_{max}]$ the best value of

k (size of the neighbrohood) that maximizes some criterion, and then to start the competition process among prototypes.

In order to compute the probability density function (pdf) $\rho(s)$ of each node s, one can use the following formulation:

$$\rho(s) = \frac{1}{\sqrt{2\pi\sigma^2} \mid \mathscr{A}_k^*(s) \mid} \sum_{\forall t \in \mathscr{A}_k^*(s)} \exp\left(\frac{-d^2(s,t)}{2\sigma^2}\right), \tag{3.3}$$

where $\mid \mathscr{A}_k^*(s) \mid = k^*$, $\sigma = \frac{d_f}{3}$, and d_f is the maximum arc weight in training set.

The competition process among prototypes is conducted in order to conquer the training samples by offering to them optimum-paths according to a path-cost function f_{min}, given by:

$$\begin{aligned} f_{min}(\langle t \rangle) &= \begin{cases} \rho(t) & \text{if } t \in \mathscr{S}, \\ \rho(t) - 1 & \text{otherwise} \end{cases} \\ f_{min}(\pi_s \cdot \langle s,t \rangle) &= \min\{f_{min}(\pi_s), \rho(t)\}. \end{aligned} \tag{3.4}$$

where $\rho(t) - 1$ is employed to avoid plateaus nearby the maxima of the pdf.

The classification process picks up a sample from the test set, connects it to its k^*-nearest neighbors in the optimum-path forest generated by the training phase, and then uses the same OPF$_{knn}$ rule employed in the competition process to conquer that sample.

3.2 Unsupervised Classification

The unsupervised OPF was proposed by Rocha et al. (ROCHA; CAPPABIANCO; FALCÃO, 2009) as a data clustering method based on optimum-path forest. Let D be an unlabeled dataset consisting of samples from a given application. Each sample $s \in D$ is usually represented by a feature vector $v(s)$ and the distance between samples s and t is given by a function $d(s,t)$. The unsupervised OPF models these samples s as nodes from graph (D, A_k), being the arcs $(s,t) \in A$ a connection of k-nearest neighbors in the feature space. The nodes $s \in D$ are weighted by a probability density value $\rho(s)$ given by:

$$\rho(s) = \frac{1}{\sqrt{2\pi\sigma^2} \mid \mathscr{A}_k(s) \mid} \sum_{\forall t \in \mathscr{A}_k(s)} \exp\left(\frac{-d^2(s,t)}{2\sigma^2}\right), \tag{3.5}$$

where $\mid \mathscr{A}_k(s) \mid = k$, $\sigma = \frac{d_f}{3}$, and d_f is the maximum arc weight in (D, A_k).

The chosen of best value of k (k^*) within [k_{min}, k_{max}], $1 \leq k_{min} < k_{max} \leq \mid D \mid$, considers the minimum graph cut among all clustering results for $k \in$[1,k_{max}] ($k_{min} = 1$). The path π_t is a sequence of adjacent samples starting from a root $R(t)$ and ending at a sample t, being $\pi_t = \langle t \rangle$ a trivial path and $\pi_s \cdot \langle s,t \rangle$ the concatenation of π_s and arc (s,t). It assigns to each path π_t a value $f(\pi_t)$ given by a connectivity function f. A path π_t is considered optimum if $f(\pi_t) \geq f(\tau_t)$ for any other path τ_t and $f(\pi_t)$ is defined by:

$$\begin{aligned} f(\langle t \rangle) &= \begin{cases} \rho(t) & \text{if } t \in R, \\ \rho(t) - \delta & \text{otherwise} \end{cases} \\ f(\pi_s \cdot \langle s,t \rangle) &= \min\{f(\pi_s), \rho(t)\}. \end{aligned} \tag{3.6}$$

for $\delta = min_{\forall (s,t) \in A_k | \rho(t) \neq \rho(s)} \mid \rho(t) - \rho(s) \mid$ and R being the root set (one element per each maximum of the pdf discovered on-the-fly).

3.3 Accuracy Measure

In the works developed in this thesis, we considered the accuracy measure proposed by Papa et al. (PAPA; FALCÃO; SUZUKI, 2009) that takes into account unbalanced datasets.

The accuracy Acc is measured as:

$$Acc = \frac{2c - \sum_{i=1}^{c} E(i)}{2c} = 1 - \frac{\sum_{i=1}^{c} E(i)}{2c}, \tag{3.7}$$

being $N(i)$, $i = 1, 2, \ldots, c$, the number of samples in the dataset $\mathscr{X}$ from class i, and $E(i)$ the partial sum error of class i.

Chapter 4

Learning Concept Drift with Optimum-Path Forest

This chapter presents the work presented by Iwashita and Papa (IWASHITA; PAPA, 2018) that evaluates the supervised OPF in different approaches to deal with concept drift.

4.1 Introduction

Our proposal is to assess the OPF robustness in concept drift environments and verify how it behaves in such situation. The main contribution of this work is to evaluate two different OPF classifiers in this context under three different perspectives (i.e., no-memory, full-memory and windowed-based approach) in public datasets.

4.2 Experiments

We evaluated three different approaches to address the problem of concept drift with traditional OPF (i.e., with complete graph) and OPFknn, as described in Subsection 2.2.1: "full-memory," "no-memory" and "window-of-fixed-size three." The synthetic datasets used in this work are: Hyperplane dataset consisting of 10 features and 90,000 samples with a drift at every 10,000 samples; and Usenet 1 and Usenet 2 datasets consisting of 99 features and 1,500 samples with a drift at every 300 samples. As the databases are labeled, all experiments performed in this work and the following chapters are considered to have "zero latency". We chose synthetic public datasets which were quite used by the articles observed in our survey and other datasets with interesting subjects.

In order to evaluate the techniques, we divided each dataset into 30 batches (equally) aiming to simulate a stream of data. For the sake of explanation, consider the Hyperplane dataset that contains $90,000$ samples, which means each batch comprises $3,000$ samples. Since the concept drifts after every $10,000$ samples, we have a change in the stream behavior every 3.3 batches (highlighted regions in Figure 4.1). Additionally, naïve OPF (i.e., without support to handle concept drift) is compared against OPF-nomemory, OPF-fullmemory, and OPF-window3. The latter classifier works with a window of 3 batches.

Since OPFknn requires one additional step to set up k_{max} parameter, which limits the extent of the neighborhood size when computing the k-nearest neighbors, we employed the following methodology: for almost all datasets, we set k_{max} as being 10% of their sizes. The only exception concerns in Hyperplane full-memory version experiment, in which we used 0.1% of the dataset size. These differences in the k_{max} values are due to the size of the datasets, and for the experiment's time exceeding the expectation that we had to abort. Since the full-memory experiment keeps adding new samples for training, it becomes too costly in terms of computational burden for training purposes.

Although the idea is to assess the OPF robustness under concept drift, we also included naïve SVM with a Radial Basis Function kernel optimized through cross-validation for comparison purposes. The "traditional SVM" parameters were optimized as follows: we divided the first batch in 50% to compose the training set and the remaining 50% to compose the validating set (used to optimize kernel parameters). Also, we consider the following proportions for the training and validating sets: $60\% - 40\%$, $70\% - 30\%$, $80\% - 20\%$ and $90\% - 10\%$. Such procedure was performed for all datasets, and we used the parameters that maximized SVM accuracy over the validating set of the aforementioned configurations (i.e., training and validating set percentages). The "no-memory," "full-memory" and "window-3" SVM versions use a grid search parameter estimation methodology with $90\% - 10\%$ of training and validating sets, respectively. In regard to SVM implementation, we used scikit-learn (PEDREGOSA et al., 2011), and with respect to OPF, we employed LibOPF[1]. Note that the comparison may not be very fair because the libraries are in different languages (the first in Python and the other in C). Finally, we employed an accuracy measure proposed by Papa et al. (PAPA; FALCÃO; SUZUKI, 2009).

Figures 4.1 to 4.3 depict the accuracy rates of all aforementioned techniques considering the datasets employed in this work. Let us have a look at Figure 4.1, which stands for the results considering Hyperplane dataset. SVM has a higher accuracy than OPF-based classifiers. Also, we observe that OPFknn obtained better results than OPF with complete graph. The OPFKnn

[1]https://github.com/jppbsi/LibOPF

version is the most time-consuming one regarding training, being SVM variants slightly more time consuming training than OPF-based classifiers.

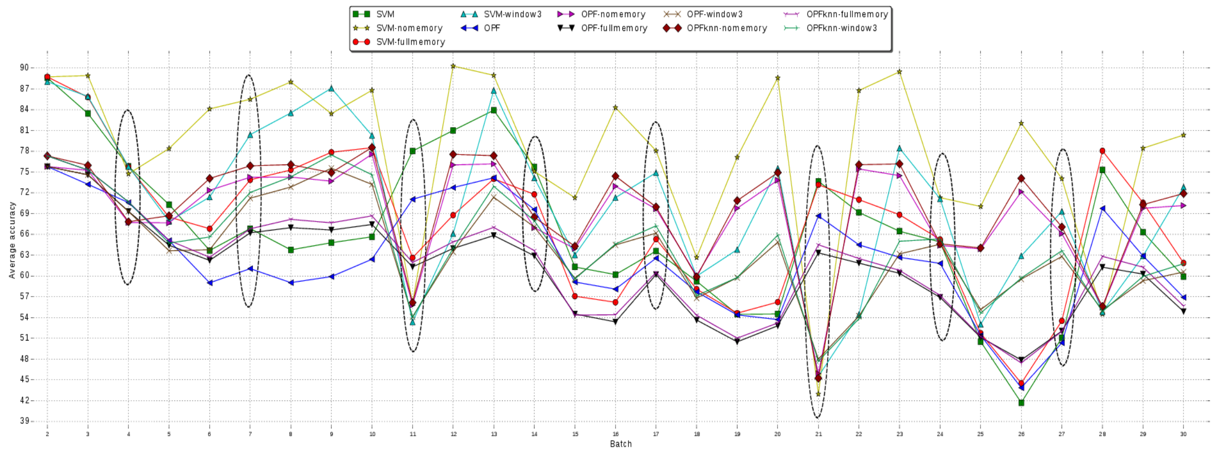

Figure 4.1: Accuracy over Hyperplane dataset stream. Some concept drifts are hilighted.

Figure 4.2 depicts the results over Usenet1 dataset stream. Traditional SVM did better in batches #1, #7 (same as traditional OPF) and #25. The concept drift-oriented SVM versions performed better in many batches, including batches #4 to #6, #10, #12, #15, #17, #20, #22 to #24 and #27 to #30. The SVM-window3 had the best performance among them, and traditional OPF obtained the best results in batches #7 and #13. The concept drift-oriented OPF versions achieved better results in batches #9, #16, #18 (same as OPFknn-window3), #19 and #26. After the last concept drift in batch #25, OPF-fullmemory obtained the worst accuracy until batch #29. It ends up that OPF-fullmemory is being penalized by a large training set with fuitless information. The concept drift-oriented OPFknn versions performed better in batches #3, #8, #11, #14, #18 (same OPF-window3) and #21. As a matter of fact, OPFknn showed to be way competitive when compared to standard OPF, being the only drawback the k_{max} parameter, which needs to be fine-tuned.

Figure 4.3 depicts the results over Usenet2 dataset stream. SVM performed well in batches #1, #8, #10, #23 and #27. The concept drift-oriented SVM versions performed better on batches #3, #4, #11, #12, #14, #16, #20, #26, #28 and #30. It had the same performance as OPF and OPFknn versions in batches #5, #6 and #17. OPF did not obtain good results, the OPF concept drift-oriented versions did much better results; besides stands out in batches like #6, #13, #15 and #22. On OPFknn versions, we have better accuracies in batches #7, #9, #18, #19, #21, #24, #25 and #29, and same performance in batches #5, #6 and #17. On batches #7, #13, #19 and #25 in which a concept drift occurs, the accuracy of all versions drops, but on batches #19 and #25 the OPFknn-fullmemory continues with similar performance.

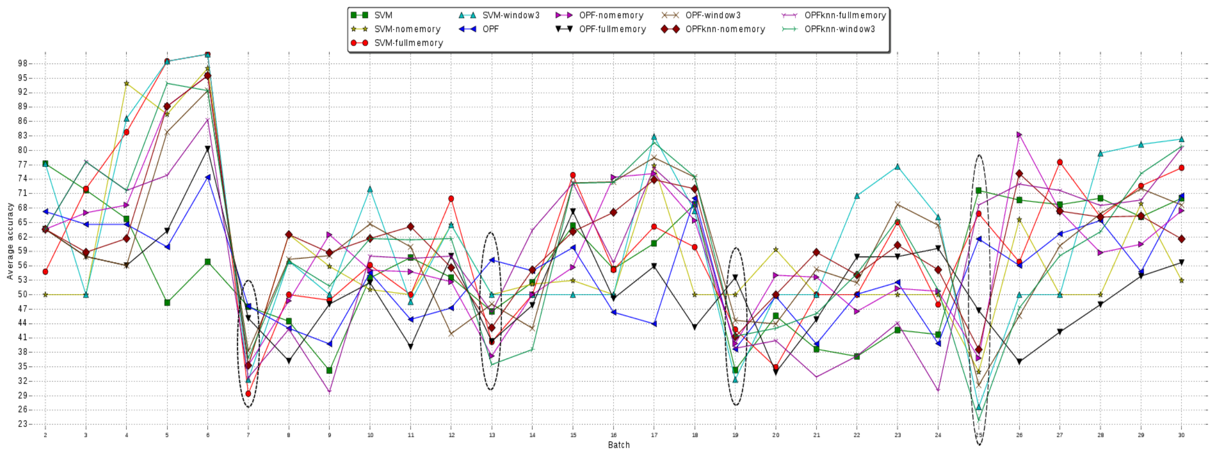

Figure 4.2: Accuracy over Usenet1 dataset stream. Some concept drifts are hilighted.

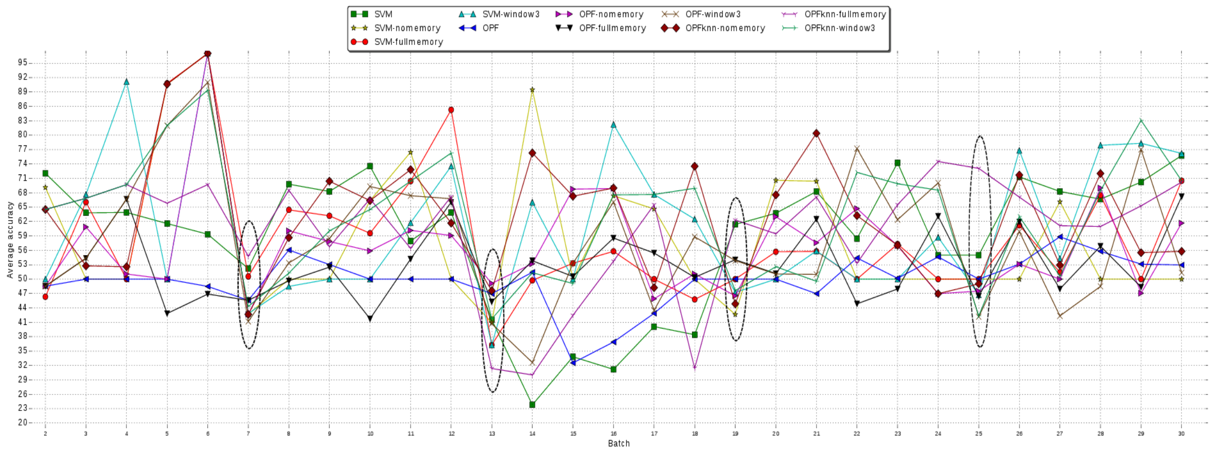

Figure 4.3: Accuracy over Usenet2 dataset stream. Some concept drifts are hilighted.

4.3 Partial Considerations

In this work, we dealt with the problem of concept drift concerning the OPF classifier. The experiments over synthetic datasets compared standard OPF, OPFknn, SVM, and three distinct versions to address concept drift on each technique. We have shown that OPF is suitable to work under these dynamic scenarios since its recognition rates were considerably better when adapted to address concept drift. Regarding to SVM versions, the OPF versions have lower time-consuming training time, and in some bases they also have a lower time-consuming testing time, which is a useful benefit in this type of environment.

OPF versions have been consistently more efficient than SVMs for training purposes. The main problem related to SVM still relies on the fine-tuning step, which is critical when one considers situations that require an on-line training. In this case, the classifier needs to be retrained every time a new batch comes to the game, and the response needs to be efficient as well, since we have a stream of data that needs to be processed on-the-fly.

Chapter 5

Learning Concept Drift with Ensembles of Optimum-Path Forest-based Classifiers

This chapter discusses the work presented by Iwashita et al. (IWASHITA; ALBUQUERQUE; PAPA, 2019), which proposes an ensemble-based OPF to deal with concept drift. This chapter also presents experiments concerning OPF dynamic-window approach to detecting drift based on accuracy values.

5.1 Introduction

The main contribution of this paper is to propose an ensemble-based approach composed of a committee of OPF classifiers to cope with the problem of concept drift handling. We considered three different perspectives with variations of streaming management and classifiers in public datasets, being the results compared to the ones obtained by standard OPF and OPF with simple concept drift handling (no-memory, full-memory, and windowed-based approach).

5.2 Proposed Approach

In this section, we evaluated three different ensemble-based approaches in three different scenarios to address the problem of concept drift with the data management described in Subsection 2.2.1: "full-memory" (OPF-fullmemory), "no-memory" (OPF-nomemory), and "window-of-fixed-size three" (OPF-window3). Also, the experiments used real-world and synthetics datasets, being the latter of great importance, since they shape an environment in which we know that concept drift really occurs, as well as which kind of change is occurring (i.e., gradual or abrupt). Although ensembles of OPF-based classifiers have been evaluated before (RIBEIRO;

PAPA; ROMERO, 2017; FERNANDES; PAPA, 2017; FERNANDES et al., 2017), they were not considered in the context of concept drift. Table 5.1 presents the main information concerning the synthetic datasets used in this work.

Table 5.1: Synthetic Datasets

Dataset	# of samples	drift time	# of features
Hyperplane	90,000	at every 10,000 samples	10
Usenet 1	1,500	at every 300 samples	99
Usenet 2	1,500	at every 300 samples	99
SEA	60,000	at every 15,000 samples	3

In regard to the real-world dataset, we used the following (Table 5.2):

Table 5.2: Real-world Datasets

Dataset	# of samples	# of features
Forest Covertype	581,012	54
Electricity	45,312	8
Poker Hand	829,201	10

We chose synthetic and real-world datasets which were quite used by the articles observed in our survey and other interesting datasets available on the internet. In order to evaluate the techniques, we divided each dataset into 30 batches (equally) aiming to simulate a stream of data. For the sake of explanation, consider the SEA dataset that contains $60,000$ samples, which means each batch comprises $2,000$ samples. Since the concept drifts after every $15,000$ samples (Table 5.1), we have a change in the stream behavior every 7.5 batches.

Therefore, the main idea of this work is to evaluate whether an ensemble version of OPF is robust enough to handle such situations or not. Additionally, naïve OPF (i.e., without support to handle concept drift) is compared against OPF-nomemory, OPF-fullmemory, and OPF-window3 (of 3 batches). With respect to the OPF implementation, we employed LibOPF[1]. Additionally, we considered the accuracy measure proposed by Papa et al. (PAPA; FALCÃO; SUZUKI, 2009).

The OPF ensemble employs three base classifiers and combines their results in three different ways (three voting mechanisms):

- Combined: the classification result is obtained by the most voted result among base classifier outputs, in which each learner has the same weight.

[1]https://github.com/jppbsi/LibOPF

- Weighted: each base classifier receives a different weight for its voting relevance based on its previous accuracy classification.
- Major: the base classifier having the highest accuracy in the previous batch receives an additional weight.

With the voting mechanisms described previously, we designed three rounds of experiments with variations of streaming managements, as described below:

- Experiment 1: it employs OPF-fullmemory, OPF-nomemory, and OPF-window3 as base classifiers (Figure 5.1), in which each approach handles the stream of batches as described in Subsection 2.2.1 with the OPF classifier. The results of each approach are further combined using the "combined," "weighted" and "major" approaches.

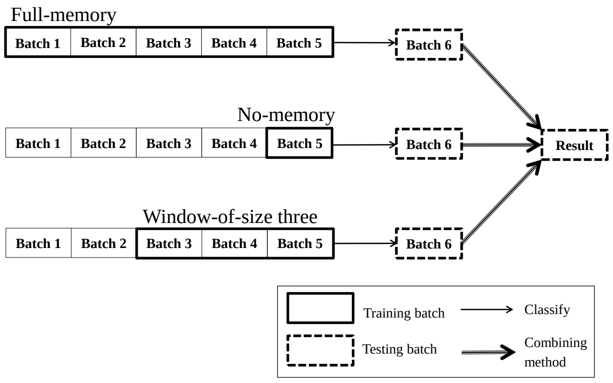

Figure 5.1: Pictorial example of the approach named as "Experiment 1."

- Experiment 2: it partitions the stream of batches into three different portions of training data for each base classifier, hereinafter called "fold1," "fold2," and "fold3," considering the data management described in Subsection 2.2.1. In regard to the full-memory (Figure 5.2) and no-memory (Figure 5.3) experiments, since one has one batch only, it is partitioned into three distinct subsets for further training one OPF classifier on each subset. With respect to the window-3 (Figure 5.4) approach, all three batches are merged into only one and further partitioned into three subsets. The results of fold1, fold2, and fold3 are further combined using the "combined," "weighted" and "major" approaches.
- Experiment 3: one classifier is trained for each batch. The ensemble considers the last three models for each new batch classification procedure, similarly to the window-3 management (Figure 5.5). The results are combined using the "combined" and "weighted" approaches.

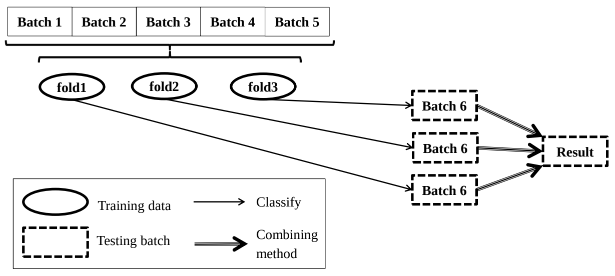

Figure 5.2: Experiment 2 with the full-memory approach.

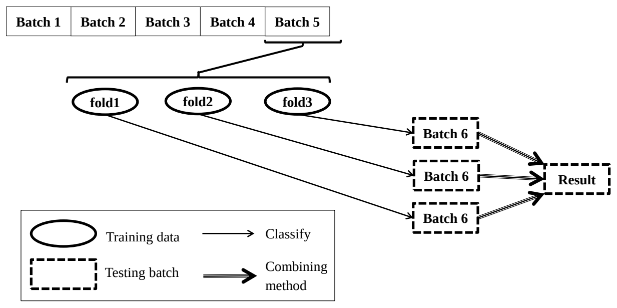

Figure 5.3: Experiment 2 with the no-memory approach.

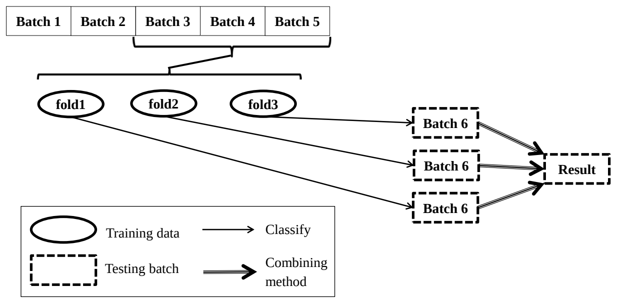

Figure 5.4: Experiment 2 with the window-of-fixed-size three.

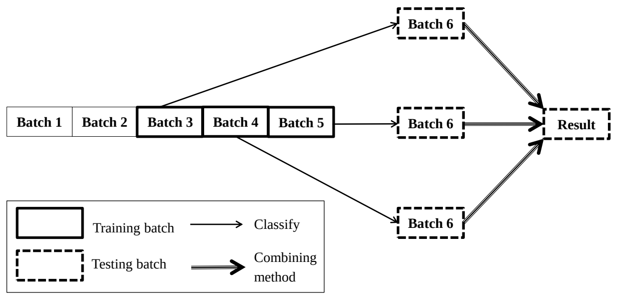

Figure 5.5: Experiment 3.

With respect to the weight values, we adopted the following methodology[2]: concerning Experiments 1 and 2, the classifiers are weighted according to their accuracies, i.e., the approach with the highest accuracy receives the weight as of 0.8, followed by the second best one with weight as of 0.5, and the last one with weight as of 0.3. In regard to Experiment 3, the most recent batch received the highest weight, i.e., suppose we are classifying the batch *i*,

[2]Notice that the weights were chosen empirically.

then the training batch $i-1$ has a weight as of 0.4, batch $i-2$ has a weight as of 0.35, and finally batch $i-3$ has a weight as of 0.25. The experiments were performed on a computer with Intel®Core(TM) i5-4690 processor, Z97M-PLUS/BR motherboard, NVIDIA Corporation GM107 [GeForce GTX 750] graphics processing unit, and Corsair DDR3 8GB 1600MHz RAM.

5.3 Experimental Results

In this section, we evaluated the three experimental approaches described in Section 5.2.

5.3.1 Experiment 1

As aforementioned, the idea of this experiment is to consider all outputs given by full memory, no-memory and window-of-size three approaches. Figure 5.6 depicts the results concerning Covertype dataset. Clearly, one can observe that standard OPF, OPF-fullmemory and the proposed OPF with majority voting ("Major") obtained the best results. Both OPF-fullmemory and OPF with majority voting were consistently similar to each other, which reflects the fact that OPF-fullmemory was also the best approach among the ones used to handle concept drift. Table 5.3 presents the mean accuracy and standard deviation concerning all batches.

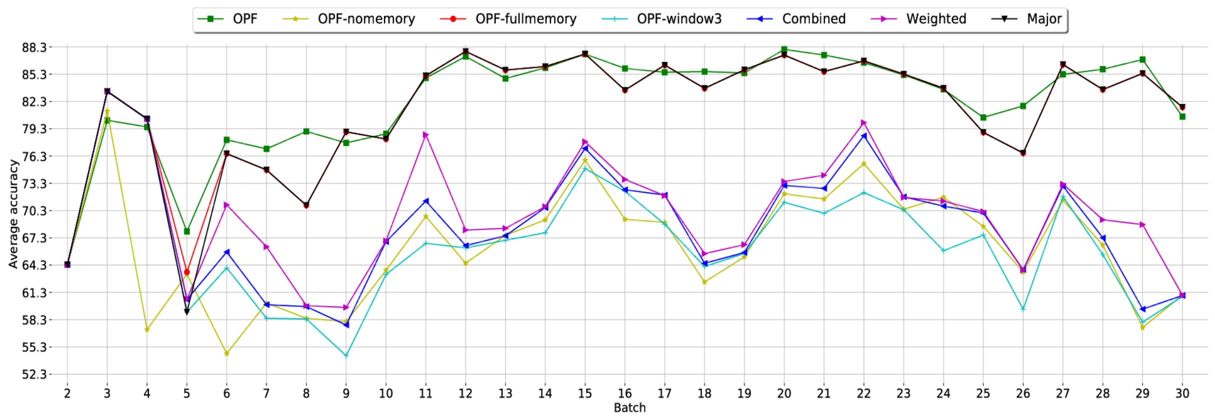

Figure 5.6: Experimental results concerning Covertype dataset with respect to Experiment 1.

Figure 5.7 depicts the results concerning the Electricity dataset. OPF with concept drift handling versions (OPF-fullmemory, OPF-nomemory, and OPF-window3) and the ensemble-based versions obtained higher accuracy when compared to traditional OPF. Only in batch #5, the traditional OPF has been more accurate than the OPF-nomemory. Table 5.4 presents the mean accuracy and standard deviation concerning all batches.

Table 5.3: Covertype dataset with respect to Experiment 1.

Type	Accuracy
OPF	82.32752648 ± 5.527916684
OPF-nomemory	66.340352 ± 6.205296248
OPF-fullmemory	81.51895655 ± 6.360053157
OPF-window3	66.63647852 ± 6.483513342
OPF-combined	68.55474907 ± 6.447446854
OPF-weighted	70.03032828 ± 6.163191174
OPF-major	81.3674181 ± 6.822249211

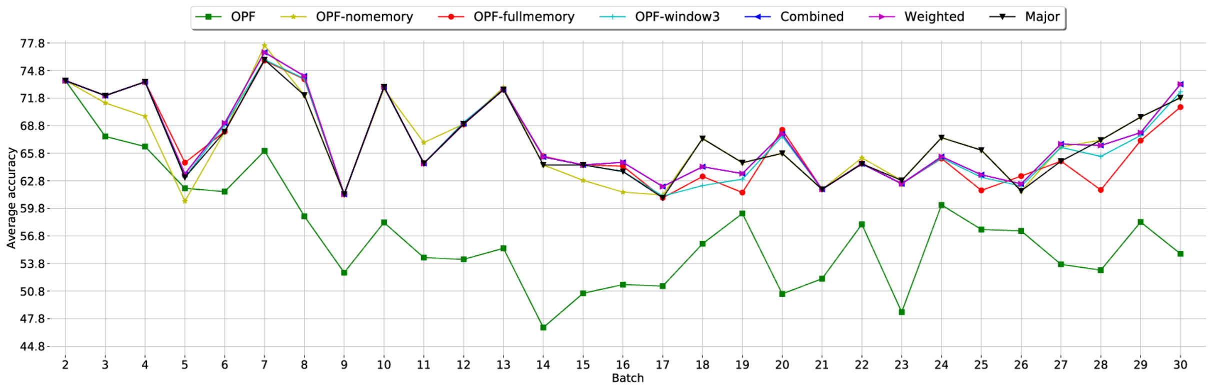

Figure 5.7: Experimental results concerning Electricity dataset with respect to Experiment 1.

The combined and weighted versions obtained a similar behavior, whereas the major version has a better performance in some batches. With respect to the previous dataset shown in Figure 5.6, one can now realize the importance of handling concept drift, since in that results standard OPF (i.e., with no concept drift handling) showed very much good results. The main idea in using such a dataset where standard OPF can outperform some approaches that can handle concept drift is just to show the proposed approach can obtain best results in any situation, i.e., where one does have or does not have a strong concept drift situation.

Table 5.4: Electricity dataset with respect to Experiment 1.

Type	Accuracy
OPF	56.94443552 ± 6.002938206
OPF-nomemory	67.06707179 ± 4.373578762
OPF-fullmemory	66.73389428 ± 4.523784782
OPF-window3	66.95818376 ± 4.505452085
OPF-combined	67.2960539 ± 4.434176779
OPF-weighted	67.2960539 ± 4.434176779
OPF-major	67.23050776 ± 4.250484987

Figure 5.8 depicts the results concerning the Hyperplane dataset. The combined and weighted versions obtained similar performances. The combined version has a drop of performance in batches #11 and #21, whereas OPF-nomemory has a better performance in batches #6, #10, #16, #20, #22, #23, among others. The major version has a similar performance as OPF-nomemory, which figured out as the two best techniques concerning this dataset.

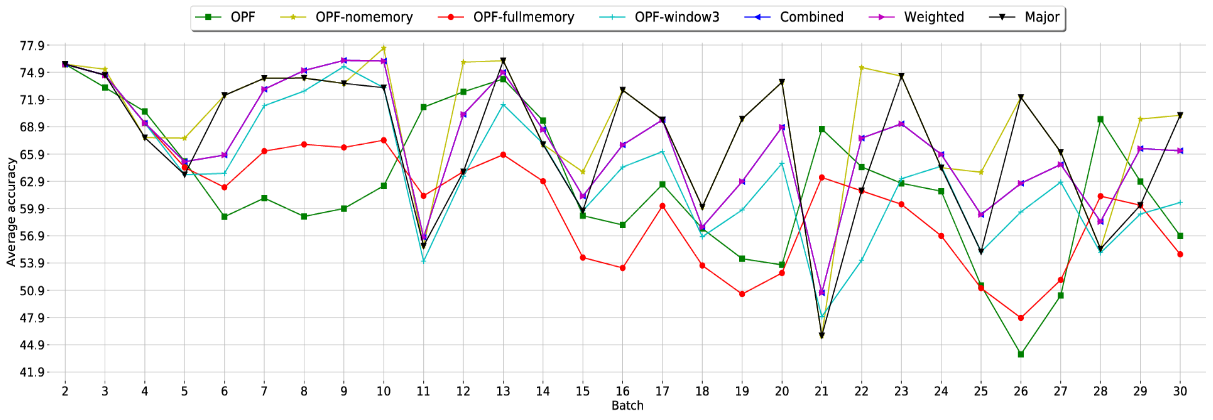

Figure 5.8: Experimental results concerning Hyperplane dataset with respect to Experiment 1.

Table 5.5: Hyperplane dataset with respect to Experiment 1.

Type	Accuracy
OPF	62.48453148 ± 7.664021067
OPF-nomemory	69.01541517 ± 7.265204095
OPF-fullmemory	60.77733197 ± 6.950052568
OPF-window3	63.79282938 ± 7.071207897
OPF-combined	66.90718755 ± 6.359478356
OPF-weighted	66.90718755 ± 6.359478356
OPF-major	67.042868 ± 7.594141989

Figure 5.9 depicts the results concerning the Poker dataset. The weighted version has a better performance than the combined version, whereas the major version has a similar performance as OPF-fullmemory. Once again, these last two approaches have obtained the best results so far. The accuracies vary consistently, thus degrading the performance of standard OPF and OPF-nomemory. Some batch transitions can really show the problem of concept drift in this dataset. Consider the transition between batches #12 and #13: clearly, one can observe the accuracy decreased from 84.8% to nearly 60% considering standard OPF. Although all the other accuracies decreased either, the proposed OPF with majority voting reached nearly 72%. Table 5.6 presents the mean accuracy and standard deviation concerning all batches.

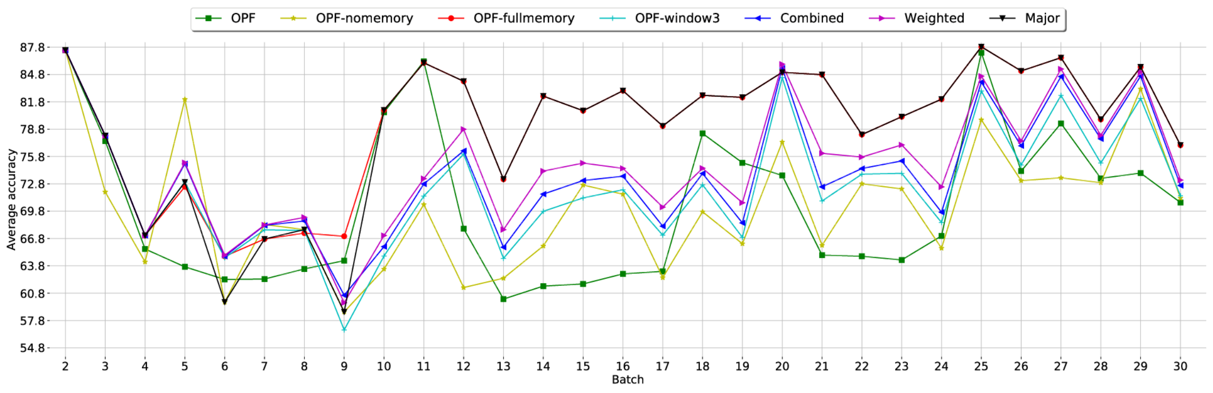

Figure 5.9: Experimental results concerning Poker dataset with respect to Experiment 1.

Table 5.6: Poker Hand dataset with respect to Experiment 1.

Type	Accuracy
OPF	70.29847297 ± 8.205813114
OPF-nomemory	70.17197476 ± 6.946167608
OPF-fullmemory	79.24812266 ± 6.830883212
OPF-window3	72.42684324 ± 6.755000663
OPF-combined	73.72377555 ± 6.649724747
OPF-weighted	74.74302866 ± 6.604097967
OPF-major	78.81981252 ± 7.820022743

Figure 5.10 depicts the results concerning the SEA dataset. The combined and weighted versions have similar behavior and also a higher accuracy when compared to traditional OPF and OPF with concept drift handling versions. This dataset showed the robustness of the proposed approaches, which obtained the best results so far, outperforming considerably standard OPF and OPF with concept drift handling with no ensemble-based learning. Table 5.7 presents the mean accuracy and standard deviation concerning all batches.

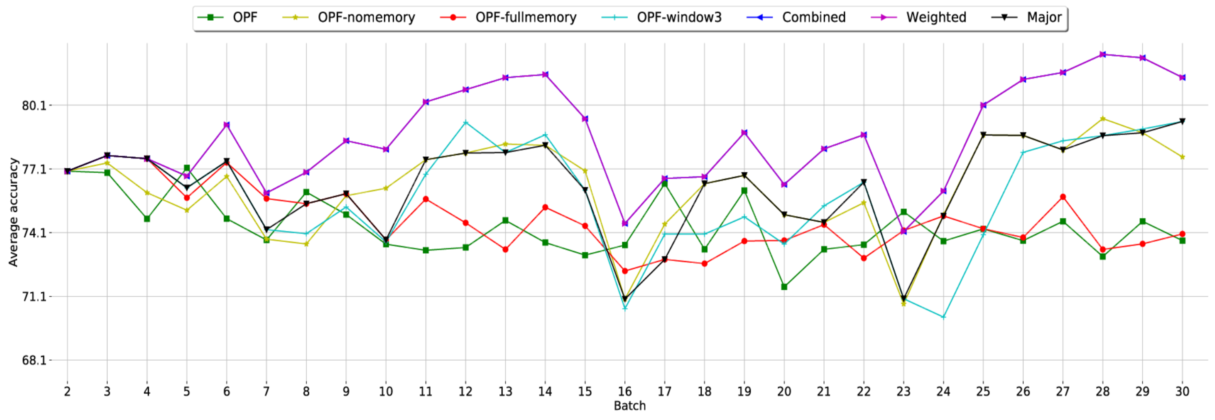

Figure 5.10: Experimental results concerning SEA dataset with respect to Experiment 1.

Table 5.7: SEA dataset with respect to Experiment 1.

Type	Accuracy
OPF	74.39154759 ± 1.346465467
OPF-nomemory	76.24827041 ± 2.128896094
OPF-fullmemory	74.69366883 ± 1.47469361
OPF-window3	75.83703666 ± 2.551433832
OPF-combined	78.6340661 ± 2.305279021
OPF-weighted	78.6340661 ± 2.305279021
OPF-major	76.34419907 ± 2.180538025

Figure 5.11 depicts the results concerning the Usenet1 dataset. Once again, the combined and weighted versions obtained similar performance, whereas the major version figured an oscillatory behavior when compared to other ensemble versions. The same behavior can be observed in the Usenet2 dataset as well (Figure 5.12). Tables 5.8 and 5.9 present the mean accuracy and standard deviation concerning all batches.

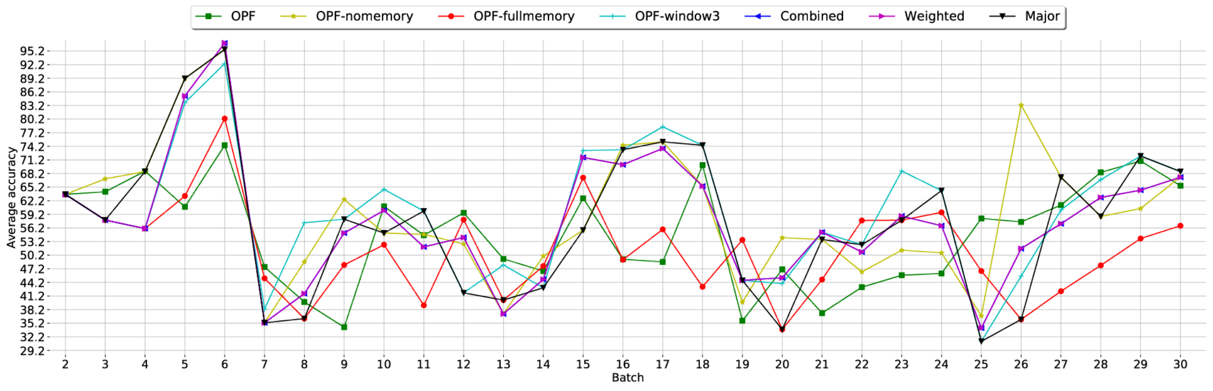

Figure 5.11: Experimental results concerning Usenet1 dataset with respect to Experiment 1.

Table 5.8: Usenet1 dataset with respect to Experiment 1.

Type	Accuracy
OPF	54.92385783 ± 11.19221173
OPF-nomemory	59.35022066 ± 14.63104797
OPF-fullmemory	51.54818866 ± 10.24161553
OPF-window3	59.95950859 ± 14.13373717
OPF-combined	57.60552966 ± 13.83043313
OPF-weighted	57.60552966 ± 13.83043313
OPF-major	57.39366248 ± 16.16345656

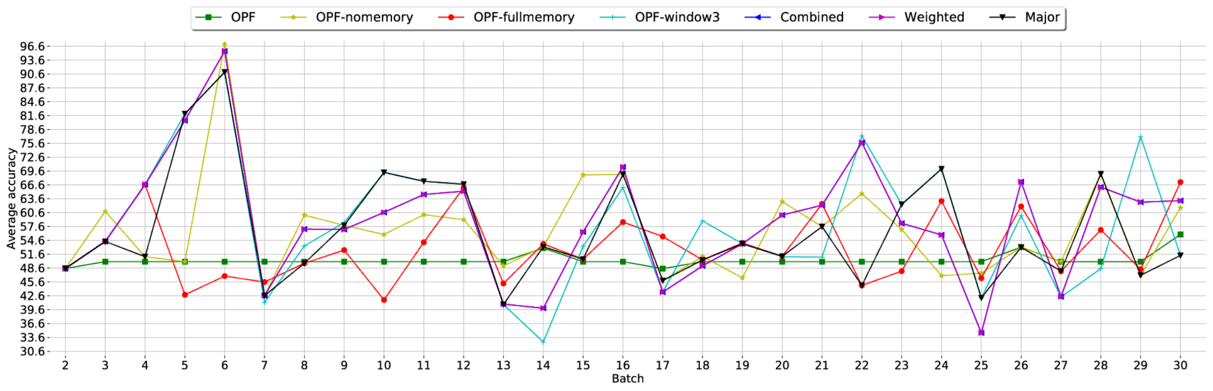

Figure 5.12: Experimental results concerning Usenet2 dataset with respect to Experiment 1.

Table 5.9: Usenet2 dataset with respect to Experiment 1.

Type	Accuracy
OPF	50.30906162 ± 1.37347363
OPF-nomemory	56.69870524 ± 10.57647197
OPF-fullmemory	52.94175238 ± 7.191195194
OPF-window3	57.96049969 ± 13.54929432
OPF-combined	58.45837903 ± 12.93718405
OPF-weighted	58.45837903 ± 12.93718405
OPF-major	56.57500459 ± 11.91377109

5.3.2 Experiment 2

In this experiment, we decided to verify whether one can improve the concept drift handling using ensemble-based learning, but not combining different approaches, i.e., OPF-fullmemory, OPF-nomemory, and OPF-window3. Therefore, the idea was, for each aforementioned approach, to partition the training data into three subsets, and then apply ensemble learning over them. In this experiment, we are comparing standard OPF, OPF-nomemory, OPF-fullmemory, OPF-window3, the proposed ensemble-based learning with combined, weighted, and major approaches, as well as the performance of each individual fold (i.e., fold1, fold2, and fold3).

Figure 5.13 depicts the results concerning the Covtype dataset. The combined and weighted versions obtained similar behavior, being the weighted version better in some batches. The major version has a worse behavior among all. In no-memory and window-3 management, the traditional OPF obtained the higher accuracy among all. A similar behavior to the one obtained in Experiment 1 can be observed, i.e., with OPF and OPF-fullmemory obtaining the most accurate results. Table 5.10 presents the mean accuracy and standard deviation concerning all batches in full-memory management.

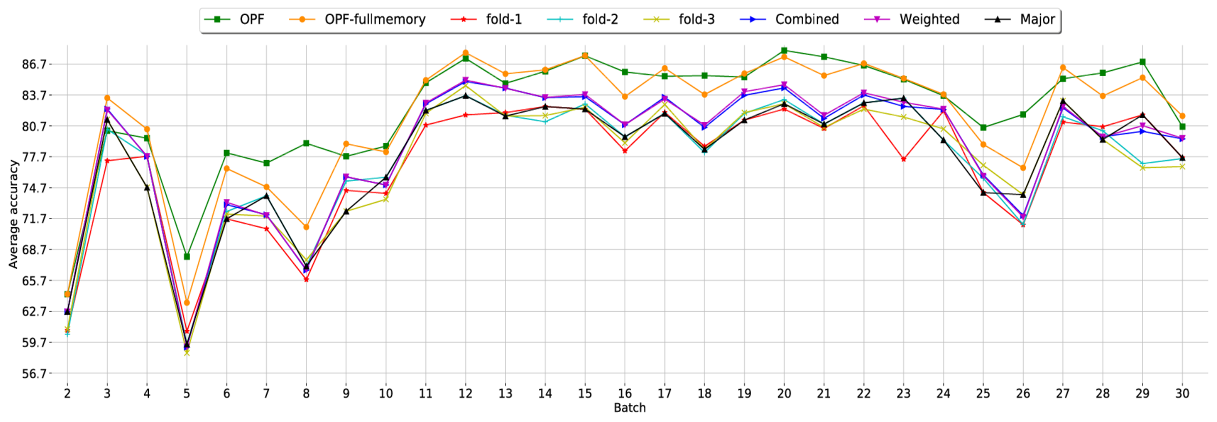

Figure 5.13: Experimental results concerning Covertype dataset with respect to Experiment 2 in full-memory management.

Table 5.10: Covertype dataset with respect to Experiment 2 in full-memory management.

Type	Accuracy
OPF	82.32752648 ± 5.527916684
OPF-fullmemory	81.51895655 ± 6.360053157
OPF-fold1	77.05269845 ± 6.149721752
OPF-fold2	77.53515266 ± 6.216085779
OPF-fold3	77.32545021 ± 6.333944787
OPF-combined	78.4888871 ± 6.553147785
OPF-weighted	78.58643359 ± 6.601479619
OPF-major	77.65701845 ± 6.154886295

In the Electricity dataset (Figure 5.14), the combined and weighted versions of full-memory, no-memory, and window-3 obtained similar behaviors, being the majority voting the one with lower accuracies. One can also observe that standard OPF did not perform well, with accuracies much below the ones obtained by the others. We can observe in the no-memory management

(Figure 5.14) that OPF-nomemory and the combined and weighted versions figured an oscillatory performance. Table 5.11 presents the mean accuracy and standard deviation concerning all batches in no-memory management.

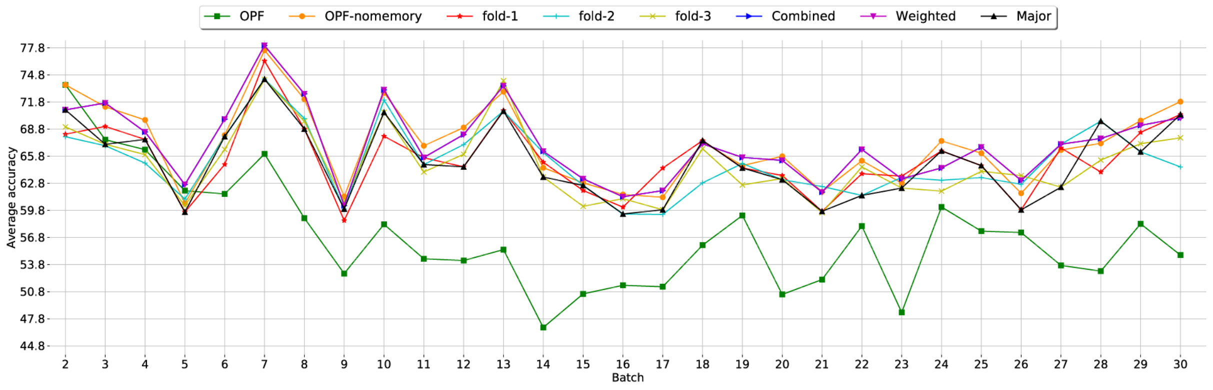

Figure 5.14: Experimental results concerning Electricity dataset with respect to Experiment 2 in no-memory management.

Table 5.11: Electricity dataset with respect to Experiment 2 in no-memory management.

Type	Accuracy
OPF	56.94443552 ± 6.002938206
OPF-nomemory	67.06707179 ± 4.373578762
OPF-fold1	65.44168362 ± 3.854310398
OPF-fold2	65.20587686 ± 3.711158446
OPF-fold3	64.99470534 ± 3.879081788
OPF-combined	67.13015469 ± 4.162979812
OPF-weighted	67.13015469 ± 4.162979812
OPF-major	65.21247934 ± 4.102278054

In the Hyperplane dataset (Figure 5.15), the combined and weighted versions of full-memory, no-memory, and window-3 obtained similar performances, being the majority voting the one with smaller accuracies. The OPF with ensemble learning and traditional OPF figured an oscillatory performance in the full-memory approach, whereas traditional OPF in no-memory management has higher accuracy only on batches #4, #11, #21, and #28. Traditional OPF in window-3 management (Figure 5.15) performed a little better when compared to the no-memory management. Interestingly, standard OPF performed better in a transition between batches #10 and #11, which seems to be a concept drift, but with the statistics of the testing data similar to the one standard OPF has been trained for (the beginning of the streaming). Table 5.12 presents the mean accuracy and standard deviation concerning all batches in window-3 management.

In the Poker dataset (Figure 5.16), the combined, weighted, and major versions of full-memory, no-memory, and window-3 obtained similar performances once more, with the weighted version the most accurate one followed by the combined one. In this dataset, one can clearly ob-

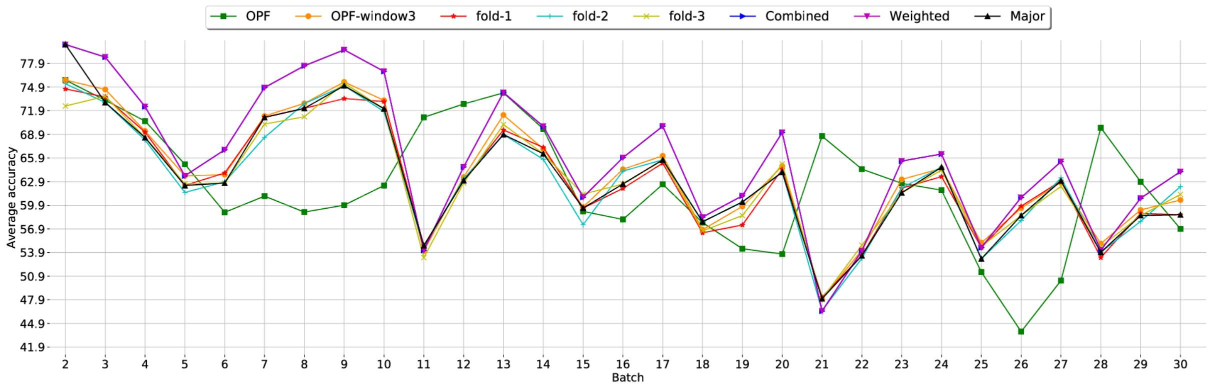

Figure 5.15: Experimental results concerning Hyperplane dataset with respect to Experiment 2 in window-3 management.

Table 5.12: Hyperplane dataset with respect to Experiment 2 in window-3 management.

Type	Accuracy
OPF	62.48453148 ± 7.664021067
OPF-window3	63.79282938 ± 7.071207897
OPF-fold1	63.05814352 ± 6.930449599
OPF-fold2	62.96584131 ± 7.003386526
OPF-fold3	63.10990428 ± 6.694532681
OPF-combined	65.928872 ± 8.586587354
OPF-weighted	65.928872 ± 8.586587354
OPF-major	63.24691903 ± 7.267281317

serve some situations where the standard OPF's accuracy drops considerably (e.g., transitions between batches #11 and #12, and between batches #20 and #21), since it is not prepared to handle concept drift. Table 5.13 presents the mean accuracy and standard deviation concerning all batches in full-memory management.

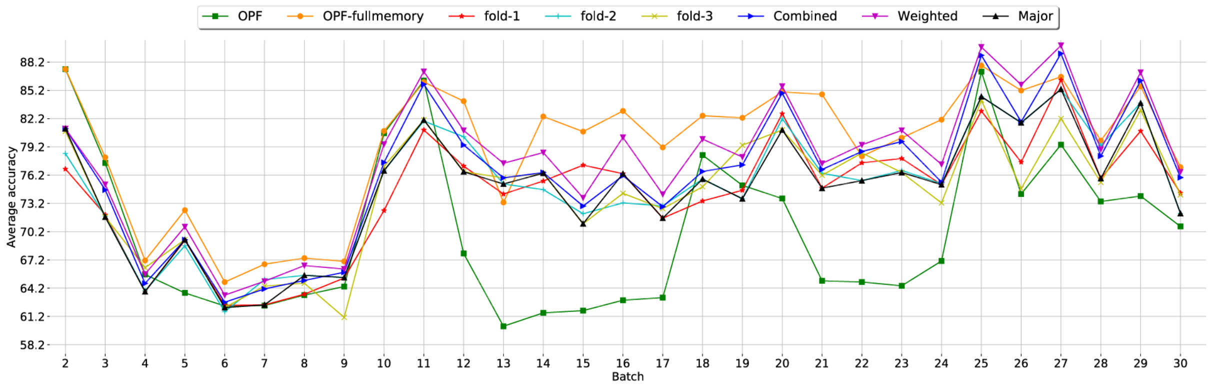

Figure 5.16: Experimental results concerning Poker dataset with respect to Experiment 2 in full-memory management.

In the SEA dataset (Figure 5.17), the combined and weighted versions of full-memory, no-memory, and window-3 obtained similar performances, performing better than traditional OPF and OPF with concept drift handling. The major version achieved the worst accuracy compared

Table 5.13: **Poker dataset with respect to Experiment 2 in full-memory management.**

Type	Accuracy
OPF	70.29847297 ± 8.205813114
OPF-fullmemory	79.24812266 ± 6.830883212
OPF-fold1	74.32355024 ± 6.097886009
OPF-fold2	74.84017507 ± 6.282530565
OPF-fold3	74.49460559 ± 6.099172864
OPF-combined	76.36024579 ± 7.0838907
OPF-weighted	77.68715672 ± 7.209030697
OPF-major	74.60479938 ± 6.335993863

to the other ensemble versions. Table 5.14 presents the mean accuracy and standard deviation concerning all batches in no-memory management.

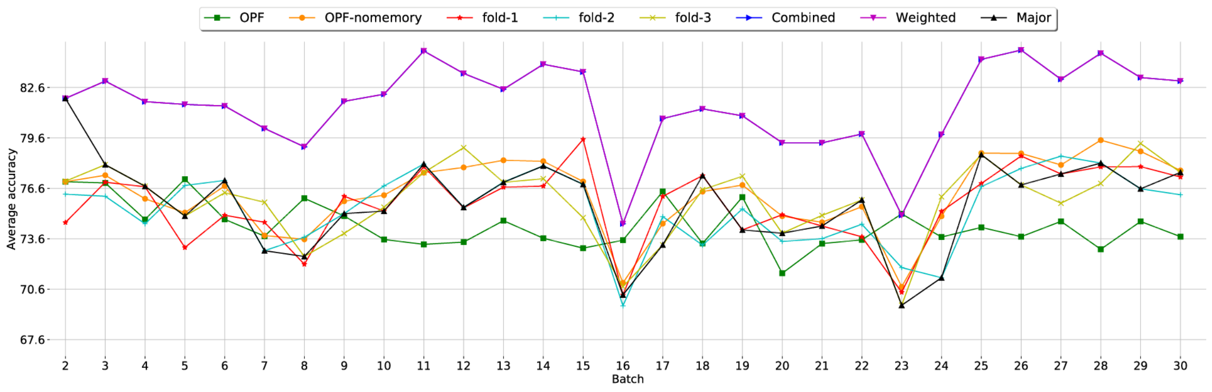

Figure 5.17: Experimental results concerning SEA dataset with respect to Experiment 2 in no-memory management.

Table 5.14: SEA dataset with respect to Experiment 2 in no-memory management.

Type	Accuracy
OPF	74.39154759 ± 1.346465467
OPF-nomemory	76.24827041 ± 2.128896094
OPF-fold1	75.625649 ± 2.204542238
OPF-fold2	75.38316921 ± 2.181240932
OPF-fold3	75.84664707 ± 2.213159411
OPF-combined	81.54980866 ± 2.490986588
OPF-weighted	81.54980866 ± 2.490986588
OPF-major	75.68724848 ± 2.655161575

In Usenet1 (Figure 5.18) and Usenet 2 (Figure 5.19) datasets, the combined and weighted versions of full-memory, no-memory, and window-3 management obtained similar performances either, with the ensemble learning with majority voting figuring an oscillatory behavior. Tables 5.15 and 5.16 present the mean accuracy and standard deviation concerning all batches in window-3 and full-memory management, respectively.

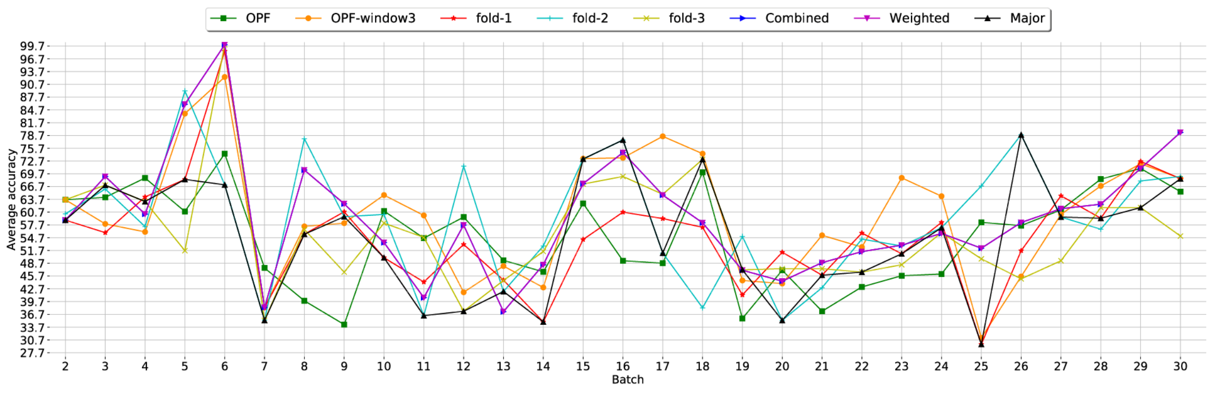

Figure 5.18: Experimental results concerning Usenet1 dataset with respect to Experiment 2 in window-3 management.

Table 5.15: Usenet1 dataset with respect to Experiment 2 in window-3 management.

Type	Accuracy
OPF	54.92385783 ± 11.19221173
OPF-window3	59.95950859 ± 14.13373717
OPF-fold1	55.46803024 ± 12.79941954
OPF-fold2	59.04749762 ± 13.80458451
OPF-fold3	55.87159655 ± 12.47894492
OPF-combined	59.7810629 ± 13.84571992
OPF-weighted	59.7810629 ± 13.84571992
OPF-major	54.87391672 ± 13.76932634

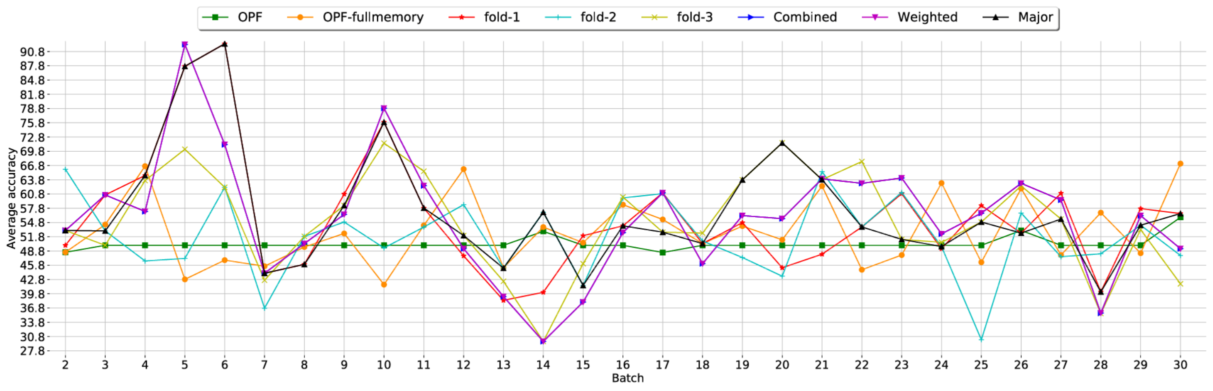

Figure 5.19: Experimental results concerning Usenet2 dataset with respect to Experiment 2 in full-memory management.

Table 5.16: Usenet2 dataset with respect to Experiment 2 in full-memory management.

Type	Accuracy
OPF	50.30906162 ± 1.37347363
OPF-fullmemory	52.94175238 ± 7.191195194
OPF-fold1	55.95265503 ± 12.2914427
OPF-fold2	51.793095 ± 8.079654346
OPF-fold3	55.08751245 ± 10.3272684
OPF-combined	55.83010672 ± 12.45587074
OPF-weighted	55.83010672 ± 12.45587074
OPF-major	57.04483903 ± 11.92551197

5.3.3 Experiment 3

In this experiment, we consider the last three batches to perform the ensemble learning step. Therefore, one classifier is trained on each batch for further combinations of results. This experiment compared traditional OPF, OPF-nomemory, OPF-window3, as well as the ensemble-based approaches combined and weighted.

Figure 5.20 presents the results over the Covertype dataset. Once again, traditional OPF obtained the best recognition rates so far, being the weighted version similar to the combined approach, but being a little more accurate. In this case, the proposed approaches did not perform well, probably because the last three batches may have conflicting information, i.e., different statistics concerning the testing batch. Table 5.17 presents the mean accuracy and standard deviation concerning all batches.

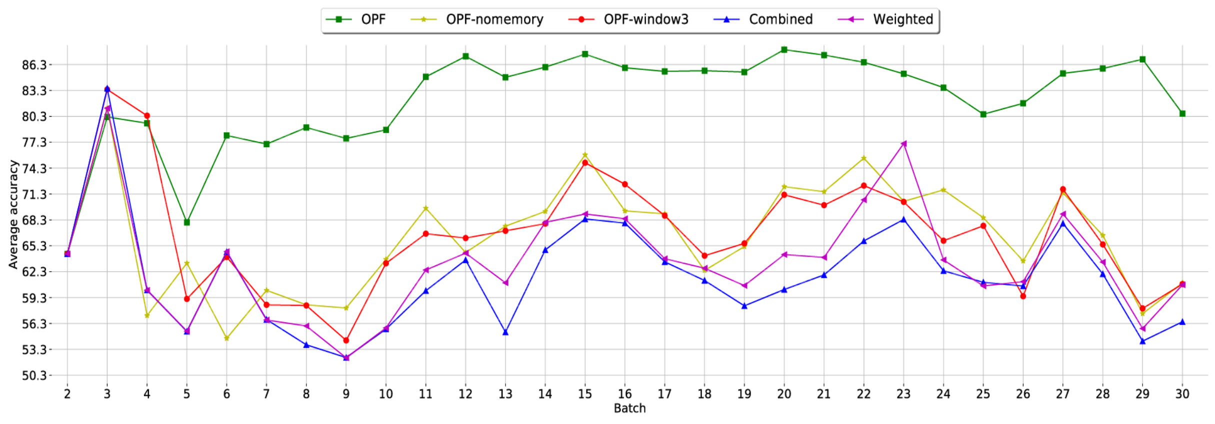

Figure 5.20: Experimental results concerning Covertype dataset with respect to Experiment 3.

Table 5.17: Covertype dataset with respect to Experiment 3.

Type	Accuracy
OPF	82.32752648 ± 5.527916684
OPF-nomemory	66.340352 ± 6.205296248
OPF-window3	66.63647852 ± 6.483513342
OPF-combined	61.75935952 ± 6.122989272
OPF-weighted	63.36128134 ± 6.217485077

Figure 5.21 depicts the results concerning the Electricity dataset. The combined and weighted version obtained similar performances, as well as an oscillatory behavior compared to OPF with concept drift with no ensemble learning can be observed as well. In this dataset, the proposed approaches outperformed the others in each batch, thus showing to be way robust to concept drift. As a matter of fact, in a few situations OPF-nomemory obtained the highest accuracies. Table 5.18 presents the mean accuracy and standard deviation concerning all batches.

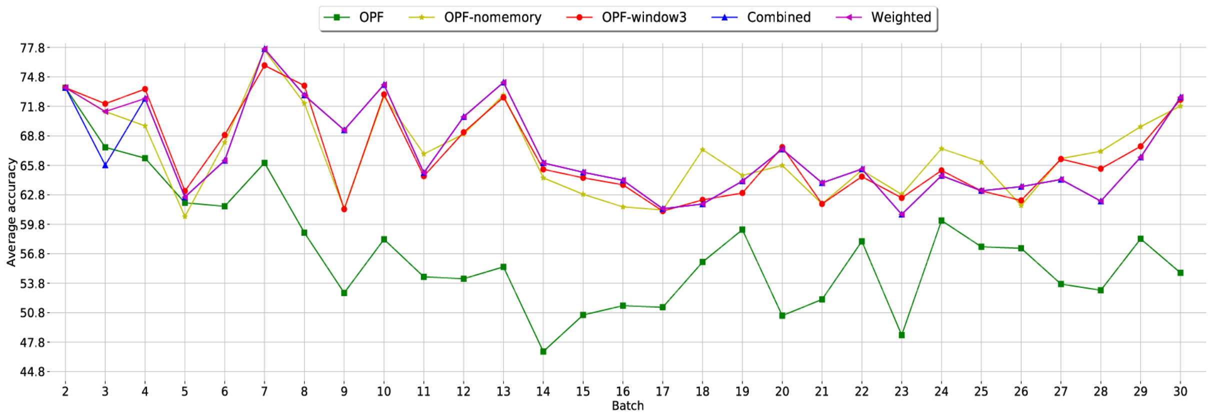

Figure 5.21: Experimental results concerning Electricity dataset with respect to Experiment 3.

Table 5.18: Electricity dataset with respect to Experiment 3.

Type	Accuracy
OPF	56.94443552 ± 6.002938206
OPF-nomemory	67.06707179 ± 4.373578762
OPF-window3	66.95818376 ± 4.505452085
OPF-combined	66.99822697 ± 4.525034793
OPF-weighted	67.18636838 ± 4.584960193

Figure 5.22 depicts the results concerning the Hyperplane dataset. Once again, the proposed approaches (combined and weighted versions) obtained the best results and with similar performances in some batches, being also slightly more accurate than OPF-window3. For other batches, one can observe that OPF-nomemory obtained very much accurate results. OPF with window-of-size 3 did not perform well with respect to the other approaches, although it has outperformed OPF with no concept drift handling. Actually, the task of choosing proper window sizes is still an open issue in the context of concept drift. The point is that smaller windows may not contain enough information about the training data, whereas bigger-sized windows may have too old and outdated information about the new data that is coming through the stream. Table 5.19 presents the mean accuracy and standard deviation concerning all batches.

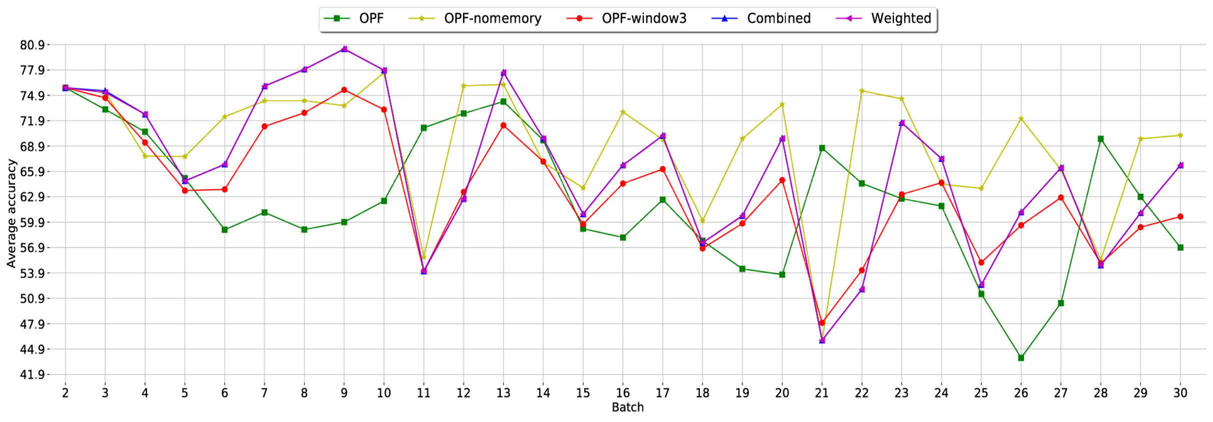

Figure 5.22: Experimental results concerning Hyperplane dataset with respect to Experiment 3.

Table 5.19: Hyperplane dataset with respect to Experiment 3.

Type	Accuracy
OPF	62.48453148 ± 7.664021067
OPF-nomemory	69.01541517 ± 7.265204095
OPF-window3	63.79282938 ± 7.071207897
OPF-combined	66.09129634 ± 8.855442297
OPF-weighted	66.08478797 ± 8.848643462

Figure 5.23 depicts the results concerning the Poker dataset. The combined and weighted versions obtained similar performances, but the weighted version a little more accurate. In this situation, a window of size 3 to handle concept drift seems to be a good alternative, since it allowed the best recognition rates for some batches. Standard OPF did perform well in some situations too, mainly in batches where concept drift-driven approaches did not obtain satisfactory results. Table 5.20 presents the mean accuracy and standard deviation concerning all batches.

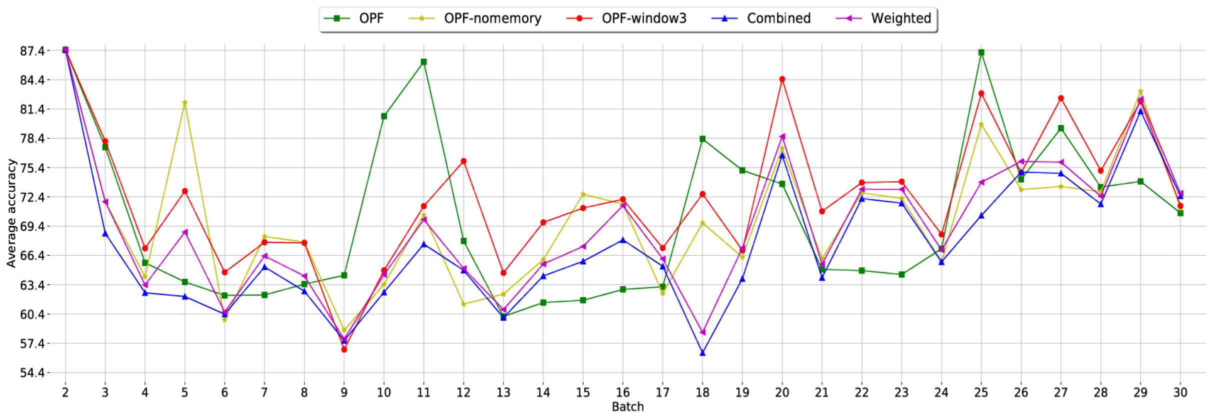

Figure 5.23: Experimental results concerning Poker dataset with respect to Experiment 3.

Table 5.20: Poker dataset with respect to Experiment 3.

Type	Accuracy
OPF	70.29847297 ± 8.205813114
OPF-nomemory	70.17197476 ± 6.946167608
OPF-window3	72.42684324 ± 6.755000663
OPF-combined	67.65484331 ± 6.826856184
OPF-weighted	69.24347752 ± 6.749471883

Figure 5.24 depicts the results concerning the SEA dataset. The combined and weighted variants obtained similar performances. As one can observe, the aforementioned approaches obtained the better performance among all, except in batch #24. In this specific situation, OPF-nomemory was slightly more accurate. Table 5.21 presents the mean accuracy and standard deviation concerning all batches.

Figures 5.25 and 5.26 depict the results concerning Usenet1 and Usenet2 datasets, respec-

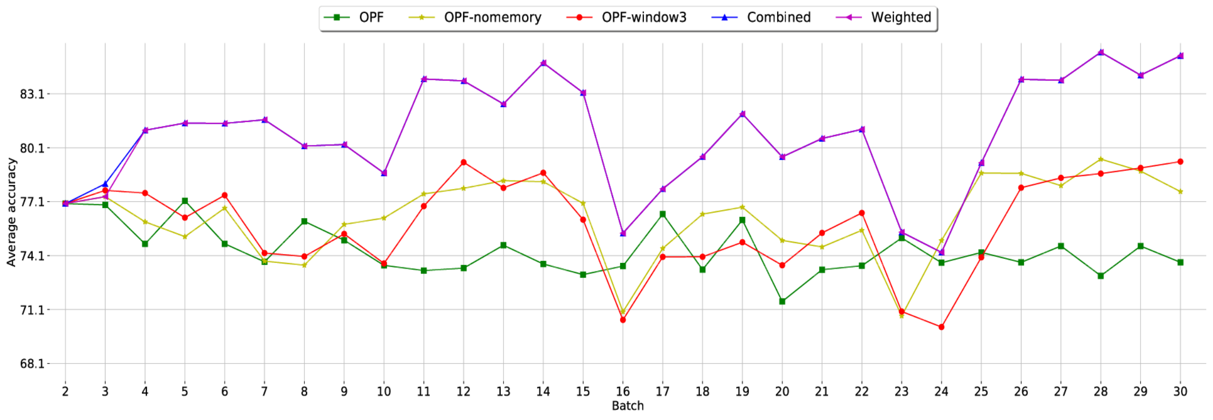

Figure 5.24: Experimental results concerning SEA dataset with respect to Experiment 3.

Table 5.21: SEA dataset with respect to Experiment 3.

Type	Accuracy
OPF	74.39154759 ± 1.346465467
OPF-nomemory	76.24827041 ± 2.128896094
OPF-window3	75.83703666 ± 2.551433832
OPF-combined	80.88808886 ± 2.995032574
OPF-weighted	80.86339066 ± 3.020802532

tively. The combined and weighted version obtained similar performances in both datasets, as one can observe in the previous experiments concerning these datasets as well. Tables 5.22 and 5.23 present the mean accuracy and standard deviation concerning all batches.

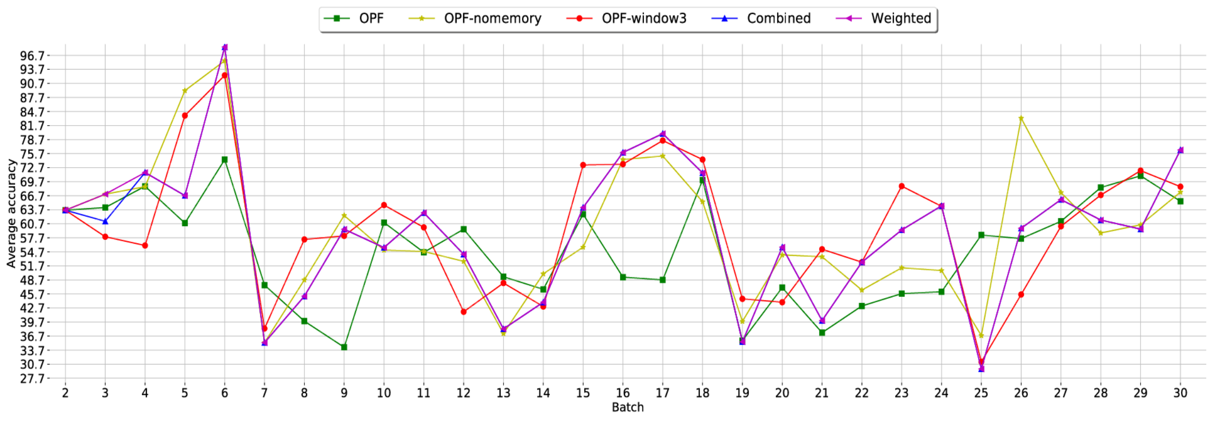

Figure 5.25: Experimental results concerning Usenet1 dataset with respect to Experiment 3.

Table 5.22: Usenet1 dataset with respect to Experiment 3.

Type	Accuracy
OPF	54.92385783 ± 11.19221173
OPF-nomemory	59.35022066 ± 14.63104797
OPF-window3	59.95950859 ± 14.13373717
OPF-combined	58.92475714 ± 14.84891883
OPF-weighted	59.12452362 ± 14.91739836

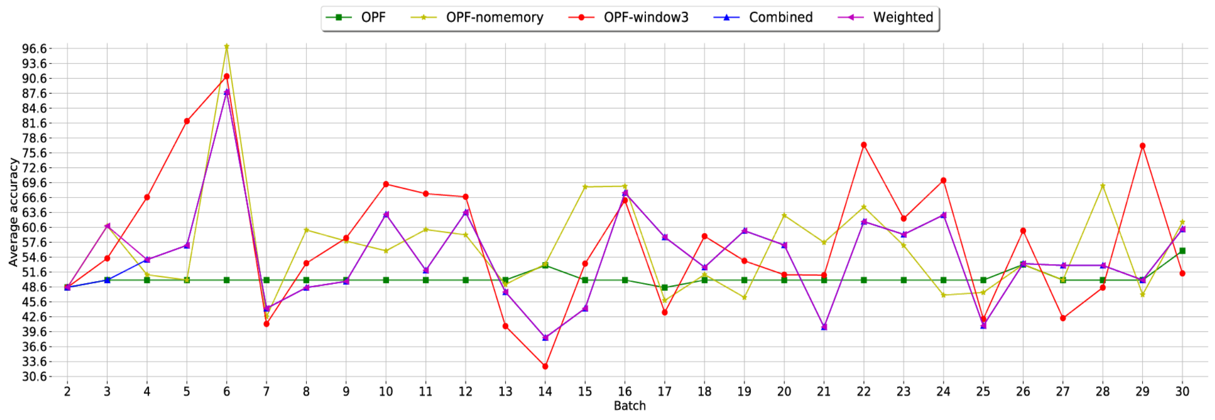

Figure 5.26: Experimental results concerning Usenet2 dataset with respect to Experiment 3.

Table 5.23: Usenet2 dataset with respect to Experiment 3.

Type	Accuracy
OPF	50.30906162 ± 1.37347363
OPF-nomemory	56.69870524 ± 10.57647197
OPF-window3	57.96049969 ± 13.54929432
OPF-combined	54.47438328 ± 9.70819266
OPF-weighted	54.84932934 ± 9.738073086

5.3.4 Discussion

In regard to Experiments 1 and 2, in most datasets they allowed us similar recognition rates, but we did not observe significant better performances with respect to the ensemble-based approaches and individual folds in the Experiment 2, just in some cases. However, the proposed approaches outperformed individual folds in almost all datasets. Also, Experiment 1 showed us the proposed approaches are pretty much suitable to handle the concept drift, and using information from different learners can improve the recognition rates. For some datasets, we observed that combining information from different concept drift handling approaches (Experiment 1) is more beneficial than using information from the same handling approach. We can verify in Experiment 2 that training in each fold is faster than training in the non-split version (OPF-nomemory, OPF-fullmemory, and OPF-window3). Although the size is pretty much the same, since OPF has a quadratic complexity for training, the learning process is faster when performed in smaller training sets (PONTI; PAPA, 2011).

The Experiment 3 allowed us to combine the three last batches, and not mixing different concept drift approaches. We have observed that such methodology brings us results that are somehow close to the ones obtained with OPF with window-of-size 3 for concept drift handling, but being faster, since we are now training on three individual batches, and not in only one.

5.4 Partial Considerations

In this work, we dealt with the problem of concept drift concerning the Optimum-Path Forest with ensemble learning. As far as we are concerned, no study aimed at considering the aforementioned method to handle changes in a data stream up to date.

The experiments over real and synthetic datasets compared standard OPF, OPF-fullmemory, OPF-nomemory, and OPF-window3 approaches, as well as three proposed distinct ensemble-based versions to address concept drift. We have shown that OPF ensemble is suitable to work under these dynamic scenarios since its recognition rates were considerably better compared to traditional OPF and OPF with concept drift handling with no ensemble learning.

We have observed that, in Experiment 1, the combined ensemble approach obtained the highest accuracy on SEA dataset, considerably improving its performance compared to traditional OPF and the OPF with concept drift handling versions with no ensemble learning. We can also verify that combined methods have better accuracy than the ones obtained on each fold individually in Experiment 2. Additionally, the main advantage concerning Experiment 3 over OPF-window3 is its training time, since training in smaller data is faster than a larger one, i.e., training three individual batches is faster than training three merged batches.

In regard to future works, we intend to dynamically learn and adjust the window size, as well as to implement OPF on data stream with incremental learning, i.e., where we can pose constraints in the training time. Therefore, the idea is not retraining the whole classifier when a new batch comes to play, but just update the trained classifier.

5.5 OPF with dynamic window

This section presents the results of OPF with drift detection based on accuracy against traditional OPF and OPF in the no-memory, full-memory, and fixed-window3 versions. The main objective of this work is to verify the OPF performance with a simple drift detector mechanism to deal with the concept drift detection. We employed a dynamic sliding window in public datasets, being the results compared to the traditional OPF and its no-memory, full-memory, and fixed-window3 versions.

The OPF verifies the accuracy of the last batch and uses it as a threshold. We considered that if the accuracy drops below a threshold, a concept drift is detected. There is no concept drift while the accuracy is above a threshold and the training set contains all previous batches to maintain the stability of the data. If concept drift occurs, i.e., accuracy drops below a threshold (which was empirically defined), the current training data is discarded and the training phase employs the newest batch.

The datasets used in this work are Electricity (Subsection 2.3.1.2), Hyperlane, SEA, Usenet1, and Usenet2 (Subsection 2.3.1.1). In order to evaluate the techniques, we divided each dataset into 30 batches (equally) aiming to simulate a stream of data.

Figures 5.27 to 5.31 depict the accuracy rates of all aforementioned techniques considering the datasets employed in this work. Let us have a look at Figure 5.27, which stands for the results considering the Electricity dataset. Traditional OPF has a lower accuracy than OPF versions. Also, we observe that the higher accuracy values alternate among the OPF versions. The detection was considered in batches #9, #17, #21, and #25.

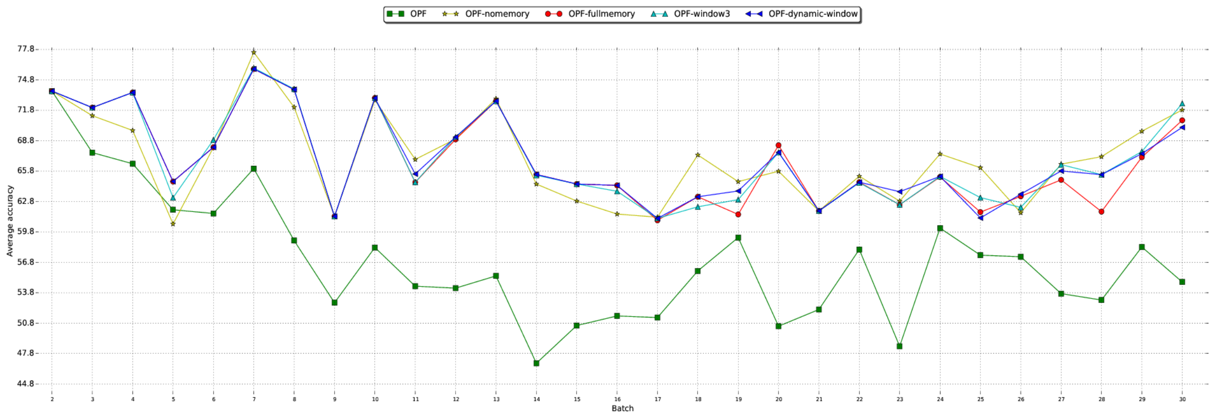

Figure 5.27: Experimental results concerning Electricity dataset with threshold 63.

Figure 5.28 depicts the results over Hyperplane dataset stream. OPF with dynamic-window has a similar behavior than OPF with fixed-window3. Dynamic window did better in batches #8, #13, #17, #20, #23, #27 and #30, whereas fixed-window3 did better in batches #6, #7, #12, #16, #21, #26 and #28. OPF-nomemory has some batches of higher accuracy. The detection was considered in batches #6, #11, #12, #15, #16, #18, #19, #21, #22, #25, #26 and #28.

Figure 5.29 depicts the results over the SEA dataset stream. OPF with dynamic-window has a similar behavior in accuracy with OPF-fullmemory until batch #17. It has a similar behavior also in the chosen data used in the training phase. After batch #17, the OPF dynamic-window detected a drift and discarded the old data, and we can see different behavior in its accuracy being more similar to the fixed-window3. The detection was considered in batches #16, #17, #23 and #24.

Figure 5.30 depicts the results over the Usenet1 dataset stream. OPF with dynamic-window starts with similar behavior to OPF-fullmemory until batch #8. After that, the value of its accuracy behaves as an alternation of the accuracy values of OPF-fullmemory and OPF-fixed-

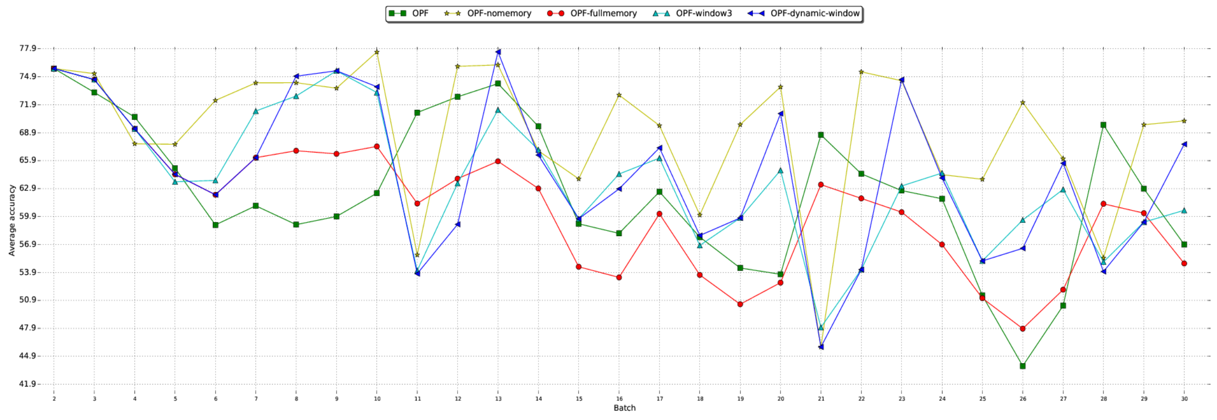

Figure 5.28: Experimental results concerning Hyperplane dataset with threshold 64.

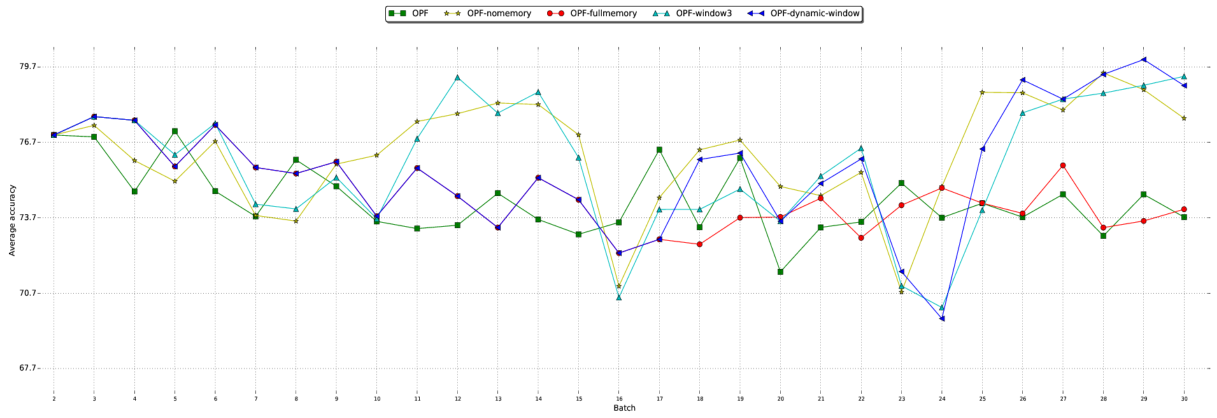

Figure 5.29: Experimental results concerning SEA dataset with threshold 73.

window3. The detection was considered in batches #8, #13, #14, #19 and #28.

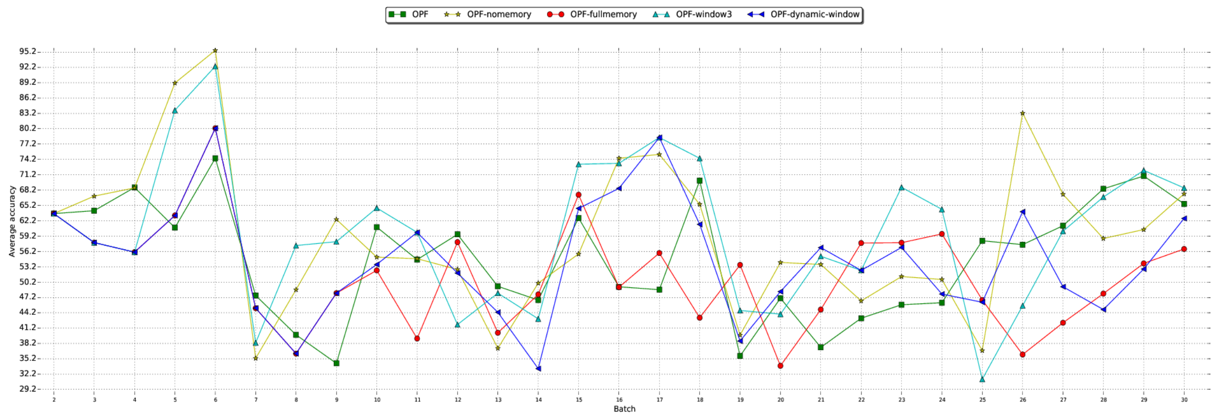

Figure 5.30: Experimental results concerning Usenet1 dataset with threshold 45.

Figure 5.31 depicts the results over the Usenet2 dataset stream. OPF with dynamic-window starts with similar behavior to OPF-fullmemory until batch #6. After that, the value of its accuracy behaves as an alternation of the accuracy values of OPF-fullmemory and OPF-fixed-window3. The detection was considered in batches #5, #7, #13, #22 and #25.

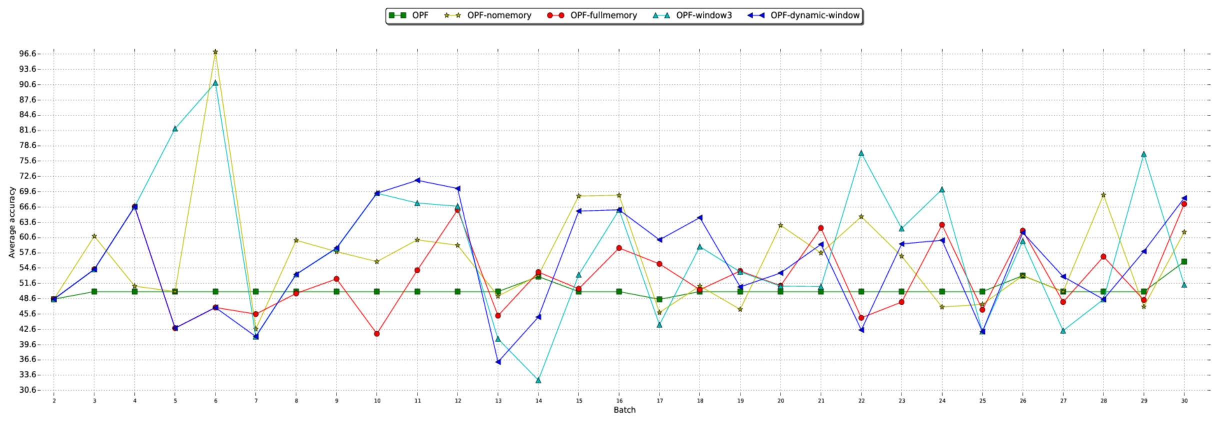

Figure 5.31: Experimental results concerning Usenet2 dataset with threshold 45.

In this work, we dealt with the problem of concept drift concerning the OPF classifier with a dynamic-window. The experiments over real and synthetic datasets compared standard OPF, OPF-fullmemory, OPF-nomemory, and OPF-window3 approaches, as well as the OPF-dynamic-window to address concept drift. We have shown that the OPF-dynamic-window accuracy has similar accuracy as OPF-fullmemory or OPF-window3, depending on how the window behaves in the stream of data. When accuracy drops below an empiric threshold, the OPF-dynamic-window employs the most recent batch on training, the same dataset used in the OPF-nomemory training phase. While accuracy is above the threshold, the batches are added on training to increase stability.

Chapter 6

INCREMENTAL OPF$_{knn}$

This chapter presents the work by Iwashita et al. (IWASHITA et al., 2021), which proposes the OPF$_{knn}$ with incremental learning. The idea of an incremental algorithm comes from the need to add new instances to the classifier without retraining the entire data, which non-stationary environments and streaming data may demand. These environments require the learner to be constantly updated since repeating the entire learning process might be prohibitive. So, adjusting the model to the new data shows to be a better choice, particularly for real-time response applications. We named our algorithm as Incremental OPF$_{knn}$ (IOPF$_{knn}$) and employed it in some databases, including non-technical losses environments. Non-technical losses stand for the energy that is consumed but not billed, which is usually referred to as commercial losses, and affect the energy grid as a whole. Frauds in energy consumption are somehow prevalent in developing countries and may harm the quality of energy and further improvements in social programs that can benefit from tax revenues. Therefore, the automatic identification of such illegal users is important to maintain decent services for the population. So the machine learning techniques have to overcome such an issue by mining information from fraudsters and legal users for further decision-making, hence requiring the learner to be constantly updated.

6.1 Introduction

Non-technical losses, also known as commercial losses, stand primarily for energy theft in power distribution systems, but not limited to it. Non-payment by customers and measurement errors in accounting and record-keeping are also frequent events that can characterize such illegal activities. Underdeveloped countries are the most affected since socioeconomic factors are crucial to shaping electricity theft behaviors (RAZAVI; FLEURY, 2019).

Research on computer-assisted non-technical losses is usually grouped into two scenarios:

(i) identification or detection and (ii) categorization. The former approaches aim at understanding the behavior of illegal profiles to identify them among hundreds of thousands of consumers. On the other hand, categorization techniques are concerned in learning the primary information used to describe users, i.e., to select the subset of features that maximizes the identification rates.

Buzau et al. (BUZAU et al., 2019) used data from smart meters to feed supervised machine learning techniques for non-technical losses detection. The authors argued that using labeled information is challenging since we most rely on imbalanced datasets. Therefore, the work employed undersampling techniques to remove the majority class, i.e., legal consumers, for further performing pattern recognition. Ramos et al. (RAMOS et al., 2011) introduced the Optimum-Path Forest (OPF) (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012) classifier in the context of non-technical losses identification. OPF works under the graph partition assumption for pattern recognition, i.e., the rationale behind it is to partition a graph induced by the training samples into clusters of samples from the very same class. The authors reported results that outperformed some state-of-the-art techniques such as Support Vector Machines (SVM). Later on, Fernandes et al. (FERNANDES et al., 2019) proposed a probabilistic OPF classifier that was able to produce soft outputs, i.e., given a profile from some consumer, OPF could output its probability of being legal or not. The main idea was to provide a framework that could monitor users over time and then learn whether they are likely to become fraudsters or not.

From another perspective, Martins et al. (MARTINS et al., 2019) developed a non-intrusive energy meter for non-technical losses identification in low-voltage AC installations. The authors argued that the proposed approach does not require on-site calibration, besides being a safe-and-easy device to be installed in service drop lines. Kim et al. (KIM et al., 2019) devised an intermediate monitor meter framework that can divide the grid into small logical sections and then provide a more in-depth analysis of the power flow for further non-technical losses inspection. The work reported real-time detection recognition rates up to 95%. Last but not least, Ramos et al. (RAMOS et al., 2016) proposed the Binary Black Hole Algorithm to characterize the profile of illegal consumers in Brazil. The authors modeled such an issue as a metaheuristic-based optimization problem, such that the accuracy of the OPF classifier was the fitness function used to find the most representative subset of features.

Most of the works mentioned above benefited from the machine learning framework, in which techniques learn by experience. Such algorithms are useful in problems like data mining, poorly-understood domains, and in environments where the learner needs to dynamically adapt to changing conditions (MITCHELL, 1997). The learners build models from sample inputs and can be categorized according to the available feedback: in supervised learning, the algorithm predicts a result and has the correct outcome; in unsupervised learning, there is no information about the outputs; and in semi-supervised learning, there is incomplete information about the correct outcome (RUSSELL; NORVIG, 1995).

Incremental learning is a paradigm that works when new instances appear, and the method adjusts the model according to them. If the entire learning process has to happen repeatedly whenever a new instance arrives, it might spend time and computation resources. Therefore, adjusting learned data according to the new instances might be a better decision, particularly for real-time response applications (GENG; SMITH-MILES, 2009). Other examples include time-dependent data generation (e.g., time-series data) and nonstationary environments, i.e., when the target concept changes over time (concept drift), and the learner should be able to self-adapt (GENG; SMITH-MILES, 2009; IWASHITA; PAPA, 2019).

As aforementioned, the Optimum-Path Forest stands for a pattern recognition framework that partitions instances (graph nodes) into a graph of optimum-path trees (OPTs). Each OPT has a key sample (prototype), which is the root tree, and the prototypes compete among themselves to conquer the remaining instances. Currently, the OPF classifier has three versions: (i) supervised (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012; PAPA; FERNANDES; FALCÃO, 2017), (ii) unsupervised (ROCHA; CAPPABIANCO; FALCÃO, 2009), and (iii) semi-supervised (AMORIM et al., 2016) ones. The supervised OPF classifier has two approaches: one that uses a complete graph (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012) and the other that uses a k-nn graph (OPF$_{knn}$) (PAPA; FERNANDES; FALCÃO, 2017). Recently, Iwashita and Papa (IWASHITA; PAPA, 2018; IWASHITA; ALBUQUERQUE; PAPA, 2019) proposed the use of OPF classifiers in the concept drift environment, and other works were developed to speed up the training algorithm taking into account the structural advantages of the data (PAPA et al., 2012; PONTI; ROSSI, 2013; PONTI; PAPA, 2011; IWASHITA et al., 2014).

Ponti and Riva (PONTI; RIVA, 2017) proposed an incremental version of the OPF with the complete graph (OPF Incremental - OPFI), in which the traditional OPF trains the initial model structure, and then the OPFI updates the model including the new instances. In this work, we propose an incremental version of the OPF with the k-nn graph (incremental OPF$_{knn}$) algorithm in order to avoid retraining all instances when including new samples in the training dataset. We performed experiments to compare the proposed approach against traditional OPF$_{knn}$. We showed the proposed approach can be so effective as standard OPF, but considerably faster in the context of non-technical losses identification and general-purpose problems. Besides, we also evaluated the proposed approach in the context of non-technical losses identification. Finally, we are not aware of previous works that aimed at modelling the problem of non-technical losses as an incremental learning task.

In a nutshell, the main contributions of this work are two-fold:

- to propose an incremental version of the OPF$_{knn}$ classifier; and
- to introduce the incremental learning paradigm in the context of non-technical losses identification.

The remainder of this work is organized as follows. Section 6.2 presents the theoretical background regarding OPF$_{knn}$, and Section 6.3 discusses the proposed approach based on the incremental learning paradigm. Section 6.4 presents the datasets and methodology, while Section 6.5 revisits the experiments. Finally, Section 6.6 states conclusions and future works.

6.2 Theoretical Background

In the OPF with k-neighborhood (OPF$_{knn}$), one connects each node to its k-nearest neighbors for the further calculation of a density value. The rationale is that high-density nodes will become the maxima of a probability density function (pdf), thus conquering the remaining nodes and forming an optimum-path forest, which is a collection of optimum-path trees rooted at each maximum (prototype). Later on, in the classification phase, a test sample is included in that optimum-path forest by finding its k-nearest neighbors and further computing its density value. The node from the training set that conquers that test sample will define its class.

The training and classification processes can be modeled as a reward-competition problem, in which samples try to conquer others by offering them optimum-path costs based on arc weights and a path-cost function. Different OPF formulations employ distinct path-cost functions, adjacency relations, and methodology to estimate prototypes. The next sections aim at explaining in more detail the OPF$_{knn}$ training and classification phases.

6.2.1 Training Step

Let $\mathscr{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ be a dataset such that $\boldsymbol{x} \in \Re^n$. Besides, let us assume that $\mathscr{X} = \mathscr{X}_1 \cup \mathscr{X}_2$, in which $\mathscr{X}_1$ and $\mathscr{X}_2$ stand for the training and test sets, respectively. Additionally, we suppose that $\mathscr{X}_1 \cap \mathscr{X}_2 = \emptyset$. Based on these considerations, one can derive a weighted graph $\mathscr{G}_{tr} = (\mathscr{A}_k, \mathscr{X}_1, w)$ such that $\mathscr{A}_k$ denotes an adjacency relation that connects each sample to its k-nearest neighbors, and $w : \mathcal{Z}_1 \times \mathcal{Z}_1 \rightarrow \Re^+$ concerns a function that computes arc weights between nodes. Figure 6.1 depicts a 2-nearest neighbors weighted graph derived from a training set with nine nodes[1].

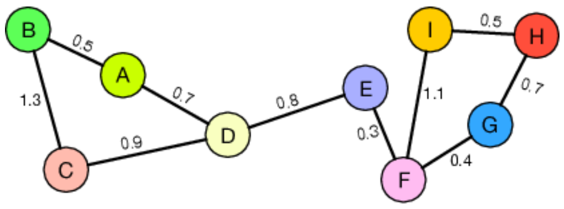

Figure 6.1: Toy example: a 2-nearest neighbors graph derived from a given training set. Edges are weighted by the distance between their corresponding nodes.

Given the graph $\mathcal{G}_{tr}$, the next step concerns computing the probability density function ρ of each node based on its neighborhood. The rationale behind computing such a value concerns the fact that the higher the density of some sample, the more likely it is to become a prototype. Unlike OPF with a complete graph, where the prototypes are placed nearby the decision boundary (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012), OPF$_{knn}$ aims at estimating prototypes in the highest density regions (PAPA; FERNANDES; FALCÃO, 2017). The pdf can be computed as follows:

$$\rho(\boldsymbol{x}_i) = \frac{1}{\sqrt{2\pi\sigma^2} \mid \mathcal{A}_k^*(\boldsymbol{x}_i) \mid} \sum_{\forall \boldsymbol{x}_j \in \mathcal{A}_k^*(\boldsymbol{x})} \exp\left(\frac{-d^2(\boldsymbol{x}_i, \boldsymbol{x}_j)}{2\sigma^2}\right), \tag{6.1}$$

where $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ denotes the distance between nodes $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ such that $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{Z}_1$ and $i \neq j$, $\sigma = \frac{d_f}{3}$, and d_f is the maximum arc weight in the training set. Besides, $\mathcal{A}_k^*(\boldsymbol{x}_i)$ denotes the adjacency relation for sample $\boldsymbol{x}_i$ considering the k^*-nearest neighbors, in which $k^* \in [1, k_{max}]$. Although Papa and Falcão (PAPA; FALCÃO, 2009) proposed to find k^* as the one that maximizes the classification accuracy over a validating set $\mathcal{G}_{val} \subset \mathcal{G}_{tr}$, in this work we considered $k^* = k_{max}$. Figure 6.2 displays the training graph from Figure 6.1 after density computation, in which the sample with the highest density value is chosen as the prototype, i.e., the dashed sample 'F'.

After the pdf computation, a competition process among prototypes takes place using the path-cost function C, which can be computed as follows:

[1]Since the symmetry is not guaranteed in the k-nearest neighbors graph, some nodes have degree greater than two.

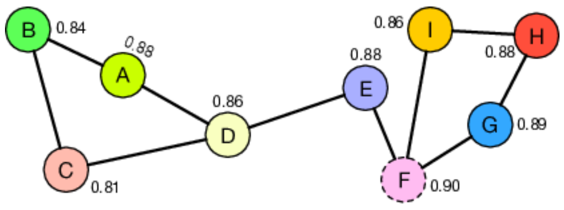

Figure 6.2: Density computation using Equation 6.1 concerning the graph depicted in Figure 6.1. The number nearby nodes stand for their density values.

$$
\begin{aligned}
C(\boldsymbol{x}_i) &= \begin{cases} \rho(\boldsymbol{x}_i) & \text{if } \boldsymbol{x}_i \in \mathscr{R}, \\ \rho(\boldsymbol{x}_i) - \delta & \text{otherwise} \end{cases} \\
C(\tau_{\boldsymbol{x}_j} \cdot (\boldsymbol{x}_j, \boldsymbol{x}_i)) &= \min\{C(\tau_{\boldsymbol{x}_j}), \rho(\boldsymbol{x}_i)\}. \qquad (6.2)
\end{aligned}
$$

where $\rho(\boldsymbol{x}_i) - \delta$ is employed to avoid plateaus nearby the maxima of the pdf such that δ is a small constant, $\mathscr{R} \subseteq \mathscr{Z}_1$ stands for the set of prototype (root) samples, $\tau_{\boldsymbol{x}_j}$ denotes a path with terminus at sample $\boldsymbol{x}_j$, and $(\boldsymbol{x}_j, \boldsymbol{x}_i) \in \mathscr{A}_{k^*}$ corresponds to an edge connecting samples $\boldsymbol{x}_j$ and $\boldsymbol{x}_i$.

The main idea of the competition process is to find the sample that maximizes Equation 6.2 for every training node. In the end, one has an optimum-path forest with trees rooted in the prototype samples, as depicted in Figure 6.3. The collection of these trees terms the name of the classifier.

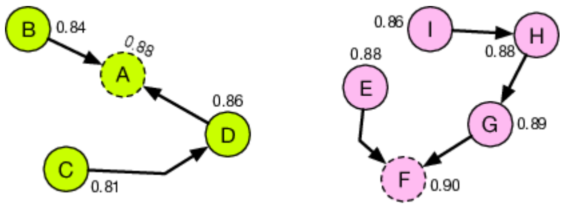

Figure 6.3: Optimum-path forest generated after the competition process using the densities shown in Figure 6.2 and arc weights computed in Figure 6.1. For the sake of explanation, we employed $\delta = 0.01$ in this example. Prototypes are highlighted, and the conquered samples are assigned to the same label (color) of its corresponding prototype (i.e., root of the tree).

A more detailed description of the OPF$_{knn}$ is implemented in Algorithm 1. Lines $1-3$ initialize the variables, and also inserts all samples in the priority queue Q. The main loop in Lines $5-14$ is in charge of the core of the algorithm. It first removes a sample $\boldsymbol{q}$ from Q with maximum path value $C(\boldsymbol{q})$ in Line 6. If $\boldsymbol{q}$ has not been conquered by any other sample, then $P(\boldsymbol{q}) = nil$ (Line 7) and $\boldsymbol{q}$ is a prototype of the connectivity map (a maximum of the pdf). Since $\boldsymbol{q} \in \mathscr{R}$, its connectivity value is reset to $\rho(\boldsymbol{q})$ (Line 8) according to the first part of Equation 6.2. It is also assigned to it the label of its corresponding conqueror for optimum-path propagation to the rest of its tree.

Algorithm 1: OPF$_{knn}$ algorithm

Input: λ-labeled graph $\mathcal{G}_{tr} = (\mathcal{Z}_1, \mathcal{A}_k^*)$ and δ value.
Output: Predecessor map P, path-cost map C, and label map L.
Auxiliary: Priority queue Q, density map ρ, and variable cst.

```
1  for all $\boldsymbol{x}_i \in \mathcal{G}_{tr}$ do
2      Compute $\rho(\boldsymbol{x}_i)$ using Equation 6.1;
3      $P(\boldsymbol{x_i}) \leftarrow nil$, $C(\boldsymbol{x_i}) \leftarrow \rho(\boldsymbol{x_i}) - \delta$, $Q \leftarrow \boldsymbol{x_i}$;
4  end
5  while $Q \neq \emptyset$ do
6      Remove from $Q$ a sample $\boldsymbol{q}$ such that $C(\boldsymbol{q})$ is maximum;
7      if $P(\boldsymbol{q}) = nil$ then
8          $L(\boldsymbol{q}) \leftarrow \lambda(\boldsymbol{q})$, and $C(\boldsymbol{q}) \leftarrow \rho(\boldsymbol{q})$;
9      end
10     for all $\boldsymbol{u} \in \mathcal{A}_k(\boldsymbol{q})$ such that $C(\boldsymbol{u}) < C(\boldsymbol{q})$ do
11         $cst \leftarrow \min\{C(\boldsymbol{q}), \rho(\boldsymbol{u})\}$;
12         if $(cst > C(\boldsymbol{u})$ then
13             $L(\boldsymbol{u}) \leftarrow L(\boldsymbol{q})$, $P(\boldsymbol{u}) \leftarrow \boldsymbol{q}$, $C(\boldsymbol{u}) \leftarrow cst$;
14             Update position of $\boldsymbol{u}$ in $Q$;
15         end
16     end
17 end
18 return $[P, C, L]$
```

The inner loop in Lines $10 - 14$ evaluates all adjacent sample u of $\boldsymbol{q}$ to which $\boldsymbol{q}$ can offer a better connectivity value (i.e., $C(\boldsymbol{u}) < C(\boldsymbol{q})$). If the path $\phi_{\boldsymbol{q}} \cdot (\boldsymbol{q}, \boldsymbol{u})$ offers a higher cost to $\boldsymbol{u}$ (Lines $11 - 12$), then the current path $\phi_{\boldsymbol{u}}$ is replaced by the new path $\phi_{\boldsymbol{q}} \cdot (\boldsymbol{q}, \boldsymbol{u})$, being the maps $C(\boldsymbol{u})$, $L(\boldsymbol{u})$, and $P(\boldsymbol{u})$ updated accordingly (Lines $13 - 14$).

6.2.2 Classification Step

The optimum-path forest computed in the training phase is then used to classify each test sample in the classification module (PAPA; FERNANDES; FALCÃO, 2017) as follows: each sample $\boldsymbol{x}_i \in \mathcal{Z}_2$ is connected to its k^*-nearest neighbors in the optimum-path forest computed earlier, and further it is evaluated the training sample that satisfies the criterium above:

$$C(\mathbf{x}_i) = \max_{\boldsymbol{x}_j \in \mathcal{A}_{k^*}(\boldsymbol{x}_i)} \{\min\{C(\mathbf{s}), \rho(\mathbf{t})\}\}, \tag{6.3}$$

such that $i \neq j$. Notice that $\boldsymbol{x}_i$ is labeled with the same label of the sample that conquers it. After this process, sample $\boldsymbol{x}_i$ is then removed from the graph.

6.3 Proposed Incremental Learning Approach

In this section, we present the proposed approach based on incremental learning for the Optimum-Path Forest classifier, hereinafter called IOPF$_{knn}$. As aforementioned, real-world problems usually suffer from two main drawbacks: (i) large datasets and (ii) dynamic environments. Both issues are somehow connected to each other, i.e., every time the statistics of the test set changes, there is a need to retrain the model again. However, when the training set is sufficiently large, it turns out to be unfeasible to update the model on-the-fly.

The issue addressed in this work, i.e., computer-aided non-technical losses identification, suits in both situations, i.e., energy companies now can keep track of hundreds of thousands of users, which, at the same time, are improving themselves and finding other ways to hack the system. Therefore, a classifier that has been trained recently may not be accurate enough to identify potential new fraudsters in the grid.

The proposed IOPF$_{knn}$ is capable of adding new training samples and self-updating without the need for re-training the whole set. Standard OPF requires a whole new training step, even if only one sample comes into the stage. The same happens with traditional pattern recognition techniques as well.

Let $\boldsymbol{x}' \in \Re^n$ be a new node that is going to be added to the training set. In this case, we must consider two important circumstances: (1) the k_{max}-nearest neighbors of $\boldsymbol{x}'$ and (2) the training nodes that have $\boldsymbol{x}'$ in their k^*-neighborhood, i.e., we need to find all training nodes that are now affected by this new node $\boldsymbol{x}'$. In other words, we are interested in finding the training nodes that have now $\boldsymbol{x}'$ in their k^*-neighborhood.

We are using $k^* = k_{max}$ since we assume the training set is growing sufficiently large and thus does not make sense to search for k^* within the interval $[1, k_{max}]$. If we have a situation in which a few samples are added to the training set, then we can assume k_{max} is suitable to be used. Notice that when the number of samples that are going to be added in the training set is considerably higher than k_{max}, one may need to consider re-training the classifier with the whole training set using standard OPF$_{knn}$. Figure 6.4 illustrates a new sample $\boldsymbol{x}'$ being added to the training set.

The proposed approach must address two distinct situations: (1) sample $\boldsymbol{x}'$ does not fall into the neighborhood of any other training sample, or (2) the other way around, i.e., sample $\boldsymbol{x}'$ does fall into the neighborhood of some training sample. Let us first analyze situation 1), which has also two different possibilities and are stated by Propositions 6.3.1 and 6.3.2.

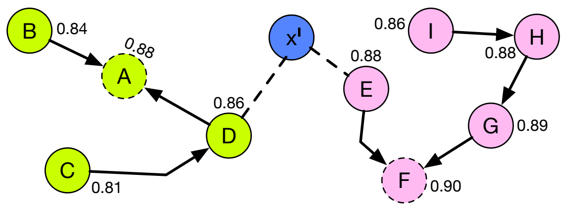

Figure 6.4: A new sample $\boldsymbol{x}'$ is added to the optimum-path forest pre-computed by standard OPF$_{knn}$, and further its k_{max}-nearest neighbors are found. We assume $k^* = 2$ in this example.

Proposition 6.3.1. *Let $\mathscr{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{k_{max}}\}$ be a set of samples such that $\boldsymbol{y}_i \in \mathscr{A}_{k_{max}}(\boldsymbol{x}')$. If $\rho(\boldsymbol{x}') \leq \rho(\boldsymbol{y}_i)$, $\forall \boldsymbol{y}_i \in \mathscr{Y}$, we have that $\boldsymbol{x}' \in \mathscr{R}$, i.e., $\boldsymbol{x}'$ will become a prototype and will not conquer any sample in $\mathscr{Y}$.*

Proof. Let us pick any sample $\boldsymbol{y}_i \in \mathscr{Y}$. Besides, let $C_{x'}(\boldsymbol{y}_i)$ be the cost that sample $\boldsymbol{x}'$ offers to node $\boldsymbol{y}_i$. When $\boldsymbol{x}'$ pops out from the priority queue Q, since it has no predecessors, i.e., $P(\boldsymbol{x}') = nil$, then $C(\boldsymbol{x}') = \rho(\boldsymbol{x}')$ (Line 8 in Algorithm 1). Besides, given that $\rho(\boldsymbol{x}') \leq \rho(\boldsymbol{y}_i)$, $C_{\boldsymbol{x}'}(\boldsymbol{y}_i) = \min\{C(\boldsymbol{x}'), \rho(\boldsymbol{y}_i))\} = \rho(\boldsymbol{y}_i)$. Since each training sample has its cost updated at least once (all samples are inserted in the priority queue Q in Line 3 of Algorithm 1) and the minimum cost a sample can be assigned is its density, we can state that $C(\boldsymbol{y}_i) \geq \rho(\boldsymbol{y}_i) \geq C_{\boldsymbol{x}'}(\boldsymbol{y}_i)$. Also, assuming the OPF employs the FIFO (first-in-first-out) policy, i.e., a sample that offers the minimum cost first has the priority over the conquered sample, $\boldsymbol{x}'$ will not conquer any sample within its neighborhood even if it offers the same optimum-path cost to it. Therefore, $\boldsymbol{x}'$ will become a prototype.

Proposition 6.3.2. *Let $\mathscr{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{k_{max}}\}$ be a set of samples such that $\boldsymbol{y}_i \in \mathscr{A}_{k_{max}}(\boldsymbol{x}')$. Suppose that $\exists \boldsymbol{y}_i \in \mathscr{Y}$ such that $\rho(\boldsymbol{y}_i) \leq \rho(\boldsymbol{x}')$. In this case, we have that $\boldsymbol{x}' \in \mathscr{R}$, i.e., $\boldsymbol{x}'$ will become a prototype and will not conquer any sample in $\mathscr{Y}$.*

Proof. Let $\boldsymbol{y}_i \in \mathscr{Y}$ be a sample such that $\rho(\boldsymbol{y}_i) \leq \rho(\boldsymbol{x}')$, $i = 1, 2, \ldots, k_{max}$. When $\boldsymbol{x}'$ is removed from the priority queue Q, it has no predecessors since it does not fall into the neighborhood of other samples. Therefore, $C(\boldsymbol{x}') = \rho(\boldsymbol{x}')$. Moreover, since $\rho(\boldsymbol{x}') \geq \rho(\boldsymbol{y}_i)$, it means that $\boldsymbol{x}'$ would be removed from Q prior to $\boldsymbol{y}_i$, i.e., there is a possibility that $\boldsymbol{x}'$ could conquer $\boldsymbol{y}_i$. Since $C_{\boldsymbol{x}'}(\boldsymbol{y}_i) = \min\{C(\boldsymbol{x}'), \rho(\boldsymbol{y}_i)\} = \rho(\boldsymbol{y}_i)$ and $C(\boldsymbol{y}_i) \geq \rho(\boldsymbol{y}_i)$, FIFO policy does not allow $\boldsymbol{y}_i$ to be conquered by $\boldsymbol{x}'$. Therefore, $\boldsymbol{x}'$ will not conquer any sample in its neighborhood, thus becoming a prototype.

We then shall consider situation 2), i.e., when $\boldsymbol{x}'$ falls in the neighborhood of some training sample. In this case, we must analyze a few distinct scenarios, as discussed below.

Proposition 6.3.3. *Let $\mathscr{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_M\}$ be a set of samples such that $\boldsymbol{x}' \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$. Besides, let $\boldsymbol{z}^\star$ be the predecessor of $\boldsymbol{y}_i$ in the optimum-path forest, i.e., $P(\boldsymbol{y}_i) = \boldsymbol{z}^\star$ (Figure 6.5). In this case, if $\rho(\boldsymbol{y}_i) < C(\boldsymbol{z}^\star)$, then $\boldsymbol{y}_i$ will not have its predecessor changed.*

Proof. Let $\rho'(\boldsymbol{y}_i)$ be the density value of $\boldsymbol{y}_i$ before adding $\boldsymbol{x}'$. Since $\boldsymbol{x}' \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$, then $\rho(\boldsymbol{y}_i) > \rho'(\boldsymbol{y}_i)$. Given that $\rho(\boldsymbol{y}_i) < C(\boldsymbol{z}^\star)$, $\boldsymbol{y}_i$ will be removed from Q after $\boldsymbol{z}^\star$, i.e., it is not possible for $\boldsymbol{y}_i$ to conquer $\boldsymbol{z}^\star$. In this case, $C_{\boldsymbol{z}^\star}(\boldsymbol{y}_i) = \min\{C(\boldsymbol{z}^\star), \rho(\boldsymbol{y}_i)\} = \rho(\boldsymbol{y}_i)$. Therefore, $C(\boldsymbol{y}_i) = \rho(\boldsymbol{y}_i)$.

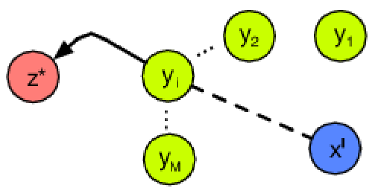

Figure 6.5: Illustration of the situation described by Proposition 6.3.3.

Proposition 6.3.4. *Let $\mathscr{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_M\}$ be a set of samples such that $\boldsymbol{x}' \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$. Besides, let $\boldsymbol{z}^\star$ be the predecessor of $\boldsymbol{y}_i$ in the optimum-path forest, i.e., $P(\boldsymbol{y}_i) = \boldsymbol{z}^\star$. In this case, if $\rho(\boldsymbol{y}_i) \geq C(\boldsymbol{z}^\star)$ and $\boldsymbol{z}^\star \in \mathscr{A}_{k^\star}(\boldsymbol{y}_i)$, then $\boldsymbol{y}_i$ will not conquer $\boldsymbol{z}^\star$ and neither any $\boldsymbol{a} \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$. Therefore, $\boldsymbol{y}_i$ will become a prototype.*

Proof. If $\rho(\boldsymbol{y}_i) \geq C(\boldsymbol{z}^\star)$, then $\boldsymbol{y}_i$ would be removed from Q before $\boldsymbol{z}^\star$, thus having no predecessor, i.e., $\boldsymbol{y}_i$ would become a prototype. Besides, $C_{\boldsymbol{y}_i}(\boldsymbol{z}^*) = \min\{C(\boldsymbol{y}_i), \rho(\boldsymbol{z}^*)\} = \rho(\boldsymbol{z}^*)$, given that $C(\boldsymbol{y}_i) \geq \rho(\boldsymbol{y}_i)$ and $C(\boldsymbol{z}^*) \geq \rho(\boldsymbol{z}^\star)$. Moreover, in case $\rho(\boldsymbol{z}^*) = C(\boldsymbol{y}_i)$, $\boldsymbol{y}_i$ would not conquer $\boldsymbol{z}^*$ due to the FIFO policy. Concerning the neighborhood of $\boldsymbol{y}_i$, let a sample $\boldsymbol{a} \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$. We have that $\boldsymbol{y}_i$ will not change $C(\boldsymbol{a})$ since $\rho(\boldsymbol{y}_i)$ has increased and, consequently, its cost.

Proposition 6.3.5. *Let $\mathscr{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_M\}$ be a set of samples such that $\boldsymbol{x}' \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$. If $C_{\boldsymbol{y}_i}(\boldsymbol{x}') > \rho(\boldsymbol{x}')$, then $\boldsymbol{x}'$ will be conquered by a sample $\boldsymbol{y}_i \in \mathscr{Y}$ that offers the maximum cost. Otherwise, i.e., if $C_{\boldsymbol{y}_i}(\boldsymbol{x}') \leq \rho(\boldsymbol{x}')$, then $\boldsymbol{x}'$ will become a prototype.*

Proof. If $C_{\boldsymbol{y}_i}(\boldsymbol{x}') \leq \rho(\boldsymbol{x}')$, $\boldsymbol{x}'$ will not be conquered by any sample in $\mathscr{Y}$ and will become a prototype. Besides, if $C_{\boldsymbol{y}_i}(\boldsymbol{x}') > \rho(\boldsymbol{x}')$, then Equation 6.3 holds:

$$C_{\boldsymbol{y}_i}(\boldsymbol{x}') = \max_{\boldsymbol{y}_i} \min\{C(\boldsymbol{y}_i), \rho(\boldsymbol{x}'))\}, \tag{6.4}$$

i.e., $\boldsymbol{x}'$ is conquered by $\boldsymbol{y}_i$.

Proposition 6.3.6. *Let $\boldsymbol{x}^*$ be a sample that belonged to the neighborhood of another node $\bar{\boldsymbol{x}}$ that now contains $\boldsymbol{x}'$ in its neighborhood. In such a situation, one needs to consider whether $\boldsymbol{x}^*$ will become a prototype or it will be conquered by some other sample.*

Proof. Let $\mathscr{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N\}$ be a set of samples such that $\boldsymbol{x}^\star \in \mathscr{A}_{k^*}(\boldsymbol{y}_i)$, and let $\boldsymbol{y}_i^\star \in \mathscr{Y}$ be the sample that satisfies the equation below:

$$C_{\boldsymbol{y}_i}(\boldsymbol{x}^\star) = \max_{\boldsymbol{y}_i} \min\{C(\boldsymbol{y}_i), \rho(\boldsymbol{x}^\star))\}. \tag{6.5}$$

If $C_{\boldsymbol{y}_i^\star}(\boldsymbol{x}^\star) \leq \rho(\boldsymbol{x}^\star)$, then $\boldsymbol{x}^\star$ will not be conquered by any sample in $\mathscr{Y}$, thus becoming a prototype. On the other hand, if $C_{\boldsymbol{y}_i^\star}(\boldsymbol{x}^\star) > \rho(\boldsymbol{x}^\star)$, then $\boldsymbol{x}^\star$ will be conquered by $\boldsymbol{y}_i^\star$.

6.4 Methodology

The experiments were divided into two rounds: (i) the first one aimed at evaluating the proposed approach in general-purpose datasets, while (ii) the second one assessed the robustness of IOPF$_{knn}$ under two datasets concerning NTL identification. All experiments aimed at comparing the proposed IOPF$_{knn}$ against its counterpart (i.e., naive) version OPF$_{knn}$ in terms of efficiency for training and effectiveness over the testing set. We expect that IOPF$_{knn}$ and OPF$_{knn}$ are similar in terms of recognition rates, being the former more effective during training.

The experimental protocol was conducted as follows. Let $\mathscr{D} = \mathscr{D}_1 \cup \mathscr{D}_2 \cup \mathscr{D}_3$ be a dataset such $\mathscr{D}_1$, $\mathscr{D}_2$ and $\mathscr{D}_3$ stand for the training, incremental, and testing sets, respectively. Besides, $|\mathscr{D}_1 \cup \mathscr{D}_2|$ stands for 50% of the entire dataset $\mathscr{D}$, whereas $\mathscr{D}_3$ accounts for the remaining 50%. The idea is to use $\mathscr{D}_1$ and $\mathscr{D}_2$ to assess the efficiency of IOPF$_{knn}$ when performing online training, while $\mathscr{D}_3$ is used for effectiveness purposes only[2].

Traditional OPF$_{knn}$ uses $\mathscr{D}_1 \cup \mathscr{D}_2$ as a single training set, while IOPF$_{knn}$ is evaluated as follows: it is first trained on $\mathscr{D}_1$ for further adding samples from $\mathscr{D}_2$ and then updating the model. Besides, the k_{max} value was set as a percentage of $\mathscr{D}_1$ and chosen empirically according to the dataset. Table 6.1 presents a brief description of each dataset and the k_{max} percentage used.

The first two datasets account for consumer profiles from commercial and industrial customers obtained from an electric power company in Brazil. The original datasets contain 8

[2]These percentages were set empirically.

Table 6.1: Dataset description and k_{max} percentages.

Dataset	**# of samples**	**# of features**	k_{max} ($\mathscr{D}_1$ percentage)
Commercial	14,856	4	1%
Industrial	9,546	4	1%
Hyperplane	90,000	10	0.1%
Cifar10	60,000	1,024	1%
Mnist	70,000	784	1%
Forest Covertype	581,012	54	0.01%
Electricity	45,312	8	1%
Poker Hand	829,201	10	0.01%

features (demand billed, demand contracted, demand measured or maximum demand, reactive energy, power transformer, power factor, installed power, and load factor), but we ended up with only the four most accurate ones for each dataset:

- Commercial: demand measured, power factor, installed power, and load factor.
- Industrial: demand contracted, demand measured, installed power, and load factor.

To accomplish such a purpose, we partitioned the datasets into training, validation, and test sets, respectively. The training dataset contains 50% of the original dataset, followed by 30% and 20% concerning the validation and test sets, respectively. The idea is to employ both training and validation sets to find the subset of features that maximize the accuracy over the test set, with the accuracy being the fitness function.

We used the well-known Particle Swarm Optimization (KENNEDY; EBERHART, 1995). Each agent is initialized with random binary positions and the original dataset is mapped to a new one that contains the features that were selected in this first sampling. In addition, the fitness function of each agent is set to the standard OPF (PAPA; FALCÃO; SUZUKI, 2009; PAPA et al., 2012) recognition rate over the validating set after training. The final subset will be the one that maximizes the curve over the range of values, i.e., the features that maximize the accuracy over the validating set. The accuracy over the test set is then assessed by using the final subset of the selected features. Notice the fitness function employed in this paper is the accuracy measure proposed by (PAPA; FALCÃO; SUZUKI, 2009), which is capable of handling unbalanced classes.

We also tripled the size of that datasets by introducing a small noise on each feature x, i.e., $x = x \pm \alpha$. In this paper we used $\alpha = 0.05$. The remaining datasets are used to assess the performance of the proposed approach in general-purpose domains.

We partitioned $\mathscr{D}_1 \cup \mathscr{D}_2$ into nine different configurations, i.e., 10% for the training set $\mathscr{D}_1$ and 90% for the incremental set $\mathscr{D}_2$ (i.e., $10-90$), 20% for the training set $\mathscr{D}_1$ and 80% for the incremental set $\mathscr{D}_2$ (i.e., $20-80$), and so on. Besides, each configuration has been randomly generated 15 times to allow statistical analysis and to report the mean accuracies and execution times.

The main idea of this experimental setup is to evaluate whether the proposed IOPF$_{knn}$ is preferable to update the training model with the incremental set rather than the traditional OPF$_{knn}$ training over the entire training set, i.e., $\mathscr{D}_1 \cup \mathscr{D}_2$. Concerning the OPF$_{knn}$ implementation, we employed the open-source library LibOPF (PAPA; SUZUKI; X,). Additionally, we considered the accuracy measure proposed by Papa et al. (PAPA; FALCÃO; SUZUKI, 2009) that takes into account unbalanced datasets, which is usually the case in the context of NTL identification.

6.5 Experiments

In this section, we discuss the experiments conducted to show the robustness of the proposed approach.

6.5.1 Non-Technical Losses Datasets

In the first experiment, we assessed the robustness of IOPF$_{knn}$ in the Commercial dataset using different setups, i.e., incremental sets with distinct sizes. The rationale is to verify if there is a specific limit in the incremental set size that tells us when it is good to prefer IOPF$_{knn}$ rather than OPF$_{knn}$. Figure 6.6 depicts such an initial experiment.

The first point to observe concerns the recognition rates over the test set. As aforementioned, we always used 50% of the entire dataset to build the training and incremental sets. Therefore, it would be expected that the accuracies reported in Figure 6.6 should be similar to each other regardless of the setting up. However, as described in Section 6.3, we set $k^* = k_{max}$, where the value of k_{max} varies according to the training set percentage. Therefore, a configuration of $10\% - 90\%$ uses a smaller value for k_{max} than the configuration of $90\% - 10\%$, thus constraining the pdf estimation to smaller neighborhoods. Such a statement explains why we have different accuracies for the very same training set size. Also, one can observe the accuracies are quite similar between IOPF$_{knn}$ and OPF$_{knn}$, thus showing the correctness of the

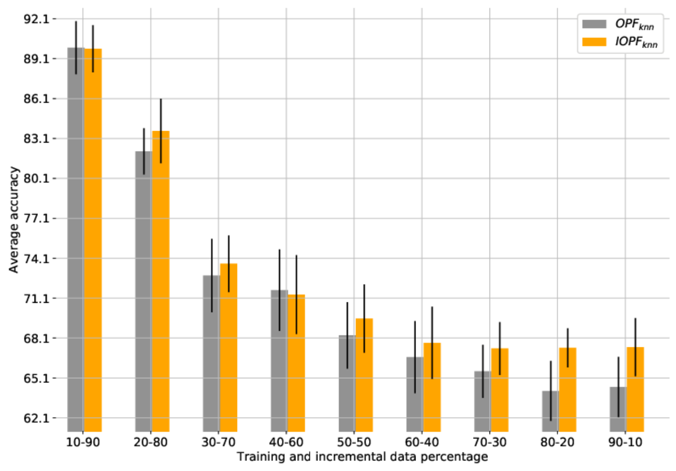

Figure 6.6: Experiment concerning Commercial dataset with different setups: recognition rates are computed over the test set.

idea presented in Section 6.3. A further statistical evaluation using the Wilcoxon signed-rank test (WILCOXON, 1945) with a significance of 5% revealed that four out of nine configurations obtained similar accuracies. Besides, the ones that were not considered alike differ up to 3% only.

Figure 6.7 depicts the training time considering the different configurations. Since we were motivated to provide a similar but faster solution, these results are the most important ones. Clearly, one can observe the training times provided by IOPF$_{knn}$ are consistently shorter than the training times spent by OPF$_{knn}$ (63% faster, on average).

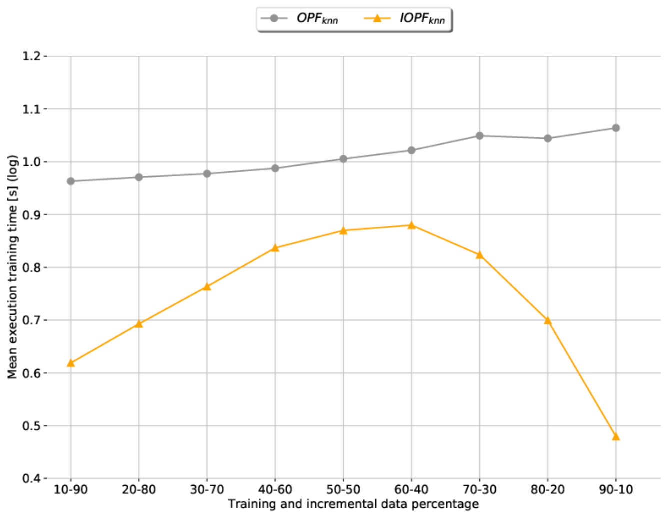

Figure 6.7: Experiment concerning Commercial dataset with different setups: training time [s].

The behavior of OPF$_{knn}$ is the expected one, i.e., the training time must not be that much different among the different configurations since the training set size is the same. Regarding IOPF$_{knn}$, the expected behavior, i.e., the smaller the incremental set, the less effective the proposed technique is, turns out to be regarded until it reaches the configuration $60\% - 40\%$. After that, IOPF$_{knn}$ becomes more effective, which can be explained by the complexity of some additional operations required, i.e., some modifications in the standard OPF$_{knn}$ to deal with online learning.

The next experiment concerns the Industrial dataset, which comprises profiles of large companies and industries. Usually, these are the consumers the electric power company is expected to deal with since they are in charge of the largest thefts. Figure 6.8 depicts the recognition rates under different configurations with respect to the dataset mentioned above.

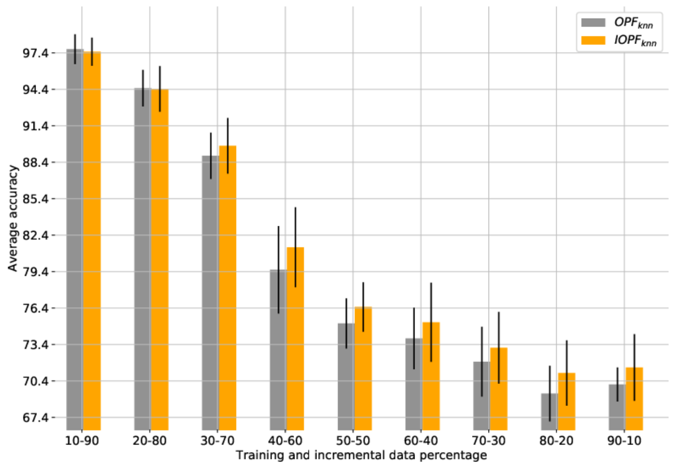

Figure 6.8: Experiment concerning Industrial dataset with different setups: recognition rates are computed over the test set.

Similar to what happened in the previous experiment, the recognition rates decreased as the training set percentage increases. The rationale behind such a scenario concerns the k_{max} parameter, which forces the pdf computation over larger neighborhoods, i.e., the rightmost columns. Basically, we are forcing fewer and bigger clusters, which may not reflect the real distribution of the data. The Wilcoxon test outcome tells us that three out of nine configurations are similar in between IOPF$_{kmax}$ and OPF$_{kmax}$. The main differences here concerning the recognition rates vary up to 2% only.

Figure 6.9 displays the mean training time concerning the Industrial dataset. Similar behavior to the one from the Commercial dataset was expected, i.e., IOPF$_{knn}$ has been faster than standard OPF$_{knn}$ for all configurations (33% faster). In real life, one shall use only one configuration for both efficiency and effectiveness, being the main idea of such experiments to stress the robustness of the proposed approach under different scenarios. The most important obser-

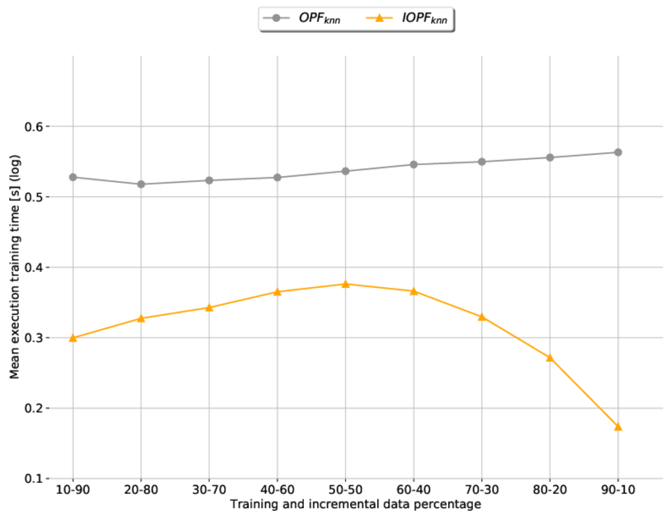

Figure 6.9: Experiment concerning Industrial dataset with different setups: training time [s].

vations we can draw come from two main points: (i) IOPF$_{knn}$ and OPF$_{knn}$ obtained very much close recognition rates in all configurations, and (ii) the proposed approach figured as the fastest one, thus corroborating the primary motivation of this work, i.e., to deal with dynamic scenarios that require online training.

6.5.2 General-Purpose Datasets

In this section, we present the experimental results concerning general-purpose datasets. We are motivated in a sense the proposed approach can be applied to a broader range of applications, not limited to non-technical loss identification, despite being our primary concern in this research.

Table 6.2 presents the mean recognition rates over the general-purpose datasets. For the sake of clarity, we presented and discussed the results concerning the configuration $50\% - 50\%$ only. We chose such a configuration since it figures training and incremental sets with the same size, i.e., we are not favoring either set. Also, this situation highlights an important aspect, i.e., the situations in which the proposed approach and standard OPF$_{knn}$ obtained similar and different recognition rates.

Concerning Hyperplane and Cifar10 datasets, both approaches obtained quite close recognition rates, while for Mnist, Electricity, and Poker Hand datasets standard OPF$_{knn}$ outperformed IOPF$_{knn}$. Besides, in the Forest Covertype dataset, the IOPF$_{knn}$ outperformed standard OPF$_{knn}$. The main problem related to medium-to-large datasets and sufficient large training and incremental sets concerns the possibility of several optimum-path forests. Although the total cost

Table 6.2: Recognition rates using the configuration $50-50$ concerning the general-purpose datasets.

Dataset	Configuration	OPF$_{knn}$	IOPF$_{knn}$
Hyperplane	50-50	58.72%	57.33%
Cifar10	50-50	55.59%	54.79%
Mnist	50-50	97.00%	91.26%
Forest Covertype	50-50	84.09%	87.04%
Electricity	50-50	69.86%	63.41%
Poker Hand	50-50	67.15%	64.42%

of those optimum-path forests is the same, the tie-zones (i.e., plateaus with the same density) may comprise samples from different classes. Therefore, depending on the order these samples are processed (i.e., the order they are added to the priority queue Q in Algorithm 1), we may have samples from different classes, but with the same cost conquering their neighbors. Such behavior will influence the final recognition rate.

The problem mentioned above has been subjected to research using different approaches, as stated by Fernandes et al. (FERNANDES; PAPA, 2019), which proposed an approach to alleviate the tie-zone problem in the context of standard OPF (i.e., the one that makes use of the complete graph). Although we could concentrate on a similar approach, we understand that it is not the focus of this work. If we choose another configuration for the Mnist dataset, e.g., $90\%-10\%$, the difference between OPF$_{knn}$ and IOPF$_{knn}$ is 1.73% only; or yet in Poker dataset with $90\%-10\%$, the difference between OPF$_{knn}$ and IOPF$_{knn}$ is 0.33%. Therefore, we argue that it is highly likely that one shall find a configuration that minimizes the difference between these approaches. Notice we are playing with the training and incremental sets only, i.e., the test set does not take part in the incremental learning step.

Now, we concentrate on the computational load for the sake of efficiency comparison. Once again, the idea is to evaluate whether the trade-off between accuracy and efficiency is worth using the proposed approach or not. Table 6.3 presents the mean execution time for training concerning OPF$_{kmax}$ and IOPF$_{kmax}$. However, we changed the configuration to $90\%-10\%$, which means we are using 90% for training purposes and only 10% to compose the incremental set. Even in that less favorable scenario, IOPF$_{kmax}$ performed faster than its counterpart approach. We mean "less favorable" for we are using much fewer incremental samples, i.e., we expect that the larger the incremental set, the faster IOPF$_{kmax}$ will be.

Table 6.3: Training time [s] using the configuration $90-10$ concerning the general-purpose datasets.

Dataset	**Configuration**	**OPF$_{knn}$**	**IOPF$_{knn}$**
Hyperplane	90-10	69.86	17.47
Cifar10	90-10	763.61	495.83
Mnist	90-10	810.71	610.98
Forest Covertype	90-10	5807.53	1012.13
Electricity	90-10	21.85	16.64
Poker Hand	90-10	6003.23	1549.73

6.5.3 Discussion

For most datasets in the experiments, we use a value of 1% of $\mathscr{D}_1$, except Hyperplane, which uses a percentage of 0.1%, and Covertype and Poker Hand, which uses a percentage of 0.01%. We chose the value of 1% for datasets with moderate size and the value of 0.1% and 0.01% for large datasets as the bottleneck of OPF_{knn} is in the chosen value of k_{max}. We can choose the percentage dimension of k_{max} depending on the size of the dataset. The k_{max} issue is an open problem, it is a parameter that the user will determine. If you do not have time problems, you can leave k_{max} with a larger size, up to the size of the training set. However, if computational time and computational cost are a problem, we recommend using a lower value for k_{max}.

Although we do not guarantee an exact match between IOPF$_{knn}$ and OPF$_{knn}$ in terms of recognition rates due to tie-zones and other characteristics of the algorithm, we showed the proposed approach is able to obtain very much close (sometimes statistically similar) results but being considerably faster for training purposes.

6.6 Partial Considerations

Incremental learning aims at dealing with dynamic scenarios and, at the same time, keeping the model updated and robust to changes in the data flow. Non-technical losses figure a well-known issue continuously faced by electric power companies, mainly in developing countries. In such a scenario, consumers try to hack the grid in order to pay fewer bills, but their profile changes too. Dealing with these thefts requires a classifier that can adjust itself quickly, thus adapting to the new consumers' behavior.

In this work, we proposed the IOPF$_{knn}$, a variant of the OPF$_{knn}$ classifier that is based on the incremental learning framework, i.e., when a new sample (or a subset of samples) is available for training, there is no need to train the model over the entire training set again. IOPF$_{knn}$ can adjust itself using that new samples only. Experiments on NTL and general-purpose datasets support the above statement. Concerning future works, we intend to elaborate on solutions that can deal with tie-zones efficiently, but without paying the price of a drop in efficiency.

Chapter 7

UNSUPERVISED OPF

In this chapter, we investigate the unsupervised OPF behavior in non-stationary environments. The unsupervised OPF algorithm was slightly modified for these experiments: since unsupervised OPF does not work with “true labels,” we make him able to calculate the accuracy of the clustering results. The unsupervised OPF here we called “OPF Cluster,” and for the “OPF Supervised,” we apply the complete graph OPF version.

We performed two steps of experiments. In the first stage of the experiments, inspired by the methodology used in Chapter 4, we follow the accuracy of the supervised and unsupervised OPF in different approaches: traditional, with a sliding window of one size and size 3, and full-memory. In the second stage, we used the accuracy decay to consider a concept drift and evaluated the detection with different threshold values. The datasets used in the experiments were Hyperplane, SEA, Usenet1, and Usenet2.

In the first stage, we performed four experiments. The first is the “traditional” way, and we train only in the first batch to classify the rest of the data stream. In the second and third experiments, we used a sliding window to go through the data and train a classifier with the most recent information; we use a window with one batch size for training and another with three batch size, respectively. In the fourth experiment, we used all the information previously received to train a classifier and further perform the classification.

Figure 7.1 depicts the accuracy rates of the first experiment considering the datasets mentioned above. We can see that the classifier trained in the first batch with supervised OPF has higher accuracy over the stream data than unsupervised OPF in Hyperplane and SEA datasets. Also, we observe that, in the Usenet datasets, the accuracy of both classifiers is similar.

Figure 7.2 depicts the accuracy rates of the second experiment, training with one batch as window size. As in the first experiment, the supervised OPF has higher accuracy over the stream

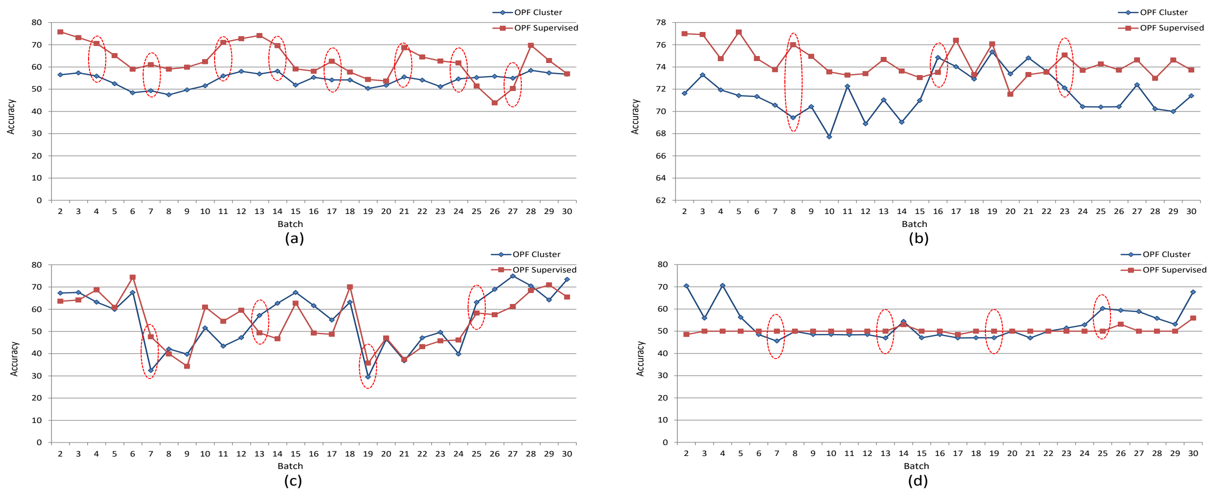

Figure 7.1: Accuracy over dataset stream in the first experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2. Some concept drifts are hilighted.

data in the Hyperplane dataset and is slightly superior in the SEA dataset. We also observe that the accuracy of both classifiers is similar in the Usenet datasets, being the unsupervised OPF a little better in some batches in the Usenet2 dataset.

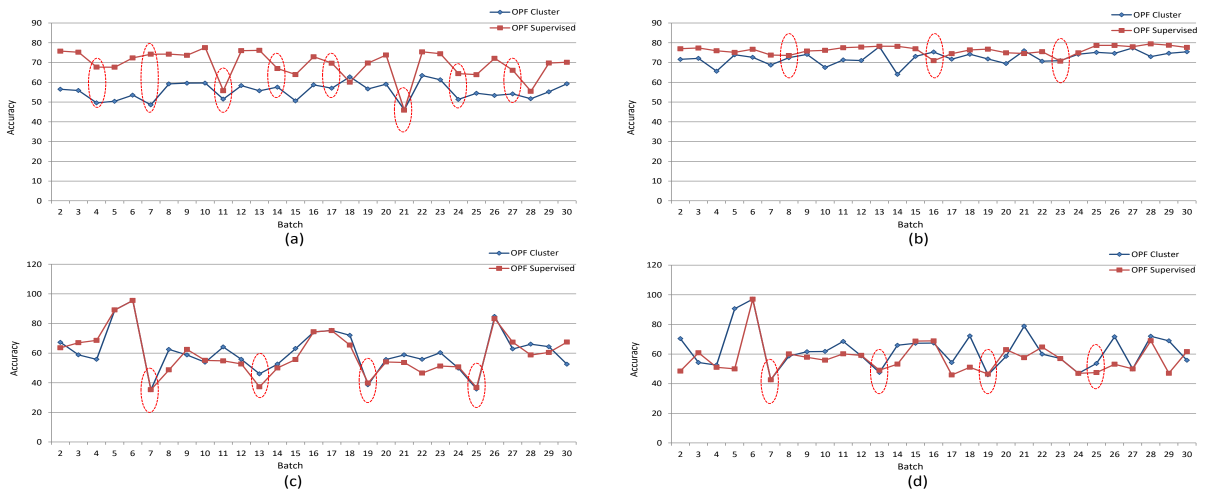

Figure 7.2: Accuracy over dataset stream in the second experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2. Some concept drifts are hilighted.

Figure 7.3 depicts the accuracy rates of the third experiment, with a training phase using three batches of window size. Again, the supervised OPF has higher accuracy over the stream data in the Hyperplane dataset and at the beginning of the SEA dataset. The accuracy in the Usenet datasets of both classifiers is similar.

Figure 7.4 depicts the accuracy rates of the fourth experiment, using all the previously seeing data to train a classifier. In the Hyperplane dataset, the supervised OPF has higher accuracy at the beginning, and from the middle of the stream onwards, both classifiers behave

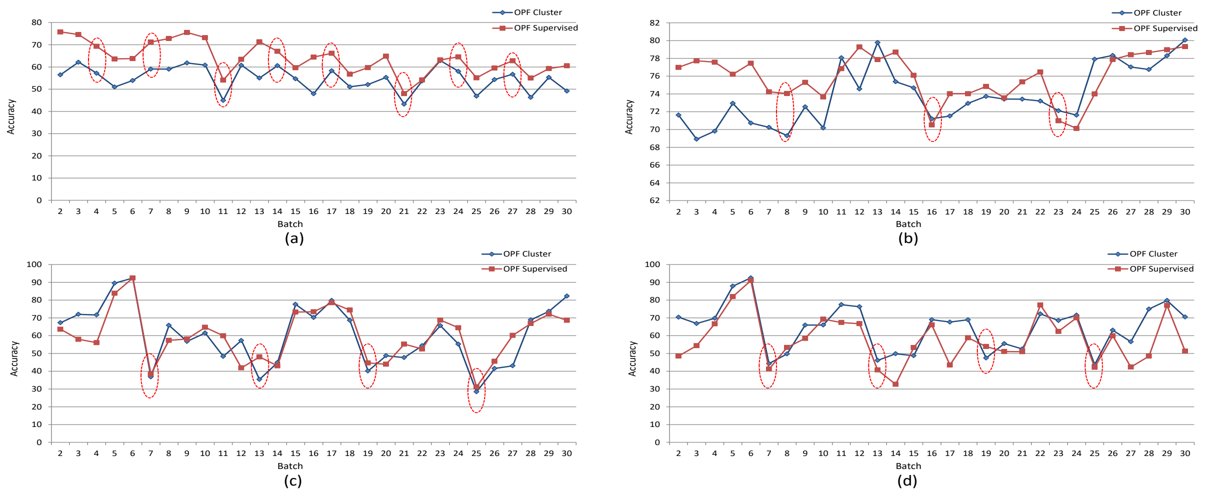

Figure 7.3: Accuracy over dataset stream in the third experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2. Some concept drifts are hilighted.

similarly. In the SEA dataset, the classifiers are similar, except at the first eight batches where supervised OPF performs better. The accuracy of unsupervised OPF in the Usenet1 dataset is slightly better in some moments, and in the Usenet2 dataset, the better performance varies between both classifiers.

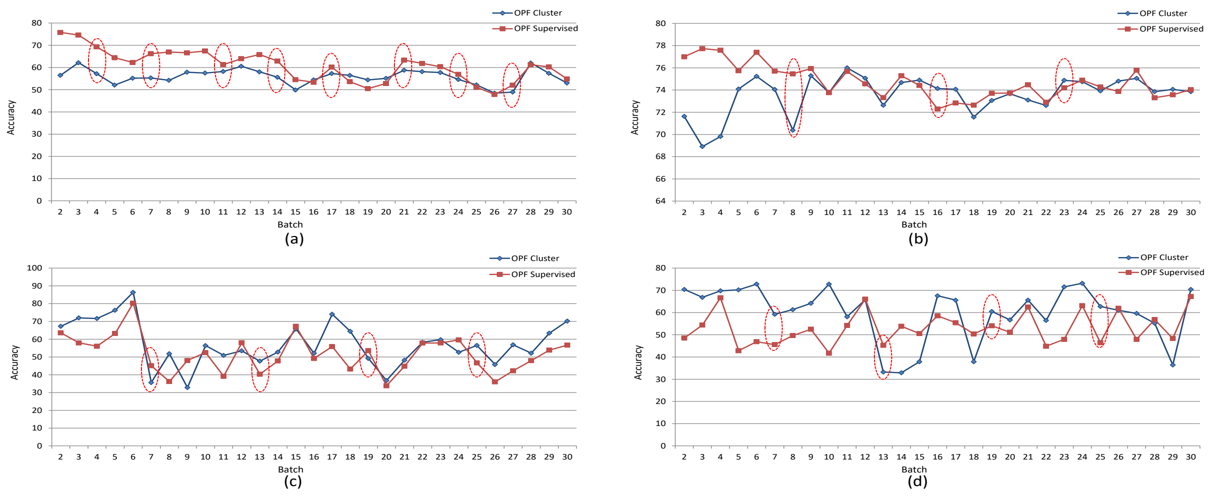

Figure 7.4: Accuracy over dataset stream in the fourth experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2. Some concept drifts are hilighted.

In the second stage, we performed the same four experiments as the previous stage but here measuring what we called “Accuracy in drift detection.” We estimated this accuracy with the number of drifts correctly assigned divided by the real number of drifts occurring in the dataset. The drift detection is determined through a threshold: when a drop in accuracy reaches a value ranging from 1%, 5%, 20%, 30%, 40%, and 50%, we assume that drift occurs.

Figure 7.5 depicts the accuracy in drift detection rates of the first experiment considering the datasets mentioned above. We can see that larger threshold values make the detection not be accused, contributing to the lack of accuracy in detecting drift. In Hyperplane, SEA, and Usenet2 datasets, the thresholds above 5% are already high. In Hyperplane, both classifiers performed similarly, and in SEA and Usenet2 datasets, the unsupervised OPF detect slightly better when the threshold is set to 1%. In the Usenet1 dataset, the supervised OPF performs better with 1% and 5% threshold, whereas the unsupervised OPF performs in the same way from 1% to 30% of a threshold value.

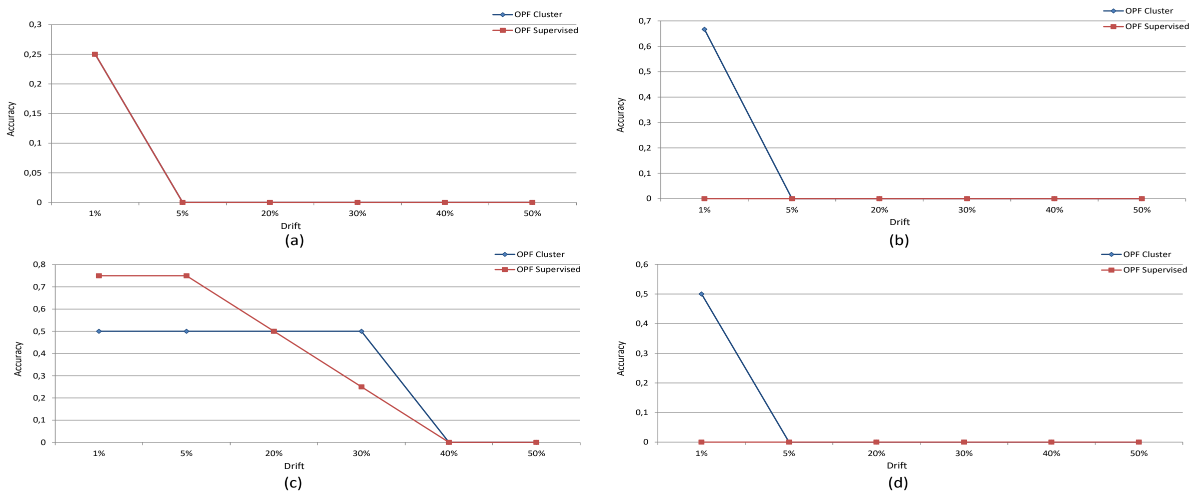

Figure 7.5: Accuracy in drift detection over dataset stream in the first experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2.

Figure 7.6 depicts the accuracy in drift detection rates of the second experiment. In the SEA dataset, unsupervised OPF was unable to detect drift, whereas, in Hyperplane, the threshold of 20% was also already high. In the Usenet1 dataset, both classifiers behaved similarly, but in threshold of 20% to 30%, the unsupervised OPF stayed with stable performance. In Usenet2, the unsupervised OPF performed better than supervised OPF when thresholds vary between 5% to 20%.

Figure 7.7 depicts the accuracy in drift detection rates of the third experiment. Again, in the SEA and Hyperplane datasets, the threshold of 20% was already high, hence supervised OPF and unsupervised OPF were unable to detect drift. In the Usenet1 and Usenet2 datasets, the unsupervised OPF behaves very well up until the threshold of 20%.

Figure 7.8 depicts the accuracy in drift detection rates of the fourth experiment. In the Hyperplane dataset, unsupervised OPF was unable to detect drift with a threshold above 5%, whereas this bad threshold for supervised OPF was 20%. In the Hyperplane dataset, when the threshold varies from 1% to 5%, the supervised OPF performed better than the unsupervised version. In the SEA dataset, they both behave equally. In the Usenet1 dataset, thresholds from

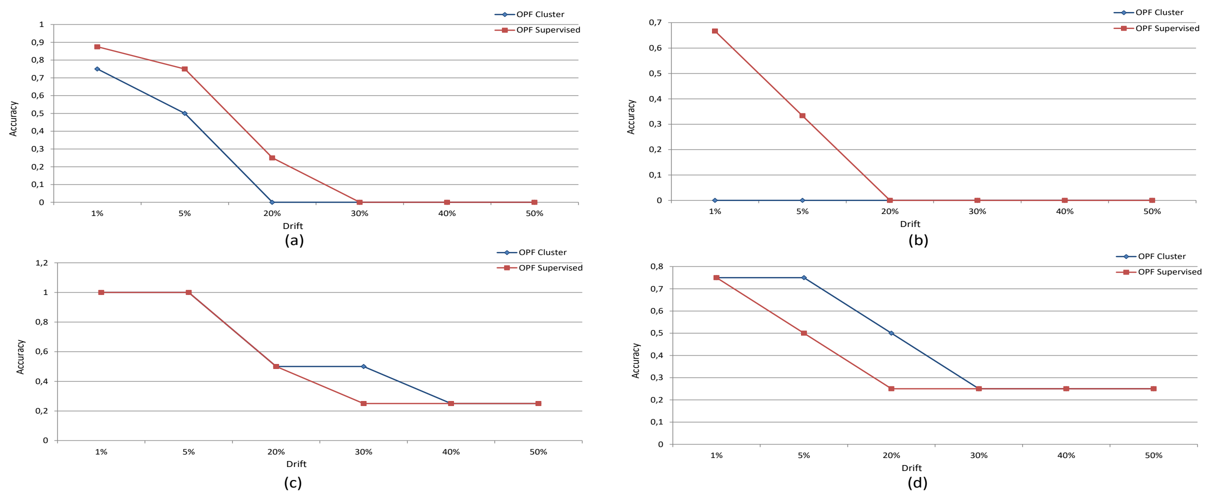

Figure 7.6: Accuracy in drift detection over dataset stream in the second experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2.

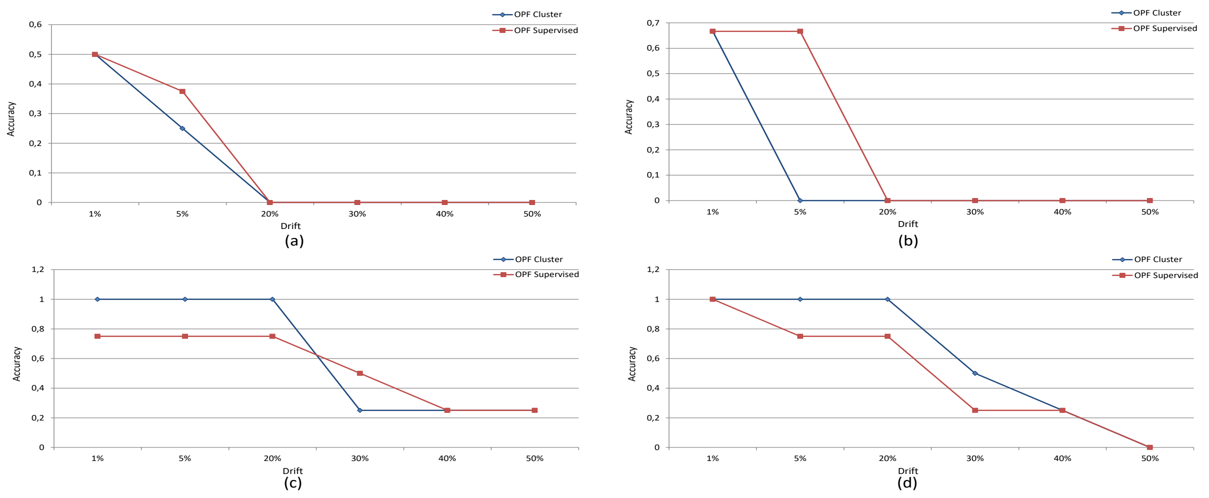

Figure 7.7: Accuracy in drift detection over dataset stream in the third experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2.

20% to 50% for unsupervised OPF did not alter the behave of values. In the Usenet2 dataset, unsupervised OPF has better accuracy when compared to supervised OPF with thresholds of 5% and 30%.

We noticed that the performance of unsupervised OPF was slightly worse than the supervised OPF. Despite this result, it was already expected that supervised OPF could overcome unsupervised OPF in accuracy since supervised OPF benefits from the information labels in the training phase. Regarding the accuracy of drift detection, as observed in the experiment results, the unsupervised OPF performed better in some situations, sometimes it was worse and sometimes behaved equally as the supervised OPF. We then consider that we can also apply unsupervised OPF in environments with concept drift.

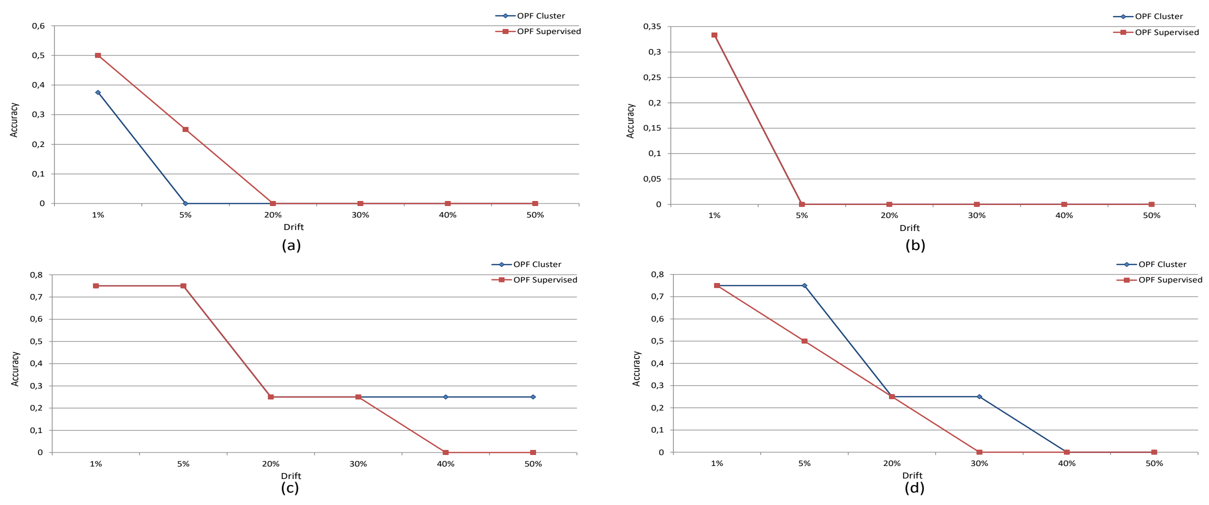

Figure 7.8: Accuracy in drift detection over dataset stream in the fourth experiment: a) Hyperplane b) SEA c) Usenet1 d) Usenet2.

Although we did not present in this work, we also performed experiments employing measures of dissimilarity in distributions to detect concept drift. Among the chosen measures, we applied Kullback-Leibler divergence (which, despite not being a distance metric, is pretty used in concept drift literature in unsupervised environments) and Hellinger distance to detect changes in the data distributions (GOLDENBERG; WEBB, 2019). However, we did not obtain relevant results, and we did not include them in the present work.

We verified the need for further studies on unsupervised OPF in environments where we do not have information about the labels. To determine desirable mechanisms for drift detection in these environments, as future works, we will examine other distance measures to apply in unsupervised data.

In situations where the available data is not labeled, we have to use some clustering method. As a concept drift example, at time t we can have n clusters, and at time $t+1$ we have n' clusters, $n \neq n'$. Identifying if a new class appears is a difficult scale problem (if a small cluster is close to another existing one), and detecting if there is a concept drift (new class appearance, or *Concept Evolution*) becomes a problem. The OPF approach has not an algorithm that forces a user-defined number of clusters. A positive point (in problems in which the number of clusters is unknown) of the unsupervised OPF is that it finds the number of clusters at runtime. As future works, we intend to fill this gap by proposing an algorithm to find a fixed number of clusters with the unsupervised OPF and to validate this approach in the problem of concept drift. Another area to be explored is: suppose that in time t we have n clusters, and in time $t+1$ we have $n' = n+1$ clusters. However, a more detailed analysis of this "new cluster" will tell us if it characterizes a concept drift (if this new cluster is significant). This new cluster may not contain a significant number of samples, being an irrelevant cluster. To solve this problem with the unsupervised OPF, we can propose two main approaches: (i) removing this irrelevant cluster or (ii) merging this cluster with the closest cluster or another more strongly connected cluster, creating an expanded cluster. By definition of closer, we can understand as the conventional distance between the prototypes of each cluster. By strongly connected cluster, we can understand as being the cluster that would offer the higher sum of costs (the unsupervised OPF aims to maximize the cost of each sample) to the merged irrelevant cluster.

Chapter 8

CONCLUSIONS

In this Ph.D. thesis, we proposed to apply supervised and unsupervised approaches based on the Optimum Path Forest algorithm in dynamic environments to deal with concept drift. We also proposed an incremental learning approach concerning supervised OPF named Incremental OPF$_{knn}$ (IOPF$_{knn}$). In Chapter 2, we presented a bibliographical review concerning concept drift. In Chapter 3, we introduced the Optimum Path Forest classifier in supervised versions (complete graph and k-nn graph) and unsupervised version. Thus, this Ph.D. thesis attempted to answer the questions: i) can OPF be efficient to handle non-stationary environments? ii) can OPF classifier be robust enough to address the concept drift problem? iii) can we provide an effective OPF implementation to handle dynamic environments?

Regarding an efficient OPF implementation to handle non-stationary environments, in Chapter 4, we presented an evaluation of the OPF classifier in a concept drift environment using *full-memory*, *no-memory*, and *window* approaches against other classifiers. In this experiment, we divided the database into a stream of batches that were received by the algorithm. The algorithm trained the first batch, and the following ones were classified and later used for training – a similar study to that done by Klinkenberg and Joachims (KLINKENBERG; JOACHIMS, 2000). As mentioned above, we employ supervised OPF in three ways: full-memory (Figure 2.2), no-memory (Figure 2.3), and a sliding window with fixed size (Figure 2.4). We compared standard OPF, OPF$_{knn}$, SVM, and three distinct versions to address concept drift on each technique in synthetic datasets. The experiments showed that OPF is suitable to work under these dynamic scenarios since its recognition rates were considerably better when adapted to address concept drift. The OPF versions have lower time-consuming training time compared to SVM versions, and in some datasets, they also have lower time-consuming testing time, which is a beneficial advantage. We also assessed the OPF classifier with a drift detection method based on accuracy in Section 5.5. In this study, we adopted the OPF classifier with a drift detection method based on accuracy values and also a window with dynamic size. The selection of a "good" window size in fixed sliding windows methods is a trade-off between fast adaptability to data (small window) and good generalization in phases where there is no concept drift (large window) (KLINKENBERG; JOACHIMS, 2000). The method used an empiric threshold to detect changes in accuracy values, and the window size is decreased to one batch-size when the accuracy drops below this threshold. If there is no drift, it adds the new batch to the training set for stability increasing. The experiments over real and synthetic datasets compared the standard OPF, OPF-fullmemory, OPF-nomemory, and OPF-window3 approaches, against the OPF-dynamic-window to address concept drift. We showed that the OPF-dynamic-window accuracy has similar accuracy as OPF-fullmemory or OPF-window3, depending on how the window behaves in the stream of data.

Concerning the OPF classifiers be robust enough to address the concept drift problem, in Chapter 5 we evaluated an ensemble of OPF classifiers in the concept drift problem. We used a committee of classifiers to make decisions using the previous approaches and with different decision rules to choose the final result. The three voting mechanisms were: "combined," "weighted" and "major." The "combined" voting mechanism chooses the most voted result among base classifier outputs as the classification result. The "weighted" voting mechanism weighs each base classifier with different values for voting relevance based on its previous accuracy classification. The "major" voting mechanism gives additional weight to the base classifier with higher accuracy in the anterior batch. The experiments in this method had three parts. In the first experiment, we combined the results of the full-memory, no-memory, and window approaches using the three voting mechanisms to obtain the committee's decision. In the second experiment, we used modified versions of the previous approach: for each full-memory, no-memory, and window method, we divide the data into three parts instead of using only one training set. In other words, in the no-memory version, the most recent batch is divided into three parts and trained with OPF, and the classification is the combination methods above. Similarly, the full-memory version includes all previously seen data into a large dataset and divided it into three parts. We trained each one with OPF, and the combination mechanisms made the decisions. In the window version, the sliding window of size three keeps the data and divided it into three parts, and the results are combined using one of the three voting mechanisms. In the third experiment, each batch trained one classifier: the committee considers the last three trained models for each new lot to be classified. The results are combined using the three approaches described above. The experiments over real and synthetic datasets compared standard OPF, OPF-fullmemory, OPF-nomemory, and OPF-window3 approaches, against the three proposed ensemble-based versions to address concept drift. We showed that the OPF ensemble is suitable to work under these dynamic scenarios since its recognition rates were considerably better compared to traditional OPF and OPF with concept drift handling with no ensemble learning. We can also verify that combined methods have better accuracy than the ones obtained on each fold individually in the second experiment. And we also discovered that the main advantage concerning the third experiment over OPF-window3 is its training time since training in smaller data is faster than a larger one, i.e., training three individual batches is faster than training three merged batches.

Regarding if we can provide an effective OPF implementation to handle dynamic environments, in Chapter 6, we proposed an OPF$_{knn}$ approach with incremental learning. Learning one sample at a time can be useful in large and continuous data stream environments. In this environment that requires the learner to be updated constantly, since repeating the entire learning process might be prohibitive, adjusting the model to the new data shows to be a better choice, particularly for real-time response applications. Whenever traditional supervised OPF executes the training step, it trains with the entire training data, regardless of the number of new samples added to data. This situation can be costly in cases where new instances are frequently added. Therefore, the idea to implement an incremental OPF that adds new instances to the optimum path forest and updates the costs of the trees without re-training the entire database is very convenient. We named our method Incremental OPF$_{knn}$ (IOPF$_{knn}$), in which the user interacts with the classifier constantly changing the labeled training dataset. We tested the problem in data provided by energy companies that made available data regularly, thus requiring the learner to be updated constantly. Although we do not guarantee an exact match between IOPF$_{knn}$ and OPF$_{knn}$ in terms of recognition rates due to tie-zones and some characteristics of the algorithm, we showed the proposed approach is able to obtain very much close (sometimes statistically similar) results but being considerably faster for training purposes.

We conducted some experiments with unsupervised OPF in non-stationary environments in Chapter 7. We compared unsupervised OPF with supervised OPF to visualize that it can also operate in these types of environments. We also performed experiments using distance measures to detect drift in non-supervised environments. However, we did not reach highly relevant results and did not present them in this work.

Concerning future works, we intend to extend IOPF$_{knn}$ work and elaborate solutions that can deal with tie-zones efficiently but without paying the price of a drop in efficiency. Unsupervised environments are a challenging area in terms of concept drift. Some authors use distance measures, statistical distances, dissimilarity measures, among others, to quantify and detect concept drift. Thus, as future works, we intend to do further study in more metrics used in the literature, and we also propose some methods to deal with concept evolution. Essentially, our proposals involve semi-supervised and unsupervised applications.

REFERENCES

ABDULSALAM, H.; SKILLICORN, D.; MARTIN, P. Classification using streaming random forests. *IEEE Transactions on Knowledge and Data Engineering*, v. 23, n. 1, p. 22–36, 2011.

AMORIM, W. P. et al. Improving semi-supervised learning through optimum connectivity. *Pattern Recognition*, v. 60, n. Supplement C, p. 72–85, 2016.

ARABMAKKI, E.; KANTARDZIC, M. Som-based partial labeling of imbalanced data stream. *Neurocomputing*, v. 262, p. 120 – 133, 2017.

BERTINI, J.; NICOLETTI, M.; ZHAO, L. Ensemble of complete p-partite graph classifiers for non-stationary environments. In: *IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2013. p. 1802–1809.

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

BUZAU, M. M. et al. Detection of non-technical losses using smart meter data and supervised learning. *IEEE Transactions on Smart Grid*, v. 10, n. 3, p. 2661–2670, 2019.

CHEN, C. H. et al. A GA-based approach for mining membership functions and concept-drift patterns. In: *IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2015. p. 2961–2965. ISSN 1089-778X.

CHEN, S.; HE, H. SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining. In: *International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2009. p. 522–529. ISSN 1098-7576.

DÍAZ, A. O. et al. Fast adapting ensemble: A new algorithm for mining data streams with concept drift. *The Scientific World Journal*, v. 2015, 2015.

DITZLER, G.; POLIKAR, R. An ensemble based incremental learning framework for concept drift and class imbalance. In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2010. p. 1–8. ISSN 1098-7576.

DITZLER, G.; POLIKAR, R. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 25, n. 10, p. 2283–2301, 2013. ISSN 1041-4347.

DITZLER, G.; POLIKAR, R.; CHAWLA, N. An incremental learning algorithm for non-stationary environments and class imbalance. In: *20th International Conference on Pattern Recognition (ICPR)*. [S.l.: s.n.], 2010. p. 2997–3000. ISSN 1051-4651.

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

ESCOVEDO, T. et al. Learning under concept drift using a neuro-evolutionary ensemble. *International Journal of Computational Intelligence and Applications*, v. 12, n. 04, p. 1340002, 2013.

FALCÃO, A.; STOLFI, J.; LOTUFO, R. The image foresting transform theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 26, n. 1, p. 19–29, 2004.

FAN, W. Systematic data selection to mine concept-drifting data streams. In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.]: ACM, 2004. p. 128–137.

FARID, D. et al. An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications*, v. 40, n. 15, p. 5895 – 5906, 2013.

FERNANDES, S. E. N.; PAPA, J. P. Pruning optimum-path forest ensembles using quaternion-based optimization. In: *2017 International Joint Conference on Neural Networks*. [S.l.: s.n.], 2017. (IJCNN), p. 984–991.

FERNANDES, S. E. N.; PAPA, J. P. Improving optimum-path forest learning using bag-of-classifiers and confidence measures. *Pattern Analysis and Applications*, Springer London, p. 703–716, 2019.

FERNANDES, S. E. N. et al. A probabilistic optimum-path forest classifier for non-technical losses detection. *IEEE Transactions on Smart Grid*, v. 10, n. 3, p. 3226–3235, 2019.

FERNANDES, S. E. N. et al. Pruning optimum-path forest ensembles using metaheuristic optimization for land-cover classification. *International Journal of Remote Sensing*, v. 38, p. 5736–5762, 2017.

GAMA, J. et al. Learning with drift detection. In: *SBIA Brazilian Symposium on Artificial Intelligence*. [S.l.]: Springer Verlag, 2004. v. 3171, p. 286–295.

GAMA, J. et al. A survey on concept drift adaptation. *ACM Computing Surveys*, ACM, v. 46, n. 4, p. 44:1–44:37, 2014.

GAO, J. et al. A general framework for mining concept-drifting data streams with skewed distributions. In: *Proceedings of the SIAM International Conference on Data Mining (SDM '07)*. [S.l.: s.n.], 2007.

GAO, Y. et al. Saccos: A semi-supervised framework for emerging class detection and concept drift adaption over data streams. *IEEE Transactions on Knowledge and Data Engineering*, p. 1–1, 2020.

GARCÍA, M. et al. Early drift detection method. In: *Fourth International Workshop on Knowledge Discovery from Data Streams*. [S.l.: s.n.], 2006. p. 77–86.

GENG, X.; SMITH-MILES, K. Incremental learning. In: ____. *Encyclopedia of Biometrics*. Boston, MA: Springer US, 2009. p. 731–735.

GOLDENBERG, I.; WEBB, G. I. Survey of distance measures for quantifying concept drift and shift in numeric data. *Knowledge & Information Systems*, v. 60, n. 2, p. 591 – 615, 2019. ISSN 02191377.

HAREL, M. et al. Concept drift detection through resampling. In: *Proceedings of the 31st International Conference on Machine Learning*. [S.l.]: JMLR Workshop and Conference Proceedings, 2014. p. 1009–1017.

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

HEGEDUS, I.; ORMÁNDI, R.; JELASITY, M. Massively distributed concept drift handling in large networks. *Advances in Complex Systems*, v. 16, n. 04n05, p. 1350021, 2013.

HEUSINGER, M.; RAAB, C.; SCHLEIF, F. Passive concept drift handling via variations of learning vector quantization. *Neural Computing and Applications*, 2020. ISSN 1433-3058. Available from Internet: <https://doi.org/10.1007/s00521-020-05242-6>.

IWASHITA, A. et al. A path- and label-cost propagation approach to speedup the training of the optimum-path forest classifier. *Pattern Recognition Letters*, Elsevier Science Inc., v. 40, p. 121–127, 2014.

IWASHITA, A. et al. An incremental optimum-path forest classifier and its application to non-technical losses identification. Computers and Electrical Engineering (submitted). 2021.

IWASHITA, A. S.; ALBUQUERQUE, V. H. C. de; PAPA, J. P. Learning concept drift with ensembles of optimum-path forest-based classifiers. *Future Generation Computer Systems*, v. 95, p. 198–211, 2019.

IWASHITA, A. S.; PAPA, J. P. Learning concept drift with optimum-path forest. In: *23rd Iberoamerican Congress on Pattern Recognition (CIARP)*. [s.n.], 2018. p. 13–15. Late breaking works. Available from Internet: <https://tinyurl.com/y2kcdc5p>.

IWASHITA, A. S.; PAPA, J. P. An overview on concept drift learning. *IEEE Access*, v. 7, p. 1532–1547, 2019.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. An ensemble of classifiers for coping with recurring contexts in data streams. In: *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*. [S.l.]: IOS Press, 2008. p. 763–764.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. *Incremental Clustering for the Classification of Concept-Drifting Data Streams*. 2008.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, v. 22, n. 3, p. 371–391, 2010. ISSN 0219-3116.

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

KHAMASSI, I. et al. Self-adaptive windowing approach for handling complex concept drift. *Cognitive Computation*, v. 7, n. 6, p. 772–790, 2015.

KIM, J. Y. et al. Detection for non-technical loss by smart energy theft with intermediate monitor meter in smart grid. *IEEE Access*, v. 7, p. 129043–129053, 2019.

KLINKENBERG, R. Using labeled and unlabeled data to learn drifting concepts. In: *Workshop notes of IJCAI-01 Workshop on Learning from Temporal and Spatial Data*. [S.l.]: AAAI Press, 2001. p. 16–24.

KLINKENBERG, R.; JOACHIMS, T. Detecting concept drift with support vector machines. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. [S.l.]: Morgan Kaufmann, 2000. p. 487–494.

KLINKENBERG, R.; RENZ, I. Adaptive information filtering: Learning in the presence of concept drifts. In: *Workshop Notes of the ICML/AAAI-98 Workshop Learning for Text Categorization*. [S.l.]: AAAI Press, 1998. p. 33–40.

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

KUBAT, M.; WIDMER, G. Adapting to drift in continuous domains. In: *Proceedings of the 8th European Conference on Machine Learning*. [S.l.]: Springer, 1995. p. 307–310.

KUNCHEVA, L. I. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In: *Proceedings of the 2nd Workshop SUEMA 2008 (ECAI 2008)*. [S.l.: s.n.], 2008. p. 5–10.

KURLEJ, B.; WOZNIAK, M. Active learning approach to concept drift problem. *Logic Journal of IGPL*, v. 20, n. 3, p. 550–559, 2012.

LI, P.; WU, X.; HU, X. Learning from concept drifting data streams with unlabeled data. *Neurocomputing*, Elsevier Science Publishers B. V., v. 92, p. 145–155, 2012. ISSN 0925-2312.

LI, P. et al. Learning concept-drifting data streams with random ensemble decision trees. *Neurocomputing*, v. 166, p. 68 – 83, 2015.

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

LIU, A.; LU, J.; ZHANG, G. Concept drift detection via equal intensity k-means space partitioning. *IEEE Transactions on Cybernetics*, Institute of Electrical and Electronics Engineers (IEEE), p. 1–14, 2020. ISSN 2168-2275. Available from Internet: <http://dx.doi.org/10.1109/TCYB.2020.2983962>.

LIU, S. et al. Concept drift detection for data stream learning based on angle optimized global embedding and principal component analysis in sensor networks. *Computers & Electrical Engineering*, v. 58, p. 327 – 336, 2017.

LOEFFEL, P. X.; MARSALA, C.; DETYNIECKI, M. Classification with a reject option under concept drift: The droplets algorithm. In: *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. [S.l.: s.n.], 2015. p. 1–9.

LU, N.; ZHANG, G.; LU, J. Concept drift detection via competence models. *Artificial Intelligence*, v. 209, p. 11 – 28, 2014.

MARTINS, A. V. et al. Non-intrusive energy meter for non-technical losses identification. *IEEE Transactions on Instrumentation and Measurement*, p. 1–1, 2019.

MASUD, M. et al. Detecting recurring and novel classes in concept-drifting data streams. In: *IEEE 11th International Conference on Data Mining*. [S.l.: s.n.], 2011. p. 1176–1181.

MASUD, M. et al. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, v. 23, n. 6, p. 859–874, 2011.

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

MIRZA, B.; LIN, Z.; LIU, N. Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift. *Neurocomputing*, v. 149, Part A, p. 316 – 329, 2015.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

MITCHELL, T. M. et al. Experience with a learning personal assistant. *Communications of the ACM*, ACM, New York, NY, USA, v. 37, n. 7, p. 80–91, July 1994. ISSN 0001-0782.

PAPA, J.; FALCÃO, A.; SUZUKI, C. Supervised pattern classification based on optimum-path forest. *International Journal of Imaging Systems and Technology*, Wiley-Interscience, v. 19, n. 2, p. 120–131, 2009.

PAPA, J. et al. Efficient supervised optimum-path forest classification for large datasets. *Pattern Recognition*, v. 45, n. 1, p. 512 – 520, 2012.

PAPA, J. P.; FALCÃO, A. X. A new variant of the optimum-path forest classifier. In: *Proceeding of the 4th International Symposium on Advances in Visual Computing*. Berlin, Heidelberg: Springer-Verlag, 2008. p. 935–944. ISBN 978-3-540-89638-8.

PAPA, J. P.; FALCÃO, A. X. A new variant of the optimum-path forest classifier. In: *Advances in Visual Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. v. 5358, p. 935–944.

PAPA, J. P.; FALCÃO, A. X. A learning algorithm for the optimum-path forest classifier. In: TORSELLO, A.; ESCOLANO, F.; BRUN, L. (Ed.). *Graph-Based Representations in Pattern Recognition*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5534). p. 195–204. ISBN 978-3-642-02123-7.

PAPA, J. P. et al. Efficient supervised optimum-path forest classification for large datasets. *Pattern Recognition*, Elsevier Science Inc., New York, NY, USA, v. 45, n. 1, p. 512–520, 2012.

PAPA, J. P.; FALCÃO, A. X.; SUZUKI, C. T. M. Supervised pattern classification based on optimum-path forest. *International Journal of Imaging Systems and Technology*, John Wiley & Sons, Inc., New York, NY, USA, v. 19, n. 2, p. 120–131, 2009. ISSN 0899-9457.

PAPA, J. P.; FERNANDES, S. E. N.; FALCÃO, A. X. Optimum-path forest based on k-connectivity: Theory and applications. *Pattern Recognition Letters*, v. 87, n. 1, p. 117–126, 2017.

PAPA, J. P.; SUZUKI, C. T. N.; X, A. Libopf: A library for the design of optimum-path forest classifiers. Software version 2.1 available at http://www.ic.unicamp.br/ afalcao/libopf/index.html.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PONTI, M.; RIVA, M. An incremental linear-time learning algorithm for the optimum-path forest classifier. *Information Processing Letters*, v. 126, p. 1 – 6, 2017.

PONTI, M. P.; PAPA, J. P. Improving accuracy and speed of optimum-path forest classifier using combination of disjoint training subsets. In: SANSONE, C.; KITTLER, J.; ROLI, F. (Ed.). *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 237–248. ISBN 978-3-642-21557-5.

PONTI, M. P.; ROSSI, I. Ensembles of optimum-path forest classifiers using input data manipulation and undersampling. In: ZHOU, Z.; ROLI, F.; KITTLER, J. (Ed.). *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 236–246.

RAMOS, C. C. O. et al. On the study of commercial losses in brazil: A binary black hole algorithm for theft characterization. *IEEE Transactions on Smart Grid*, v. 9, n. 2, p. 676–683, 2016.

RAMOS, C. C. O. et al. A new approach for nontechnical losses detection based on optimum-path forest. *IEEE Transactions on Power Systems*, v. 26, n. 1, p. 181–189, 2011.

RAZA, H.; PRASAD, G.; YUHUA, L. Adaptive learning with covariate shift-detection for non-stationary environments. In: *14th UK Workshop on Computational Intelligence (UKCI)*. [S.l.: s.n.], 2014. p. 1–8.

RAZAVI, R.; FLEURY, M. Socio-economic predictors of electricity theft in developing countries: An indian case study. *Energy for Sustainable Development*, v. 49, p. 1 – 10, 2019.

RIBEIRO, P. B.; PAPA, J. P.; ROMERO, R. A. F. An ensemble-based approach for breast mass classification in mammography images. In: *SPIE Medical Imaging*. [S.l.: s.n.], 2017. p. 101342N–1–101342N–8.

ROCHA, L.; CAPPABIANCO, F.; FALCÃO, A. Data clustering as an optimum-path forest problem with applications in image analysis. *International Journal of Imaging Systems and Technology*, Wiley Periodicals, v. 19, n. 2, p. 50–68, 2009.

RODRIGUES, D. et al. Eeg-based person identification through binary flower pollination algorithm. *Expert Systems with Applications*, v. 62, p. 81–90, 2016.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995. ISBN 0-13-103805-2.

SALGANICOFF, M. Density-adaptive learning and forgetting. In: *Proceedings of the Tenth International Conference on Machine Learning*. [S.l.: s.n.], 1993. p. 276–283.

SCHLIMMER, J.; GRANGER, R. Beyond incremental processing: Tracking concept drift. In: KEHLER, T.; ROSENSCHEIN, S. (Ed.). *Proceedings of the Fifth National Conference on Artificial Intelligence*. [S.l.: s.n.], 1986. p. 502–507.

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

SETHI, T. S.; KANTARDZIC, M. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, v. 82, p. 77 – 99, 2017.

SILVA, B. da et al. Dealing with non-stationary environments using context detection. In: *Proceedings of the 23rd International Conference on Machine Learning*. [S.l.]: ACM, 2006. p. 217–224.

SILVA, J. de A.; HRUSCHKA, E. R.; GAMA, J. An evolutionary algorithm for clustering data streams with a variable number of clusters. *Expert Systems with Applications*, v. 67, p. 228 – 238, 2017.

SPINOSA, E. et al. Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. [S.l.]: ACM, 2008. (SAC'08), p. 976–980. ISBN 978-1-59593-753-7.

STANLEY, K. *Learning Concept Drift with a Committee of Decision Trees*. [S.l.], 2003.

STREET, W.; 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*. [S.l.]: ACM, 2001. p. 377–382.

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.

SUSNJAK, T.; BARCZAK, A.; HAWICK, K. 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.

TENNANT, M. et al. Scalable real-time classification of data streams with concept drift. *Future Generation Computer Systems*, v. 75, p. 187 – 199, 2017.

TSYMBAL, A. *The Problem of Concept Drift: Definitions and Related Work*. [S.l.], 2004.

VIVEKANANDAN, P.; NEDUNCHEZHIAN, R. Mining data streams with concept drifts using genetic algorithm. *Artificial Intelligence Review*, Kluwer Academic Publishers, v. 36, n. 3, p. 163–178, 2011.

WADEWALE, K.; DESAI, S. Survey on method of drift detection and classification for time varying data set. *International Research Journal of Engineering and Technology*, v. 2, n. 9, p. 709–713, 12 2015.

WANG, H.; ABRAHAM, Z. Concept drift detection for imbalanced stream data. *The Computing Research Repository (CoRR)*, abs/1504.01044, 2015.

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.

WIDYANTORO, D. Exploiting unlabeled data in concept drift learning. *Jurnal Informatika*, v. 8, n. 1, p. 54–62, 2007. ISSN 1411-0105.

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

XU, S.; WANG, J. Dynamic extreme learning machine for data stream classification. *Neurocomputing*, v. 238, p. 433 – 449, 2017.

YALCIN, A.; ERDEM, Z.; GURGEN, F. Ensemble based incremental SVM classifiers for changing environments. In: *22nd International Symposium on Computer and Information Sciences (ISCIS)*. [S.l.: s.n.], 2007. p. 1–5.

YANG, Y.; WU, X.; ZHU, X. *Combining Proactive and Reactive Predictions for Data Streams*. New York, NY, USA: ACM, 2005. 710–715 p. (KDD ’05).

ZAREMOODI, P.; BEIGY, H.; SIAHROUDI, S. K. Novel class detection in data streams using local patterns and neighborhood graph. *Neurocomputing*, v. 158, p. 234 – 245, 2015.

ZAREMOODI, P.; SIAHROUDI, S. K.; BEIGY, H. A support vector based approach for classification beyond the learned label space in data streams. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2016. (SAC ’16), p. 910–915.

ZHANG, Y. et al. Three-layer concept drifting detection in text data streams. *Neurocomputing*, v. 260, p. 393 – 403, 2017.

ZLIOBAITE, I. Learning under concept drift: an overview. *Computing Research Repository by Cornell library*, p. –1–1, 2010.

GLOSSARY

ADAGoLF – *Age-based Drift handling Gossip Learning Framework*

ADWIN – *ADaptive WINdowing*

ALCSD – *Adaptive Learning with Covariate Shift-Detection*

CCP – *Conceptual Clustering and Prediction*

CDC – *Concept Drift Committee*

CDDGoLF – *Concept Drift Detection Gossip Learning Framework*

CDGFM – *Concept Drift Genetic-Fuzzy Mining*

CPp-AbDG – *Complete P-partite Attribute-based Decision Graph*

CV – *Conceptual Vectors*

DARLING – *Density-Adaptive Reinforcement Learning*

DDD – *Diversity for Dealing with Drifts*

DDM – *Drift Detection Method*

DELM – *Dynamic Extreme Learning Machine*

DWM – *Dynamic Weighted Majority*

ECSMiner – *Enhanced Classifier for data Streams with novel class Miner*

EDDM – *Early Drift Detection Method*

EDIST2 – *Error Distance Approach for Drift Detection and Monitoring*

EDTC – *Ensemble Decision Trees for Concept drifting data streams*

EI-kMeans – *Equal Intensity k-means space partitioning*

EM – *Ensemble Model*

ESOS-ELM – *Ensemble of Subset Online Sequential Extreme Learning Machine*

FAE – *Fast Adapting Ensemble*

FDCD – *Financial Distress Concept Drift*

FDP – *Financial Distress Prediction*

FEAC-Stream – *Fast Evolutionary Algorithm for Clustering data Streams*

FLORA – *FLOating Rough Approximation*

FRANN – *Floating Rough Approximation in Neural Networks*

GLVQ – *Generalized Learning Vector Quantization*

KBS – *Knowledge-Based Sampling*

KDD – *Knowledge Discovery and Data mining*

LFR – *Linear Four Rates*

LOCE – *LOcal Classifier Ensemble*

Learn ++.CDS – *Learn++ with the Synthetic Minority class Oversampling TEchnique*

Learn ++.NIE – *Learn++ and Nonstationary and Imbalanced Environments*

Learn ++.NSE – *Learn++ and NonStationary Environments*

Learn ++.SMOTE – *Learn++ and SMOTE*

MC-NN – *Micro-Cluster Nearest Neighbour*

MD3 – *Margin Density Drift Detection*

MST – *Minimum Spanning Tree*

NEVE – *Neuro-EVolutionary Ensemble*

NTL – *Non-Technical Loss*

OLINDDA – *OnLIne Novelty and Drift Detection Algorithm*

OPF$_{knn}$ – *Optimum-Path Forest with k-neighborhood*

OPFI – *Optimum-Path Forest Incremental*

OPF – *Optimum-Path Forest*

OPT – *Optimum-Path Trees*

RBF – *Radial Basis Functions*

RL-CD – *Reinforcement Learning with Context Detection*

RLS-SOM – *Reduced labeled Samples-Self Organizing Map*

RSLVQ – *Robust Soft Learning Vector Quantization*

RePro – *REactive plus PROactive*

SACCOS – *Semi-supervised Adaptive ClassifiCation Over data Stream*

SCANR – *Stream Classifier And Novel and Recurring class detector*

SEA – *Streaming Ensemble Algorithm*

SERA – *SElectively Recursive Approach*

SUN – *Semi-supervised classification algorithm for data streams with concept drifts and UNlabeled data*

SVMLearn++ – *SVM and Learn++*

SVM – *Support Vector Machines*

SVSCLASS – *Support Vector-based Stream CLASSifier*

UCB – *Uncorrelated Bagging*

k-NN – *k-Nearest Neighbor*

Appendix A
LEARNING CONCEPT DRIFT WITH ENSEMBLES OF OPTIMUM-PATH FOREST-BASED CLASSIFIERS

This appendix extends the work presented by Iwashita et al. (IWASHITA; ALBUQUERQUE; PAPA, 2019). We performed the same experiments described in Chapter 5 with the synthetic datasets: Hyperplane and SEA. Regarding the real-world dataset, we used the Forest Covertype, which contains 581,012 samples and 54 features, being used by many stream-based classifiers.

Although the idea is to assess the OPF robustness under concept drift, we also included naïve SVM with a Radial Basis Function kernel optimized through cross-validation for comparison purposes. The "traditional SVM" parameters were optimized as follows: we divided the first batch in 50% to compose the training set and the remaining 50% to compose the validating set (used to optimize kernel parameters). Also, we consider the following proportions for the training and validating sets: $60\% - 40\%$, $70\% - 30\%$, $80\% - 20\%$ and $90\% - 10\%$. Such procedure was performed for all datasets, and we used the parameters that maximized SVM accuracy over the validating set of the aforementioned configurations (i.e., training and validating set percentages). The other SVM versions use a grid search parameter estimation methodology with $90\% - 10\%$ of training and validating sets, respectively. Regarding SVM implementation, we used scikit-learn (PEDREGOSA et al., 2011). Finally, we employed an accuracy measure proposed by Papa et al. (PAPA; FALCÃO; SUZUKI, 2009).

The figures below contain the results presented in Chapter 5, but in this appendix, we add the results of SVM versions.

Figures A.1 to A.3 show the results evaluated in Experiment 1 and described in Section 5.2. Figure A.1 depicts the results concerning the Covertype dataset. We can observe that standard

SVM obtained the best results among the SVM versions. Table A.1 presents the mean accuracy and standard deviation concerning all batches.

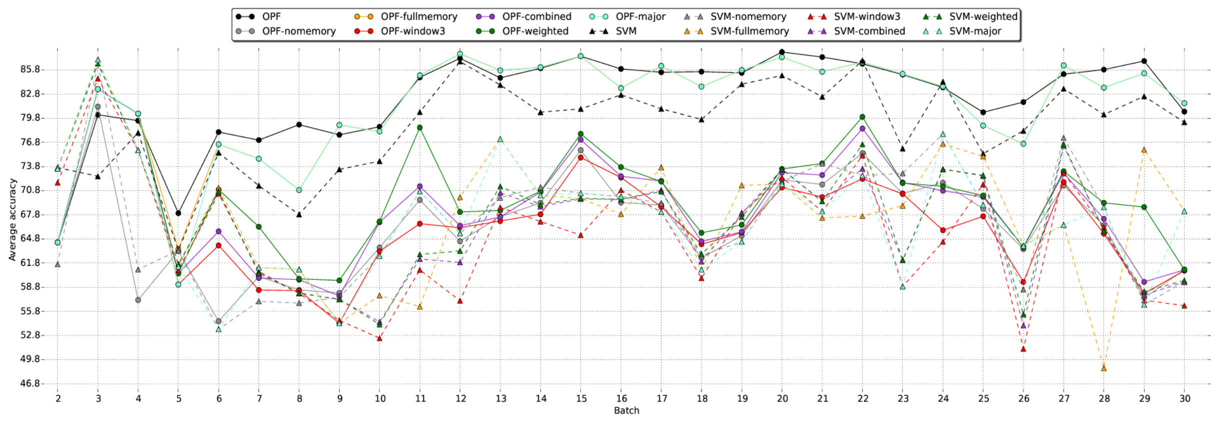

Figure A.1: Experimental results concerning Covertype dataset with respect to Experiment 1.

Table A.1: Covertype dataset with respect to Experiment 1.

Type	Accuracy
OPF	82.32752648 ± 5.527916684
OPF-nomemory	66.340352 ± 6.205296248
OPF-fullmemory	81.51895655 ± 6.360053157
OPF-window3	66.63647852 ± 6.483513342
OPF-combined	68.55474907 ± 6.447446854
OPF-weighted	70.03032828 ± 6.163191174
OPF-major	81.3674181 ± 6.822249211
SVM	78.7832119 ± 5.549011807
SVM-nomemory	66.69459997 ± 7.552454182
SVM-fullmemory	67.99529252 ± 7.904770131
SVM-window3	65.19059759 ± 7.751632536
SVM-combined	66.92436969 ± 7.48059647
SVM-weighted	67.21876483 ± 7.472223996
SVM-major	67.27513031 ± 7.29145594

Figure A.2 depicts the results concerning the Hyperplane dataset. The SVM versions in batches #6 to #10 seem to has better performance than traditional SVM, but in batch #11 they present a drop in performance whereas traditional SVM has an improvement. The SVM-major in some batches has interesting accuracy values. Table A.2 presents the mean accuracy and standard deviation concerning all batches.

Figure A.3 depicts the results concerning the SEA dataset. Among the SVM versions, the results seem to be more accurate than traditional SVM, with some batches of exceptions. Table A.3 presents the mean accuracy and standard deviation concerning all batches.

Figures A.4 to A.6 show the results evaluated in Experiment 2. Figure A.4 depicts the results concerning the Covertype dataset in full-memory management. We can see that the SVM versions have better behavior than SVM-fullmemory. Table A.4 presents the mean accuracy and standard deviation concerning all batches in full-memory management.

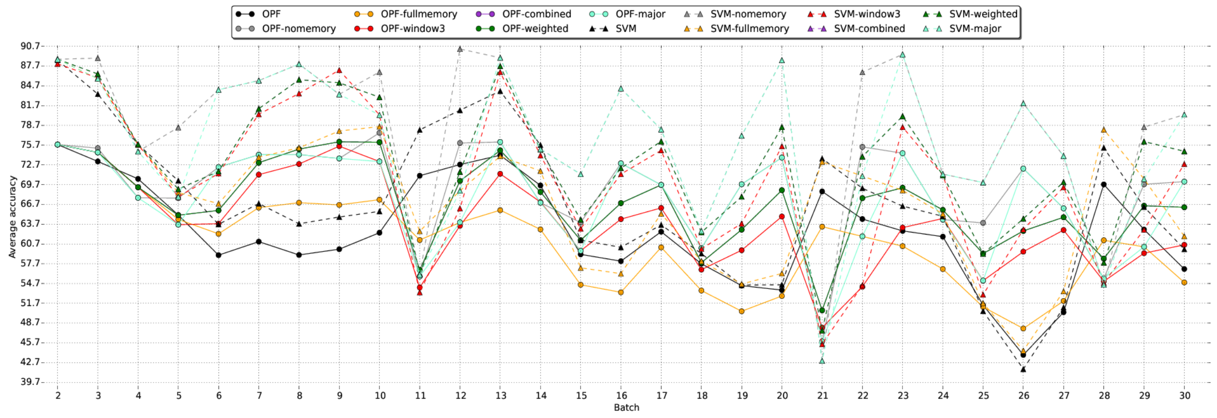

Figure A.2: Experimental results concerning Hyperplane dataset with respect to Experiment 1.

Table A.2: Hyperplane dataset with respect to Experiment 1.

Type	Accuracy
OPF	62.48453148 ± 7.664021067
OPF-nomemory	69.01541517 ± 7.265204095
OPF-fullmemory	60.77733197 ± 6.950052568
OPF-window3	63.79282938 ± 7.071207897
OPF-combined	66.90718755 ± 6.359478356
OPF-weighted	66.90718755 ± 6.359478356
OPF-major	67.042868 ± 7.594141989
SVM	66.68400903 ± 10.72104972
SVM-nomemory	77.9653311 ± 11.55842702
SVM-fullmemory	67.393422 ± 10.28898787
SVM-window3	70.15282272 ± 11.27674476
SVM-combined	72.91148562 ± 10.0056331
SVM-weighted	72.91148562 ± 10.0056331
SVM-major	75.71327959 ± 11.22732262

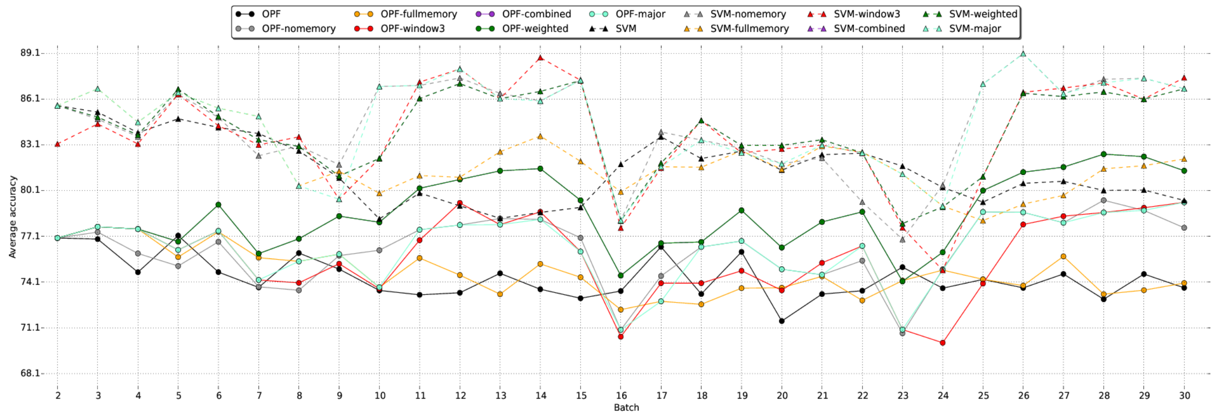

Figure A.3: Experimental results concerning SEA dataset with respect to Experiment 1.

In the Hyperplane dataset (Figure A.5), the combined and weighted SVM versions obtained similar performances, being more accurate than traditional SVM except in three batches. Table A.5 presents the mean accuracy and standard deviation concerning all batches in no-memory management.

Table A.3: SEA dataset with respect to Experiment 1.

Type	Accuracy
OPF	74.39154759 ± 1.346465467
OPF-nomemory	76.24827041 ± 2.128896094
OPF-fullmemory	74.69366883 ± 1.47469361
OPF-window3	75.83703666 ± 2.551433832
OPF-combined	78.6340661 ± 2.305279021
OPF-weighted	78.6340661 ± 2.305279021
OPF-major	76.34419907 ± 2.180538025
SVM	81.50678338 ± 2.119882634
SVM-nomemory	84.387488 ± 3.026719665
SVM-fullmemory	82.1416731 ± 2.194377926
SVM-window3	83.80219638 ± 3.334783039
SVM-combined	84.00122831 ± 2.664180205
SVM-weighted	84.00122831 ± 2.664180205
SVM-major	84.5856531 ± 2.932968912

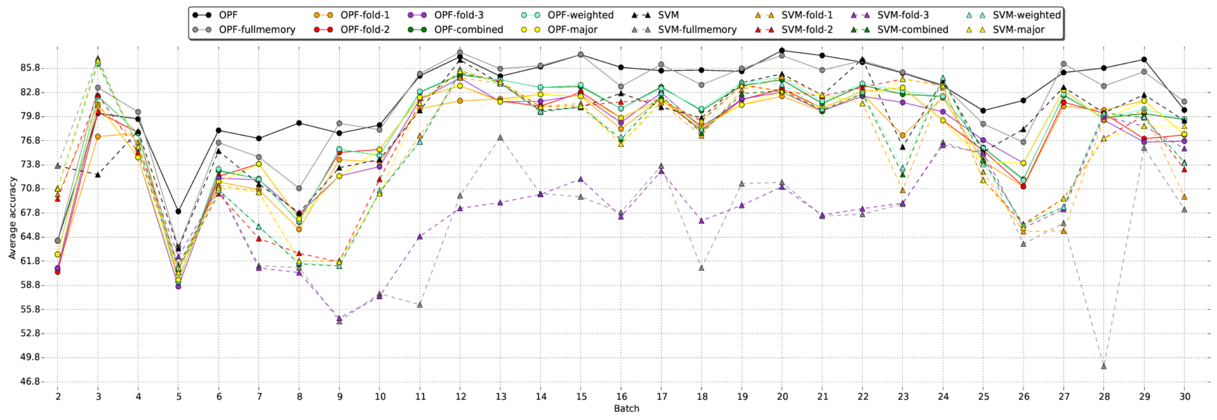

Figure A.4: Experimental results concerning Covertype dataset with respect to Experiment 2 in full-memory management.

Table A.4: Covertype dataset with respect to Experiment 2 in full-memory management.

Type	Accuracy
OPF	82.32752648 ± 5.527916684
OPF-fullmemory	81.51895655 ± 6.360053157
OPF-fold1	77.05269845 ± 6.149721752
OPF-fold2	77.53515266 ± 6.216085779
OPF-fold3	77.32545021 ± 6.333944787
OPF-combined	78.4888871 ± 6.553147785
OPF-weighted	78.58643359 ± 6.601479619
OPF-major	77.65701845 ± 6.154886295
SVM	78.7832119 ± 5.549011807
SVM-fullmemory	67.99529252 ± 7.904770131
SVM-fold1	75.4320561 ± 7.40372963
SVM-fold2	76.36477648 ± 7.629993757
SVM-fold3	69.56091117 ± 6.700559242
SVM-combined	75.80669193 ± 7.455094698
SVM-weighted	75.98013331 ± 7.575125596
SVM-major	76.57805514 ± 7.514583676

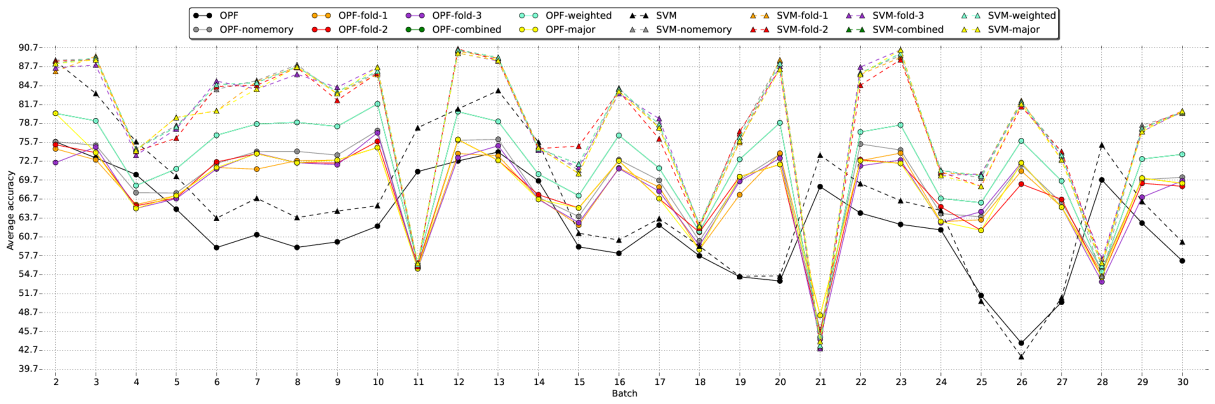

Figure A.5: Experimental results concerning Hyperplane dataset with respect to Experiment 2 in no-memory management.

Table A.5: Hyperplane dataset with respect to Experiment 2 in no-memory management.

Type	Accuracy
OPF	62.48453148 ± 7.664021067
OPF-nomemory	69.01541517 ± 7.265204095
OPF-fold1	67.92163028 ± 6.590813201
OPF-fold2	68.15354062 ± 6.862876752
OPF-fold3	67.82526386 ± 7.16983859
OPF-combined	72.09896266 ± 8.563747093
OPF-weighted	72.09896266 ± 8.563747093
OPF-major	68.2630471 ± 7.031192091
SVM	66.68400903 ± 10.72104972
SVM-nomemory	77.9653311 ± 11.55842702
SVM-fold1	77.74081683 ± 11.2355244
SVM-fold2	77.69870045 ± 11.24403346
SVM-fold3	77.86011476 ± 11.30176483
SVM-combined	78.04638724 ± 11.37294124
SVM-weighted	78.04638724 ± 11.37294124
SVM-major	77.6620691 ± 11.23552217

In the SEA dataset (Figure A.6), the SVM versions and traditional SVM performed similarly in various batches. The results of SVM versions seem to be more accurate than traditional SVM in batches #10 to #15, whereas traditional SVM was a little more accurate in few batches. Table A.6 presents the mean accuracy and standard deviation concerning all batches in window-3 management.

Figures A.7 to A.9 show the results evaluated in Experiment 3. Figure A.7 presents the results over the Covertype dataset. Once again, traditional SVM obtained the best recognition rates considering SVM versions. Table A.7 presents the mean accuracy and standard deviation concerning all batches.

Figure A.8 depicts the results concerning the Hyperplane dataset. The combined and weighted versions of SVM obtained similar performances, being sometimes more accurate than traditional SVM. Table A.8 presents the mean accuracy and standard deviation concerning all

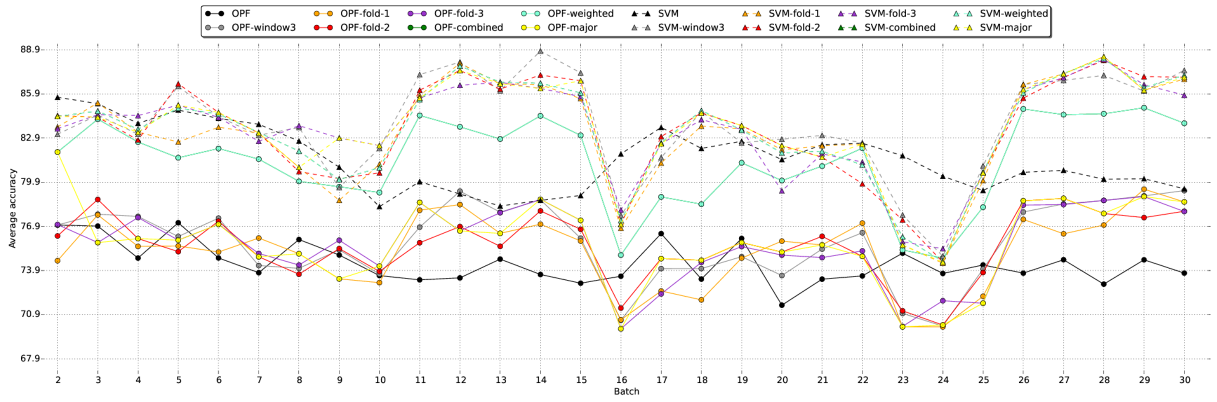

Figure A.6: Experimental results concerning SEA dataset with respect to Experiment 2 in window-3 management.

Table A.6: SEA dataset with respect to Experiment 2 in window-3 management.

Type	Accuracy
OPF	74.39154759 ± 1.346465467
OPF-window3	75.83703666 ± 2.551433832
OPF-fold1	75.21760424 ± 2.51672435
OPF-fold2	75.60642824 ± 2.16436392
OPF-fold3	75.68674859 ± 2.52686664
OPF-combined	81.34387231 ± 2.938457534
OPF-weighted	81.34387231 ± 2.938457534
OPF-major	75.76676728 ± 2.795645255
SVM	81.50678338 ± 2.119882634
SVM-window3	83.80219638 ± 3.334783039
SVM-fold1	83.19033838 ± 3.540041707
SVM-fold2	83.471199 ± 3.476069957
SVM-fold3	83.46943507 ± 3.163171141
SVM-combined	83.46486041 ± 3.393117611
SVM-weighted	83.46486041 ± 3.393117611
SVM-major	83.59102159 ± 3.371619002

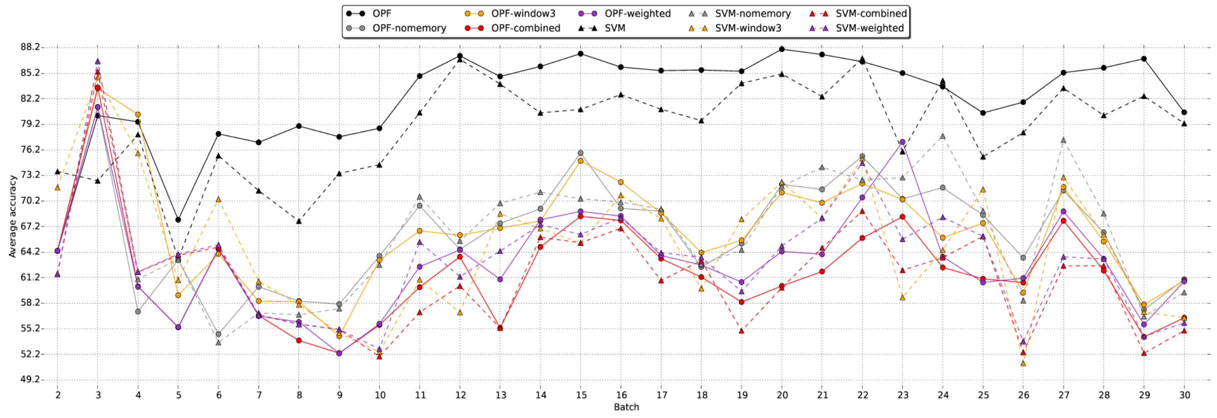

Figure A.7: Experimental results concerning Covertype dataset with respect to Experiment 3.

batches.

Figure A.9 depicts the results concerning the SEA dataset. The SVM versions and tradi-

Table A.7: Covertype dataset with respect to Experiment 3.

Type	Accuracy
OPF	82.32752648 ± 5.527916684
OPF-nomemory	66.340352 ± 6.205296248
OPF-window3	66.63647852 ± 6.483513342
OPF-combined	61.75935952 ± 6.122989272
OPF-weighted	63.36128134 ± 6.217485077
SVM	78.7832119 ± 5.549011807
SVM-nomemory	66.69459997 ± 7.552454182
SVM-window3	65.19059759 ± 7.751632536
SVM-combined	61.29515555 ± 6.557138516
SVM-weighted	63.40883021 ± 6.754922019

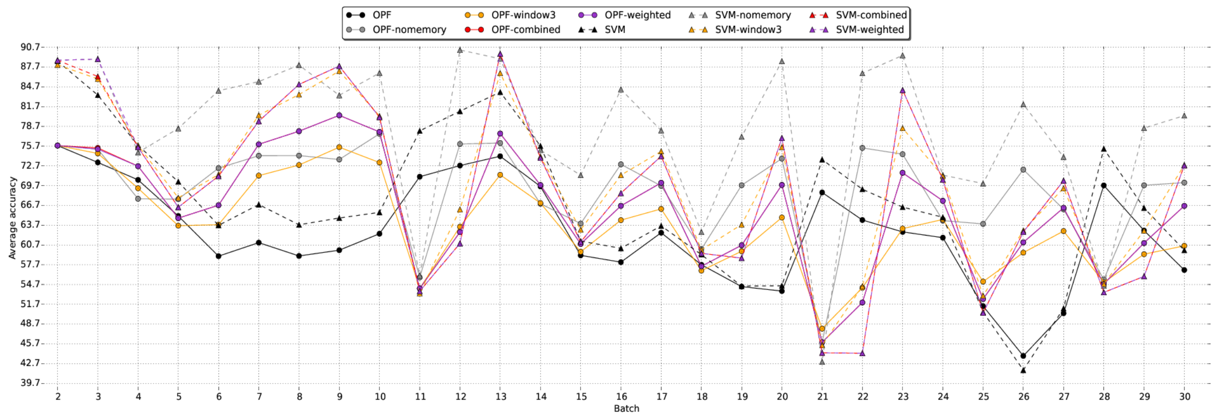

Figure A.8: Experimental results concerning Hyperplane dataset with respect to Experiment 3.

Table A.8: Hyperplane dataset with respect to Experiment 3.

Type	Accuracy
OPF	62.48453148 ± 7.664021067
OPF-nomemory	69.01541517 ± 7.265204095
OPF-window3	63.79282938 ± 7.071207897
OPF-combined	66.09129634 ± 8.855442297
OPF-weighted	66.08478797 ± 8.848643462
SVM	66.68400903 ± 10.72104972
SVM-nomemory	77.9653311 ± 11.55842702
SVM-window3	70.15282272 ± 11.27674476
SVM-combined	69.20823466 ± 13.07394588
SVM-weighted	69.29923652 ± 13.20099646

tional SVM performed similarly in various batches. Among the SVM versions, the results seem to be more accurate than traditional SVM in batches #10 to #15 and #26 to #30, whereas traditional SVM was a little more accurate in few batches. Table A.9 presents the mean accuracy and standard deviation concerning all batches.

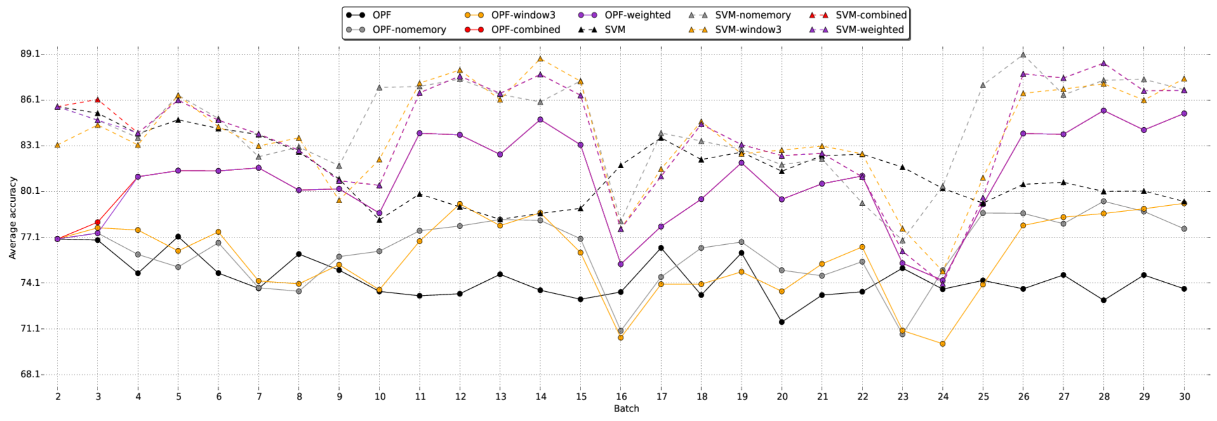

Figure A.9: Experimental results concerning SEA dataset with respect to Experiment 3.

Table A.9: SEA dataset with respect to Experiment 3.

Type	Accuracy
OPF	74.39154759 ± 1.346465467
OPF-nomemory	76.24827041 ± 2.128896094
OPF-window3	75.83703666 ± 2.551433832
OPF-combined	80.88808886 ± 2.995032574
OPF-weighted	80.86339066 ± 3.020802532
SVM	81.50678338 ± 2.119882634
SVM-nomemory	84.387488 ± 3.026719665
SVM-window3	83.80219638 ± 3.334783039
SVM-combined	83.77519379 ± 3.641911321
SVM-weighted	83.72836431 ± 3.619883153