Adaptive machine learning algorithms for data streams subject to concept drifts

Pierre-Xavier Loeffel

SORBONNE UNIVERSITÉS

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Pierre-Xavier Loeffel

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Algorithmes de machine learning adaptatifs pour flux de données sujets à des changements de concept

soutenue le 04 décembre 2017

devant le jury composé de :

M. Bernd Amann	Examinateur
M. Albert Bifet	Examinateur
M. Antoine Cornuéjols	Examinateur
M. Marcin Detyniecki	Directeur de thèse
M. Joao Gama	Rapporteur
Mme. Ludmila Kuncheva	Rapportrice
M. Vincent Lemaire	Examinateur
M. Christophe Marsala	Directeur de thèse

Adaptive Machine Learning Algorithms For Data Streams Subject To Concept Drifts

Résumé

Dans cette thèse, nous considérons le problème de la classification supervisée sur un flux de données sujets à des changements de concepts. Apprendre à partir de ces flux de données représente un challenge immense. En effet, un algorithme d'apprentissage doit être capable d'apprendre à partir d'une série d'observations et doit pouvoir obtenir des bonnes performances de prédiction sous les contraintes d'un temps de calcul et d'une mémoire ordinateur limités. Un autre challenge est le fait que la distribution de probabilité cachée (le concept) qui génère les observations, puisse changer avec le temps (changement de concept). Un algorithme d'apprentissage doit donc avoir la flexibilité de pouvoir s'adapter à ces changements de distributions.

Afin de surmonter ces difficultés, nous pensons qu'un algorithme d'apprentissage doit combiner plusieurs caractéristiques. Il doit apprendre en ligne, ne pas faire d'hypothèses sur le concept ou sur la nature des changements de concepts et doit être autorisé à s'abstenir de prédire lorsque c'est nécessaire.

Les algorithmes en ligne sont un choix évident pour traiter les flux de données. De par leur structure, ils sont capables de continuellement affiner le modèle appris à l'aide des dernières observations reçues.

La structure instance based a des propriétés qui la rende particulièrement adaptée pour traiter le problème des flux de données sujet à des changements de concept. En effet, ces algorithmes font très peu d'hypothèses sur la nature du concept qu'ils essaient d'apprendre ce qui leur donne une flexibilité qui les rend capable d'apprendre un vaste éventail de concepts. Une autre force est que stocker certaines des observations passées dans la mémoire peux amener de précieuses meta-informations qui pourront être utilisées par la suite par l'algorithme. Pour finir, cette structure permet de baser la mise à jour du modèle sur des preuves concrètes d'obsolescence et de fait, permet de s'adapter aux changements de concept sans avoir besoin de les détecter explicitement.

Enfin, dans cette thèse, nous mettons en valeur l'importance de permettre à un algorithme d'apprentissage de s'abstenir de prédire lorsque c'est nécessaire. En effet, les changements de concepts peuvent être la source de beaucoup d'incertitudes et, parfois, l'algorithme peux ne pas avoir suffisamment d'informations pour donner une prédiction fiable. Dans ces cas-là, plutôt que d'essayer de donner une prédiction à n'importe quel prix, nous pensons qu'une meilleure stratégie consiste à déconnecter automatiquement l'algorithme en lui permettant de s'abstenir de prédire.

Summary

In this thesis, we investigate the problem of supervised classification on a data stream subject to concept drifts. A stream of data is a source which continuously (and potentially endlessly) emits data. Learning from these data streams is a tremendous challenge. The learning algorithm must be capable of learning out of sequential data and must obtain good predictions performances under the constraints of limited running time and computer memory.

Another major challenge is that the hidden probability distribution (the concept) which generates the observations might change over time (concept drift). This means that the observations used to learn can't be assumed to be i.i.d. anymore and that a successful learning algorithm must have the flexibility to adapt to these changing distributions.

In order to deal with these challenges, we claim that a successful learning algorithm must combine several characteristics. It must be able to learn and adapt continuously, it shouldn't make any assumption on the nature of the concept or the expected type of drifts and it should be allowed to abstain from prediction when necessary.

On-line learning algorithms are the obvious choice to handle data streams. Indeed, their update mechanism allows them to continuously update their learned model by always making use of the latest data.

The instance based (IB) structure also has some properties which make it extremely well suited to handle the issue of data streams with drifting concepts. Indeed, IB algorithms make very little assumptions about the nature of the concept they are trying to learn. This grants them a great flexibility which make them likely to be able to learn from a wide range of concepts. Another strength is that storing some of the past observations into memory can bring valuable meta-informations which can be used by an algorithm. Furthermore, the IB structure allows the adaptation process to rely on hard evidences of obsolescence and, by doing so, adaptation to concept changes can happen without the need to explicitly detect the drifts.

Finally, in this thesis we stress the importance of allowing the learning algorithm to abstain from prediction in this framework. This is because the drifts can generate a lot of uncertainties and at times, an algorithm might lack the necessary information to accurately predict. In these cases, instead of trying to output a prediction at all cost, we have argued that it might be better to automatically disconnect the algorithm by allowing it to abstain from prediction.

Acknowledgment

This PhD thesis has been financed by Région Ile de France as part of the Wasteforecaster project. I am very grateful to them for allowing this thesis to happen.

Publications

- P-X Loeffel et al. Classification with a reject option under Concept Drift: The Droplets algorithm. **IEEE International Conference on Data Science and Advanced Analytics (DSAA 2015)**, Paris, France
- P-X Loeffel et al. Improving the Prediction Cost of Drift Handling Algorithms by Abstaining. **IEEE International Conference on Data Mining (ICDM 2016)**, Barcelone, Spain
- P-X Loeffel et al. Memory management for data streams subject to concept drift. **24th European Symposium on Artificial Neural Networks (ESANN 2016)**, Bruges, Belgium
- P-X Loeffel et al. Droplet Ensemble Learning on Drifting Data Streams. **16th International Symposium on Intelligent Data Analysis (IDA 2017)**, London, United Kingdoms

Contents

Chapter 1

Introduction

Over the course of the last few years, the quantity of data generated in real time has exploded. Health monitoring systems, electricity consumption data, stocks prices on the financial markets are a few examples of *data stream*s: a source which continuously (and potentially endlessly) generates data. The data generated by these streams can be used for the purpose of making predictions about the future value of a variable of interest. For instance, a trader would be interested to make use of past informations about a company to predict its future stock's price value. To this end, a machine learning algorithm can be used. These algorithms can automatically learn a model out of past data and this model will output a prediction when given an input generated by the stream. Unfortunately, these streams of data represent a tremendous challenge for machine learning algorithms.

For a start, the pace at which the data are generated by the stream can be a problem for a machine learning algorithm. Indeed, following up on the example of financial markets, the price of a stock is often refreshed every millisecond. This means that an algorithm used for prediction in this environment must have a running time small enough to handle each observation received while at the same time obtain good prediction performances.

Another challenge is the limited amount of computer memory available to the learning algorithm. Indeed, because the stream is potentially endless, it isn't possible to store all the received data into memory or to endlessly grow a model without setting a threshold on its memory consumption.

Finally, many things change over time and this is also the case for data streams. The relationship linking the input data to the target variable can evolve over time. An example would be a learning algorithm trained to detect whether a person is wealthy or not, compared to the average citizen. It might be the case that the learning algorithm would have figured that in the 18th century, a monthly income of 1000$ is enough to be classified as "wealthy", whereas this very same income would merely classify this citizen as "middle class" (if not "poor") in today's world. When the *concept* (the hidden probability distribution that the algorithm is trying to learn) which generates the observations and their labels changes over time, it is said that a *concept drift* has occurred. These drifts are a major challenge for a learning algorithm

because they can significantly decrease its prediction performances if they are not taken into account. The additional difficulty is that they are often unpredictable and can happen in many different ways. Consequently, a machine learning algorithm aiming at learning in this kind of evolving environment should be capable of adaptation to these changes.

Although it is sometimes possible to get an a-priori information about the expected evolution of the stream over time as well as the nature of the concept learned, we set the framework of this PhD thesis in the most general scenario: we don't make any assumption on the nature of the concept or how this concept is expected to drift (or remain stationary) over time.

In this framework, we claim that a learning algorithm aiming at achieving good prediction performances on a data stream subject to concept drifts must combine several properties: it must be capable to learn and adapt continuously, its prediction model shouldn't make any assumption on the nature of the concept it is trying to learn, its adaptation procedure should be capable to deal with any kind of drift, it should be allowed to abstain from prediction when necessary and its running time and memory consumption should be constrained. We now present how we managed to get these properties in this thesis:

On-line learning algorithms [91] are particularly well suited when it comes to continuously learn and update a model. Indeed, contrarily to batch learning algorithms which need a full dataset in order to learn a static model, an on-line learning algorithm is able to constantly update its learned model with the latest observation received. When the concept drifts over time, this structure grants the required flexibility to adapt to the new concept whereas when the concept remains static, it allows the algorithm to make use of all the available observations to infinitely fine tune its learned model. The challenge with these algorithms is to come up with an adaptation / forgetting mechanism which is suited for the type of drifts encountered.

Many on-line learning algorithms have been devised with the purpose of handling drifting data streams. Some of them are regular batch learning algorithms which have been adapted to the on-line setting. They include sets of rules [6], decision trees [11, 29] or ensemble method [60] for instance. Amongst all the type of learning algorithms, we claim that instance based methods have many characteristics which makes them particularly well suited to overcome some of the challenges discussed above.

Instance based learning algorithms [91] retain (some of the) past observations into memory. They postpone the generalization part of the learning process until a prediction is required. When this is the case, a local model is built around the latest observation and used to determine its label. The local model is then discarded and the latest observation is added to the memory.

This structure is naturally well suited to the on-line learning framework, as the update of the model is simply performed by adding the latest observation to the memory without the need to retrain the algorithm from scratch. Furthermore, when the concept drifts, forgetting the outdated model is achieved by deleting from memory the observations which aren't relevant with the new concept. In particular, when the concept changes locally, this forgetting mechanism allows for a bespoke update of the local parts of the model which are outdated without loosing the valuable information accumulated elsewhere.

Because the running time and memory consumptions of these algorithms will be a function

of the number of observations saved into memory, they can easily be tweaked to deal with the constrains of the problem at hand, simply by limiting the number of observations retained into memory. In both cases (whether it is for the purpose of adapting to concept change or for the purpose of constraining the memory consumption and running time of the algorithm), the challenge with this strategy is to accurately select the observations which will be allowed to remain into computer memory.

Due to their lack of assumptions regarding the nature of the concept learned, the models that can be learned from instance based methods are also generally much more flexible than the ones of algorithms assuming that the concept has a particular characteristic (e.g. linear relationship between the observations and their labels). This ensures that the learning algorithm will be able to adapt to a wide range of concepts, which can prove useful when no assumption can be made regarding the nature of the future concepts.

Finally, keeping past observations into memory allows for comparison of the latest observation received with the ones stored into memory. This brings extra meta-informations which can be used by the algorithm. In particular, when no assumption can be made about the expected nature of future drifts, we claim that adaptation should rely on hard evidences of obsolescence and we show in this thesis that this strategy is capable of adapting to a wide range of drifts.

Furthermore, these saved observations could also be used by the learning algorithm to know which parts of the feature space have already been explored, allowing to confidently predict on the regions which have been extensively covered or conversely, adopt a cautious stance when an observation is received in an unexplored region of the feature space. In the latter case, we claim that, in order to retain good predictions performances, a learning algorithm should be allowed to abstain from prediction, particularly when wrong predictions are costly.

Prediction with a reject option is a field of machine learning which focuses on devising algorithms that can abstain from prediction when the estimated reliability of a prediction is not high enough. The underlying idea is to decrease the proportion of observations for which a prediction is given in order to increase the percentage of accurate predictions.

In the particular case of drifting data streams, the hidden probability distribution generating the observations might change in many unexpected ways, and consequently sometimes, a learning algorithm might not have all the necessary information in order to accurately predict on an unlabeled observation. For instance, if a drift occurs in a previously unexplored part of the feature space or if the new hidden probability distribution in force after the latest drift completely differs from the old one, without additional prior information, the first predictions given by a machine learning algorithm in these regions would probably be closer to "guesses" than confident predictions.

In such cases, these "guesses" can end up being costly predictions errors and therefore we propose to abstain from prediction as a way to avoid this. The challenge here is to accurately spot the observations which might lead to a wrong prediction while maximizing the coverage (i.e. the proportion of observations for which a prediction is given).

In this thesis, we address the problem of learning from a data stream subject to concept

drifts under the constrains of limited computer memory and running time and without assuming any knowledge about the nature of the concept learned and the type of drifts encountered. We claim that algorithms combining the instance based and on-line learning structures are particularly well suited to overcome these challenges and that, in this environment, a learning algorithm should be allowed to abstain from predicting when there isn't enough available information to accurately predict.

In chapter 2 we lay down the framework associated with the problem of learning on a data stream subject to concept drifts. We also briefly introduce on-line learning algorithms.

Chapter 3 intends to break down the different algorithm structures that can be used when learning in an evolving environment. We discuss each structure's strengths and weaknesses and present some of the main learning algorithms based on them.

In chapter 4, we claim that the *instance based structure* has many properties which make it naturally well suited to address the problem of learning on data streams subject to concept drifts. In particular, we propose to take advantage of the meta-informations brought by storing past observations into memory. This idea is experimented with a novel algorithm (The Droplets Ensemble Algorithm) which combines instance based and ensemble learning structures and uses past observations to keep track of the local expertise of each base learner in the explored parts of the feature space. When a new unlabeled observation is received, this mechanism ensures that the base learner(s) which managed to obtain the best prediction performances on similar observations will be selected for prediction. We also propose an adaptive mechanism which ensures that the area of expertise associated with the base learners remains up to date with their current performances. This is done by reducing the area of expertise of the base learners which predict poorly and associating the area located around the latest observation with the base learner which obtained the best prediction accuracy in this part of the feature space.

In chapter 5, we investigate the benefits of *abstaining from prediction* in our framework. We claim that, because of the uncertainty that they create, concept drifts can significantly diminish the prediction performances of a learning algorithm. Indeed sometimes, there aren't enough information available to the algorithm to accurately predict. In the particular case where costs can be associated with good and bad predictions, we claim that an algorithm should be allowed to abstain when the estimated reliability of its prediction is not high enough. In order to show this point, we perform an in depth study with several state of the art drift handling algorithms which are given an ensemble of reliability estimators. When the estimated reliability of their prediction is not high enough, the algorithms abstain from prediction. We showed through a set of experiments that their overall prediction cost could be largely improved by abstaining to predict on difficult observations.

Chapter 6 aims at combining the findings of chapters 4 and 5 together in order to overcome the initial problem of accurately predicting on a data stream subject to concept drifts. Here again, the idea is to take advantage of the *meta-informations* provided by past observations saved into memory by abstaining from prediction when there isn't enough information to accurately predict the label associated with the latest observation. This happens, either if the

latest observation is received in an unexplored part of the feature space either if it is received in an area where observations with conflicting labels have been received. A new algorithm (the Droplets algorithm) is developed to test these ideas. Its main strengths and weaknesses are discussed and its predictive performances are compared to a batch of state of the art drift handling algorithms. The results obtained showed that the Droplets algorithm managed to over-perform other drift handling algorithms by consistently obtaining good results on datasets reproducing different types of drifts.

In chapter 7, we look into the problem of limited computer memory and running time in the case of instance based learning algorithms. We show that both of these issues can be resolved by setting an upper bound on the maximum number of observations allowed into memory. We then claim that, instead of using the time stamps of the observations to decide whether they should remain in memory or not, better prediction performances can be achieved by saving the observations which will minimize the differences in the predictions and model obtained with infinite memory. This is because the information brought by the time stamp of an observation is simply not enough to decide whether it should remain into memory or not as it might lead to the selection of noisy, redundant or even outdated observations. We implement this idea with the Droplets algorithm presented in chapter 6 and show through a set of experiments that this memory management technique obtains much better performances than discarding the observations that would be outside a temporal window.

Finally chapter 8 concludes and goes through a set of perspectives.

Chapter 2

Framework

Throughout this chapter, we lay down a general framework for machine learning, data streams, concept drifts and on-line learning algorithms.

2.1 Formal Machine Learning framework

In this section, we briefly introduce the machine learning framework as it will be used in this thesis.

2.1.1 Main goal of machine learning

We live in an era where data are present almost everywhere. For instance, there are more than one million transactions per hour performed on Walmart's website, around 269 billions emails sent each day over the world and big corporations often have databases holding petabytes of data. Being able to extract informations from these data constitute one of the major challenge of our time. However, these huge datasets are just too big to be analyzed by a human being which wouldn't be able to make sense of them and automated methods for data analysis are required.

That's the goal of *machine learning.* Machine learning is defined as "a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty" [78].

2.1.2 Supervised learning

In supervised learning, the aim is to learn a mapping from an input space X to an output space Y. To this end, a data set of N labeled observation (called *training set*) is usually provided $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$. We call $\boldsymbol{x}_i$ the i^{th} observation with $\boldsymbol{x}_i = \left\{x_i^1, ..., x_i^d\right\} \in X$ and where d is the dimension of the input space. The d values contained in the vector $\boldsymbol{x}_i$ are called features (or attributes). For instance, the feature's vector could hold the characteristics of a human being, such as its age, weight, height and so on. We call $y_i \in Y$ the label which is associated with $\boldsymbol{x}_i$. It is usually assumed that this label can take two types of values.

In the *classification* setting, the label $y_i \in \{1, ..., C\}$ belongs to a finite set of categorical variables. For instance, the categorical labels which could be associated with the characteristics of the human being could be $Y = \{male, female\}$. When $C = 2$ we refer to this as the *binary classification* setting whereas when $C > 2$ we call this the *multi-class classification* setting.

In the *regression* setting, the label y_i is a real number. Examples of real valued labels could be the body temperature or number of miles traveled per year for a given person.

In both classification and regression settings, the assumption is that there is a hidden function h such that $h(\boldsymbol{x}_i) = y_i$ and the goal is to use the provided training set to create an estimate $\hat{h}$ of this function. This estimate is then used for prediction $\hat{h}(\boldsymbol{x}_i) = \hat{y}_i$ (the hat symbol is used to denote an estimate) on previously unseen observations (this is called *generalization*).

In other words, it is assumed [91] that the couples $\{(\boldsymbol{x}_i, y_i)\}$ are independent and identically distributed (i.i.d.) from a joint distribution $P(X, Y)$ and the goal of a machine learning algorithm is to estimate this joint distribution in order to make predictions.

We call the joint distribution $P(X, Y)$ as well as the true hidden function h the *concept* that the machine learning algorithm is trying to learn.

2.1.3 Parametric vs non-parametric models

In order to estimate the unknown function h, two types of models can generally be used: *parametric* and *non-parametric models*.

Parametric models Parametric models rely on assumptions regarding the nature of h. For instance, in the case of linear regression, the assumption is that the relationship between the observations and their labels is linear. Once a parametric model has been selected, the problem of estimating h is greatly simplified. Indeed, what is left to do is to define a procedure to automatically fit the model's parameters to the training data.

These models have the advantage of being computationally fast to use on unseen observations but the strong assumptions made might result in poor prediction performances if they do not accurately reflect the reality [78].

Non-parametric models On the other hand, non-parametric models do not make explicit assumptions about the nature of h.

Due to this lack of assumptions, the model is given much more flexibility to estimate h than in the parametric case which might result in better prediction performances because the estimate is closer to the reality. The downside is that in order to accurately estimate the reality, non-parametric models generally require more observations than non-parametric ones [78].

2.1.4 Source of prediction errors

In this section, we review some of the main sources of prediction errors.

Reducible vs irreducible errors The accuracy of the predicted value $\hat{y}$ depends on two quantities called *reducible* and *irreducible errors*.

The reducible error comes from the fact that the chosen model will generally not be a perfect estimate of the true hidden function h and that inaccuracy will introduce some errors. However, this error could potentially be reduced by using the best machine learning method to estimate h.

On the other hand, even if a machine learning algorithm was able to perfectly estimate h, it wouldn't predict y perfectly [52]. For instance, in real life, the sensor which records the data on which the algorithm is learning might not be very accurate, introducing some variations around the true values of the observations. Another reason is that, it might be the case that, in order to perfectly predict y, some features which can't be measured would need to be provided. This is called the irreducible error because regardless of how well h is estimated, this error can't be reduced.

Thus, supervised learning algorithms aim at creating estimates of h which minimize the reducible error.

The bias-variance trade-off As discussed in Section 2.1.3, non-parametric models do not make any explicit assumptions about the nature of h, which gives a lot of flexibility to their estimates. One of the consequences is that, if the training dataset is slightly modified, the obtained estimate of h might be very different from the estimate obtained with the initial training set.

Variance refers to the error resulting from the sensitivity of the model to changes in the training set. The more flexible the learner is, the more variance it has. High variance can result into over-fitting (the issue of over-fitting is discussed hereafter) the training set.

Conversely, parametric models tend to make assumptions regarding the nature of h which means that if the training set is slightly modified, the new estimated function $\hat{h}$ is unlikely to be very different from the original one.

Bias refers to the error introduced by making assumption about the nature of h. The less flexible a model is, the more bias it has. High bias can result into under-fitting the training set.

Therefore, the bias-variance dilemma refers to the problem of choosing a model that simultaneously minimizes two different sources of errors (which is impossible) [52].

Over-fitting Over-fitting refers to the fact that, as the flexibility of the chosen model increases, the overall prediction error achieved on the training set will tend to decrease whereas the overall prediction error achieved on the test set will tend to increase. This is because the learning algorithm tries to find patterns in the training data that doesn't exist and ends up learning patterns which have been generated by noise rather than by the true hidden function.

The consequence is a large test set error because the patterns founds on the training set did not exist.

2.1.5 Measures of models performances

Once a model has been chosen and trained, some metrics are needed in order to assess the quality of its estimate of h. The performances are generally evaluated on a set of previously unseen observations: the *test set*.

In order to asses how well a model is performing, several different measures can be used and this choice will often depend on the problem at hand.

For instance, in the regression setting, one of the most common performance measure is the mean squared error (MSE):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

If the predictions $\hat{y_i}$ of the model are close to the true value y_i, then the MSE will be small. On the other hand, if there is a substantial difference between the predictions and the true value, then the MSE will be large.

In the classification setting, the most common performance measure is the average miss-classification rate computed with 0-1 loss function:

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{\{y_i \neq \hat{y_i}\}}$$

where the indicator function $\mathbb{I}_A$ is equal to 1 is A is true and 0 otherwise. This function simply computes the average number of times where the predicted label was different from the real one.

2.2 Data Streams

The results achieved in recent years by machine learning algorithms in the classical *off-line setting* (i.e. when the whole dataset of observations on which the algorithm is learning is available at training time) have been impressive. However, many data are generated in real time nowadays like sensors reading, the prices of stocks in financial markets or data generated by social networks for instance.

A *data stream* is defined as a source of data which emits observations sequentially, in real time, infinitely and potentially at a very high speed.

In this section, we start by giving a brief formalization of the data streaming framework and then present the tremendous challenges faced by an algorithm learning in this environment.

2.2.1 Formalization

At time t_0 a stream starts to emit unlabeled observations $\{\boldsymbol{x}_{t_0}, \boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}, ...\}$ which are received one by one, at regular time intervals (i.e. $\forall k \in \mathbb{N} : t_{k+1} - t_k = u$ where $u \in \mathbb{R}^{+*}$ is a constant) and endlessly. Later on, the label y_t which is associated with observation $\boldsymbol{x}_t$ is released. In this thesis, we assume that y_t is always released before reception of $\boldsymbol{x}_{t+1}$.

The goal of a machine learning algorithm in this framework is similar to the classical off-line setting: create a learner $h : X \to Y$, which can predict as accurately as possible the label y_t which is associated with observation $\boldsymbol{x}_t$. However, data streams bring new challenges that must be dealt with.

2.2.2 Challenges

We now go through the main challenges that must be overcome by an algorithm learning from a data stream.

Sequential observations and on-line learning: Receiving the observations one by one, without having access to a whole dataset from scratch requires a learning algorithm which can generalize from the received observations and constantly update itself when a new observation is available. *On-line learning* algorithms are particularly well suited for this framework as they are capable of continuously using the latest observations received to extend the existing model's knowledge. The topic of on-line algorithms will be covered in depth in section 2.4.

Constrained memory usage: Because a data stream is potentially infinite, it isn't possible to store all the past observations into memory. Thus, the memory consumption of a learning algorithm in this framework must remain constrained and the model can't expect to have access to all the previously received observations.

Constrained computational time: Another challenge is the limited computational time which is given to the algorithm for prediction and to update its model. Indeed, when an unlabeled observation $\boldsymbol{x}_t$ is received, the prediction and update functions of the model must be completed before reception of observation $\boldsymbol{x}_{t+1}$. This can prove particularly challenging if the data are streamed at a very high frequency (e.g. each consecutive observations are received every millisecond).

Concept drifts and adaptivity: The general assumption made for the off-line setting (that the observations are independent and identically distributed (i.i.d.) from a stationary distribution), generally doesn't hold in the data streaming framework.

Indeed, the observations generated by a data stream are often related to each other, which prevents the independence assumption. For instance, if a camera records a video stream, it could be expected that the image recorded at time t would be related to the image recorded at time *t+1*.

Furthermore, many things evolve over time in our world and, because each observation is a snapshot of an object at a given point in time, it is expected that the characteristics of this object (and therefore of the observations) will also evolve over time. The consequence is that it can't be assumed that the observations are identically distributed from an unique distribution. An example would be the evolution of the price of a stock where the price can switch from a prolonged regime of low volatility to another prolonged regime of high volatility.

Thus, in the data streaming framework, no assumptions is made on the distribution of the data which is allowed to unexpectedly change over time in any way conceivable: a phenomenon referred as *concept drift*. One of the consequences is that learning in this environment requires an algorithm which can adapt to such changes, as a non-adaptive model trained under the classical i.i.d. assumption will inevitably become obsolete over time. The topic of concept drift will be presented in further details in section 2.3.

2.3 Concept Drift

In this section, we define and characterize concept drifts as it was done in the work of Gama et al.[41].

2.3.1 Definition

In the data streaming framework, it is assumed that each couple $\{\boldsymbol{x}_t, y_t\}$ is generated by a hidden joint distribution $P_t(X, Y)$. According to the Bayes rule, $P_t(X, Y)$ can be decomposed as follows:

$$P_t(X, Y) = P_t(Y/X) P_t(X)$$

where $P_t(Y/X)$ is the posterior probability of the label given the observation and $P_t(X)$ is the prior distribution of the observations. We call *concept* the joint law $P_t(X, Y)$ that the learner h_t is trying to approximate at time t.

In our framework, we focus on the case where the concept is allowed to evolve over time and concepts changes are assumed to be unpredictable. In some real world scenarios, the changes could probably be anticipated as the concept might display a seasonality pattern, but the underlying idea here is that if a learner can be constructed to successfully operate in the general setting, then it will also work for the particular case of reoccurring concepts (reoccurring concepts will be described formally in Section 2.3.2.2).

Formally, it is said that the concept drifts at time $t + 1$ if:

$$P_t(X, Y) \neq P_{t+1}(X, Y)$$

2.3.2 Characterization

Webb et al. [102] have provided a rigorous characterization of the drifts that we summarize here. The interested reader is referred to the original paper for further details. To our knowledge, this is the most advanced and formal characterization of concept drifts.

The notations used here are the ones of the original paper. We have kept the distinction between quantitative and qualitative characterizations of drift used by the authors. Quantitative measures of drift aim at quantifying a concept drift whereas the qualitative characterization aims at providing formal definitions of the different types of drifts.

2.3.2.1 Quantitative measures of drift

Drift magnitude is a metric which quantifies the degree of difference between two consecutive concepts. Formally, we denote $D(t, t+m)$ the distance between the concepts at the start time t and the end time $t+m$ of the drift. The authors chose to use the Hellinger [45] distance to quantify this. This distance is one of the most widely used when it comes to compare probability distributions.

Drift duration is defined as the elapsed time over which a period of drift occurs. For instance, if a drift starts a time t and finishes at time $t+m$, the duration of the drift is $(t+m)-t$.

Drift rate quantifies how fast the concept is changing at time t. This is defined as:

$$Rate_t = \lim_{n\to\infty} nD\left(t - \frac{0.5}{n}; t + \frac{0.5}{n}\right)$$

2.3.2.2 Qualitative characterization

A period of *concept stability* is defined as a time interval $[t; t+m]$ such that

$$D(t, t+m) = 0$$

The authors also introduce S_a and E_a respectively to denote the starting time and ending time of the a^{th} stable concept ().

The *drift frequency* measure counts the number of drifts that occurred during time interval $[t, t+m]$. It is defined as:

$$F_{[t,t+m]} = |\{w | t \leq S_w \leq t+m\}|$$

An *abrupt drift* denotes a sudden change from concept a to concept $a+1$. The authors argue that an abrupt drift is defined according to a parameter $\delta \in \mathbb{N}^*$ which sets the maximum duration over which an abrupt drift can occur. They claim that the value of δ depends on the context of the data stream. According to the authors, an abrupt drift verifies:

$$S_{a+1} - E_a \leq \delta$$

In this thesis, we consider that a drift is abrupt if $\delta = 1$ (i.e. the concept changes within one observation only). This is because this definition is the most restrictive for an abrupt concept change.

Conversely, we say that an *extended drift* occurred if

$$S_{a+1} - E_a > \delta$$

A *gradual drift* over m time steps is defined as:

$$\forall t \in [E_a, S_{a+1}] \;\; D(t, t+m) \leq \mu$$

with μ a parameter which defines the maximum difference allowed between two successive concepts. This definition doesn't necessarily assume that the drift rate should be constant.

A *recurring concept* denotes the reappearance of a previously seen concept. Formally:

$$\exists a \exists b \; with \; a \neq b \mid D(S_a, S_b) = 0$$

For instance, this type of drift can occur with datasets showing an element of seasonality.

2.3.2.3 Statistical characterization

Gama et al.[41] also gave a statistical characterization. According to the Bayes equation derived in the concept drift definition, if $P_t(X, Y) \neq P_{t+1}(X, Y)$ then:

1. Either $P_t(Y/X) \neq P_{t+1}(Y/X)$ and $P_t(X) = P_{t+1}(X)$. When the posterior distribution of the label changes, we say that a *real concept drift* occurred. This means that the probability that a given observation is associated with a particular label has changed.

2. Either $P_t(Y/X) = P_{t+1}(Y/X)$ and $P_t(X) \neq P_{t+1}(X)$. When the prior distribution of the observation changes, we say that a *virtual concept drift* occurred. This means that the probabilities of occurrence of an observation in some (or all) parts of the feature space have changed.

3. Either $P_t(Y/X) \neq P_{t+1}(Y/X)$ and $P_t(X) \neq P_{t+1}(X)$. This case is also referred as *real concept drift*. In this case, both the probabilities of occurrence of an observation in some parts of the feature space and the probability that a given observation is associated with a particular label have changed

2.4 On-line Learning

In order to handle drifting concepts, on-line algorithms are the most widely used. In this section, we start by presenting these algorithms and we then review some of the main procedures used to evaluate their performances on data streams.

2.4.1 Presentation

In the on-line learning setting, the observations are presented to the learning algorithm one by one. Contrarily to the off-line setting which relies on the assumption that the observations are i.i.d., there are no such assumption here. This means that an observation is allowed to be dependent of the value of the previous one and that the concept is allowed to change over time.

The operating process of an on-line algorithm also differs from the one of an off-line algorithm: When an observation $\boldsymbol{x}_t$ is received at time t, the model h_{t-1} learned at the previous time step is used to output a prediction : $\hat{y}_t = Predict\,(h_{t-1}; \boldsymbol{x}_t)$. The true label y_t is then released and the prediction error ℓ_t of the model is computed according to the chosen loss function : $\ell_t = \ell\,(y_t, \hat{y}_t)$ (with ℓ the chosen loss function). The learner is then updated using the latest labeled observation: $h_t = Update\,(h_{t-1}; \{\boldsymbol{x}_t, y_t\})$. The whole process then restarts with observation $\boldsymbol{x}_{t+1}$.

Therefore, on-line learning algorithms base their predictions on a model continuously updated with the latest observation instead of repeatedly using a static model learned from a fixed training set. This update mechanism makes these algorithms particularly well suited to deal with evolving data streams.

2.4.2 Evaluation procedures

In order to evaluate the performances of a learning algorithm, it is necessary to define which observations will be used for training and which ones will be used to test the model learned. In the traditional off-line setting, a training and test sets can be used. In particular, it is common to analyze and average the performances of the models produced with different arrangements of the dataset, a procedure known as *cross-validation*. However, cross-validation can't be used in the framework of evolving data streams as the observations might be dependent of each other and shuffling them would mix their temporal order.

Furthermore, because the data are not i.i.d., the performance of the learning algorithm might also change over time. For instance, an algorithm might obtain good prediction performances during a period of time followed by poor ones later on as a result of an unexpected drift. Therefore, it is necessary to keep track of the evolution of the performances over time by continuously taking snapshots of the model's performances.

Gama et al. [41] reviewed some of the most common evaluation procedures that we summarize here.

2.4.2.1 Holdout

In this case, the model learned is evaluated on a test set at regular time intervals. The test set is constituted of observations received from the stream which have not been used yet to train the algorithm. The procedure is as follows:

1. Train the algorithm with observations $\{t_1, ..., t_k\}$ which results in h_{t_k}.
2. Test h_{t_k} on observations $\{t_{k+1}, ..., t_{k+l}\}$ and record the prediction performance.
3. Update h_{t_k} with observations $\{t_{k+1}, ..., t_{2k}\}$ which results in $h_{t_{2k}}$.
4. Test $h_{t_{2k}}$ on observations $\{t_{2k+1}, ..., t_{2k+l}\}$ and record the prediction performance.
5. Repeat this procedure until all the remaining observations have been used.

Note that in order to avoid testing the algorithm twice on the same observation, it is necessary to have $k \geq l$. Also note that in this case, t_1 is used as a shortcut to designate the observation $\{x_{t_1}, y_{t_1}\}$.

This approach is best suited for batch learning algorithms.

2.4.2.2 Prequential

Here, the idea is to use each new unlabeled observation to first evaluate the prediction given by the algorithm (test) and then, once the label is released, to update the model (train). There are three type of prequential evaluations: interleaved test-then-train which uses a landmark window, fading factor or sliding window. These methodology are the most widely used to assess the performance of on-line algorithms.

Interleaved test-then-train: With this method, the error at time T is computed as the average of the prediction errors since the stream started:

$$L_T = \frac{1}{T} \sum_{t=1}^{T} \ell_t$$

This method has the advantage of making use of all the observations available to test the algorithm. However, there is no explicit forgetting mechanism, and the plot obtained will get smoother as the number of observations considered increases. This means that the evolution in the prediction performances due to concept drifts might not appear properly in the plot, especially if a drift occurs after a large number of observations is received.

Fading factors: In order to get a better estimate of the recent performances of the algorithm, it is possible to apply a decay factor $\alpha \in]0; 1[$ to the past errors. This method ensures that recent changes in the evolution of the performance are tracked more accurately. In the case where $\alpha = 1$, this comes to the regular interleaved test-then-train. Gama et al. [40] proposed to use the following equation:

$$L_T = \frac{S_T}{N_T}$$

with

$$\begin{array}{llll} S_T = \ell_t + \alpha S_{T-1} & and & S_1 = \ell_1 \\ N_T = 1 + \alpha N_{T-1} & and & N_1 = 1 \end{array}$$

Sliding window: Another way to estimate the recent performances of the learning algorithm is to use a sliding window. In this case, the performance achieved by the algorithm on the latest k observations is computed as:

$$L_T = \frac{1}{k} \sum_{t=T-k+1}^{T} \ell_t$$

where k denotes the size of the window.

Just as fading factors, a sliding window has the advantage of allowing to observe the reactivity of the algorithm after concept changes. Note that a sliding window of infinite size would result in the interleaved test-then-train evaluation.

2.5 Conclusion

In this chapter we have laid down the framework associated with the problem of supervised learning on data streams subject to concept drifts. We have used some of the most widely used definitions in the literature to formally define concept drifts and we have introduced on-line learning algorithms which are the most widely used in this framework.

Chapter 3

Taxonomy of drift handling algorithms

Creating an adaptive algorithm capable of learning a predictive model from an evolving data stream requires to make many choices regarding its structure. Most of these choices are dictated by the problem at hand.

In this section we review the different types of structures that can be given to a drift handling algorithm, present some of the state of the art algorithms implementing these structures and discuss whether they are adapted or not to the problem at hand. The taxonomy of methods used here is largely inspired by the work of Gama et al. [41] although we have completed it with some of the latest algorithms. As the authors point out, the idea here is to see the creation of an adaptive learning algorithm as a set of modular components which can be permuted and combined together. Therefore, some of the algorithms presented here might belongs to several categories at the same time.

The list of methods and algorithms provided here is not exhaustive and some algorithms which haven't been described here will be described in later chapters.

3.1 Observations management

In order to evolve, an adaptive learning algorithm should define a process to learn the latest concept as well as a forgetting mechanism to get rid of the outdated model. In this section, we discuss how the observations received from the data stream can be used in order to achieve these two goals.

Figure 3.1 details the different types of strategies available.

3.1.1 Learning

When it comes to learning, the goal is to choose which data will be used for learning the latest concept. The common assumption behind the overwhelming majority of learning algorithms is that the most recent data should be the most representative of the new concept and thus should be used for learning.

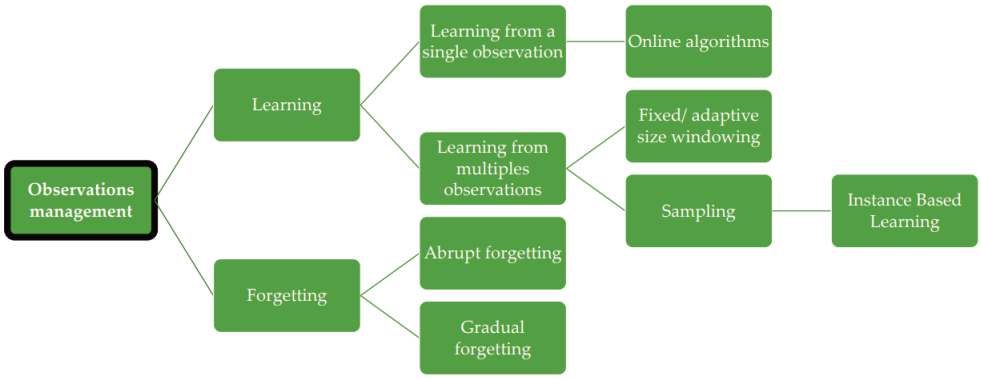

Figure 3.1: Types of observations management

3.1.1.1 Learning from a single observation

In its most basic form, learning from a data stream can be performed with a single observation (usually the latest one). Upon reception of an unlabeled observation, the previously learned model is used to output a prediction. Once the true label which is associated with the observation is received, the prediction error is computed and the model is updated accordingly. Thus, the updating process is a function of the previously learned model, the latest observation and the prediction error on this observation. The latest observation is then discarded and won't be available anymore to the algorithm.

Some of the main algorithms based on this structure include WINNOW [79], VFDT [29], STAGGER [92], DWM [60], SVM [96], IFCS [20] and GT2FC [21].

The advantage of this mechanism is that it is naturally well suited for continuous learning. When the concept remains stable, the model learned will be continuously fine tuned with the latest observations. However, when the concept drifts, there isn't an explicit forgetting mechanism and adaptation happens by diluting the previously learned concept over time. This adaptation takes place through a parameter controlling the trade-off between speed of adaptation and robustness to noise.

3.1.1.2 Learning from multiples observations

Another solution is to maintain a set of observations into memory on which the model learns.

Fixed size windowing: One way of doing this is to use a sliding window of fixed size over the recent observations which adds the newest observation to the set and discard the oldest one. At each time step, the model is then retrained using the observations in the window.

Some of the main algorithms based on this structure include FLORA [103], Mitchell et al. [77], Lazarescu et al. [72].

The advantage of using a window of fixed size over the recent observations is that it provides a simple mechanism for learning (by continuously adding the latest observation to the window) and forgetting (by removing the oldest observation from the window). However, selecting an appropriate window size is a challenging task: if the window is too small, the model might react quickly to abrupt drifts but also over-react to noisy observations. Furthermore, a small window might not hold enough observations to accurately learn the current concept. Conversely, if the window size is large, the model learned will most likely be much more robust to noisy observations and hold enough observations to learn an accurate model when the concept remains stable. The downside is that, with a large window, the model might not be able to adapt quickly enough to abrupt changes.

Adaptive size windowing: One way of solving this problem would be to use a window of variable size. In this case the size of the window is automatically set according to a change detector. The idea is to let the window grow as long as no change has been detected and to shrink its size when a change is detected, ensuring that the observations kept into the window are relevant with the current concept.

Some of the main algorithms based on this structure include FLORA2 [103], Klinkenberg and Joachims [57], Maloof and Michalski [76], Zliobaite [107], Klinkenberg [83], Gama et al. [39], and Kuncheva and Zliobaite [69, 108, 109], Koychev and Lothian [62].

Provided that the change detector is capable of accurately spotting concept changes, this solution ensure ensures that only the relevant data will be used to learn the model. However, we highlight in Section 3.2.2.2 that it is very difficult to implement a change detector that can deal with a wide range of scenarios (e.g. abrupt and gradual drifts) and to take a relevant decision once an alarm is triggered (e.g. should the whole model be discarded or only some parts of it?).

Furthermore, the common assumption behind windowing (whether it is of fixed or adaptive size) is that the recent observations are the most relevant and that a model should learn on a contiguous set of observations. In the first case, it could be argued that recent observations aren't necessarily the most relevant as it is the case with noisy observations. In the later case, it seems overly restrictive to constrain the learning set to contiguous data. In the case of reoccurring concepts for instance, the relevant data could be fragmented over time.

Sampling: Another solution would be to sample the observations which will be used to learn from. Sampling algorithms try to summarize the characteristics of the whole stream by retaining into memory the observations which have been selected according to a given probability distribution. In particular, this probability distribution might be biased in favor of the most recent observations in order to account for the drifts.

Some of the main algorithms based on this structure include those of Vitter [100], Al-Kateb et al. [5], Efraimidis and Spirakis [31], Aggarwal [2].

This mechanism is well suited to overcome the problems raised by learning on contiguous

data. Indeed, sampling gives the algorithm a better chance of learning from a diversified set of observations (e.g. the data generated by the stream might have a temporal dependence and consequently, it could be expected that information brought by storing observations x_t and x_{t+1} would be smaller than the information brought by storing observations x_t and x_{t+k}, with k large). Furthermore, by reducing the number of observations which will be fed to the algorithm, its running time might be significantly improved. Finally this strategy will work well, when there is a constrain on the maximum memory available but where it is desirable to retain as much of the past knowledge as possible.

The downside is that the memory will be cluttered with a "little bit of everything". This means that when the concept drifts frequently, the memory will hold a few observations belonging to each of the past concepts. The relevance of this strategy can be questioned as the information brought by the observations related to past concepts might be very small (e.g. it is unclear how much information would be brought to the model by the last 2 observations related to one of the past concept). Because they might belong to different concepts, the saved observations could also confuse the learning algorithm as they might conflict with each other (e.g. $x_t = x_{t+k}$ and $y_t \neq y_{t+k}$).

A particular type of sampling algorithms are the instance based algorithms which do not try to explicitly generalize from the observations saved in memory (i.e. they don't maintain a model). These algorithms have the liberty to select an arbitrary subset of observations when it comes to chose which ones will be used to learn. They can rely on a wide range of custom criteria to select these observations such as time or a probability distribution (as discussed previously) but they can also use spatial relevance (use the location of an observation in the feature space) or consistency (assess whether an observation in memory is different from its neighbors). These criteria can be used independently or combined. Kuncheva and Gunn [67] have proposed a taxonomy of drift handling instance based methods.

Some of the main algorithms based on this structure include IB3 [4], IBLStreams [93], ANNCAD [70], IBL-DS [8], SAM-kNN [75], Lazarescu [71].

Because of the flexibility that they provide, these algorithms are particularly well suited to handle data streams subject to concept drift. They make very little assumptions regarding the nature of the concept and they can use many different criteria to decide which observations should be kept into memory. The updating step of the model is easy as it only requires to add the observations to the memory.

The downside is that these algorithms have to create a local model each time a prediction is required. This step might increase the running time of the algorithm.

3.1.2 Forgetting

We now review some of the main forgetting mechanisms used to diminish the effect of past observations on the learned model. Ultimately, choosing the appropriate forgetting mechanism depends on the expected type of drifts (e.g. abrupt, gradual, reoccurring, ...) as well as the characteristics of the data (e.g. is the the data stream subject to a lot of noise?).

3.1.2.1 Abrupt forgetting

Abrupt forgetting refers to the fact that an observation is either fully taken into account to update the current model, either is not used at all. We can further distinguish the algorithms relying on an abrupt forgetting mechanism by the criteria used to select the observations that should be forgotten.

Windows: The window-based algorithms described and discussed in Section 3.1.1.2 are examples of abrupt forgetting mechanisms as past observations are either in or outside of the window. The observations can be removed one by one as in the case of a sliding window of fixed size or as a whole batch as in the case of a window of adaptive size.

When the observations are removed one by one from memory, as it is the case with a sliding window of fixed size, the result will be a model which will gradually forget the past. This strategy is best suited for gradual drifts but might lack reactivity in cases of abrupt drifts of high magnitude for instance.

Conversely, when a whole chunk of observations is deleted at once, as it is the case with an adaptive window, the model learned with the remaining observations will be capable of quick adaptation to abrupt concept drifts. This, however, is a risky strategy as it leaves the algorithm exposed to the problem of "catastrophic forgetting". Indeed, if noisy observations trigger the forgetting mechanism by mistake, this could lead to the loss of valuable accumulated information. This type of forgetting mechanism might also struggle with slow and gradual drifts that might never be detected.

In both cases, it is interesting to note that the criteria used to get rid of the old observations is "time" and that both strategies rely on the assumption that old observations are outdated and thus should be deleted in priority.

Sampling: Because an observation is either inside or outside the sample, sampling also uses the abrupt forgetting strategy. The criteria discussed earlier (e.g. time, a probability distribution, error, ...) to learn from a sample can also be used to select the observations which will be forgotten.

For instance, in Salganicoff [88, 87] (DARLING), a weight is associated with each observations stored into memory. This weight is decreased as a function of the proximity and number observations around the observation of interest. When this weight drops below a given threshold, the observation is removed from memory.

3.1.2.2 Gradual forgetting

Gradual forgetting refers to the fact that the effect an observation has on the current model never totally vanish. In this respect, this strategy can be seen as a full memory approach as the contribution of past observations is never null. This strategy is closely linked with the algorithms learning from one observation which were presented in Section 3.1.1.1.

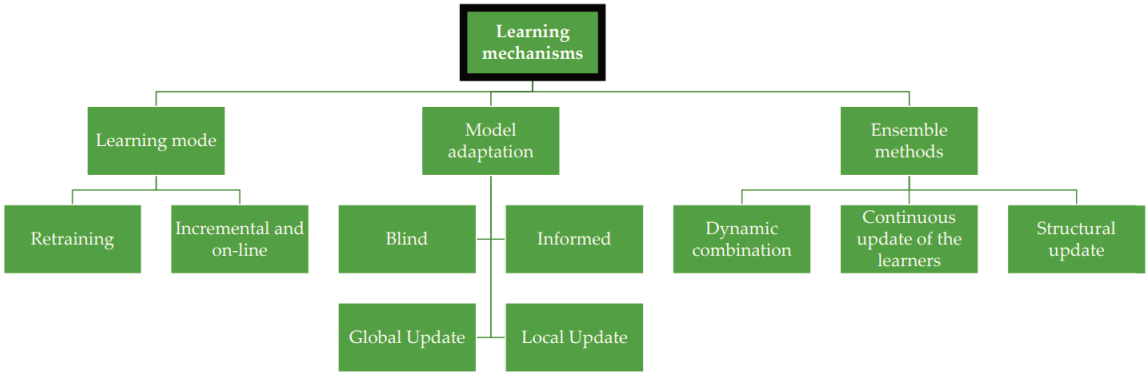

Figure 3.2: Types of learning mechanisms

Fading factor: One way of implementing this mechanism is to give a weight to each observation which will decrease over time according to a parameter α (the fading factor). The underlying idea is that the older an observation is, the less relevant it is and thus the less weight it should be given.

Formally: when a new observation $\boldsymbol{x}_t$ is received at time t, the stored statistics S_{t-1} are updated according to an aggregation function G such as: $S_t = G(\boldsymbol{x}_t, \alpha S_{t-1})$ with $\alpha \in]0;1[$ the fading factor.

Some examples of the algorithms based on this structure include Cohen and Strauss [27], Koychev [61], Law and Zaniolo [70] and Klinkenberg [83], Salganicoff [87].

Here again, the main criteria used to allocate a weight to an observation (and thus to forget the past) is time. This mechanism is best suited for gradual drifts with constant drift rate (ideally a drift rate which matches the value of the fading factor) and where past observations keep having a positive effect on the learned model.

On the other hand, allowing all of the past observations to have an influence (even if it is a very small one) on the current model could also be questioned, particularly in the case where they have a negative effect on the learned model (i.e. they decrease its performances). An example would be noisy observations which would be allowed to have an effect on the learned model. Finally, the major difficulty with this method is to accurately estimate the value which must be associated with the fading factor. This could prove very troublesome when the drifts occur at irregular frequencies and when they are of different types (e.g. abrupt or gradual).

3.2 Learning mechanisms

This section is concerned with the possible strategies used to generalize from the observations: whereas the previous section was focusing on which observations will be used to learn a model, here the emphasis is on how to learn this model.

Figure 3.2 gives an overview of the different learning mechanisms.

3.2.1 Learning mode

We now go through the main strategies used to update the learning algorithm once a new labeled observation have been received.

3.2.1.1 Retraining

In this case the old model is deleted and a new one is learned from scratch with the observations which have been kept into memory. Most of the time, these algorithm are combined with a drift detector which triggers the update of the model.

Some of the main algorithms based on this structure include Gama et al. [39], Street and Kim [94], Zeira et al. [105], Klinkenberg and Joachims [57].

The advantage of this solution is that it allows to easily convert an off-line learning algorithm to the on-line setting. Moreover, when this strategy is combined with a drift detector, it could also obtain good performances in the cases of abrupt drifts of high magnitude which make the whole model out of date.

The major drawback is that it is computational expensive to learn a whole model from scratch. In particular, this might be an issue when the data are streamed at a very high frequency. This methodology is more suited to the case where the observations are received by batches or where the model doesn't require regular updates (i.e. when the drift frequency is small).

3.2.1.2 Incremental and on-line

In this setting, the model (and the associated statistics), are updated with each new observation without the need to retrain the algorithm from scratch. Thus, the algorithm learned at time t is a function of the algorithm learned at time *t-1* and the current observation (although the updating function might access to previous observations stored into memory).

Some of the main algorithms based on this structure include CVFDT [47], Salperwyck et al. [90], Learning vector quantization [59], OnlineTree2 [80], WINNOW [79], MBW [26], VFDT [29], VFDTc [38] and FIMT-DD [49].

This structure is designed for frequent model updates and when it isn't necessary to forget the whole learned model at once, making it particularly useful to handle gradual drifts. One of the major drawback is that the update of the model will take place one example at a time and this might end up being too slow in cases were quick adaptation is required. Kuncheva and Plumpton [68] proposed to use an adaptive learning rate to deal with this issue.

3.2.2 Model Adaptation

When the concept drifts, adaptation is required in order to maintain a high prediction accuracy. We now discuss some of the main adaptation strategies.

3.2.2.1 Blind

In this case, the model is updated without explicit detection of concept changes: they are proactive. Fixed size windows methods discussed earlier are examples of blind adaptation where the model is continuously retrained with the observations in the window. Gradual forgetting methods based on fading factors are also an example of blind adaptation.

Some of the main algorithms based on this structure include Widmer and Kubat [103], Klinkenberg and Renz [58], Aggarwal et al. [1], VFDT [29].

The main advantage of this strategy is its simplicity. It solves the adaptation problem without needing to develop a complex drift detection module. Furthermore, by avoiding to wait for an alarm to be triggered, these algorithms can theoretically start updating their learned model right from the very first observation received after the drift which can make them more responsive than a model relying on a drift detector (which will need to gather more evidence of concept changes before triggering an alarm). The downside of course, is that it leaves the algorithm vulnerable to noisy observations. They might also be too slow in cases where the model learned must be quickly discarded.

3.2.2.2 Informed

In this case, adaptation of the model is triggered by a drift detector which raises an alarm: these methods are reactive. The idea is to track a metric of interest (any performance metric, the distribution of the data received or any other statistic which is specific to the model) and to assess whether the changes in this metric were caused by a drift or not.

Some of the main algorithms based on this structure include Salperwyck et al. [89], Bifet and Gavalda [9], CVFDT [47], EWMA [86], FLORA2, FLORA3 [103], Gama et al. [38][39], Ikonomovska et al. [49], Zeira and Maimon [105], EDDM [12], Kuncheva [65, 66].

One of the major advantage of using a detection mechanism is the additional information provided. Once a change has been correctly detected, it is indeed possible to characterize and quantify the extent of this change. This additional information can then be used to select the update mechanism which is the best suited to handle the drift that just occurred.

The downside is that accurately detecting drifts is a very challenging task. Drift detection requires to set a threshold parameter for detection. If this threshold is too low, noisy observation might trigger a false alarm. If the threshold is too high, the change detector might not be able to detect gradual concepts changes or might react too late[1]. Furthermore, deciding on the metric that should be tracked is not necessarily trivial and taking the best action once an alarm is triggered might also be difficult. For instance, if the metric of interest is the error rate, a consistently high error rate on the last observations doesn't necessarily mean that the whole model should be discarded. Indeed, the latest observations might have been concentrated in a particular part of the feature space where the predictive power of the model is weak. But on the other parts of the feature space, the model might still perform well. Therefore in this

[1] Faithfull and Kuncheva worked on the problem of threshold estimation [33]

example, it is not obvious which action should be taken as the error rate explode without further informations.

3.2.2.3 Global update

Global update occurs when the whole model learned is discarded at once and reconstructed from scratch.

Some of the main algorithms based on this structure include Gama et al. [39], Zeira et al. [105], Klinkenberg and Joachims [57].

This is a dangerous strategy which requires prior knowledge on the type of drifts in the considered data stream. Indeed, by triggering this update mechanism the algorithm is exposed to the issue of catastrophic forgetting, where valuable information would be discarded by mistake.

A scenario where global update would be well suited would be in the case of reoccurring concepts with abrupt drifts. If a detector is able to recognize that a model stored in memory is suited for the current concept, it could select it and would mean that no time is wasting learning the new concept from scratch, ensuring to get good performances quickly.

3.2.2.4 Local update

In this case, the model is only modified locally and the learned rule is otherwise preserved. The underlying idea is that updates are only performed when they are required. This implies that the algorithm has to include a mechanism to track the performance of the learned model on each parts of the feature space.

Some of the main algorithms based on this structure include CVFDT [47], Ikonomovska et al. [49].

This strategy is best suited for local concept drifts as it preserves the knowledge acquired elsewhere. It is a very cautious mechanism which waits for hard evidences to dismiss the outdated parts of the model. By doing so, it prevents the issue of catastrophic forgetting from happening. When there are no prior informations about the nature of the drift, it is impossible to know whether the drift will make the whole model outdated or not. In this case, we argue that the only option is to gradually adapt the outdated parts of the model as evidences that they are outdated are gathered.

The downside is that, when there is some prior information about the nature of the drift, this strategy will be too slow in certain scenarios (e.g. when an abrupt drift makes the whole model outdated).

3.2.3 Ensemble methods

Ensemble methods maintain a set of models into memory and combine their predictions. The underlying idea is to create a diversified ensemble of base learners which will compensate for each base learner weaknesses. The final prediction can be obtained by different aggregation

methods. One way could be to ask the base learners to vote. When each vote is given the same weight, the prediction of the ensemble will be obtained by a majority vote. Another way could be to weight the votes as a function of the recent performances of each base learner (weighted vote). These weights could evolve over time, ensuring that the base learners which performed the best recently are given a higher weight.

Kuncheva [64] provided a categorization of ensemble methods for evolving data that we use here. As the author points out, these categories are not necessarily exclusive and it is possible to combine them.

3.2.3.1 Dynamic combination

In this case, the base learners are trained in advance and the learned rule remains unchanged later on. As the concept drifts over time, the base learners are dynamically combined to respond to the changes by modifying the combination rule. This is usually done with weights which are adjusted to reflect their expected accuracy.

Some of the main algorithms based on this structure include Hedge β [36], WINNOW [79, 85], Weighted Majority [73], Blum [16], Tsymbal et al. [99], Widmer and Kubat [103].

Storing a model into memory without updating it at a later stage can obtain very good results in the case of reoccurring concepts. Indeed, if the algorithm manages to recognize that a model is already available to deal with the current concept, it can be selected and it won't be necessary to gather more observations. By skipping the updating step of the base learners, this strategy might also have a better running time than other classical ensemble strategies described below.

This can also prove to be a dangerous strategy when the concepts never reoccur. In this case, when the new concepts will become too different from the ones which have been used to learn the models stored in memory, the algorithm will not be able to find the necessary expertise in its memory to deal with the current concept.

3.2.3.2 Continuous update of the learners

Another way is to allow the base learners to retrain or update their model using the latest observations received. It is possible to combine continuous update of the base learners with dynamic combination by also allowing the combination rule to change over time.

Some of the main algorithms based on this structure include On-line bagging and boosting [81], Kolter and Maloof [60], Rodriguez and Kuncheva [85], Ganti et al. [43], Fern and Givan [34].

This strategy deals with the shortfalls of the dynamic combination by allowing the base learners to keep their model updated. This will give the ensemble algorithm even more flexibility to adapt to concept changes, However, this strategy might not work as well as the dynamic combination in the case of reoccurring concepts as it will need to learn the concept

from scratch again. Finally, constantly retraining the algorithms with the latest observations might increase the running time of the algorithm.

3.2.3.3 Structural update

Structural update refers to a strategy where base learners are added and removed from the ensemble. The criteria used to select the base learner which will be added/removed from the ensemble could be time (replace the oldest), past performances (replace the loser) or merit to the ensemble.

Some of the main algorithms based on this structure include Elwell and Polikar [32], Kolter and Maloof [60], Wang et al. [101], Street and Kim [94], Bouchachia [19].

This is an abrupt adaptation mechanism (as the base learners are abruptly removed and added from the ensemble).By abruptly deleting a base learner, the algorithm looses the previously acquired knowledge which could be an issue in the case of reoccurring concepts for instance. Furthermore, the speed of adaptation of the ensemble is limited by the rate at which base learners are replaced. For instance, if all the L base learners are outdated and that a new base learner can be added every X observations, it will take XL observations to completely forget the outdated concept which might prove to slow in the case of frequent abrupt drifts.

3.3 Conclusion

In this chapter, we have reviewed some of the main mechanisms that can be used in order to devise an algorithm capable of adapting to concept changes. Each of these strategies has its own strengths and weaknesses and is suited for a particular scenario. Therefore, the most important task is to start by identifying if it is possible to get an idea of the types of concepts drift that will occur on the stream. Once this information has been extracted, it will be much easier to know which type of algorithm or strategy should be used.

In this thesis, we didn't make any assumption on the characteristics of the stream or expected type of concept drifts. In this framework, one way to address this issue could be to implement a drift detector which automatically detects the type of drift and selects an adaptation strategy accordingly. However, as we have discussed in Section 3.2.2.2, drift detection is a very challenging topic. Instead, we chose to investigate adaptation to concept drifts without explicit detection. Within this framework, we claim that adaptation should happen as a consequence of hard evidences proving the old model wrong and not because of a temporal criteria. Finally, when dealing with so much uncertainty, it seems reasonable to allow the algorithm to abstain from prediction when the reliability of its predictions is too low. We show that an algorithm can take advantage of the instance based structure in order to reach theses goals and consequently we investigate these ideas in the in the following chapters.

Chapter 4

Instance-based structure to handle concept drifts

4.1 Introduction

In this chapter, we show that the instance based (IB) structure is particularly well suited for the problem of learning on a data stream subject to concept drifts. One of the key advantage is that, keeping (some of) the past observations into memory brings additional informations such as: which parts of the feature space have been explored and how well they have been explored. The second key advantage is that using the IB structure transforms the learning and adaptation problems into choosing which observations should be saved into memory. Contrarily to most IB methods which use a simple temporal criterion to select the observations that will remain into memory (i.e. delete the old observations which are assumed outdated), we use a criterion based on "conflicting observation" (i.e. as long as there is no new observation which conflicts with an old observation, the old observation should remain in memory if possible).

In this chapter, we show how these two points can be used by a meta-learning algorithm[1]. To this end, we propose a new algorithm which combines ensemble and IB learning: the Droplet Ensemble Algorithm (DEA). Contrarily to state of the art ensemble methods which select the base learners according to their performances on recent observations, DEA determines the regions of expertise of its base learners (BL) in the feature space and dynamically selects the subset of BLs which is the best suited to predict on the region of the feature space where the latest observation was received.

The chapter is organized as follows: Section 4.2 describes the Droplets Ensemble Algorithm, Section 4.3 presents the experimental protocol whereas Section 4.4 show and discuss the results of the experiments. Finally Section 4.5 concludes.

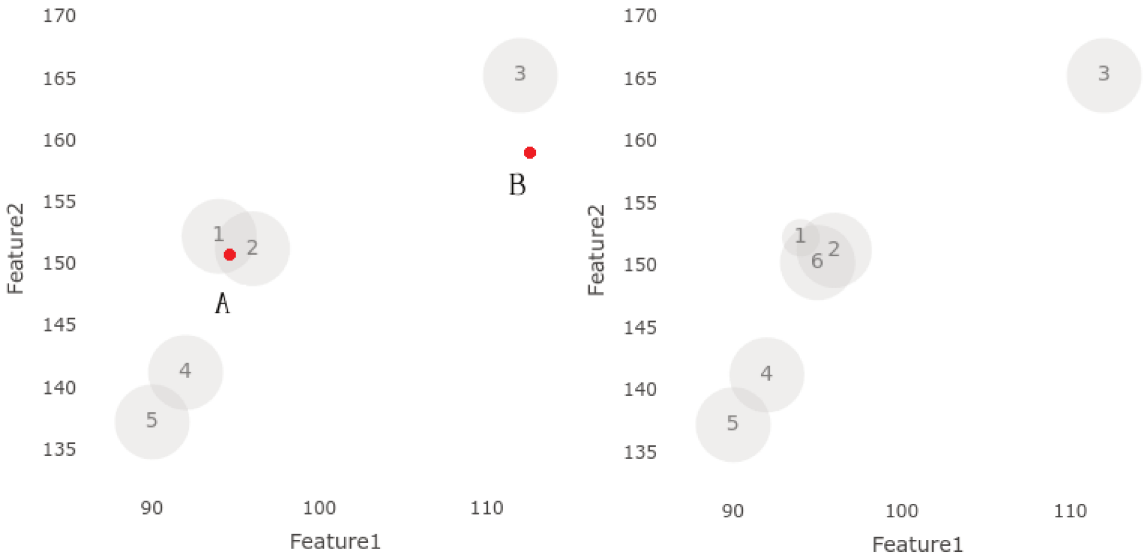

Figure 4.1: Example of map learned in 2 dimensions. Left: before update of the model with the 6^{th} observation (received at point A). Right: after update of the model with the 6^{th} observation.

4.2 The Droplets Ensemble Algorithm

Our main goal in designing our new ensemble algorithm dealing with a data stream subject to concepts drift, is to take into account the local expertise of each of its BL on the region of the feature space where the latest observation was received. This means that it gives more weight to the predictions of the BL which demonstrated an ability to predict accurately in this region.

We propose DEA (Droplets Ensemble Algorithm), an ensemble learning algorithm which dynamically maintains an ensemble of n BL $\left(F = \left\{f^1, ..., f^n\right\}\right)$ along with an ensemble of p Droplets $\left(Map = \left\{D^1, ..., D^p\right\}\right)$ up to date with respect to the current concept.

The BL can be any learning algorithms, as long as they are able to classify on a data stream subject to concept drifts.

A Droplet is an object which can be represented as a k-dimensional hypersphere (with k the dimension of the feature space). Each Droplet D^t is associated with an observation x_t and holds a pointer to a BL: f^i $(i \in \{1, ..., n\})$. The values taken by x_t correspond to the coordinates of the center of the Droplet in the feature space whereas f^i corresponds to the BL which managed to achieve the lowest prediction error on a region of the feature space defined around x_t.

Figure 4.1 shows an example of Map learned where the numbers represent the time step at which each Droplet has been received.

[1] We will study the advantage of the IB structure at the base-learning level in Chapter 6.

Algorithm 4.1 Model Prediction

Inputs: $F = \{f^1, ..., f^n\}$: Ensemble of base learners,
$Map = \{D^1, ..., D^p\}$: Ensemble of existing Droplets,
x_t: Latest unlabeled observation,
x_{const}: Normalization constants
Output: $\hat{y_t}$: Estimated class for x_t

$x_t^{norm} \leftarrow Normalize\,(x_t, x_{const})$
$OD_t \leftarrow Get\ overlapped\ Droplets\,(Map, x_t^{norm})$
If $(OD_t \neq \emptyset)$
  Foreach $D^h \in OD_t$ $(h \in \{a, ..., u\})$
    $\hat{y_t^h} \leftarrow Predict\,(D^h, x_t)$
  End Foreach
  $\hat{y_t} \leftarrow Majority\ Vote\left(\hat{y_t^a}, ..., \hat{y_t^u}\right)$
Else
  $D^{nn} \leftarrow Get\ Nearest\ Droplet\,(Map, x_t^{norm})$
  $\hat{y_t} \leftarrow Predict\,(D^{nn}, x_t)$
End If

We now go through the algorithm in details.

4.2.1 Model Prediction

At time t, upon reception of a new unlabeled observation x_t the first step is to normalize the values of x_t according to a vector of normalization constants x_{const} found on the initialization step[2]. Then OD_t, the set of Droplets which contains the normalized coordinates of the latest observation is computed. If $OD_t \neq \emptyset$, the predicted value for this observation is given by a simple majority vote of the BL associated with the overlapped Droplets in OD_t. On the other hand, if $OD_t = \emptyset$, the learner associated with the nearest Droplet D^{nn} is used for prediction. For instance, in the left plot of Fig. 1., if an observation is received at the position of point A, the BL associated with D^1 and D^2 will be used for prediction whereas if an observation is received at the position of point B, only the BL associated with D^3 will be used for prediction.

The prediction process is summarized in Algorithm 4.1.

4.2.2 Model Update

Once the true label y_t associated with the latest observation x_t is released, each BL f^i (with $i \in \{1, ..., n\}$) predicts on the latest observation and the vector of the prediction errors $e_{t+1} = \{e_{t+1}^1, ..., e_{t+1}^n\}$ (with $e_{t+1}^i \in \{0, 1\}$) is set aside. The BL are then updated with $\{x_t, y_t\}$.

[2]This is simply done by computing the average μ^i as well as the standard deviation σ^i of each feature on the initialization set and by transforming the i^{th} feature of x_t into $\frac{x_t^i - \mu^i}{\sigma^i}$

The next step is to search for the BL which will be associated to the new Droplet D^t. This is done by summing the prediction errors achieved by each BL on the N nearest Droplets, where N is a parameter defined by the user. If an unique BL minimizes this sum, it is associated to the new Droplet, otherwise (if at least 2 BL minimizes the sum of prediction error) the search space is expanded in turns to the $N+1, N+2, N+3, ...$ nearest Droplets until a single best performer is found.

The new Droplet D^t is then added to the feature space at the coordinates of x_t^{norm}. This Droplet is given a default radius $R_{default}$ (where $R_{default}$ is a parameter defined by the user), stores the vector of prediction errors e_{t+1} and creates a pointer to the best BL f^k found on the previous step.

The algorithm then goes through the set of overlapped Droplets OD_t and if it is not empty, it decreases the influence of the Droplets in OD_t which have outputted a wrong prediction on x_t. This is done by shrinking their radius which will make them less likely to predict on a future observation received in this region of the feature space. Formally, for each Droplet u in OD_t :

1. Compute the overlap between D^u and the latest Droplet:
   $Overlapp_u = R_{default} + R_u - \|x_u^{norm} - x_t^{norm}\|$ (where $\|.\|$ denotes the Euclidean distance)
2. Update the radius of D^u: $R_{u,t+1} = R_{u,t} - \frac{Overlapp_u}{2}$.
3. Delete D^u if $R_{u,t+1} \leq 0$.

For instance the right plot of Fig. 1. shows the updated model after reception of an observation at the position of point A and where the BL associated with D^1 outputted a wrong prediction on the 6^{th} observation whereas the BL associated with D^2 predicted correctly.

Finally, a memory management module is ran at each time step to ensure that p, the user defined parameter for the maximum number of Droplets allowed in memory is not exceeded. If the memory is full, the algorithm uses 3 different criteria to select the Droplet which will be removed:

1. Remove the Droplet with the smallest radius.
2. If all the Droplets have the same radius, remove the Droplet which has outputted the highest number of wrong prediction.
3. If criteria 1. and 2. failed, remove the oldest Droplet.

Algorithm 4.2 summarizes the model update process.

4.2.3 Running time and memory requirements

Running time: Provided that each of the base learner runs in constant time at each time step, the temporal complexity of both the prediction and update steps of DEA is $\mathcal{O}(i.p)$ with i is the number of observations generated by the stream so far and p the maximum number of Droplets allowed in memory.

Algorithm 4.2 Model Update

Inputs: $R_{default}$: Default radius of a Droplet,
$F = \{f^1, ..., f^n\}$: Ensemble of base learners,
$Map = \{D^1, ..., D^p\}$: Ensemble of existing Droplets,
x_t: Latest unlabeled observation,
y_t: True label latest observation,
p: Maximum number of Droplets allowed in memory,
OD_t: Set of overlapped Droplets at time t
Output: Updated DEA

Foreach f^i in F
    $e^i_{t+1} \leftarrow Get\ Prediction\ Error\left(f^i_t, \{x_t, y_t\}\right)$
    $f^i_{t+1} \leftarrow Update\ Base\ Learner\left(f^i_t, \{x_t, y_t\}\right)$
End foreach
$f^k \leftarrow Search\ best\ base\ learner\left(Map, \{x_t, y_t\}\right)\ ,\ k \in \{1, ..., n\}$
$D^t \leftarrow Create\ Droplet\left(R_{default}, x^{norm}_t, f^k, e_{t+1}, sum\ errors = 0\right)$
$Map \leftarrow Add\ Droplet\left(Map, D^t\right)$
Foreach $D^u \in OD_t$
    $R_{u,t+1} \leftarrow Update\ Radius\left(R_{u,t}\right)$
    If $R_{u,t+1} \leq 0$
        $Map \leftarrow Remove\ Droplet\left(Map, D^u\right)$
    End if
End foreach
If $\left(Card\left(Map\right) \geq p\right)$
    $Map \leftarrow Memory\ Management\left(Map\right)$
End if

Space requirements: As previously explained, the maximum number p of Droplets saved into computer memory is constrained and so is the number n of base learners. This means that, as long as each of the n base learner constrains its memory consumption at each time step, the space complexity of DEA will be $\mathcal{O}(n+p)$ which is independent of the number of observations generated by the stream so far.

4.2.4 Handling concept drifts using the instance based structure

The first key point with this algorithm is that the IB structure allows to use space on top of time and prediction errors as criteria to weight the BLs. The addition of this extra feature for the meta-learning algorithm results in learning the BL(s) which is/are expected to perform the best for each regions of the feature space. These regions are automatically defined according to the values associated with the past observations and the frequency at which each of these values has been observed.

Adaptation to concept drifts happens by locally reducing the influence of poorly performing BLs and associating new Droplets with the BL which achieved the best performances in the region of the feature space where the latest observation was received.

Note 1: In this case, the meta-learner adapts to a consequence of the concept drift rather than the concept drift itself. Adaptation to concept drifts is done by the BLs. However, each BL relies on a model which carries assumptions which might not be suited for the current concept or for a particular region of the feature space at a given time. Therefore, the meta-algorithm adapts to the evolution of the best available BL for each region of the feature space.

This raises the second key point with this algorithm: using the IB structure transforms the learning/adaptation problem into choosing which observations will be saved into memory. This is an interesting property because we claim that instead of simply relying on the assumption that old Droplets are outdated (and thus that time should be the criteria used to forget), we claim that each observation in memory is "good until proven wrong" (and thus that forgetting must happen based on this idea of conflicting information). Indeed, as long as the base learner which is associated to a Droplet has not been proved outdated by a better performing BL, it will be used for prediction, regardless of how old is the Droplet associated with it. The main advantage of relying on this criterion is that it allows to adapt to concept drifts without explicitly having to detect them, thus removing a very challenging problem from the adaptation process.

Note 2: This algorithm bear some resemblance with the IB3 algorithm [4] and Salganicoff [87] which keep track of the classification performance (i.e. the number of correct and incorrect predictions) of each observations saved into memory. When a particular observation performs poorly, it is removed from memory. It should be noted however, that IB3 was only designed for the incremental learning setting and can't handle concept drifts.

Dataset	Features	Classes	Observations	# Drifts	Type of Drift
Agrawal	9	2	100 000	0	N/A
Random Tree	10	6	100 000	15	Abrupt, Reoc.
Waveform	21	3	100 000	Continuous	Real, Local
LED	7	10	100 000	Continuous	Real, Local
KDD Cup	41	2	494 000	N/A	N/A
Rotating Check	2	2	419 600	Continuous	Real, Gradual
SPAM	500	2	9324	N/A	N/A
Usenet	658	2	5 931	N/A	N/A
Airlines	7	2	153 200	N/A	N/A
Multi Dataset	3	2	200 000	3	Abrupt, Real
Multi Dataset NO	3	2	200 000	Continuous	Abrupt, Real, Virt.
Weather	8	2	18 100	N/A	N/A
SEA	3	2	50 000	3	Real, Abrupt
Chess	2	8	200 000	0	N/A
Transient Chess	2	8	200 000	Continuous	Virt., Reoc.
Mixed Drift	2	15	600 000	Continuous	Incr., Abrupt, Virt.
Moving Square	2	4	200 000	Continuous	Real, Gradual
Interchanging RBF	2	15	200 000	9	Real, Abrupt
Moving RBF	10	5	200 000	Continuous	Real, Gradual
Cover Type	54	7	581 000	N/A	N/A
Electricity	8	2	45 300	N/A	N/A
Outdoor Stream	21	40	4 000	N/A	N/A
Poker Hand	10	10	829 200	N/A	N/A
Rialto	27	10	82 200	N/A	N/A
Rotating Hyp.	10	2	200 000	Continuous	Real, Gradual

Table 1. Datasets used and their characteristics

4.3 Experimental Framework

In this section, we describe the datasets on which the experiments have been conducted, their characteristics as well as the experimental protocol used.

4.3.1 Datasets

A total of 25 artificial and real world datasets have been used. These datasets have been chosen for the diversity of their characteristics, which are summarized thereafter:

Most of these datasets have frequently been used in the literature dedicated to streams subjects to concept drifts. Also, please note that in this table, an "N/A" value doesn't mean that there is no concept drift. It means that, because the dataset comes from the real world, it is impossible to know for sure the number of drifts it includes as well as their type.

The first 4 datasets from Agrawal to LED have been generated using the built-in generators of MOA[3]. A precise description of theses datasets can be found in the following papers [22, 3, 29]. The KDD Cup 10 percent dataset was introduced and described in [97]. Rotating Check board was created in [32] and the version CB (Constant) dataset was used (constant drift rate). SPAM was introduced in [54] and Usenet was inspired by [53]. Airlines was introduced in the 2009 Data Expo competition. The dataset has been shrunk to the first 153 000 observations.

[3] http://moa.cms.waikato.ac.nz/

Multidataset is a new synthetic dataset created for this paper. Every
50 000 observations, the concept drifts to a completely new dataset, starting with Rotating checkboard, then Random RBF, then Rotating Hyperplane and finally SEA. In the basic version, the successive concepts overlap each other whereas in the No Overlap (NO) version the datasets are shifted and the data are randomly generated on each dataset.

Finally, all the datasets listed after Weather have been retrieved from the repository[4] given in the paper of Losing et al. [75].

All the datasets used as well as the code of the DEA algorithm and the results of the experiments are available at the following link[5].

4.3.2 Benchmarks

We now present some of the state of the art, drift handling algorithms which will be used as base learners and benchmark.

ADACC was introduced in [51]. It maintains a set of BL which are weighted every τ time steps according to their number of wrong predictions. It then randomly selects one BL from the worst half of the ensemble and replaces it by a new one which is protected from deletion for a few time steps. The final prediction is given by the current best performer. The algorithm also includes a mechanism to remember past concepts.

Dynamic Weighted Majority (DWM) is an ensemble method introduced in [60]. Each of its BL has a weight which is reduced in case of a wrong prediction. When a BL's weight drops bellow a given threshold, it is deleted from the ensemble. If all the BL output a wrong prediction on an instance, a new classifier is added to the ensemble.

ADWIN Bagging (Bag Ad) was introduced in [13] and improves the On-line Bagging algorithm proposed by Oza and Rusell [81] by adding the ADWIN algorithm as a change detector. When a change is detected, the worst performing BL is replaced by a new one.

Similarly, **ADWIN Boosting** (Boost Ad) improves the on-line Boosting algorithm of Oza and Russell [81] by adding ADWIN to detect changes.

Leveraging Bagging (Lev Bag) was introduced in [14] and further improves the ADWIN Bagging algorithm by increasing re-sampling (using a value λ larger than 1 to compute the Poisson distribution) and by adding randomization at the output of the ensemble by using output codes.

Hoeffding Adaptive Tree (Hoeff Tree) was introduced in [11] and uses ADWIN to monitor the performance of the branches on the tree. When the accuracy of a branch decreases, it is replaced with a more accurate one.

AccuracyUpdatedEnsemble (AUE) described in [25] maintains a weighted ensemble of BL and uses a weighted voting rule for its final prediction. It creates a new BL after each chunk of data which replaces the weakest performing one. The weights of each BL are computed according to their individual performances on the latest data chunk.

Finally, **SAM KNN** [75], best paper award at ICDM 2016, is a new improvement method of the KNN algorithm. It maintains the past observations into 2 types of memories (short and

[4]https://github.com/vlosing/driftDatasets
[5]https://mab.to/o5iNvZdhH

long term memory). The task of the short term memory is to remain up to date according to the current concept whereas the long term memory is in charge of remembering the past concepts. When a concept change is detected, the observations from the short term memory are transfered to the long term memory.

4.3.3 Experimental Setting

The performances of the classifiers will be evaluated according to the 0-1 loss function: $\mathscr{L}(y, \hat{y}) = \mathbb{I}_{\{y \neq \hat{y}\}}$ (where $\mathbb{I}$ is the indicator function), and the goal is to minimize the average error $\left(\frac{1}{n}\sum_{i=1}^{n}\mathscr{L}(y_i, \hat{y}_i)\right)$ over the n observations received so far.

MOA have been used to conduct the experiments and provide the implementation of the classifiers. DEA was also implemented in MOA. The code for SAM KNN was directly retrieved from the link provided in their paper [75][6].

All the parameters of all the classifiers were set to default values (except for the training period which was set to 100 observations for all the learners and for the number of observations allowed in memory which was set to 400 for DEA and SAM KNN) and for all the datasets. In the case of the Droplets algorithm, the default radius was set to 0.1 and the minimum number of neighbors considered was set to 20 for all the experiments. We used all the algorithms described in this paper as BL for DEA (they were chosen because of their availability on MOA) with the exception of SAM KNN and Majority Vote. The simple majority vote algorithm (which uses the same BL as DEA) was used as a base-line for performance comparison.

Leaving all the parameters to default values for all the datasets is required because there is no assumptions regarding the structure of the data or the type of drifts the classifiers will have to deal with. Therefore, it wouldn't be relevant to optimize parameters that would be suitable for a particular concept, at a particular time and for a particular dataset.

The goals of the experiments were to compare the performance of DEA against one of the currently best adaptive algorithm (SAM KNN), assess how DEA was faring against another ensemble algorithm which is given the same BL (Majority Vote) and assess whether DEA was able to outperform each of its BL.

For each dataset, the performance of the algorithms was computed using the prequential method (interleaved test-then-train) described in section 2.4.2.2.

4.4 Results and discussion

The accuracy (percentage of correct classifications) obtained by each algorithm on each dataset are reported in Table 2. Bold numbers indicate the best performing algorithm. The bottom 2 lines show the average accuracy as well as the average rank obtained by each algorithm on all the datasets.

The results indicate that DEA managed to obtain the best average accuracy as well as the best average rank on the 25 datasets considered. In particular, the average rank obtained

[6] https://github.com/vlosing/driftDatasets

Dataset	DEA	SAMKNN	ADACC	DWM	Bag Ad	Lev Bag	Hoeff Tree	Boost Ad	AUE	Maj Vote
Agrawal	94.20	92.53	84.83	81.46	**94.87**	93.97	94.24	90.13	90.80	93.62
Rand Tree	51.75	32.27	29.08	30.25	46.93	**52.36**	35.76	19.80	48.35	43.89
Waveform	83.26	82.15	76.35	77.37	82.95	**84.84**	81.55	81.35	80.49	82.87
LED	73.51	71.12	64.67	63.46	73.94	73.88	**73.95**	73.69	73.93	73.86
KDD Cup	**99.91**	99.90	99.84	99.62	99.87	99.91	99.78	98.91	82.44	82.71
Rotating Check	92.94	91.39	79.65	76.70	84.68	93.03	84.39	**93.95**	82.68	87.41
SPAM	**96.60**	94.95	94.91	92.30	90.95	96.04	90.60	95.54	65.36	69.45
Usenet	60.30	57.24	61.00	60.85	56.43	61.95	56.91	59.17	61.07	**62.28**
Airlines	66.40	65.16	62.44	60.25	68.14	65.41	66.26	62.72	67.36	**68.54**
Multi Dataset	95.11	92.56	90.99	89.53	93.80	**95.31**	92.52	93.71	92.90	93.96
Multi Dataset NO	90.25	85.09	55.00	54.39	88.28	**90.86**	88.73	87.93	89.19	88.02
Weather	76.52	76.02	73.56	72.22	75.00	**78.14**	73.53	74.40	74.55	76.74
SEA	87.77	85.45	85.12	84.72	86.77	**88.33**	86.76	82.33	86.85	87.64
Chess	**94.92**	78.22	14.45	13.84	55.35	94.84	58.49	12.64	88.02	76.67
Transient Chess	**94.98**	85.37	58.03	57.07	56.11	89.52	37.27	56.12	26.03	39.68
Mixed Drift	76.64	**91.57**	37.56	35.63	59.54	75.17	55.70	42.76	68.23	64.91
Moving Square	99.08	97.36	**99.14**	83.31	88.02	87.85	74.89	79.66	67.18	88.67
Inter RBF	98.45	98.00	**98.59**	97.14	88.92	94.10	56.81	94.14	91.61	96.17
Moving RBF	59.18	**86.98**	44.65	45.80	52.31	55.02	38.72	45.43	54.70	53.09
Cover Type	92.69	**93.58**	90.38	84.29	84.02	90.40	80.54	92.55	82.69	86.65
Electricity	**90.66**	82.54	89.55	82.34	83.63	89.49	82.83	87.34	78.98	87.32
Outdoor Stream	69.43	**88.25**	66.88	58.32	58.91	60.21	57.20	57.70	40.08	38.63
Poker Hand	88.09	79.77	79.36	75.64	73.46	85.68	65.54	**91.74**	68.65	80.47
Rialto	70.92	**81.90**	71.21	45.27	49.46	60.43	30.65	18.34	47.35	40.76
Rotating Hyp	86.89	81.42	82.78	83.46	88.02	86.87	86.45	76.58	87.20	**88.44**
Average Accuracy	**83.62**	82.83	71.60	68.21	75.21	81.75	70.00	70.74	71.87	74.10
Average Rank	**2.48**	4.72	6.44	7.96	5.28	2.96	7.12	6.72	6.36	4.96

Table 2. Percentage of correct classification achieved on each dataset.

demonstrates the ability of DEA to perform consistently well regardless of the characteristics of the dataset and of the type of drifts encountered. This is an interesting property because it is often impossible to predict how the stream will evolve over time and thus, an algorithm which can deal with a very diversified set of environments could be useful as it wouldn't be possible to pick right from the beginning the algorithm which is the best suited for the whole dataset.

This good performance also confirms that using the local expertise of the BL as a selection criteria to decide which subset will be used for prediction should be considered as a way to improve the performances of an ensemble learning algorithm. Indeed, DEA over-performed the ensemble learning algorithms which rely on the latest performances to weight their BL (ADACC, DWM, AUE, ...) as well as a Majority Vote algorithm which simply ask all the algorithms to collaborate for prediction, independently of the observation received.

4.5 Conclusion

In this chapter, we have claimed that IB learning algorithms had some properties which made them particularly well suited for the problem of learning on a data stream subject to concept drift. The two mains properties highlighted here were:

1. Using the IB structure at the meta level allows to add space on top of time and prediction error as criteria to weight the BLs of the ensemble. By doing so, it is possible to learn the area of expertise of each BL in the feature space and use them accordingly.

2. Within the IB structure, use a criterion of "conflicting information" as a forgetting mechanism (instead of time to select the observations which should be deleted from memory). By doing so, it is possible to adapt to concept drifts without explicitly detecting them.

In order to test these ideas, we have proposed the Droplets Ensemble Algorithm (DEA), a novel algorithm which combines the properties of an instance base learning algorithm with the ones of an ensemble learning algorithm. It maintains into memory a set of hyper-spheres, each of which includes a pointer to the BL which is the most likely to obtain the best performance in the region of the feature space around that observation. When a new observation is received, it selects the BLs which are likely to obtain the best performance in this region and use them for prediction. The algorithm adapts to changes by locally reducing the weight of the BLs which performed poorly and constantly adding new Droplets associated with BLs which are expected to perform well locally.

The experiments carried on a set of 25 diversified datasets, reproducing a wide variety of drifts show that our algorithm is able to over-perform each of its base learners, a majority vote algorithm using the same base learners as well as SAM KNN (one of the currently best adaptive algorithm) by obtaining the best average accuracy and rank. These results indicate that our algorithm is well suited to be used as a general purposed algorithm for predicting on data streams with concept drifts and that the IB structure should be considered when designing an algorithm learning in this environment.

Chapter 5

Dealing with uncertainties by abstaining

5.1 Introduction

In this chapter, we investigate the problem of prediction with a reject option in the framework of data streams subject to concept drifts. Our goal is to assess whether abstaining should be considered as a way to improve the performances of a drift handling algorithm when costs can be associated with the outcome of predictions.

To this end, we consider the specific problem of regression with a constraint on the precision of each predictions; a scenario which can appear in cost-sensitive applications of machine learning such as medicine or financial forecasting. Concept drifts can diminish the reliability of the predictions over time and it might not be possible to output a prediction which satisfies the constraints on the precision. In this case, we claim that if the costs associated with a good and with a bad prediction are known beforehand, the overall prediction cost can be improved by allowing the regressor to abstain. To this end, we propose a generic method, compatible with any regressor, which uses an ensemble of reliability estimators to estimate whether the constraints on the precision of a given prediction can be met or not. In the later case, the regressor is allowed to abstain. Empirical results on 30 datasets including different types of drifts back our claim.

5.2 Framework and proposed method

In this section, we lay down the framework, introduce our problem with a real life example and propose a generic method aimed at improving the performance of any regression algorithm able to learn on a stream of data subject to concept drifts. We also discuss the choices made for the parameters and how we constrain the suitable abstention costs.

5.2.1 Framework

In the supervised regression setting, the goal is to learn a predictor $h : \mathcal{X} \to \mathcal{Y}$, (with $\mathcal{X}$ the input space and $\mathcal{Y} \in \mathbb{R}$ the output space) capable of generating accurate output predictions $\hat{y} = h(x) \in \mathbb{R}$ for any unlabeled observation $x \in \mathcal{X}$. Each observation $z \in \mathcal{Z}$ (with $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$) is generated by a stream which starts emitting observations $\{z_{t_1}, z_{t_2}, ...\}$ at time t_0. Each observation z_t is generated according to a hidden distribution $\mathcal{D}_t$ over $\mathcal{Z}$. The distribution $\mathcal{D}_t$ (also referred as *concept*) can change over time and we will say that the concept $\mathcal{D}_t$ that the predictor is trying to learn, has drifted at time t if $\mathcal{D}_{t-1} \neq \mathcal{D}_t$. When the concept is stable, the observations are assumed to be i.i.d. realizations of a single concept whereas when the concept is drifting they are only assumed independent from each other.

The regressor h is further allowed to abstain from prediction, a framework commonly known as *selective regression* [104]. In this case, the regressor h is associated with a *selection function* $g : \mathcal{A} \to \{0, 1\}$ (where the input space $\mathcal{A}$ can change from one selection function to the other) whose meaning is as follows:

$$(h, g)(x_t) = \begin{cases} \emptyset & if\ g(a_t) = 0 \\ h(x_t) & if\ g(a_t) = 1 \end{cases}$$

where $\emptyset$ denotes an abstention on the unlabeled observation x_t and where $a_t \in \mathcal{A}$.

The problem considered here is to produce a predictor h (or a predictor associated with a selection function (h, g)) that minimizes the expected cost from prediction on the n observations received so far: $\frac{1}{n}\sum_{i=1}^{n} C(h(x_i), y_i)$ (where $C : (\mathcal{Y} \cup \emptyset) \times \mathcal{Y} \to [0; 1]$ is a cost function) under the constraint of a required precision threshold ϵ. Formally, we define the $\epsilon - tube$ cost function presented in [55] as:

$$C_{\epsilon}^{0-d-1}(\hat{y}, y) = \begin{cases} 0 & if\ |y - \hat{y}| \leq \epsilon\ and\ g(a) = 1 \\ 1 & if\ |y - \hat{y}| > \epsilon\ and\ g(a) = 1 \\ d > 0 & if\ g(a) = 0 \end{cases}$$

where ϵ is a threshold determined by the problem at hand and d is the cost associated with an abstention.

Note: The problem considered here shares some similarities with the task of giving *predictions intervals* [56]. The difference is that, in the case of a prediction interval, the goal is to estimate the interval into which the observation will fall with a given probability in order to quantify the uncertainty in the point forecasts whereas here, the interval is constrained by the requirements of the problem at hand. In both cases though, the performance of the underlying algorithm is assessed by checking whether the observation falls into the predicted interval.

5.2.2 Real life example

In order to illustrate the problem, we describe the case of an investor which is betting on a "Binary Tunnel Option". A Binary Tunnel Option is a financial product that rewards the

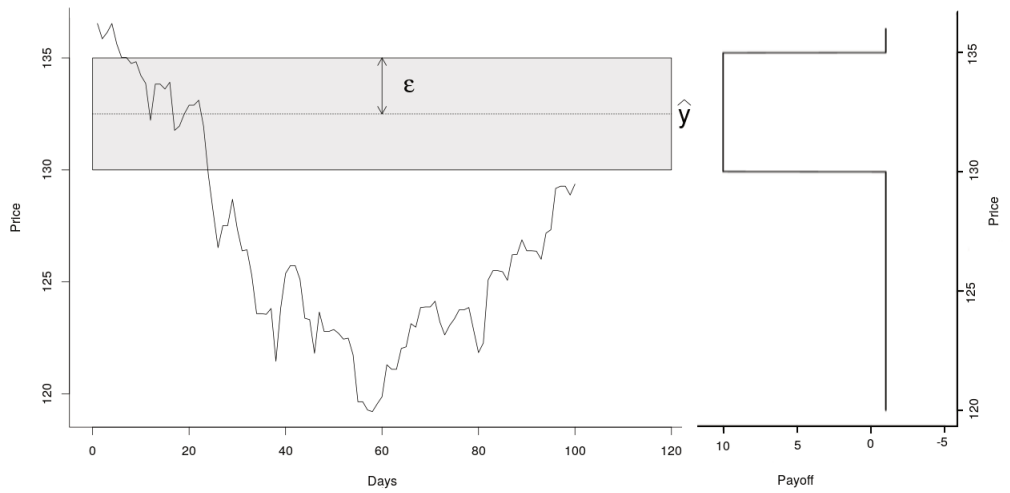

Figure 5.1: Target range (left) and payoff function (right) of a Binary Option Tunnel

buyer of the product if he manages to accurately forecast whether the price of the underlying asset will be in a given range at the maturity date of the option.

For instance, assume that the current price of stock S is 129€ (see the left hand side of Fig. 5.1). Based on his machine learning algorithm, an investor predicts that the price of stock S will be $\hat{y} = 132.5$€ in 20 days. He can then call a broker and ask him to create a Binary Tunnel Option for which the thresholds are for instance $\hat{y} - \epsilon = 130$€ and $\hat{y} + \epsilon = 135$€ (thus, in this case, $\epsilon = 2.5$). If the broker manages to find another investor willing to sell this option, the first investor can buy the option for a price p (1€ for instance) and will get a profit (for instance 10€) only if the price of stock S is within the 130€-135€ range at the closing price in 20 days. Otherwise, if the price of stock S at the maturity date is outside this range, the investor loses his initial investment (1€). The payoff of the option in this particular example is shown on the right hand side of Fig. 5.1.

5.2.3 Proposed method

The underlying idea of the proposed method is that by allowing a predictor to abstain, it is possible to achieve better performances at the cost of a smaller coverage (the proportion of observations for which a non-empty prediction is given). Therefore, we propose a generic method which assess whether the constraint on the required precision can be met. When this is not the case, the predictor is allowed to abstain in order to avoid a costly error.

Note: For the rest of this chapter, "reliability estimate" is used to refer to an estimate of the prediction error.

Algorithm 5.1 Generic Method

Inputs: unlabeled observation: x_t, predictor learned at previous step: h_{t-1}, ensemble of reliability estimator: $\left\{RE^1, ..., RE^m\right\}$, ensemble of confidence thresholds: $\left\{q^1, ..., q^m\right\}$, ensemble of inputs specific to each reliability estimator: $\left\{Q^1, ..., Q^m\right\}$, selection function: g

01: $\hat{y_t} \leftarrow h_{t-1}(x_t)$
02: $\hat{\mathbf{r_t}} \leftarrow Compute\ REs\left(RE^1, ..., RE^m, Q^1, ..., Q^m\right)$
03: $\mathbf{a_t} \leftarrow Apply\ Thresholding\left(\hat{\mathbf{r_t}}, q^1, ..., q^m\right)$
04: **if** $g(\mathbf{a_t}) = 0$ **then**
05: $\quad \emptyset \leftarrow (h, g)(x_t)$ //Abstain
06: **else**
07: $\quad \hat{y_t} \leftarrow (h, g)(x_t)$ //Predict
08: **end if**
09: $h_t \leftarrow Update(h_{t-1}, x_t)$ //Update the learned model

5.2.3.1 Method

The full method is detailed in Algorithm 5.1 and is described here: at time t, upon reception of x_t, the predictor h_{t-1} learned at time $t-1$ outputs an estimated prediction $\hat{y_t}$. The estimate $\hat{y_t}$ is then set aside. A set of m reliability estimators $RE^i,\ i = 1, ..., m$ then associates reliability estimates $\hat{\mathbf{r_t}} = \left\{\hat{r_t^1}, ..., \hat{r_t^m}\right\} \in (\mathbb{R}^+)^m$ to $\hat{y_t}$ where small values of $\hat{r_t^i}$ indicate that the reliability estimator is confident that the prediction $\hat{y_t}$ is close to the target value y_t, and large values of $\hat{r_t^i}$ indicate a lack of confidence.

For each reliability estimator, a confidence threshold q^i is set and if $\hat{r_t^i} \leq q^i$, the prediction is deemed as reliable according to this reliability estimator. The final decision of each RE $\left(\mathbf{a_t} = \left\{a_t^1, ..., a_t^m\right\}\right)$ is then aggregated through a selection function g and the prediction $\hat{y_t}$ is used if the selection function assessed the prediction as reliable. Otherwise, $\hat{y_t}$ is discarded and the predictor abstains for this observation.

Indeed concept drifts can diminish the reliability of the predictions over time and when the reliability of a given prediction is too small, it might not be possible to output a prediction which satisfies the constraints on precision required by the problem at hand. In such cases, abstaining should be considered as a way to improve the overall expected cost.

One major advantage of this method is that the REs do not depend on a particular predictor and thus can be used with any base algorithm, as long as it is able to deal with concept drift on a stream. There are, however, some constraints on the REs, as they must be able to operate on-line, use limited memory, have a low processing time and be able to cope with non-stationary distribution.

Another advantage of our approach is it relies on an ensemble of REs: Previous empirical evaluations of existing reliability estimates showed that the best RE depends on the regressor and on the dataset [17, 24, 98]. These results are of particular interest in the framework of concept drift, where the characteristics of the dataset can evolve over time. To tackle this issue, several studies have been carried showing the interest of ensemble approaches [98, 17]

for the estimation of the reliability of individual prediction. We proposed here to use a simple majority vote but a different aggregating technique could have been used [18]. Rather than the choice of a specific ensemble method, the emphasis here is put on being able to reject prediction estimates with low reliability regardless of the base regressor used and of the characteristics of the dataset considered.

Building up on these previous findings, the contribution of the proposed method is to show the necessity to allow abstention when learning on a data stream subject to concept drifts.

Note: At this point, it should be emphasized that the generic method previously proposed has no impact on the update of the hypothesis h. Whether the selection function g choses to abstain or predict, the hypothesis h learned after update with the latest observation, is the same as the one obtained without the selection function.

5.2.4 Setting the parameter's value

Many of the REs used require to set some parameters. In the framework of concept drift, setting an appropriate value for the parameter of an algorithm is a difficult task as a given parameter would only be optimal at a given time [63], for a given concept, on a given dataset and for a given algorithm. Consequently, we chose to use non optimized parameters which are set to default values regardless of the datasets or the base learner and the task of optimally setting parameters is left for future work.

We also applied the same principle for the numerical confidence threshold q of each RE. As previously stated, each RE is an estimate of the prediction error and thus, because they evaluate the same value, the same threshold q was used for all the linear REs $(\hat{y} - y)$ whereas q^2 was used for the REs with a quadratic form $(\hat{y} - y)^2$. Finally, a value also had to be given to the confidence threshold q which also depends on the problem at hand. Because each RE is an estimate of the prediction error, we chose to set $\epsilon = q$ for all the experiments.

Indeed, when ϵ is small, the problem considered requires a lot of precision and thus the requirement for being confident on a particular observation should be tougher. On the other hand, if ϵ gets larger, the need for precision decreases and thus the requirements on the confidence estimators should also be looser.

5.2.5 Defining which abstention costs are suitable

Ultimately, the value of d depends on the problem at hand, however, for the remainder of this chapter, we will set $d = \frac{1}{2}$ which is the "worst" abstention cost that we will consider. Indeed, regardless of the algorithm considered, if $d \in \left[0; \frac{1}{2}\right]$, there exist some cases where abstaining can improve the performances. Conversely, if $d > \frac{1}{2}$, there are some algorithms for which abstaining will never improve the performances. For this reason, we restrict our framework to $d \in \left[0; \frac{1}{2}\right]$ and we will choose to never abstain if $d > \frac{1}{2}$.

Proof: The use of the $\epsilon - tube$ cost function transforms a regression problem into a binary classification problem where the goal at each observation is to predict $\hat{y} \in A$ where $A =$

$[y - \epsilon; y + \epsilon]$.

In the binary classification setting, regardless of the problem at hand, the Bayes rule equipped with the true posterior probabilities, will always minimize the probability of misclassification [28]. For instance, on a given observation x, if the true posterior probabilities are $P(Y = A/X = x) = 0.6$ and $P\left(Y = \bar{A}/X = x\right) = 0.4$ then, the Bayes rule will always predict the class associated with the highest posterior probability. In this particular example, the expected cost from prediction is $0.6 \times 0 + 0.4 \times 1 = 0.4$ and abstaining should be considered only if $d \leq 0.4$ (the expected cost from abstaining is always known and is equal to d). More generally, the worst expected prediction cost is achieved if $P(Y = A/X = x) = P\left(Y = \bar{A}/X = x\right)$ and is 0.5. Thus, regardless of the dataset considered, if $d > \frac{1}{2}$ then the Bayes rule should never abstain.

Because any algorithm will have a worse misclassification rate than the Bayes rule with the true posterior rule, the cases where abstaining improves the expected prediction cost of the Bayes rule will also improve the expected prediction cost of any other algorithm. For this reason, we restrict our framework to $d \in \left[0; \frac{1}{2}\right]$ and more particularly to the "worst" case $d = \frac{1}{2}$.

5.2.6 Choosing a selection function

For the choice of selection function g, we used a simple majority vote which chooses to use the initial prediction $\hat{y}$ only if an absolute majority of reliability estimators marked the prediction as reliable. Here again, the performances can be enhanced by a wiser choice of selection function.

5.3 Related work

We start this section by reviewing some of state of the art regression algorithms, able to adapt to concepts change, process observations upon reception and use limited computer memory. These algorithms will be used in the experimental section. We then review regression algorithms which can abstain when a prediction is deemed unreliable. Finally, we discuss why these algorithms are not suitable to tackle our problem.

5.3.1 Regression algorithms for data stream subject to concept drift

The overwhelming majority of the algorithms able to handle drifting concepts have been designed to predict in the classification setting. However, a few algorithms have been devised for the regression setting.

Shaker and Hullermeier [93] proposed **IBLStreams**, an instance-based algorithm able to learn under the classification and regression settings. The algorithm is able to autonomously optimize the composition and the size of the case base. The latest observation is first added to the case base and then, the algorithm checks whether some of the past observations should be removed, either because they have become redundant, either because they are outliers. The recent observations, however, are excluded from removal.

Ikonomovska et al. [49] developed **FIMT-DD**, an incremental algorithm for learning regression trees from data streams. The algorithm is equipped with mechanisms for adaptation and drift detection which allow the local update of the tree if necessary. In order to constrain the consumption of memory, a method for disabling bad split points is included.

In [6], Almeida et al. devised **AMRules**, a rule learning algorithm for regression problems on data streams where each rule is created as a linear combination of attribute values. Each rule uses the Page-Hinkley test [82] to detect changes and model adaptation happens by pruning the rule set. The algorithm also allows to differentiating the importance of the training observation by the use of weights.

Duarte and Gama [30] proposed **Random AMRules**, an on-line ensemble method that combines a set of rules created by the AMRules algorithm. A mechanism prevents the base models from being correlated by randomly choosing the set of attributes considered for each base rule. The final prediction of the model is a simple linear combination of the predictions produced by the base models where the weight of each model can be set either uniformly, either according to the performance of the base model.

Note: These 4 algorithms are used as a baseline later in the experimental section.

5.3.2 Regression with a reject option

The issue of selective prediction has been abundantly addressed in the classification setting, however, similarly to the algorithms developed for data streams, only a few models were devised for the regression setting.

El-Yaniv and Wiener [104] developed a strategy for learning selective regressors which are guaranteed to achieve $\epsilon-$point-wise optimality (when the regressor is able to achieve results which are arbitrarily close the optimal regressor in hindsight, on the set of observations for which a prediction is given) under the assumption that the observations are i.i.d. realizations of a static concept $\mathcal{D}$.

Kegl [55] devised MedBoost, a boosting algorithm for regression that uses the weighted median of base regressors as final regressor. The special case where the base regressors as well as the final decision abstain is briefly considered.

In [95], a special type of on-line linear regression (Know What It Knows Linear Regression) is introduced by Strehl and Littman. The authors devised 2 uncertainty measures for the least-squares estimate and allow the algorithm to abstain from prediction when the confidence in this estimate is not high enough. Unfortunately, despite its on-line learning ability, the algorithm isn't suited to deal with drifting concepts as it assumes that the concept doesn't change over time.

The case for abstention in the regression setting also appears within the framework of conformal prediction [7, 42]. In this framework, it is possible to give guarantees on the accuracy of an algorithm under the assumption that the observations are i.i.d. realizations of a static concept $\mathcal{D}$. Unfortunately, this assumption doesn't hold in the framework of concept drifts.

5.3.3 Shortfalls of the related works

On the one hand, the existing regressors suited to learn on a data stream subject to concept drift never abstain from prediction in their current form. On the other hand, the state of the art algorithms for regression with a reject option have not been devised to operate under a stream of data subject to concept drifts. Therefore, we propose to improve the performances of drift handling algorithms by allowing them to abstain.

5.4 Experimental study

In this section, we describe the reliability estimators used, characterize the types of concept drifts reproduced in the synthetics datasets, describe the experimental protocol as well as the success metrics.

5.4.1 Description of the On-line Reliability Estimators

We chose to implement 7 of the reliability estimators (REs) presented in the work of Rodrigues et al. [84] as they are well suited for data streams. Here, rather than the particular reliability estimates used, the emphasis is on creating a diversified set of REs, suited to operate on a data stream subject to concept drifts. A brief description of the reliability estimators is given below the interested reader is referred to the original paper for further details.

Similarity-based reliability estimate: The underlying idea of this estimate is to use temporal similarity: Given the ordered arrival of observations, it can be argued that the latest observation x_t should be more similar to the "recent" observations $\{x_{t-1}, x_{t-2}, ..., x_{t-k}\}$ than the older ones $\{x_{t-k-1}, x_{t-k-2}, ...\}$. Thus, if the mean squared error has been low on a sliding window of recent observations, the RE is confident that the prediction error at time t will also be low. Formally, the RE is defined as follows:

$$R_{MSE} = \frac{\sum_{t \in B} (\hat{y}_t - y_t)^2}{|B|}$$

where B is the set of the $|B| = k$ most recent observations.

Local Sensitivity: The principle of this RE is to perturb the label associated with the latest observation and assess to which extent the prediction of the algorithm learned with the perturbed observation is modified. If there is little difference in the prediction, the RE is confident that the initial prediction is good. Formally: at time t, upon reception of an unlabeled observation x_t, the algorithm outputs an initial prediction $h_{t-1}(x_t) = \hat{y}_0$. Two artificial observations are then created by modifying the estimated label by a value $\delta_1 > 0$: $\{(x_t, \hat{y}_0 + \delta_1), (x_t, \hat{y}_0 - \delta_1)\}$. Two copies h^1_{t-1} and h^2_{t-1} of the algorithm h_{t-1} are then created. The first copy is trained with the first artificial observation whereas the second uses the second one. Then each copy computes a prediction $\hat{y}_{\delta_1} = h^1_{t-1}(x_t)$ and $\hat{y}_{-\delta_1} = h^2_{t-1}(x_t)$ for the unlabeled observation x_t initially received. This process is repeated k times with different

values of δ, resulting in a set of $2k$ predictions $A = \{\hat{y}_{\delta_1}, \hat{y}_{-\delta_1}, ..., \hat{y}_{\delta_k}, \hat{y}_{-\delta_k}\}$. The 2 REs derived from these predictions are:

$$R^1_{LSA} = \frac{\sum_{i=1}^{k} (\hat{y}_{\delta_i} - \hat{y}_{-\delta_i})}{k}$$

and

$$R^2_{LSA} = \frac{\sum_{j \in A} \hat{y}_j}{2k} - \hat{y}_0$$

Dual Perturb and Combine: Conversely to Local Sensitivity Analysis, the idea is to perturb the attribute values of the latest observation and assess to which extent the prediction of the algorithm changes. Formally: the algorithm outputs an initial prediction $h_{t-1}(x_t) = \hat{y}_0$. A set of k artificial unlabeled observations is then created: $\forall i = 1, ..., k \ : \ x^i_t = x_t + \delta_i$ with δ_i the vector of modifiers δ_{ij}, one for each attribute dimension j and with $\delta_{ij} \sim N\left(0, \sigma^2_j\right)$. A prediction $\hat{y}_i$ is then generated for each artificial observation by computing $h_{t-1}\left(x^i_t\right) = \hat{y}_i$. The 2 REs derived from these predictions are:

$$R^1_{DPC} = \frac{\sum_{i=1}^{k} (\hat{y}_i - \bar{y})^2}{k}$$

where $\bar{y}$ is the average of all the perturbed predictions $\hat{y}_i, \ i = 1, ..., k$ and the original prediction $\hat{y}_0$.

$$R^2_{DPC} = \frac{\sum_{i=1}^{k} (\hat{y}_i - \hat{y}_0)}{k}$$

On-line Bagging Sensitivity: Here, the idea is to compare the prediction of the base model trained with all the observations to the predictions of k multiple versions of the base model trained with different subsets of the observations seen so far.

$$R^1_{BAG} = \frac{\sum_{i=1}^{k} (\hat{y}_i - \bar{y})^2}{k}$$

where $\bar{y}$ is the average of all the predictions $\hat{y}_i, \ i = 1, ..., k$ given by the k models and $\hat{y}_0$ is the prediction obtained with the base model.

$$R^2_{BAG} = \frac{\sum_{i=1}^{k} (\hat{y}_i - \hat{y}_0)}{k}$$

Values of k and δ: As explained in section 5.2.4, we chose to use a fixed set of parameters, regardless of the base learner or the dataset. The chosen values are roughly in line with the values used in the paper of Rodrigues et al. Table 5.1 summarizes the choices made for each parameter.

Table 5.1: Parameters used for the reliability estimators

	k	δ
R_{MSE}	10	-
R_{LSA}	5	$\delta \sim N(0, 0.1)$
R_{DPC}	10	$\forall i, j:\ \delta_{i,j} \sim N(0, 0.1)$
R_{BAG}	10	-

5.4.2 Experimental protocol

We used the Java platform MOA [15] which provides an environment for running experiments in the framework of data streams subject to concept drifts. We used 4 regressors (described in section 5.3.1) which were already implemented in the platform (with the exception of IBLStream which was developed as an add-on[1]). The confidence estimators were directly implemented in MOA.

Similarly to the parameters of the reliability estimates, we chose to use the default values (set in MOA) of the parameters of each regressor, regardless of the dataset. The underlying idea remains the same: as we are not allowed to make any kind of assumption regarding the type of drift encountered, there would be little point in optimizing a set of parameters that would only be relevant at a given time, on a particular dataset and for a particular concept.

In the case of ϵ (the value associated with the tube cost function), it was previously stated that ϵ is a threshold set before the algorithm is ran and which depends on the problem at hand. Therefore, in order to simulate different requirements on the precision level, 2 thresholds were tested on the synthetic datasets:

- The first threshold was assumed to be "low" (i.e. a good prediction is hard to achieve as the $\epsilon-$tube is small).
- The second threshold was assumed to be "high" (i.e. it is easier to output a prediction which is within the $\epsilon-$tube considered).

The 2 thresholds for ϵ were determined with hindsight by computing the variance of the target variable on the whole dataset and using a different multiple of this number for each threshold.

5.4.3 Success Metrics

In order to assess the benefits from abstention, we have computed the percentage of improvement in the overall cost between each regressor and its abstaining version. Formally, for a given dataset with n observations, we started by computing the absolute difference between the 2 overall costs achieved:

$$\text{abs diff} = \sum_{i=1}^{n} \left[C_{\epsilon}^{0-d-1}\left(h_{base}\left(x_i\right), y_i\right) - C_{\epsilon}^{0\hat{-}d-1}\left[\left(h, g\right)\left(x_i\right), y_i\right] \right]$$

[1]The code for their add-on can be recovered from this link: https://www.uni-marburg.de/fb12/kebi/research/software/iblstreams

where h_{base} is the base version (without a selection function) of the algorithm that predicts all the time, (h, g) is the abstaining version of the algorithm described in section 5.2.3.1 and

$$\hat{C_\epsilon^{0-d-1}}\left[(h,g)\left(x_i\right), y_i\right] = \frac{1}{10}\sum_{j=1}^{10} C_\epsilon^{0-d-1}\left[(h,g)_j\left(x_i\right), y_i\right]$$

is used to denote the average cost achieved by the 10 copies of the abstaining regressor (this point is explained thereafter) on the particular observation (x_i, y_i).

Recall that both abstaining and base versions of the algorithm are updated in the same way and will thus result in the same hypothesis h learned, regardless of the output of the selection function g. Thus, the 10 copies of the abstaining regressor will have learned exactly the same hypothesis h but might output different predictions from each other. This is the case because there is an element of randomness associated with some of the reliability estimators which we chose to overcome by averaging the results of the 10 copies.

The percentage of improvement from the fully predicting version to the abstaining version was then computed as:

$$\text{improvement} = -\frac{\text{abs diff} \times 100}{\sum_{i=1}^{n} C_\epsilon^{0-d-1}\left(h_{base}\left(x_i\right), y_i\right)}$$

Thus, on each dataset, a negative value (e.g. -10.3) indicates that the algorithm that was allowed to abstain managed to achieve an overall cost which is lower (in this case 10.3% lower) than the base algorithm. Conversely, a positive number indicates that the base algorithm managed to over-perform the abstaining version.

Note: Because we are comparing the difference of performance of one base algorithm to his abstaining version, we are guaranteed that this difference can only be attributed to the decision to abstain (or not) and not by the underlying ability of a particular algorithm to learn on a given dataset.

5.5 Synthetic datasets

We start by presenting each synthetic dataset and explain why it was used. We then present and discuss the results achieved. Synthetic datasets are useful to experiment in an environment where the type of drift can be controlled.

5.5.1 Presentation

5.5.1.1 Drifts of controlled magnitude

Two batches of 10 datasets have been created to assess the effects of drifts with gradually increasing magnitudes and to "force" local drifts on the feature's joint density (commonly known as *covariate shift*). For each dataset, the dimension of the feature space was set to 2 and the number of observations generated to 1000. A unique drift was introduced at time t_{501}.

For the first batch, we generated 10 datasets for which $H\left(\mathcal{D}_{t_{500}}, \mathcal{D}_{t_{501}}\right) = \{0.12, 0.2, ..., 0.99\}$ respectively ($H\left(.,.\right)$ is the Hellinger distance [45] which is used to measure the drift magnitude in Section 2.3.2.1). This was achieved by randomly generating 10 000 pairs of multivariate normal distributions, computing the Hellinger distance for each pair and retaining the 10 pairs which had the closest value to the desired magnitudes. The first multivariate normal distribution was then used to generate the first 500 observations whereas the second one was used for the rest of the dataset.

For the second batch, we generated 10 datasets such as

$$H\left(f_{t_{500}}\left(X\right), f_{t_{501}}\left(X\right)\right) = \{0.1, 0.2, ..., 0.98\}$$

with $f_t\left(X\right)$ the joint density of the features at time t. This was done by generating a random multivariate normal distribution as the joint law of (X, Y) before the drift and deducing the laws[2] of X and (Y/X). We then randomly generated another multivariate normal distribution for the law of X after the drift and the Hellinger distance was computed between the laws of X before and after the drift. This process was also repeated 10 000 times and the 10 pairs which had the closest value to the desired magnitudes were kept. We then obtained the joint density[3] of (X, Y) after the drift by multiplying the original conditional density Y/X with the new joint density of X. The observations after the drift were then generated with a simple rejection sampling algorithm.

5.5.1.2 Drifts of controlled type, frequency and area of effect

In comparison to the 2 batches of datasets described previously, these 3 datasets (based on the Friedman's function [37]) are useful to assess the effect on the performances of different types of drifts (gradual, abrupt, local and global) and of different drift frequencies (several drifts appear on the same dataset).

In this case, there are 10 continuous attributes and their values are independently distributed with uniform distribution on $[0, 1]$. The first 5 attributes are used to compute the target value whereas the last 5 are useless. The basic target value is computed as follows:

$$y = 10sin\left(\pi x_1 x_2\right) + 20\left(x_3 - 0.5\right)^2 + 10x_4 + 5x_5 + \sigma$$

with $\sigma \sim N\left(0, 1\right)$ a random number. 3 datasets of 1000 observations were implemented, following the work of Ikonomovska [48]:

Local expending abrupt drift: In this dataset, 3 local drifts are introduced at times, t_{251}, t_{501} and t_{751}. From t_1 to t_{250}, the goal is to learn the initial Friedman's function. A local drift is then introduced at time t_{251} such as

$$\forall x \in R_1 = \{x_2 < 0.3 \wedge x_3 < 0.3 \wedge x_4 > 0.7 \wedge x_5 < 0.3\}$$

[2] In this case, X is also a multivariate normal distribution.

[3] Note that in this case, the joint density (X, Y) is not necessarily a multivariate normal distribution. Consequently, there is more diversity in the set of joint densities considered than in the first batch.

$$y_{R_1} = 10x_1x_2 + 20\,(x_3 - 0.5) + 10x_4 + 5x_5 + \sigma$$

If $x \notin R_1$, the target value is unchanged. At time t_{501}, a second local drift is introduced on R_2, such as

$$\forall x \in R_2 = \{x_2 > 0.7 \wedge x_3 > 0.7 \wedge x_4 < 0.3\}$$

$$y_{R_2} = 10cos\,(x_1x_2) + 20\,(x_3 - 0.5) + e^{x_4} + 5x_5^2 + \sigma$$

and R_1 is further expended by removing the last inequality from its definition ($x_5 < 0.3$). Finally, at time t_{751}, a third local drift is introduced by further expending R_1 and R_2. In both cases, the last inequalities from their modified definitions are removed ($x_4 > 0.7$ and $x_4 < 0.3$ respectively).

Global reoccurring abrupt drift: In this dataset, the drifts appear over the whole input space $\mathcal{X}$. There are 2 drifts at times, t_{501} and t_{751}. The new target function after the first drift is

$$y_{gl} = 10sin\,(\pi x_4x_5) + 20\,(x_2 - 0.5)^2 + 10x_1 + 5x_3 + \sigma$$

whereas it reverts to the initial target function after the second drift.

Global and slow gradual drift: Here, two gradual drifts are introduced at times t_{501} and t_{751}. In order to simulate a gradual drift, the observations are generated in parallel according to 2 different concepts and the sigmoid function is used for the probability of selecting one concept over the other. At time t_{501}, a new target function is introduced

$$y_{glr_1} = 10sin\,(\pi x_4x_5) + 20\,(x_2 - 0.5)^2 + 10x_1 + 5x_3 + \sigma$$

and the examples are slowly shifting from the initial target function to y_{glr_1} such as, at time t_{750}, the probability of selecting the new target function is 1. The same principle apply after the second drift where the target function

$$y_{glr_2} = 10sin\,(\pi x_2x_5) + 20\,(x_4 - 0.5)^2 + 10x_3 + 5x_1 + \sigma$$

gradually replaces y_{glr_1}.

5.5.1.3 Comparison between stable and drifting concept

Here the goal was to assess which performances could be achieved when the concept remains stable and to compare the difference in performance when a drift is introduced on this same dataset.

To this end, 2 datasets based on the regression version of the hyperplane generator (Shaker and Hullermeier [93]) have been created. This generator randomly creates a d-dimensional

Table 5.2: Values used for the ϵ-tube cost and the confidence thresholds on each datasets

	Low ϵ	High ϵ	Size	# Drifts
0.1-(X); ... ; 0.99-(X,Y)	0.1	0.75	1000	1
Hyperplane Regression	0.02	0.08	1000	1
Fried Local Expending Abrupt	1	3	1000	3
Fried Global Slow Gradual	1	3	1000	2
Fried Global Reoccurring Abrupt	1	3	1000	2
S&P 500	0.0005	N/A	6692	N/A
CAC 40	0.0004	N/A	6637	N/A
Apple	0.005	N/A	8927	N/A
EUR/USD	0.00005	N/A	2295	N/A
Gold	0.001	N/A	1565	N/A
Hyperplane Regression No Drifts	0.02	0.08	1000	0

hyperplane in a unit hyper-cube. The goal here is to predict the distance of each observation received to the hyperplane. In our experiment, both datasets have a feature space of dimension 8 and holds 1000 observations.

The first dataset has been generated according to a single stable concept whereas the second one is strictly identical (i.e. it has exactly the same observations), up to time t_{501} where a single abrupt drift is introduced. The drift was introduced by generating another random hyperplane in the hypercube.

5.5.2 Results

The improvements (as defined in Section 5.4.3) achieved on each dataset and by each learner are presented in Fig. 5.2. As previously stated, a negative value indicates that globally the performances of the learners were improved by abstaining whereas a positive value indicates that the performances of the base versions were better. The values used for the ϵ-tube cost function, the confidence thresholds, the size of each dataset and the number of drifts included in them are given in table 5.2.

5.5.2.1 Influence of the drifts type on the performances

Despite the large variety of drifts (global, local, abrupt, gradual, different magnitudes ...) reproduced, the results of the experiments globally indicate that the proposed method is able to significantly improve the performances of the underlying algorithm (up to -43% on the hyperplane dataset) regardless of the type of the drift. This further indicates that abstaining should be considered when dealing with data streams subject to concept drifts.

Mixed results were achieved when there was no drift at all (on the hyperplane dataset with no drift), with 3 learners out of 4 for which the performance was significantly improved in the case of a small ϵ and only 2 learners out of 4 had better results when ϵ was high.

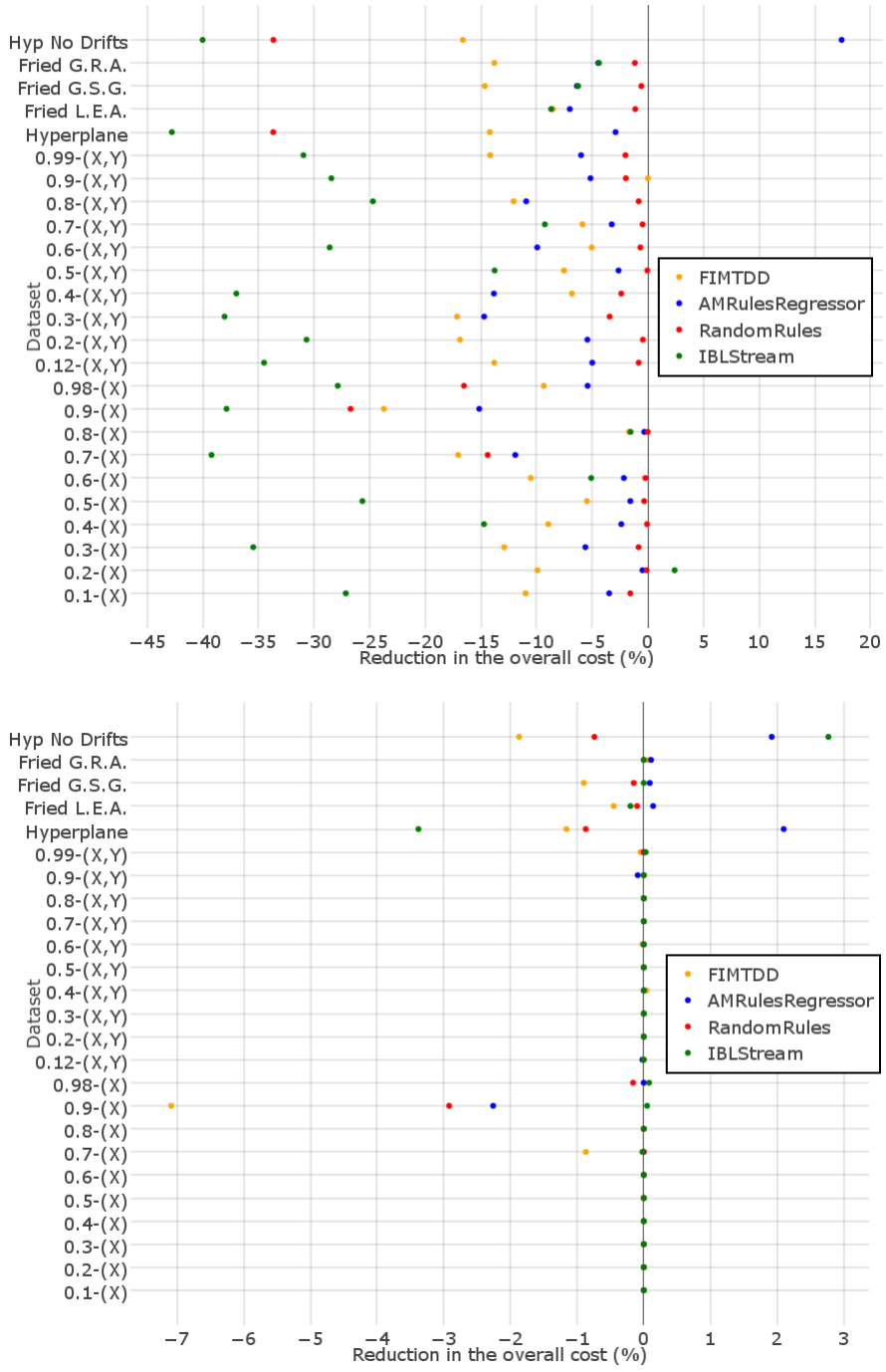

Figure 5.2: Percentage of reduction in the overall cost gained by allowing the algorithm to abstain. The upper plot is for a low ϵ whereas the lower plot is for a high ϵ.

5.5.2.2 Analysis of the results against the learner used

The improvement in the overall performance can change widely from one learner to the other (when the dataset and ϵ are fixed), especially for a small ϵ. For instance, on the hyperplane dataset with one drift, for a small ϵ, the abstaining version of IBLStream managed to improve the performances by 43% whereas the performances were only improved by 3% for AMRulesRegressor. Overall, despite their drift handling capabilities, the proposed method managed to improve the performances of the 4 algorithms by allowing them to abstain. The algorithm which in general, benefited the most from abstention was IBLStream whereas the one that benefited the least was RandomRules.

5.5.2.3 Analysis of the results against the value of ϵ used

Intuitively, when ϵ is small, it is harder for the learner to predict within the $\epsilon-$tube and thus the number of wrong predictions increases. The results suggest that in this case, the REs globally managed to filter some of the predictions that would not have met the precision constrain as the abstaining version over-performed the fully predicting version on most of the datasets (an overall increase in the cost would have suggested that the predictions filtered by the REs met the precision constrain). On the other hand, when ϵ is large, the confidence threshold increases and most of the predictions are not filtered by the REs anymore. This leads to overall performances which are globally equal to the performances obtained when predicting all the time.

In order to further assess the effect that the required precision threshold ϵ has on the performance of the proposed method, we have conducted an in-depth analysis on the Hyperplane Regression dataset with one drift using different thresholds
($\epsilon = \{0.02, 0.04, 0.06, 0.08, 0.1, 0.15, 0.2, 0.3\}$). For each ϵ, the improvement (defined in section 5.4.3) was computed. We also added the best (respectively worst) improvement for each learner, which was computed using the abstaining copy of the learner which achieved the smallest (respectively largest) overall cost with hindsight. In other words, when computing the value of *abs diff* (defined in 5.4.3), $C_\epsilon^{\hat{0-d-1}}\left[(h,g)\left(x_i\right), y_i\right]$ was replaced by $C_\epsilon^{0-d-1}\left[(h,g)_{\mathbf{j}}\left(x_i\right), y_i\right]$ where the j^{th} copy verifies: $\forall k \in \{1, ..., 10\}$:

$$\sum_{i=1}^{n} C_\epsilon^{0-d-1}\left[(h,g)_{\mathbf{j}}\left(x_i\right), y_i\right] \leq \sum_{i=1}^{n} C_\epsilon^{0-d-1}\left[(h,g)_{\mathbf{k}}\left(x_i\right), y_i\right]$$

(respectively $\geq$).

The shape of the curves obtained in Fig. 5.3, further proves that the interest for abstaining is correlated to the value of ϵ. The results also show that for half of the tested algorithms, abstaining never led to worse performances (regardless of the value of ϵ) whereas for the other half, there is only a limited range of ϵ values for which the averaged overall cost increased (the worst case is 3.78% for $\epsilon = 0.1$ with IBLStream). Finally, apart from AMRulesRegressor for which the variability between the abstaining copies was the largest (for $\epsilon = 0.02$, the best abstaining copy reduced the overall cost by 9.9% whereas the worst copy increased the

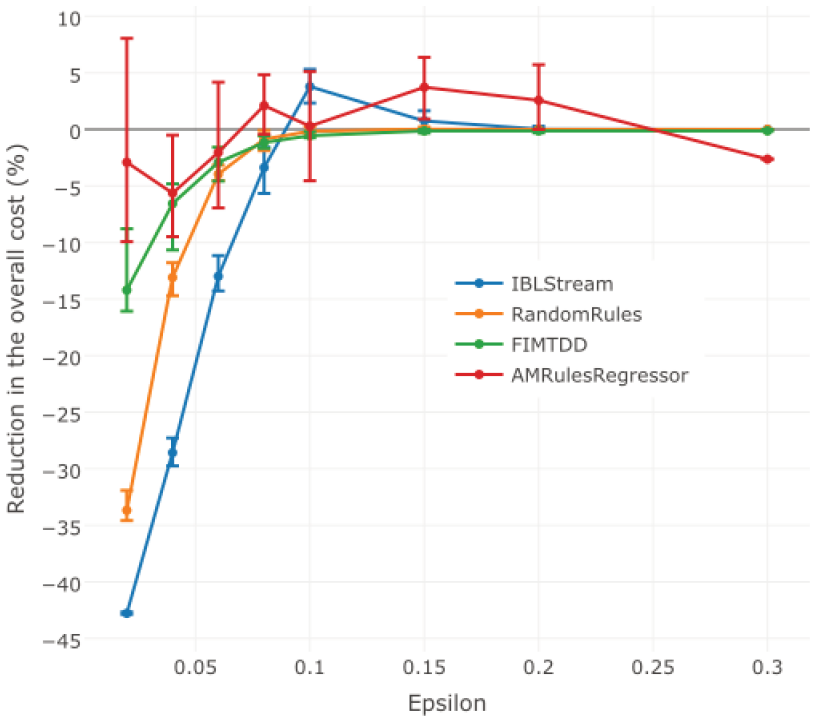

Figure 5.3: Comparison of the average percentage of reduction in the overall cost as a function of the value of ϵ. Error bars indicate the performance achieved with the best (respectively worst) copy of the abstaining algorithm.

overall cost by 8%), the variability observed within the abstaining versions of the other learners remained globally limited.

5.5.2.4 Evolution of the improvement over time

We also studied the evolution of the difference in performances over time between the base and the abstaining version of an algorithm. We show here the evolution of the results in the case of the RandomRules algorithm on the 0.9-(X) and Fried Global Reoccurring Abrupt Drift datasets (respectively upper and lower plot of Fig. 5.4) with a small ϵ.

To obtain these plots, we started by computing the summed costs of each version of the algorithm on a rolling (but non overlapping) window of 10 observations. In the case of the abstaining version, we further averaged the sum. Formally:

$$s_k^{base} = \sum_{i=(k-1)\times 10+1}^{k\times 10} C_\epsilon^{0-d-1}\left(h_{base}\left(x_i\right), y_i\right)$$

and

$$\hat{s_k^{abs}} = \frac{1}{10}\sum_{j=1}^{10} s_{k,j}^{abs}$$

with $k = \{1, 2, ..., 100\}$, $s_{k,j}^{abs}$ the summed cost of the j^{th} copy of the abstaining algorithm on the observations $\{(k-1)\times 10+1, ..., k\times 10\}$. The values shown on each plot are $s_k^{base} - \hat{s_k^{abs}}$. Thus, a *positive* number indicates that the cost of the abstaining algorithm is lower than the base version whereas a *negative* number indicates that the base over-performed the abstaining version.

The plots show that there are periods of time where abstaining clearly improves the performance and periods of time where it makes no difference.

Each RE has its own strengths and weaknesses and is designed to estimate a particular aspect of what makes a prediction reliable. For instance, the similarity-based reliability estimate will efficiently discard observations leading to a large prediction error when the recent observations also had a large prediction error whereas the local sensitivity reliability estimate will use the estimated "flatness" of the values taken by target variable on a small area to decide whether to abstain or not. Thus, periods of over-performance of the abstaining algorithm are difficult to explain because they are the result of a combination of factors that led the ensemble of REs to accurately filter the predictions that would have led to a prediction error larger than the ϵ-tube.

These factors can appear under a stationary concept (for instance, between t_{250} and t_{400} on the 0.9-X dataset[4]) and will not necessarily appear because the concept has drifted (for instance, on the Fried G.R.A. dataset, the gradual drift introduced at t_{501} left the performances

[4]Remember that this dataset has an unique and abrupt drift at time t_{500} and that the concept is stable before and after.

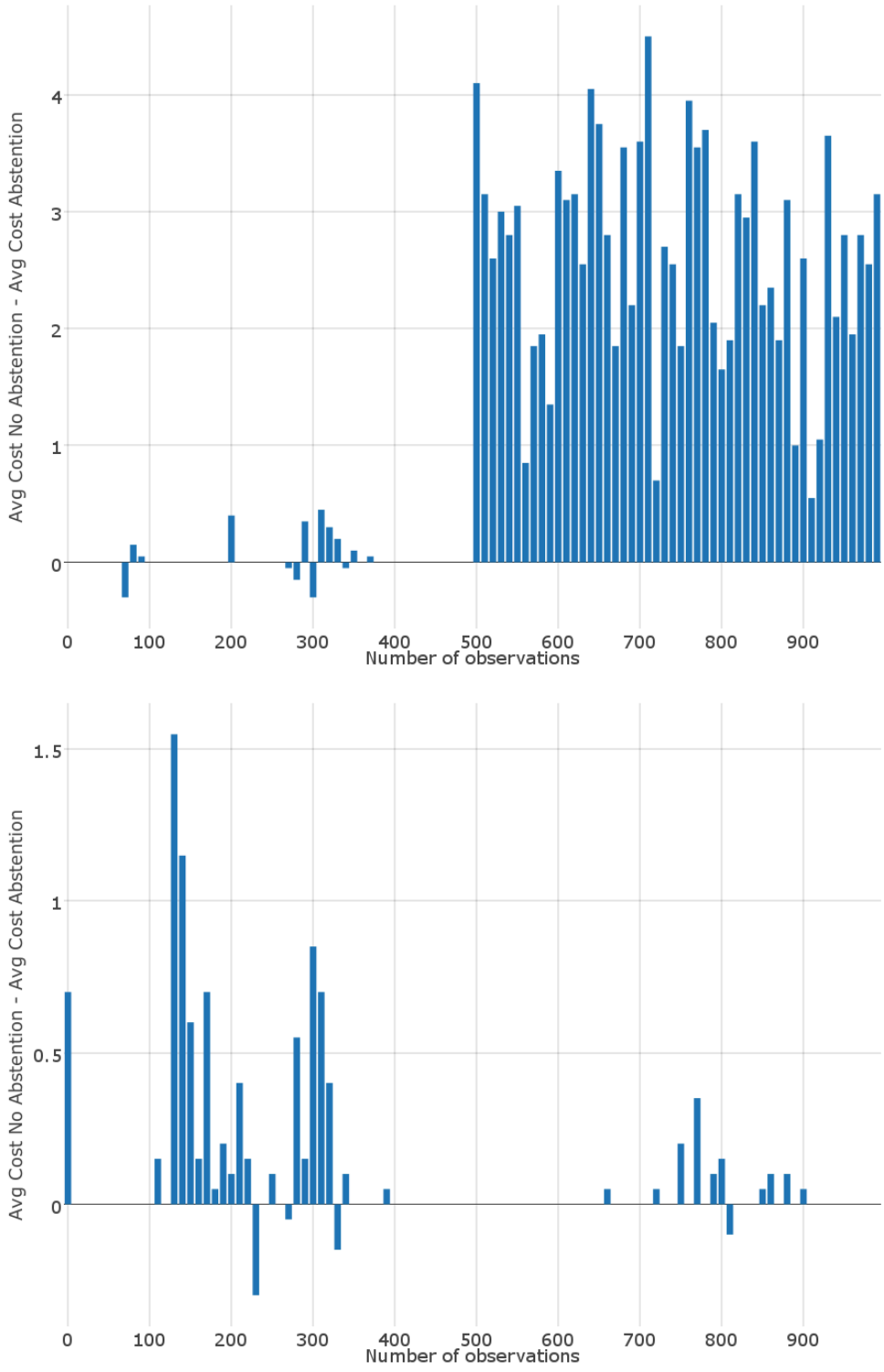

Figure 5.4: Evolution of the improvement in performance over time of the RandomRules algorithm on the 0.9-(X) and Fried G.R.A. datasets (upper and lower plot respectively) computed on a rolling (and non-overlapping) window of 10 observations. Positive values indicate that the abstaining version over-performed the base version of the algorithm.

of the base and abstaining version exactly similar up to t_{650}). However, the plots back our claim that when the concept drifts, allowing the algorithm to abstain can improve the performances and therefore that it should be considered as a performance enhancing technique.

5.6 Real life datasets with concept drifts

Following up on our introductory real life example presented in section 5.2.2, we ran a batch of experiments on several financial datasets. These datasets were chosen because they provide real life examples of streams subject to concept drifts.

5.6.1 Presentation

Each dataset is based on a particular financial asset (a stock, an index of stocks, a precious metal an exchange rate between 2 currencies) and has 7 attributes. The first 5 attributes are based on the observation of the opening price, highest price, lowest price, closing price and volume of transaction for that asset on a given day. For the last 2 attributes, we have added the average as well as the variance computed with the closing prices of the last 10 days.

Our framework assumes that the observations are independent realizations of a single hidden concept (when the concept is stable) or independent realizations of a set of concepts (when the concept drift). In both cases, the observations are assumed to be independent from each other. Unfortunately, this assumption clearly doesn't hold in the case of time series where the value of an observation at time t depends on its value at time $t-1$. Therefore, we chose to transform the time series of the 7 attributes into series of returns which can be assumed to be independent from each other.

The transformation was done as follows: for a given time series $\{p_{t_1}, ..., p_{t_n}\}$, we have computed the return:

$$r_t = \frac{p_t - p_{t-1}}{p_{t-1}}$$

for each time $t \in \{t_2, ..., t_n\}$, where p_t is the value of the time series at time t. Thus, at time t, the learner receives an observation

$$x_t = \left\{ r_t^{Open}, r_t^{High}, r_t^{Low}, r_t^{Close}, r_t^{Volume}, r_t^{Average}, r_t^{Variance} \right\}$$

and must predict the target variable $y_t = r_{t+1}^{Close}$.

5.6.2 Results

We give the results achieved on the real life datasets with a small ϵ (see Figure 5.5) which is in line with the goal of the investor (the narrower the tube, the larger the expected payout of the option). All the results indicate that allowing abstention led to an improvement of the performances and tend to confirm what was observed on synthetic datasets. This good performance is explained by the increased difficulty to accurately predict on extremely noisy datasets subject to a wide range of drifts.

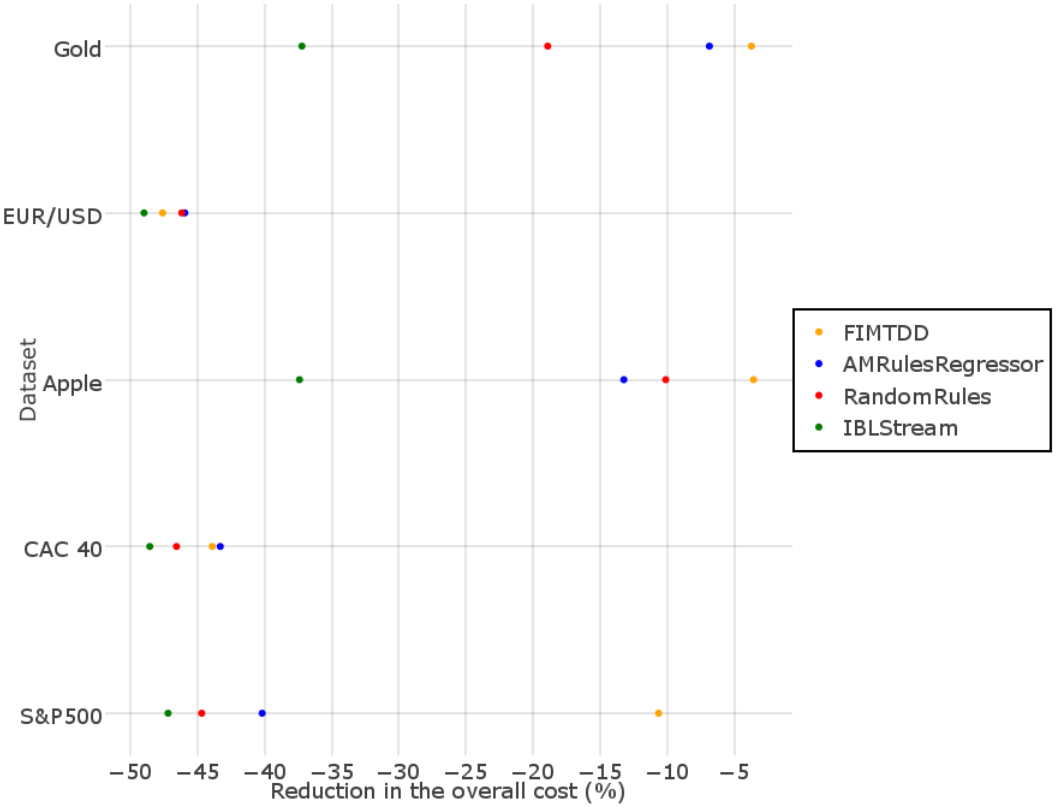

Figure 5.5: Percentage of reduction in the overall cost gained by allowing the algorithm to abstain. The plot was obtained with a low ϵ

In order to concretely describe what these results mean for our investor, we calculate the amount of money that he would have saved by allowing his machine learning algorithm (IBLStream) to abstain on the stock of Apple. To this end, we chose to use a cost function which attributes fixed values to the price and the payout of the created binary tunnel option as well as the cost of abstaining. In real life, the true price of such option would be calculated with complicated formulas which depends on many factors (such as the volatility and price of the underlying asset, the time until expiration, the selected boundaries, ...) and which we omit for the sake of simplicity.

Therefore and without loss of generality, assume that the investor has an initial capital of 10 000€, that the price of the option is always equal to 1€ (the amount of money lost if the prediction is wrong), that the payout (i.e. the amount of money received if the prediction is correct) is always 1€ and that if the investor chooses to abstain, he will leave the money at the bank which will charge him a fixed 0.1€ overnight. Thus, the cost function is then given by:

$$C_\epsilon(\hat{y}, y) = \begin{cases} 1 & if\ |y - \hat{y}| \leq \epsilon\ and\ g(a) = 1 \\ -1 & if\ |y - \hat{y}| > \epsilon\ and\ g(a) = 1 \\ -0.1 & if\ g(a) = 0 \end{cases}$$

In this case the version of the algorithm which predicts all the time managed to output 1483 good predictions and 7437 wrong predictions. This results in a final capital of 4046€. On the other hand, the abstaining version of the algorithm gave 533 wrong predictions, 162 good predictions and abstained on 8225 observations resulting in a final capital of 8806.5€. Thus, by allowing its algorithm to abstain, the investor managed to save almost 50% of its initial capital (note that the performance is better than the -37.4% achieved in the experiment because the cost of abstention in this illustration is smaller).

5.7 Conclusion

Learning on a data stream subject to concept drifts is a challenging task. Drifting concepts can significantly diminish the performance of a learner over time and undermine the confidence in the outputted predictions. In this chapter, we claim that when costs can be associated with good and bad predictions, allowing a predictor to abstain must be considered in order to reduce the overall prediction cost (and thus improve performances).

In order to show this, we considered the specific problem of regression whit a constrain on the expected precision level associated with each prediction. We proposed a generic method which can be used with any regressor and which filters the predictions that would not have met the precision constraint.

We experimented this strategy on 30 datasets including different types of drifts, with 4 state of the art algorithms and with 2 levels of expected precision. We assessed the performance of our method by comparing the overall prediction cost of the base version of an algorithm (which predicts all the time) to the performance of the same algorithm equipped with the proposed method (which allows it to abstain when the confidence is not high enough). Globally, the

results indicate that when the need for precision is high, allowing the algorithm to abstain significantly improves the overall prediction cost whereas when the need for precision is low, the overall prediction cost is the same as the one achieved by predicting all the time.

Furthermore, the evolution of the difference in performance over time between the base version and the abstaining version of each regressor showed that, concept drifts can be the cause of an over-performance of the abstaining version and therefore that abstaining must be considered as an enhancing method to reduce the overall prediction cost. Indeed, when the required precision level cannot be achieved, allowing the algorithm to abstain based on an ensemble of reliability estimators acts as an automatic way to "disconnect" the algorithm during some of these adverse periods.

Chapter 6

Combining instance based structure and abstention

6.1 Introduction

In this chapter, we build up on the findings of chapters 4 and 5 by combining the instance based structure with the setting of prediction with a reject option. Our goal is to show that by merging these 2 properties, it is possible to solve the following problems, listed in the introduction chapter:

1. Adaptivity to a wide range of concepts changes.
2. Endless improvement of the learned model when the concept remains stable.
3. Achieving superior prediction performances compared to other state of the art algorithms.

The issues of constrained computer memory and computational time are left aside for the time being and will be addressed in chapter 7.

In order to test this idea, we propose the Droplets algorithm, a novel on-line algorithm which takes advantage of the meta-informations brought by the instance based structure to estimate whether abstaining from prediction should be considered or not for a particular unlabeled observation. When the informations available are judged insufficient to obtain a reliable prediction, the algorithm automatically abstain from prediction (without the need to calibrate a confidence threshold). This happens in two cases:

- Either if the observation has been received in a previously unexplored part of the feature space.
- Either if the observation has been received in a part of the feature space where the labels of neighboring past observations are in conflict with each other.

Adaptation to concept changes is achieved by using the principle of "conflicting observations", which was introduced in chapter 4.

We empirically show that this algorithm is able to outperform 10 of the best state of the art algorithms on datasets reproducing different types of drifts and that it can accurately spot the observations which are hard to classify.

6.2 Framework

In this chapter, the considered framework (supervised classification on a stream of data subject to concept drifts), goal (predict as accurately as possible the label y_i associated with x_i) and performance metric (0-1 loss function) are identical to chapter 4 and thus won't be presented in details again.

6.3 The Droplets Algorithm

In this section we start by presenting the Droplets algorithm and then discuss its main strengths and weaknesses.

6.3.1 Presentation

The broad principles underpinning the Droplets algorithm are similar to the ones of DEA presented in chapter 4. The difference is that DEA is a meta-learning algorithm whereas the Droplets algorithm is a base-learning algorithm.

Every new observation can be thought as a Droplet falling on a *d-dimensional* hyperplane (where d is the dimension of the input space X), thus the hyperplane corresponds to the feature space. We will refer to this hyperplane as the "map" for the rest of this chapter. The "chemical" composition of a droplet is defined by its class and the droplets which belong to different classes are mutually repulsive, so they can't cover the same parts of the map.

The first step is to set aside the first N observations received (where N is a user defined parameter). These observations will be used as an initializing set. Its goal is twofold:

1. Initialize the map with the first Droplets.

2. Estimate the range of values taken by each feature.

Once the minimum and maximum values of each feature on the initializing set have been determined, a *d-dimensional* normalization vector $\overrightarrow{D}$ is computed as follows: For the k^{th} feature, the k^{th} value of $\overrightarrow{D}$ will be equal to: $max\left(x_1^k, ..., x_N^k\right) - min\left(x_1^k, ..., x_N^k\right)$.

The map is then updated as follows: At first it is empty. The first labeled observation (x_1, y_1) from the initializing set is received and the observed values of its features are normalized (according to the normalization vector $\overrightarrow{D}$ found beforehand: x_1 becomes x_1^{norm}). The normalized observation is then inputed in the map as an hypersphere with a radius of size $R_{default}$ (where $R_{default}$ is a user defined parameter) and which is centered at the coordinates

equal to the normalized feature's values: x_1^{norm}. The class y_1 of this observation is then attributed to this hyper-sphere. The following observations are then released in turn and inputed in the map in the same fashion.

If at some point the hypersphere of latest observation (x_i, y_i) overlaps one or more existing hyper-spheres, the classes associated with the overlapped hyper-spheres are assessed. For all the overlapped hyper-spheres belonging to y_i, the class of the latest observation, nothing happens. However, for each overlapped hyper-spheres belonging to a class which is different from y_i, the overlap:

$$\lambda_k = R_k + R_{default} - \|x_k^{norm} - x_i^{norm}\|$$

between the k^{th} droplet and the latest one is computed (where $\|.\|$ denotes the Euclidean distance) and set aside. The identifiers of all of these Droplets are also saved into a list called *indexes*. This list holds all the Droplets which should have their radius decreased.

For the first hyper-sphere belonging to the previously created set *indexes*, the value:

$$\Delta_1 = \frac{R_1 + R_{default} - \|x_1^{norm} - x_i^{norm}\|}{2} + \varepsilon = \frac{\lambda_1}{2} + \varepsilon$$

used to remove the overlap between this hyper-sphere and the newest hyper-sphere is computed. This value includes an arbitrary small $\varepsilon > 0$, added to prevent the hyper-spheres from being tangent. Δ_1 is then subtracted to the radius size of this droplet (R_1 becomes $R_1 - \Delta_1$). The value Δ_2 is then computed for the second hyper-sphere belonging to *indexes* and so on. In order to make sure that the latest observation doesn't overlap with any of the Droplets belonging to the *indexes* set, its radius R_i is set to the most restrictive value:

$$R_i = max\left(R_{default} - \Delta_{max}; 0\right)$$

where Δ_{max} is the largest value obtained at the previous step. The new Droplet is then inputed to the map and is centered in x_i^{norm}. Finally, the radius of the updated Droplets is checked and if an hyper-sphere ends up with a radius equal to 0 or less, it is removed from the map.

Algorithm 6.1 details the updating process of the radii sizes.

This updating process is then repeated each time the label associated with the latest observation is in conflict with one or more overlapped droplets.

Once the initializing set is over, the first unlabeled observation is received. In order to output a prediction, the algorithm will simply check if the normalized feature's values of the latest observation belong to an existing hypersphere. If this is the case, the algorithm will predict that they both have the same class, otherwise, the algorithm will abstain from predicting and wait for the true label to be released. Note that the update of the map doesn't start until the true label is released. Algorithm 6.2 gives the details of the prediction process.

Once the true label is released, the update process described previously restarts. The full algorithm[1] is summarized in Algorithm 6.3.

[1] A video of the algorithm running on the SEA dataset is available here: https://www.youtube.com/watch?v=M9rBYLD7SkY

Algorithm 6.1 Update of the radii sizes

Inputs:
$indexes$: the identifiers of the droplets that should be updated in the map
map_{i-1}: holds all the Droplets saved into memory at the previous time step
ε : an arbitrarily small positive number
$R_{default}$: parameter setting the default value of the radius

Foreach:$k \in indexes$
  $\Delta_k \leftarrow \frac{R_k + R_{default} - \|x_k^{norm} - x_i^{norm}\|}{2} + \varepsilon$
  $R_k \leftarrow R_k - \Delta_k$
End Foreach
$\Delta_{max} \leftarrow max\,(\Delta_1, ..., \Delta_k, ...)$
$R_i \leftarrow max(R_{default} - \Delta_{max}; 0)$
$map_{i-1} \leftarrow Add\ Droplet\ to\ map\,(x_i^{norm}, y_i, R_i)$
$map_i \leftarrow Remove\ unnecessary\ Droplets\ from\ map\,(map_{i-1}, Criteria : R \leq 0)$
Return map_i

Algorithm 6.2 Prediction

Inputs:
map_{i-1}: holds all the Droplets saved into memory.
x_i^{norm}: the normalized features value of the latest observation

$\hat{y_i} \leftarrow \emptyset$ //The prediction is initialized as empty
Foreach:$k \in map_{i-1}$
  If $\|x_k^{norm} - x_i^{norm}\| \leq R_k$ **Then**
    $\hat{y_i} \leftarrow y_k$
    Return $\hat{y_i}$
  End If
End Foreach
Return $\hat{y_i}$

Algorithm 6.3 The Droplets algorithm

Inputs:
$R_{default}$: parameter setting the default value of the radius
N: size of the initializing set
ε: an arbitrarily small positive number

$\overrightarrow{D} \leftarrow Compute\ normalization\ constants\,(x_1, ..., x_N)$
$map_0 \leftarrow \emptyset$ //Initialize the map to empty
Foreach: x_i with $i \in \{1, 2, ...\}$
$x_i^{norm} \leftarrow Normalize\ observation\left(x_i, \overrightarrow{D}\right)$
// **Prediction step**
If $i > N$
$\hat{y}_i \leftarrow Predict\,(map_{i-1}, x_i^{norm})$ // Algorithm 6.2
End If
// **Map update**
$y_i \leftarrow True\ Label\ of\ x_i$
$indexes \leftarrow Get\ Overlapped\ Droplets\ with\ conflicting\ labels\,(map_{i-1}, x_i^{norm}, y_i)$
$map_i \leftarrow Update\ radii\,(map_{i-1}, indexes, R_{default}, \varepsilon)$ // Algorithm 6.1
End Foreach

6.3.2 Main strengths

In this section, we discuss the main strengths of the Droplets algorithm. In particular, we articulate the discussion according to the different algorithm structures that were presented in chapter 3.

6.3.2.1 Adaptation to concept drifts without explicit detection

As explained in chapter 3, some of the state of the art drift handling algorithms rely on a drift detector in order to trigger the update of their model. Although this adaptation strategy can potentially achieve performances which are superior to the blind adaptation method, it is extremely challenging to set up without prior information on how the data stream is expected to evolve.

For a start, the drift detector has to monitor the right metric: it could track the evolution of any performance metric of the algorithm like its accuracy, precision or recall over time. It could also track the evolution of any statistic associated with the observed data themselves like their mean value, variance, or their empirical distribution. The issue is that, a drifting concept might trigger one of these metric without triggering the others which make the choice of which one to monitor very difficult without prior information.

Even in the case where the detector accurately detects a drift, for instance, following an explosion in the error rate, the obtained information might not be enough to deliver an appropriate response. Indeed, the explosion of the error rate in itself might not be sufficient

to decide whether the whole model should be discarded or if it is simply necessary to get rid of some parts of the model (where the latest observations might have been received for instance).

Furthermore, assuming a proper metric has been found, a value will have to be associated with the trigger which will result into an additional parameter which can be hard to calibrate. For instance, if the drift detector relies on two windows (a short term one and a long term one) and monitor the estimated distributions of the observations on each of these window to trigger an alarm when these distributions are too different, the chosen parameter might be either too high and risk missing gradual drifts or too low and risk being triggered by noisy observations.

Therefore, in the general case where we no assumption about the nature of the future drifts can be made, an algorithm able to adapt to any type of drifts without explicitly detecting them seems better suited than an algorithm designed to handled some particular drifts. This also has the advantage of resulting in a simpler algorithm as a drift detection module isn't required.

6.3.2.2 Automatic abstention on tricky observations

By limiting the influence of each Droplets, the algorithm is prevented from making clueless assumptions on parts of the feature space that wouldn't have been explored previously. Instead, it will only predict when more information will be made available in this region.

In the second case, the reception of Droplets with conflicting labels on a particular part of the feature space will lead to their radii being diminished which in turn will create blank spaces on the map. This reflects a zone of uncertainty where the algorithm will also abstain until more information is received.

As a consequence, the algorithm will be able to distinguish which observations are potentially hard to classify and can abstain from predicting in those cases and will automatically focus on the other observations, further increasing the confidence that the end user can put in the predictions. This feature has been obtained because we made use of the meta-informations brought by the instance based structure, which we claimed was well suited to handle data streams subject to concept drifts.

Although to our knowledge, none of the existing drift handling algorithm has been explicitly designed to abstain from predicting after reception of an unlabeled observation. Some algorithms (like Boosting [key-19]) associate a degree of confidence with their predictions, leaving to the end user the task of setting a meaningful threshold above which he/she is willing to trust the algorithm. However, as we have seen in the previous section, setting a relevant parameter might not always be easy and the use of a fixed value can be questioned when dealing with non-stationary environments which often require flexibility in the value of the parameters. Furthermore, most of the algorithms producing confidence estimates of their prediction rely on the assumption that the distribution linking the observations to their class is stationary. This assumption doesn't hold when concept drift occur and hence the validity of the confidence estimate produce can be questioned. Therefore, an algorithm which can abstain without having to explicitly specify a confidence threshold could prove to be very useful in an evolving environment.

6.3.2.3 Forgetting observations which have been proved wrong

As discussed in chapter 4, the instance based structure also allows to use a criteria of "conflicting information" to decide which observations should be deleted from memory. This means that the adaptation process is based on hard evidences of past observations having been proved wrong instead of just assuming that they should be deleted because they are old (which is what most of the algorithm presented in chapter 3 do).

A consequence of this forgetting mechanism is that the algorithm isn't forced to keep temporally contiguous observations into memory. Indeed, it isn't clear why an algorithm should be forced to keep contiguous data, especially in the cases of noisy observations or reoccurring concepts. In the particular case of non-overlapping and reoccurring concepts, this means that the observations associated with past concepts won't be forgotten and that the learned model will be available as soon as the concept reoccurs.

6.3.2.4 Flexible structure allowing to learn a wide range of concepts

Because of the very limited assumptions they make about the nature of the concept they are trying to learn, instance based methods are well suited to learn a very wide range of concepts. This is very useful because in the general setting the characteristics of the future concepts can't be predicted and consequently, a great flexibility is required.

6.3.2.5 Implicit handling of appearing/disappearing classes

Finally, although it wasn't one of the initial requirement of this thesis, the Droplets algorithm doesn't require any previous information about the classes that will be encountered and thus can naturally adapt to appearing and disappearing classes.

6.3.3 Identified weaknesses

We now review the identified weaknesses of the Droplets algorithm.

6.3.3.1 Curse of dimensionality

Like the majority of instance based methods, the algorithm could suffer in high dimensions due to the increased volume of the map that needs to be filled by hyper-spheres. Because of its cautious handling of observations received in previously unexplored parts of the feature space, high dimensions won't necessarily result in a decrease of the prediction performance of the algorithm but will significantly reduce its coverage.

In this case, one solution could be to feed the algorithm with more data. Another solution has been proposed by T. Hastie and R. Tibshirani [44] in the case of the KNN which could be adapted to our algorithm. A third solution could be to use algorithms which would shrink the dimension of the feature space.

6.3.3.2 Initial normalization step doesn't scale well

The normalization step that takes place during initialization can be troublesome in some cases. Indeed, using the observed maximum and minimum values of each features on the initializing set to determine the normalization vector can make the algorithm sensitive to outliers. A simple way to deal with that issue would be to compute the observed mean and standard deviation for each feature and use these values for normalization. This method is described in details in section 4.2.1.

Furthermore, setting the normalization vector to a constant value can also be problematic in cases where the range of values taken by each feature changes from one scale to another. For instance, if the observed values of the first feature on the initialization set are uniformly distributed on the $[-10; 10]$ set, the normalization constant found (and thus the default radius size) with this set might not remain relevant if the range of values later becomes very large (i.e. uniform distribution over $\left[-10^{10}; 10^{10}\right]$), which will lead to a radius size which is too small and in turns to many blank spaces. Conversely, if the range of observed value becomes very small (i.e. uniform distribution over $\left[-10^{-10}; 10^{-10}\right]$), it will lead to an over-sized radius size where the Droplets will constantly overlap each other and where the algorithm will not be able to learn accurately the concept. We haven't yet studied this type of drift which has been left for futures researches.

6.3.3.3 Setting a proper parameter value is difficult

The Droplets algorithm has only one parameter. However, finding a suitable value for the default radius size can also prove difficult. As we have advocated in this thesis, in the framework of data streams subject to concept drifts, the value giving the best results for a given parameter might change over time. Therefore, the idea of setting an unique value for the default radius size could be challenged.

We haven't worked on this particular issue. The answer probably lies in adaptive parameters.

6.3.3.4 Trade-off between slow computational time and insufficient information

Another weakness coming from the instance based structure is that, by keeping (some of the) past observations into memory, the prediction step is likely to become increasingly slow as the number of observations kept into memory increases. This is because the algorithm requires to go through the observations saved into memory in order to build a predicting model on the fly. In cases where the data are streamed at a very high frequency, this could prove to be an issue.

One way to deal with this issue would be to constrain the number of observations kept into memory which will have the effect of setting an upper bound on the computational time. However, constraining the number of observations into memory can also decrease the quality of the model learned, especially in high dimensions where a lot of observations are required in order to accurately learn the concept. This problem is specifically addressed in chapter 7.

6.4 Experimental Framework

In this section, we describe the datasets on which experiments have been conducted, the drifts they include and the experimental protocol.

6.4.1 Datasets and experimental protocol

We decided to include both semi-artificial and artificial datasets and tried to cover different types of drifts (abrupt, incremental and reoccurring).

Evaluation procedure: In order to assess the predictive performances of the classifiers at every single time step, we used the experimental protocol presented in [19]. This is a slightly modified version of the classical interleaved test-then-train evaluation procedure: For each dataset, a set of 1 000 temporally indexed observations is generated $(x_i, y_i), i \in \{1, ..., 1000\}$. Then, for each time step i, 500 observations have been randomly generated according to the concept in force at that time: $\left\{\left(x_i^1, y_i^1\right), ..., \left(x_i^{500}, y_i^{500}\right)\right\}$. The classifiers are then trained on $(x_i, y_i), i \in \{1, ..., 1000\}$ but their accuracy at each time step i is assessed on $\left\{\left(x_i^1, y_i^1\right), ..., \left(x_i^{500}, y_i^{500}\right)\right\}$. The initializing set is composed of the first 100 observations $(x_i, y_i), i \in \{1, ..., 100\}$ and the test set composed of the remaining observations $(x_i, y_i), i \in \{101, ..., 1000\}$.

This evaluation procedure means that the performance of each algorithm is evaluated on $900 \times 500 = 450000$ observations per datasets. The advantage is that it gives a much more reliable idea of the accuracy of the classifiers at a given time than what would have been obtained with only one observation. The drawback is that it forces the use of datasets where the user can generate the labeled observations for each time step.

In order to have full control over the generating process of the observations, we used our own implementation of all the datasets described bellow.

Random RBF: This dataset was initially introduced in [13]. The idea is to generate a fixed number of centroids in d-dimensions where each center will be characterized by random coordinates, a random standard deviation, a particular class label and a random prior probability of being selected. The new observations are then randomly generated according to the previous parameters. Drifts are introduced by offsetting the coordinates of the centers in a random direction according to a Gaussian distribution with zero mean and a given standard deviation.

We used 3 Random RBF datasets, two for incremental drifts: RBF_2 with 6 classes, RBF_3 with 30 classes and 12 dimensions and one for abrupt drifts: RBF_1 with 3 classes. In the case of incremental drifts, the centroids are offset at each time step. In the case of abrupt drifts, there are 4 different concepts evenly distributed on the dataset (the drifts happen at $t = \{250, 500, 750\}$). In each cases the data were generated without noise and constrained in a $[0, 10]^2$ square (or hyper-cube for RBF_3).

Rotating Hyperplane: This synthetic dataset was initially introduced in order to assess the performance of CVFDT against VFDR in [47]. The idea is to generate the data uniformly in a d-dimensional hyperplane according to the following decision boundary

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$$

with x_i the i^{th} feature and w_i the weight of this feature. If the left hand part of equation (8) is superior or equal to w_0 the label of the observation will be positive and negative otherwise. Concept drifts are introduced by changing the weights. d was set to 2 and some noise has been added to the dataset by randomly switching the class of each observation with a 15% probability. The abrupt drifts happen at times $t = \{166, 332, 498, 664, 830\}$.

Weather Temperatures: This dataset is composed of one feature and 3 classes, created out of the historical temperatures of the city of Paris[2] from the 17th of January 2006 to the 15th of December 2014. 3 temperatures are available: the highest, lowest and average temperature of each day. After removing the outliers from the data, the average of the highest and lowest temperatures for each day of the year have been computed (the average of the highs and lows of every 1st of January, 2nd of January and so on ...). This gives 2 bands that are used as decision boundaries. If the observed temperatures is above the average of the highs it will be given the class "warm". The temperatures below the average of the lows will be labeled as "cold" and the ones in between "medium".

The underlying idea is to reproduce incremental, reoccurring drifts. In order to keep the size of the dataset consistent with the other datasets, only the first 1000 observations starting from the 17th of January 2006 were kept. For each day, 501 observations were generated by uniformly drawing temperatures between the maximum and the minimum observed that day.

6.4.2 Benchmarks

We now present some of the state of the art, drift handling algorithms which will be used as benchmarks.

DWM (Dynamic Weighted Majority) [60] is an ensemble method able to dynamically add or remove on-line learners, weighted according to their past performances. The predictions are given by a weighted-majority vote of the learners.

Learn++.NSE [32] is an incremental ensemble classifier based on the same ensemble structure as DWM. Its novelties lie in the mechanism used to weight the vote of the classifiers. Here the weights are dynamically updated according to the classifier's current and past error and in its passive drift detection mechanism.

Accuracy Updated Ensemble [25] also has a similar structure as DWM but processes the observations in chunk. One novelty is that after each data chunk it replaces the worst performing classifier by a new one, trained on the latest data chunk, no matter how good the

[2]The data can be downloaded at the following address:
http://www.wunderground.com/personal-weather-station/dashboard?ID=I75003PA1.

worst performing one is. Another novelty is that the weights of each classifiers are now updated according to their prediction error and the mean square error of a randomly predicting classifier on each data chunk instead of simply decreasing their weight according to a pre-set parameter. Finally, it is equipped with a memory surveillance module that prune the Trees when the memory limit is reached.

DACC (Dynamic Adaptation to Concept Changes) [50] has been proposed as an ensemble algorithm able to adapt to concept changes. It is also based on the same principles as DWM in the sense that it also maintains a set of on-line learners, weighted according to their performance. One improvement is the deletion strategy which doesn't delete automatically the worst performers of the ensemble but select them randomly from the worst performing half of the committee. Another improvement is the final vote which instead of using a simple weighted vote, rely either on a weighted vote of the learners with weights belonging to the upper half of the committee either on the prediction of the best performer of the ensemble.

On-line Bagging [81] is an extension of the well known Bagging algorithm [23]. The authors noticed that when the size of the training set tends to infinity, the number of occurrence of each example in the training set tends to a $Poisson(1)$ distribution. Thus, in an on-line framework, for every new example each base model is trained k times with $k \sim Poisson(1)$. The prediction is performed in the same way, by taking an unweighted vote of the base learners.

ADWIN Bagging [13] adds ADWIN (ADaptive sliding WINdow) [10] in order to detect changes and to estimate the weights in the On-line Bagging [81] algorithm.

Leveraging Bagging [14] has been proposed as an improvement of On-line Bagging [81]. The authors managed to improve the On-line Bagging algorithm by randomizing even more the classification process. Instead of using $\lambda = 1$ in the Poisson(λ) distribution to generate the weight of each new example, they use a larger value of λ. Their claim is that, by doing so, they increase the diversity of the weight given to every new example and thus modify the set on which the base classifiers will be trained. The second improvement comes from the use of output codes in order to add randomization at the output of the ensemble. According to them, the main benefits of this process are that it can reduce the effects of correlations between classifiers and further increase the diversity of the ensemble. Finally, they use ADWIN [10] to detect changes in error of the classifiers and replace the worst performing by new ones.

On-line Boosting [81] is designed to extend the AdaBoost.M1 algorithm [35] to the on-line framework. It is similar to On-line Bagging [81] except that it increases (respectively decreases) the parameter λ associated with the Poisson distribution for the next base learner if the latest example has been misclassified (respectively correctly classified).

Hoeffding Adaptive Trees [11] is an extension of the Hoeffding trees [46]. Hoeffding trees was initially developed in order to cope with a potentially infinite stream of data under the assumption that the distribution of the data wouldn't change. Hoeffding trees has been implemented in VFDT (Very Fast Decision Trees) with the addition of some heuristics. CVFDT (Concept-adapting Very Fast Decision Trees) [47] was developed as an extension of VFDT able to deal with change of concept. Finally, Hoeffding Adaptive Trees were developed to further improve CVFDT by getting rid of the parameter governing the size of the sliding window of instances. Instead of that, it uses ADWIN as a change detector to remove and replace the

branches of the tree with poor accuracy.

6.4.3 Implementation of the algorithms

MOA[3] and its extension[4] have been used to conduct the experiments and provide the implementation of the classifiers. All the parameters of the classifiers were set to default. This is required because there is no assumptions regarding the structure of the data or the type of drifts the classifiers will have to deal with. Therefore, it wouldn't be relevant to optimize parameters that would be suitable for a particular concept, at a particular time and for a particular dataset. In the case of the Droplets algorithm, the default radius was set to 0.1 for all the experiments. We also added the KNN algorithm with a fixed parameter of K=1 as a baseline.

6.5 Results and discussion

The results of the experiments empirically show the following facts:

1. Overall, the Droplets algorithm manages to obtain a better prediction performance than other state of the art adaptive learners, regardless of the type of drift encountered.
2. The prediction performance of the algorithm improves when the concept remains stable.
3. The algorithm manages to provide reliable predictions by discarding potentially difficult observations without relying on a fixed confidence threshold.

6.5.1 Consistently high and stable accuracy

In order to assess the performances of the Droplets against other algorithms, the results have been decomposed in two tables: the performances of the algorithms on the observations where the Droplets gave a predictions and their performances on all the observations where the Droplets didn't predict. The underlying idea is to avoid producing misleading results with a table reporting the overall accuracy on the test set where the Droplets could have achieved a very high performance at the cost of a very high proportion of unclassified observations (e.g. a performance of 100% by classifying correctly a single observation).

Table 6.1 shows the average accuracy (in percentage) of all the classifiers on the observations where the Droplets algorithm gave a prediction. The performance of the algorithms on the observations for which the Droplets algorithm abstained from prediction are shown and discussed in the following section.

The last line of the table indicates the percentage of unclassified observations by the Droplets algorithm on each dataset. Bold numbers indicate the best performing algorithm for each dataset.

[3] http://moa.cms.waikato.ac.nz/

[4] https://sites.google.com/site/moaextensions/

	RBF_1	RBF_2	RH	Temp	RBF_3	Avg Acc	Avg Rank
Droplets	86.32	74.20	77.65	**84.27**	**100.00**	**84.49**	**1.6**
DWM	73.90	69.40	77.16	69.11	97.36	77.39	5.2
Leveraging Bagging	79.17	**74.88**	71.94	72.36	96.18	78.91	4.0
Hoeffding Adaptive Trees	67.61	73.74	69.95	65.10	95.22	74.32	7.6
ADWIN Bagging	78.32	73.89	72.64	68.00	95.59	77.69	5.2
Online Bagging	72.29	73.87	69.68	66.55	95.41	75.56	6.8
Online Boosting	81.58	73.35	73.92	69.10	99.03	79.40	4.2
Accuracy Updated Ensemble	20.34	42.75	59.57	34.29	29.87	37.36	10.8
Learn++.NSE	40.07	60.44	65.67	43.29	2.05	42.30	10.0
DACC	**88.97**	71.08	**79.25**	80.40	98.75	83.69	3.0
1-NN	68.07	62.91	64.56	56.23	**100.00**	70.35	7.4
Unclassified observations (%)	15.05	33.62	24.89	5.98	98.15	-	-

Table 6.1: Average accuracy (in percent) on the observations for which the Droplets gave a prediction

The results show that Droplets algorithm managed to get the top spot twice out of 5 datasets. The average accuracy and the average rank indicate that overall, the algorithm has a steady performance compared to the other classifiers. Indeed, although some classifiers managed to obtain better performances on some datasets (DACC for instance outperformed the Droplets algorithm twice), their average accuracy and rank show that they are not able to perform consistently well, regardless of the dataset at hand or the type or drift encountered. This shows that, when the algorithm is confident enough to predict, it is capable of performing consistently well regardless of the characteristics of the environment it is learning from. This is a useful property because, in the general framework where there is no prior information on how the dataset at hand will evolve over time, it means that the Droplets algorithm is well equipped to adapt to a very broad range of drifts.

Note that the performance achieved on RBF_3 is an extreme case where the algorithm managed to get the perfect score at the cost of 98.15% of unclassified observations. The perfect score was achieved because of the high dimensions: the incrementally moving centroids never overlapped each other, which never gave the opportunity to output a wrong prediction. On the other hand, the very high percentage of unclassified observations comes from a perpetual discovery of unexplored parts of the map.

In order to assess the stability of the performance of the Droplets algorithm against the set of observed data, we re-runed the algorithm 45 times on each dataset. For each time step i, we randomly switched one of the 500 observation t_i^j with t_i. This means that the concept at a given time doesn't change compared with the experiments previously carried, but that the observed data (which will be used to update the model), representing this concept is now a different one. This has the effect of completely modifying the set of observations on which the algorithm learns. Table 6.2 gives an overview of the mean and variance achieved for the

	RBF_1	RBF_2	RH	Temp	RBF_3
Accuracy	87,79±0,31	73,32±0,36	77,47±0,40	84,28±0,37	100±0.00
Unclassified	15,43±0,91	34,12±1,89	25,23±3,13	5.00±0,34	98,15±0,01

Table 6.2: Average performance and variance of the Droplets over 45 runs

accuracy and the percentage of unclassified observations.

Here again, the results indicate that the performances of the Droplets algorithm are very stable as there is little variance in the observed accuracy and percentage of unclassified observations over the 45 different datasets. In particular, this shows that the learned model doesn't over-fit the past observations.

6.5.2 Reliable predictions under concept drift

In this section, we assess whether the only parameter of the Droplets algorithm ($R_{default}$, the default radius associated with a new Droplet) acts as a confidence threshold or not and whether the observations on which the algorithm chose to abstain were indeed harder to classify.

6.5.2.1 The radius size doesn't act as a confidence threshold

In the selective classification framework, there is generally a trade-off between risk and coverage [106] (the accuracy of the algorithm improves as the percentage of classified observations diminish and the other way around) and the performance of this type of classifier is usually assessed against the curve connecting these 2 quantities. Figure 6.1 shows the evolution of the percentage of accurate predictions along with the percentage of unclassified observations as different values are given to the $R_{default}$ parameter (from 0.02 to 0.38 with a step of 0.04).

The results indicate that the parameter's value yielding the best accuracy is also the one minimizing the percentage of unclassified observations, regardless of the dataset and of the type of drifts encountered. The shape of the curve also shows there isn't a trade-off between accuracy and coverage because the accuracy of the Droplets increases with the coverage. This leads to draw the conclusion that the parameter governing the default radius size doesn't act as a confidence threshold.

This is an important property because a confidence threshold is parameter which is hard to calibrate. Indeed, it could be unclear whether an accuracy of $x\%$ and a coverage of $y\%$ would be better or not than an accuracy of $y\%$ and a coverage of $x\%$. In this case, because of the absence of trade-off between these two quantities, the user of the algorithm simply has to optimize this value to end up with an unique best value.

The default radius sizes that gave the best results (i.e. the highest accuracy and the lowest rate of unclassified observations) are respectively 0.22, 0.1, 0.18 and 0.1 for RBF_1, RBF_2, RH and Temperatures.

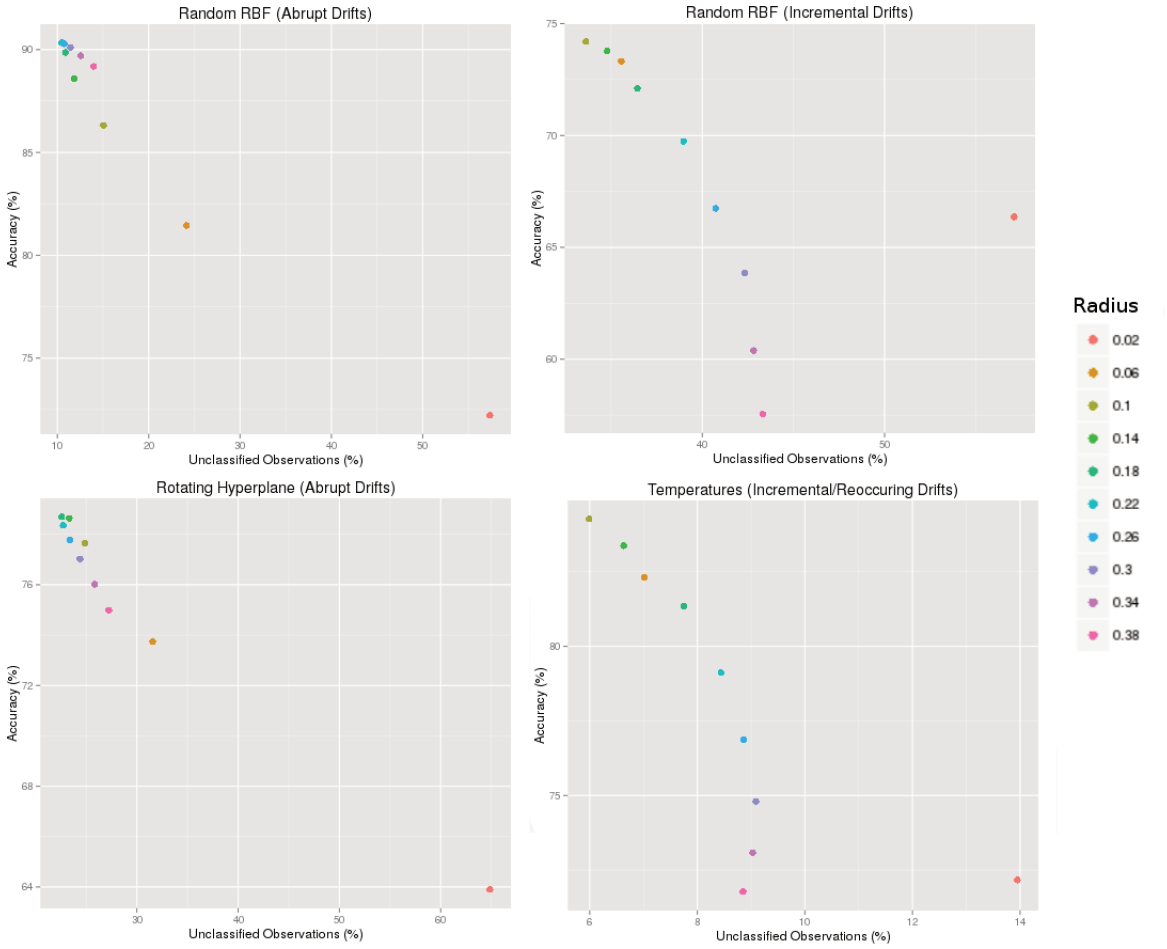

Figure 6.1: Evolution of the accuracy against the percentage of unclassified observations for different default radius

	RBF_1	RBF_2	RH	Temp	RBF_3	Avg Acc	Avg Rank
DWM	50.76	51.30	72.65	53.39	86.64	62.95	3.8
Leveraging Bagging	46.74	**55.86**	63.65	46.53	87.03	59.96	4.6
Hoeffding Adaptive Trees	44.72	54.94	60.77	48.55	86.37	59.07	6.0
ADWIN Bagging	49.09	55.05	64.34	50.70	85.40	60.92	4.2
Online Bagging	43.29	55.03	63.59	49.60	85.38	59.38	5.8
Online Boosting	52.50	54.80	65.26	55.72	88.37	63.33	3.0
Accuracy Updated Ensemble	25.22	37.18	54.84	32.43	31.78	36.29	9.4
Learn++.NSE	26.57	47.00	52.97	36.54	2.93	33.20	9.0
DACC	**70.57**	50.83	**75.85**	**66.36**	88.99	**70.52**	**2.4**
1-NN	37.55	42.99	54.87	49.31	**98.30**	56.60	6.4

Table 6.3: Average accuracy (percentage) on the observations where the Droplets algorithm didn't predict

6.5.2.2 Ability to accurately discard the tricky observations

Table 6.3 shows the accuracy achieved by the other algorithms on the observations for which the Droplets algorithm chose to abstain. Bold numbers indicate the best performing algorithm for each dataset.

The results show that the Droplets managed to successfully discard the observations which were harder to classify. Indeed, except in the case of the algorithms that were already performing poorly on the observations where the Droplets predicted (AUE on RBF_1, RBF_3 and Learn++.NSE on RBF_3), the performance of all learners decreased on the observations which were unclassified by the Droplets algorithm. This result suggest that, by choosing to abstain when an observation is received in a previously unexplored part of the feature space, or when it is received in an area where the labels of neighboring observations are in conflicts, the Droplet algorithm managed to capture an important aspect of what makes a prediction reliable.

It could be argued that, despite achieving a worse performance on these observations, the achieved accuracy of some of the algorithms remains good enough in some cases (e.g. DACC achieved almost 89% on RBF_3). Ultimately, it will be up to the end user to decide whether the problem at hand requires the highest possible accuracy and if he is willing to sacrifice coverage for this. Here we claim that when the goal is to obtain stable and high performances, the Droplets algorithm should be considered.

Note that on these observations, a majority of classifiers didn't even manage to outperform by at least 5% a simple 1-NN on RBF_3 and Temperatures datasets.

The bottom part of Figure 6.2 shows the evolution of the accuracy of the top 4 classifiers on the RBF_1 dataset. The displayed accuracy has been obtained by computing the percentage of accurate predictions for each of the 500 observations tested at every time step.

Despite being one of the dataset were the Droplets algorithm performed the worst, the results show that it managed to recover quickly from the abrupt drifts. The shape of the curve also indicate that the performances of the Droplets algorithm continuously improve when the concept remains stable which was one of the requirement of the introduction chapter of this

thesis.

The top plot of Figure 6.2 shows the percentage of unclassified observations over time. This reflects the uncertainty of the Droplets algorithm over time. In the particular case of the RBF_1 dataset, the observations are constrained in the unit hypercube. Therefore, once the initializing set is over, the map will have been extensively explored and new observations are unlikely to appear in previously unexplored parts of the map. This means that the observed increase in uncertainty reflects the fact that observations with conflicting labels where received. As is expected, this uncertainty explodes right after the drift, and gradually goes down as more information is accumulated over time.

An idea for futures research would be to use the evolution of this uncertainty as a drift detector.

6.6 Conclusion

In this chapter, we have claimed that an on-line algorithm combining the instance based structure with the framework of prediction with a reject option would manage to obtain good predictions performances on a data stream subject to concept drifts (which was the main goal presented in the introduction chapter).

We claim that the instance based structure brings meta informations which can be taken advantage of when dealing with drifts. We have already demonstrated that point in chapter 4 for the meta learning level and showed in this chapter that this is also true at the base learning level.

Furthermore, when dealing with the uncertainty associated with evolving environments, an algorithm outputting reliable predictions can prove particularly useful, especially when wrong predictions are costly. In this case, the instance based structure can also be used to automatically filter the observations which are harder to classify.

In order to implement these ideas, we have introduced the Droplets algorithm, which shares some similarities with DEA presented in chapter 4. This algorithm keeps track of the past observations and choses to forget the observations which have proved to be outdated. It can deal with a broad range of drifts (abrupt, incremental or reoccurring) without needing to explicitly detect them and produces reliable predictions by abstaining from predicting on potentially difficult observations without relying on a fixed confidence threshold (which would be hard to calibrate and would induce a trade-off between coverage and accuracy).

Experimental results show that the Droplets algorithm managed to achieve the best average rank and accuracy on 5 datasets and against 10 state of the art classifiers. This result prove the stability of the performances achieved by this algorithm, regardless of the environment encountered. The results also show that it is able to accurately distinguish which observations are easily classifiable, as the performance of the other learners drops on the observations where the Droplets algorithm chose to abstain.

This means that the Droplets algorithm is well suited for tasks where wrong predictions are costly and where accuracy is more important than coverage.

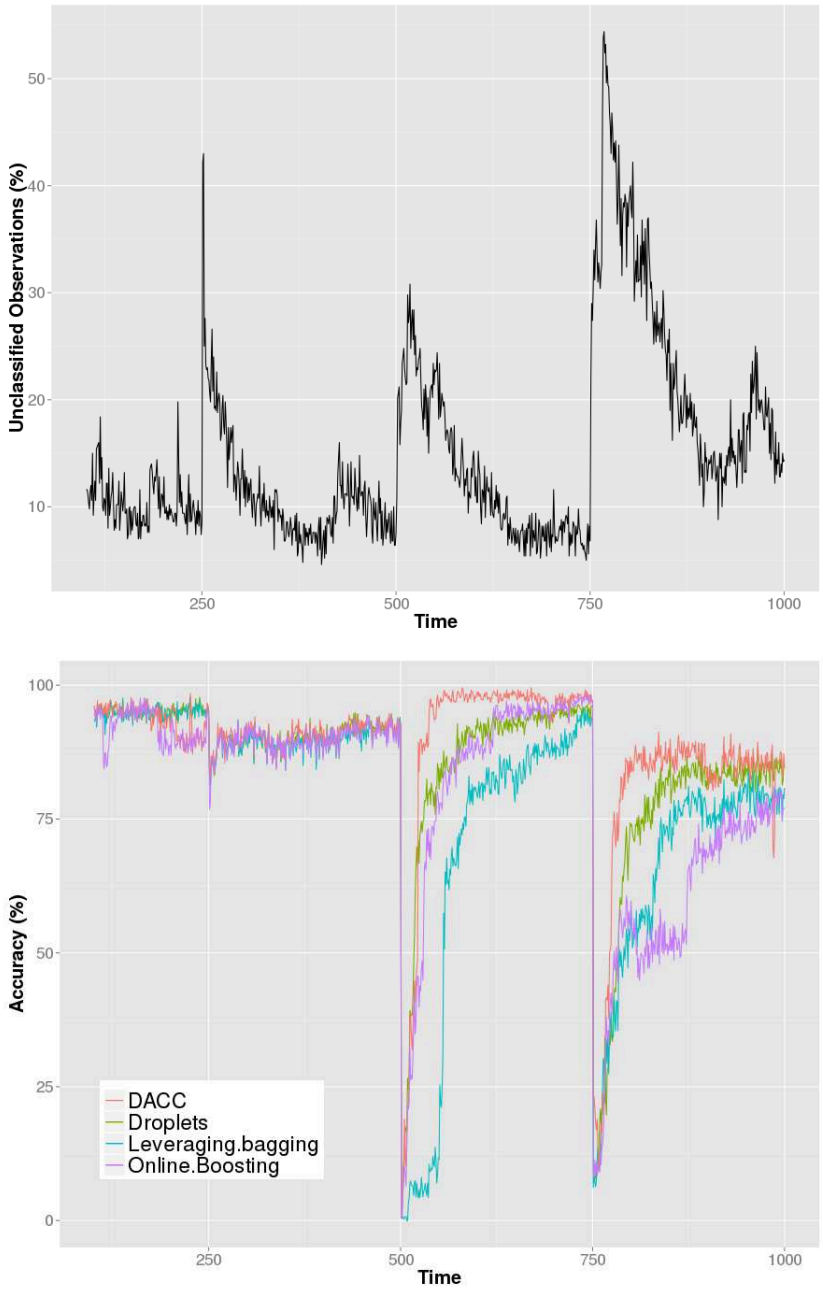

Figure 6.2: Evolution of the percentage of unclassified observations (top) and evolution of the accuracy of the top 4 classifiers on RBF_1 (bottom)

Chapter 7

The problem of limited memory and running time

7.1 Introduction

In the previous chapters, we have argued that the instance based structure was particularly well suited to handle the problem of accurately predicting on a data stream subject to concept drifts and we have proposed the Droplets algorithm as an instantiation of this idea. Data streams however, can potentially be infinite and the observations might be streamed at a very high frequency. Therefore, it is necessary for an instance based learning algorithm to be able to deal with these 2 components.

In this chapter, we propose a new way to address the problems of constrained computer memory and running time for this class of algorithms. We start by showing that both of these problems can be resolved by constraining the number of instances kept into memory. This, in turns, raises the question of selecting the observations that should be retained into memory. We argue that, instead of using time as a criterion to select the observations which will remain into memory, it is better to retain observations that minimize the differences in outputted prediction and rule learned with the infinite memory algorithm. We call this the Rule Preserving (RP) criterion.

This idea is implemented for the Droplets algorithm. Experimental results on 6 artificial and semi-artificial datasets reproducing various types of drifts show that this strategy achieves better results than a simple temporal window.

7.2 Framework

In this chapter, the considered framework (supervised classification on a stream of data subject to concept drifts), goal (predict as accurately as possible the label y_i associated with x_i) and performance metric (0-1 loss function) are identical to chapter 4 and thus won't be presented in details again.

The difference however is that we now take into account 2 additional constraints: M which is the maximum computer memory available to the algorithm (in Megabyte: MB) and S which is the elapsed time between reception of 2 consecutive unlabeled observations (in seconds).

We also make the following assumptions:

1. M is known beforehand and is constant over time.
2. S is also known beforehand and also constant over time. This means that observations are received at regular time intervals.

7.3 Constraining the number of observations in memory

If we assume that the computer memory used by an instance based algorithm is proportional to the number of past observations stored into memory, constraining the memory consumption of this algorithm can easily be achieved by constraining the number of instances kept into memory.

Furthermore, the time complexity of an instance based learning algorithm is also a function of the number of instances kept into memory (as the algorithm has to loop over these observations at each time step) and can also be constrained by setting an upper bound on the maximum number of observations allowed into memory.

Thus, assuming that the space (in MB) required to store each observations can be known beforehand (e.g. w MB for each instance in memory) and that the running time of the algorithm can be determined as a function of the number of observations into memory (e.g. z additional seconds per observations in memory), we can solve the two issues of limited computer memory and running time by setting a parameter on the Maximum Number of Observations (MNO) allowed in memory when the algorithm starts running:

$$MNO = Min\left(\left\lfloor \frac{M}{w} \right\rfloor, \left\lfloor \frac{S}{z} \right\rfloor\right)$$

where $\lfloor . \rfloor$ is the floor function.

Note: It is easy to see that the computer memory consumption of an instance base algorithm should grow linearly with the number of observations kept into memory. However, it might not be the same for the running time. The running time might, for instance, grow exponentially with the number of observations kept into memory. This doesn't change anything as, constraining the number of observations into memory will also set an upper bound on the running time.

7.4 Selecting the observations kept into memory

7.4.1 Understanding the goal behind forgetting

When selecting the observations that should remain into memory it is important to distinguish between two points:

On the one hand, we have presented some of the main forgetting mechanisms in chapter 3. These mechanisms aim at deleting observations from memory in order to adapt to concept drifts.

On the other hand, it is also possible to devise a forgetting mechanism which will also choose which observations should be removed from memory, but with the goal of constraining the memory consumption and the running time of the algorithm.

In this chapter, we aren't concerned with the adaptation problem. We assume that the instance based algorithm at hand is already capable of dealing with drifts and therefore capable of selecting the observations that should and shouldn't remain in memory in order to adapt to concept changes. Therefore, under this assumption, all the observations left into memory are deemed useful according to the algorithm and the goal is to select which of these observations should be deleted in priority in order to meet with the constrains on computer memory and running time.

7.4.2 Why a sliding window isn't suitable

One of the most commonly used solution is to maintain the memory consumption (and running time) of an instance based algorithm constant over time by using a rolling window. The window holds the last p observations (with p a user defined parameter) and when a new observation is released, it is added to the window and the oldest observation is deleted. The window can be either of fixed or adaptive length [41].

Regardless of the case (fixed or adaptively sized window), the first underlying assumption here is that time should be used as a criterion to decide whether an observation should be deleted from memory or not. The second underlying assumption is that it is necessary to keep contiguous observations. We have already discussed the shortfalls of these 2 assumptions in Section 6.3.2.3 when the goal is to adapt to changes. These shortfalls remain valid in the framework of forgetting to meet constrained memory and running time.

7.4.3 The Rule Preserving criterion

Instead, we propose a Rule Preserving (RP) criterion which aims at yielding predictions and learned rule as close as possible to the predictions and learned rule that would have been obtained if the algorithm had access to infinite memory.

Achieving this goal can be done by establishing a ranking of the observations saved in memory and by removing in priority the redundant observations or the observations which are unlikely to have a significant effect in the outputted prediction of the algorithm. Thus, deleting an observation at the top of this ranking would significantly change the learned rule and outputted prediction whereas an observation at the bottom of the ranking would have little effect on the learned rule and outputted prediction. Consequently, we claim that these observations should be deleted in priority.

We further claim that this strategy leads to better prediction performances than a simple time stamp criterion. In the next section, this idea is tested for the Droplets algorithm which was introduced in Chapter 6.

7.5 The Droplets algorithm with memory management

In this section, we explain how the Droplets algorithm (presented in chapter 6) has been modified using the principles described above, in order to deal with constrained memory and running time on data streams subject to concept drifts.

7.5.1 Criteria used to rank the Droplets

As explained in the previous section, in order to maintain an upper bound on the running time and computer memory consumption, it is necessary to establish a ranking of the observations (Droplets) into memory. This ranking should favor the observations which are likely to have a big impact on the rule learned and on the outputted predictions. In the case of the Droplets algorithm, we have used 2 different criteria.

7.5.1.1 Volume not overlapped

The first criterion used to rank the Droplets is the Volume Not Overlapped (VNO) of each hypersphere. This is the volume of the Droplet which is not shared with at least another Droplet of the same class (recall that Droplets associated with different classes are not allowed to overlap each other).

Because the outputted prediction will only depend on the area covered by the hyper-spheres and, because there is no assumption on the distribution of the observations (i.e. in which part of the map the next unlabeled observation will be received), the higher this value, the more likely it is that a Droplet will be the only one used for the prediction.

Conversely, when this value is very small (e.g. $VNO = 0$), this means that all, or the majority of, the volume covered by the Droplet is also covered by at least another Droplet of the same class. Thus, if an observation is received in that part of the map, the Droplet can be safely deleted as there is already another Droplet which will be able to output a prediction. This is reason why Droplets with low VNO are deleted in priority.

Note: To compute the VNO, Monte Carlo simulations were used, randomly sampling P points (where P is a fixed parameter) in the hypersphere of interest and assessing the fraction of points that belong to at least another hypersphere.

7.5.1.2 Radius size

In some cases, there could be a tie in the ranking based on the VNO (in particular, for all the Droplets that have $VNO = 0$) and a second criterion is required to select the Droplet that should be deleted. The first criterion aimed at preserving the outputted prediction. The goal of this second criterion is to preserve as much as possible the model learned after update with the latest labeled observation. In other words: minimizes the change between the model obtained with infinite memory and the model obtained with constrained memory.

The model obtained after update is a function of the latest observation received. Two cases can arise when adding the latest Droplet to the map:

1. Either the latest Droplet doesn't overlap an existing Droplet with a conflicting class (and in this case, there is no need to update the radius of the existing Droplets).

2. Either the latest Droplet overlaps one or more existing Droplets with a conflicting class (and in this case, there is a need to update the radius of at least one of the existing Droplets).

In the first case, the Droplet for which the deletion would minimize the difference in the model learned after update is the Droplet with the smallest VNO.

In the second case, the size of the latest Droplet will be a function of the value of the largest overlap it has with the other conflicting Droplets. For instance, if there are n Droplets overlapping the latest one (received at time t+1), the size of the new Droplet will depend of $max\,(\lambda_1, ..., \lambda_n)$ with:

$$\lambda_i = R_{default} + R_i - \|x_{t+1}^{norm} - x_i^{norm}\|, \forall i \in \{1, ..., n\}$$

λ_i will increase if R_i increases and $\|x_{t+1}^{norm} - x_i^{norm}\|$ decreases. Conversely, λ_i will decrease if R_i decreases and $\|x_{t+1}^{norm} - x_i^{norm}\|$ increases. Because we don't make any assumption on the parts of the map which are most likely to received the next observations, the distance $\|x_{t+1}^{norm} - x_i^{norm}\|$ can't be used as a criterion to decide which λ_i is most likely to end up having the highest value.

Therefore, it is assumed that Droplets with a large radius are more likely to end up having a larger overlap with the future Droplets than Droplets with a small radius. As a consequence, Droplets with small radius can be deleted priority as they are less likely to have an impact on the size of future Droplets and thus on the rule learned after update of the map.

This leads to pick the radius as a second criterion for deletion (in case of tie for VNO) and delete in priority the Droplets which have a small radius.

7.5.2 The algorithm

The full Droplets algorithm equipped with the RP criterion is detailed in Algorithm 7.1. The differences with the base Droplets algorithm presented in Algorithm 6.3 are highlighted in red. We now go through the memory management algorithm step by step:

When a new observation is received, the algorithm first checks whether the MNO is reached. If this is the case, it gets the index in the map of the observation which is at the bottom of the ranking (in Algorithm 7.1, this index is u) and remove this Droplet from memory.

A new unlabeled observation x_i is then received and the process follows the same steps as for the basic Droplets algorithm.

Then, once the map has been updated, the algorithm updates the ranking of all Droplets in the map.

A video comparing the models learned with the 3 types (infinite memory, RP criterion and simple time window) of Droplets algorithms can be found here[1].

[1] https://www.youtube.com/watch?v=_uLhRX9FXxc

Algorithm 7.1 The Droplets algorithm with Rule Preserving criterion.

Inputs:
$R_{default}$: parameter setting the default value of the radius
N: size of the initializing set
ε: an arbitrarily small positive number
MNO: parameter setting the maximum number of observations allowed in memory

$\overrightarrow{D} \leftarrow Compute\ normalization\ constants\,(x_1, ..., x_N)$
$map_0 \leftarrow \emptyset$ //Initialize the map to empty
$ranking_0 \leftarrow \emptyset$ //Initiliaze the ranking to empty
Foreach: x_i with $i \in \{1, 2, ...\}$
  $n \leftarrow Get\ number\ observations\ in\ memory\,(map_{i-1})$
  If $n = MNO$ **Then**
    $u \leftarrow Get\ index\ Droplet\ at\ the\ bottom\ of\ the\ ranking\,(ranking_{i-1}, map_{i-1})$
    $map_{i-1} \leftarrow Remove\ Droplet\ from\ map\,(map_{i-1}, Criteria : index = u)$
  End If
  $x_i^{norm} \leftarrow Normalize\ observation\left(x_i, \overrightarrow{D}\right)$
  // **Prediction step**
  If $i > N$
    $\hat{y}_i \leftarrow Predict\,(map_{i-1}, x_i^{norm})$ // Algorithm 6.2
  End If
  // **Map update**
  $y_i \leftarrow True\ Label\ of\ x_i$
  $indexes \leftarrow Get\ Overlapped\ Droplets\ with\ conflicting\ labels\,(map_{i-1}, x_i^{norm}, y_i)$
  $map_i \leftarrow Update\ radii\,(map_{i-1}, indexes, R_{default}, \varepsilon)$ // Algorithm 6.1
  $ranking_i \leftarrow Update\ Ranking(Ranking_{i-1}, map_i)$
End Foreach

7.5.3 Temporal complexity

The temporal complexity of the RP is $O\left(C.(MNO)^2\right)$ with C the chosen number of Monte Carlo simulations and MNO the maximum number of observations allowed in memory. This ensures that the running time of the algorithm will be constrained for fixed values of MNO and C.

7.6 Experiments

In this section, we lay down the experimental protocol and experimentally show that the RP is better than a simple temporal window at replicating the predictions of the infinite memory Droplets algorithm. We also show that the performances achieved by the RP are superior to the ones of the simple temporal window.

7.6.1 Experimental protocol

In order to compare the performances achieved by the RP criterion against the commonly used temporal window, we have carried out two sets of experiments on 6 artificial and semi-artificial datasets[2] including 1000 observations and reproducing different types of drifts (the drifts are evenly distributed on the datasets). The descriptions of the datasets RBF, Rotating Hyperplane and Temperatures can be found in [74] and a description of SEA is given in [94].

The default radius of the Droplets for all the experiments was set to 0.1 whereas the number of Monte Carlo simulations was set to $C = 1000$. In each case, the first 100 observations were kept for initialization purpose and the performance was assessed on the remain observations.

Three versions of the Droplets were implemented. Their only difference is the way they manage the Droplets in memory. The first version is based on a Temporal Window (TW) that adds the latest Droplet to the memory and continuously drops the oldest Droplet once the memory is full whereas the second version uses the RP criterion to decide which Droplet should be deleted. Finally, the third version of the Droplet is being given access to infinite memory.

We chose to compare the Droplets algorithm against itself in order to make sure that any difference in the achieved performances would effectively come from the the chosen memory management procedure.

7.6.2 Results

7.6.2.1 The RP criterion outperforms a temporal window

In the first set of experiments, we compared the differences between the achieved performances of the RP and the TW on the 6 datasets.

Because the Droplets allow to abstain from prediction, we monitored the difference in the percentage of correct predictions (accuracy) P_{acc} as well as the difference in the percentage

[2]The datasets used can be downloaded here: http://webia.lip6.fr/~loeffel/ESANN2016/

of unclassified observations P_{unc} of the 2 algorithms. Therefore, for each dataset, the values displayed on the y-axis of Figure 7.1 are equal to:

$$P_{acc}(RP) - P_{acc}(TW)$$

and

$$P_{unc}(TW) - P_{unc}(RP)$$

The values displayed on the x-axis indicate the maximum number of observations (MNO) that were allowed in memory. For both charts, a positive number indicates that the RP outperforms the temporal window.

The results indicate that globally the RP has a better accuracy and a lower percentage of unclassified observations than a simple temporal window, regardless of the dataset, the type (or absence) of drift and the number of observations allowed in memory.

The differences in performances can be as significant as +15% in correctly classified observations and -16% of unclassified observations for a MNO of 40. As the MNO increases, the coverage of both methods converges to a point where there is not much difference (less than 2%) between them but where the RP still retains an hedge over the temporal window.

7.6.2.2 The RP replicates well the outputs of the Droplets with infinite memory

The second set of experiments was aimed at assessing the ability of the RP to replicate the predictions obtained by the infinite memory algorithm.

At each time step, the prediction obtained with the infinite memory Droplets was compared to the prediction obtained with the RP. These predictions were then classified into 4 categories: exactly the same output, prediction (for the infinite memory Droplets) to unclassified (for the RP), unclassified to prediction and two different predictions. For comparison purposes, we performed the same experiment for the Droplets equipped with the TW which was also compared to the infinite memory algorithm.

The results for the SEA dataset with 20 drifts are shown in Figure 7.2. For a given MNO (shown on the x-axis), the left bar represents the proportion of each of the 4 types of outcome achieved by the RP whereas the right bar represents the proportion of outcome achieved by the TW. The results on the 5 remaining datasets were similar but have not been reproduced here.

The results indicate that despite the constrain on memory usage, the predictions outputted by the RP are for the vast majority, exactly the same as the ones obtained with infinite memory. Even when there are as little as 20 observations allowed in memory, the RP managed to replicate the prediction of the infinite memory Droplets 58% of the time.

They also show that the RP is consistently better at replicating the predictions of the infinite memory Droplets than the TW, regardless of the MNO.

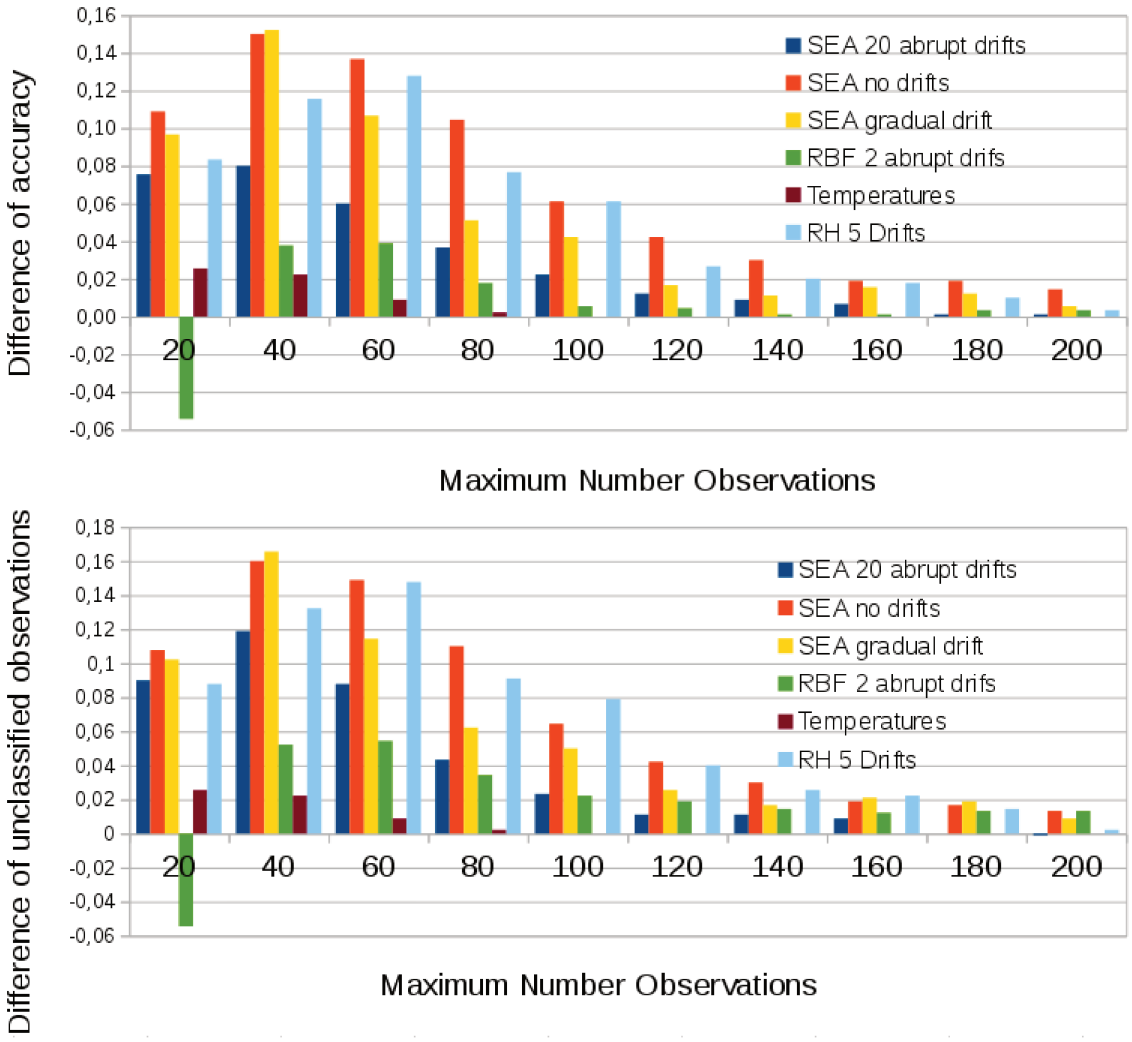

Figure 7.1: Difference in performances: Temporal Window vs Rule Preserving criterion

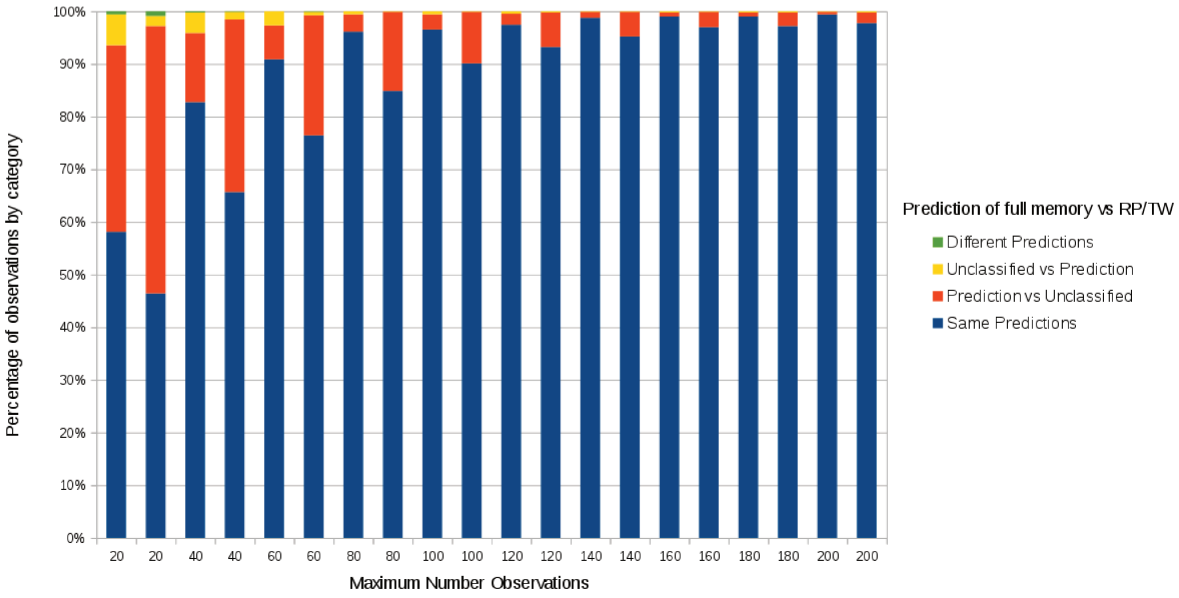

Figure 7.2: Outputs by categories of the RP (left) vs TW (right) on the SEA dataset with 20 abrupt drifts

Finally, in the cases where the outputted prediction of the RP was different than the one obtained with infinite memory, the majority of these predictions ended up being unclassified by the RP which is the safest outcome when lacking information.

7.7 Conclusion

A successful instance based algorithm must be able to keep its memory consumption and running time constrained while at the same time retaining good prediction performances.

In this chapter, we have argued that the issues of limited computer memory and running time could be overcame by limiting the number of observations that an instance based learning algorithm was allowed to keep into memory. Under the assumption that the instance based algorithm at hand is already capable of adapting to concept changes, we have proposed a new criterion (the Rule Preserving criterion) to select the observations that would be allowed to remain into memory. This criterion aims at minimizing the differences between the model learned with infinite memory and the model learned with constrained memory.

We have applied this criterion to the particular case of the Droplets algorithm by establishing a ranking of the Droplets saved into memory. The ranking was based on to 2 criteria which assessed whether the deletion of a particular Droplet would produce a significant difference in the model learned with infinite memory or not and whether deleting this particular Droplet would have an effect on the predictions given by the algorithm.

The results achieved on several datasets reproducing different types of drifts, indicate that

the RP criterion obtains better prediction performances than a temporal window which selects the observations based on their time stamp. They also indicate that the RP criterion is capable of replicating very well the predictions of the infinite memory algorithm, even under extreme memory constrains.

Chapter 8

Conclusion

8.1 Considered problem and challenges

Machine learning algorithms have proved to be very useful when it comes to automatically extract informations from large datasets. However, not all machine learning algorithms are suited to handle data streams. Indeed, classic machine learning algorithms were created to learn a static rule out of a dataset, fully available at the training step. This type of algorithm is no longer suited for data streams as a full dataset is not available during the training step and that the observations are received sequentially. This raises the need of using an algorithm which can maintain a model up to date with the observations received so far.

Another issue is that it can no longer be assumed that the observed data are generated from an unique hidden probability distribution (the concept). Indeed, the observed values generated by a data stream often represent a snapshot of the characteristics of an object at a given time. As the object might evolve over time, so are its characteristics. For instance, the electrocardiogram monitoring the cardiac activity of a person, will behave differently according to the current activity of that person. Therefore, a suitable algorithm must be capable of adapting to these concept changes.

Finally, the chosen learning algorithm must also deal with some constraints such as the available amount of computer memory or the frequency at which the data are streamed. On the one hand, it must ensure that its memory consumption remains within the allowed limits (which is not necessarily straightforward when the amount of data to process is potentially infinite) and on the other hand it must also ensure that its running time is smaller than the time it takes to receive two consecutive observations from the stream (which can also prove very challenging when the data are streamed at a very high frequency).

To sum up, the considered problem in this thesis is supervised learning (in both classification and regression settings) on a data stream subject to concepts drifts and under the constraints of limited computer memory and running time. The goal is to predict as accurately as possible the labels associated with the received observations. We didn't make any assumption on the type of the drifts or about the nature of the concepts that could be expected.

8.2 Contributions

In order to deal with these challenges and under the previously stated framework, we have claimed that a successful learning algorithm had to combine several properties: it must be capable to learn and adapt continuously, its prediction model should not make any assumption on the nature of the concept it is trying to learn, its adaptation procedure should be capable to deal with any kind of drift, it should be allowed to abstain from prediction when necessary and its running time and memory consumption should be constrained.

We now discuss how we obtained each of these properties.

8.2.1 On-line architecture for continuous learning and adaptation

On-line learning algorithms are the obvious choice to handle data streams and have already been widely used in the literature for this purpose. Because of their structure, they can continuously update their learned model by always making use of the latest data. This mechanism ensures that when the concept remains stable, the algorithm will always fine tune its learned model. Conversely, when the concept drifts, this very same adaptation mechanism will allow the algorithm to adapt to the changes.

All the proposed algorithms in this PhD thesis are on-line (DEA in chapter 4, the abstaining mechanism in chapter 5, the Droplets in chapter 6 and the strategy for memory management in chapter 7). The major challenges for an on-line algorithm aiming a predicting in this environment are to be flexible enough to learn a wide range of concepts and to define an adaptation / forgetting mechanism which would be suited for a wide range of drifts. We now explain how we solved these issues by taking advantage of the instance based structure.

8.2.2 The benefits of the instance based structure

The one major contribution of this thesis is our claim that the instance based (IB) structure has some properties which make it extremely well suited to handle the issue of data streams with drifting concepts. We now go through some of the main properties.

8.2.2.1 Flexibility to learn a wide range of concepts

IB algorithms make very little assumptions about the nature of the concept they are trying to learn. This grants them a great flexibility which make them likely to be able to learn from a wide range of concepts. This is in contrast with the model based architecture which often make prior assumptions on the nature of the concept. When wrong, these assumptions can hinder proper learning of the concept at hand. In our case, we didn't assume any prior knowledge on the expected evolution of the concept over time and consequently this flexibility proved to be a key factor in the success of DEA and the Droplets algorithm. This point has been demonstrated through the various experiments conducted in chapters 4 and 6 where the obtained results showed that the proposed IB algorithms were capable of regularly ranking amongst the top performing algorithms against several (8 algorithms for DEA and 9 algorithms

for the Droplets) state of the art drift handling algorithms and on a wide range of datasets (25 datasets for DEA and 5 datasets for the Droplets), with very different characteristics .

8.2.2.2 Stored observations bring valuable meta-informations

Storing some of the past observations into memory can bring valuable meta-informations which can be used by a learning algorithm to enhance its performances. Examples of how these informations could be used were presented with the Droplet Ensemble Algorithm (DEA, chapter 4) and the Droplets algorithm (chapter 6).

DEA is an ensemble learning algorithm relying on the IB structure which uses past observations to keep track of the area of expertise of its base learners on the feature space. This information is used to select the subset of its base learners which is the best suited to predict on the latest observation received.

In the case of the Droplets algorithm, the past observations were used to keep track of the regions of the feature space which have been explored and to assess whether the algorithm could confidently predict or not (this point is detailed further in the next subsection) for a given unlabeled observation.

8.2.2.3 Powerful adaptation mechanism

Adaptation without explicit detection: The IB structure also allows a great deal of flexibility on the choice of the criterion used to decide which observations will be allowed to remain into memory in order to maintain the model in line with the current concept (e.g. time, spatial relevance, probability distribution, ...). These criteria can be used independently or combined. Amongst these criteria, we chose to rely on hard evidences of obsolescence of past observations to trigger the forgetting mechanism.

As explained in chapters 4 and 6, a past observation was deemed out of date if a newest observation with similar characteristics (i.e. with similar observed features values) proved it wrong (i.e. has a label which conflicts with the label of the old observation). Two consequences of this are that the model is constantly updated with the latest observations (which fits our on-line requirement) and that it isn't necessary to detect concept drifts anymore in order to trigger the update of the model.

As we have argued in Section 6.3.2.1, detecting concept drifts can prove to be a very difficult task in the absence of prior information about the stream. Indeed it might not be easy to accurately spot the metric that must be tracked (e.g. should a performance metric or a statistic over the observations be tracked ?) and to choose the best action when an alarm is triggered from this metric (e.g. should the whole model be discarded or just some parts of it?). Furthermore, setting a threshold above which an alarm will be triggered can also prove challenging as it adds another parameter to the algorithm which will most likely need to be constantly fined tuned and will further complicate the model.

Therefore, we argued that it is easier to perform adaptation without explicit detection.

Cautious adaptation to drifts: Another consequence of this update mechanism is that all the updates of the model will be local (i.e. they will only happen around the latest observation) and therefore, that the adaptation process will always be gradual, regardless of the type of drifts. It could be argued that this strategy might not always be the best, especially when the whole model becomes outdated after an abrupt drift. However:

1. This adaptation procedure ensures that only the parts of the model which must be updated are effectively updated. For instance, when the concept drifts locally, this forgetting mechanism allows for a bespoke update of the local parts of the model which are outdated without loosing the valuable informations accumulated elsewhere. Therefore, by avoiding to throw away the accumulated knowledge, the model is less prone to catastrophic forgetting (when all the accumulated knowledge is deleted by mistake).

2. It is impossible to know beforehand whether a drift will make the whole former model out of date or not. This would require prior informations on the type of expected drifts which we didn't allow. Because we don't know which type of updates will be required for the whole model, we use a "safe" updating mechanism which rely on hard evidence of obsolescence.

This cautious approach allowed the IB methods developed in this thesis to achieve the best average accuracy and average rank against other state of the art algorithms and on datasets reproducing a wide range of drifts as shown by the results achieved in tables 2 and 6.1. These results demonstrate the capabilities of this adaptation mechanism (both DEA and Droplets algorithms share this characteristic) to perform consistently well, regardless of the nature of the drifts or the characteristics of the concept at hand.

8.2.2.4 Simple solution to running time and memory constraints

The IB structure offers a simple and natural solution to the problems of constraining the computer memory consumption and running time of the algorithm. This is achieved by limiting the maximum number of observations allowed into memory. In order to decide which observations should be deleted to meet these constrains and under the assumption that the learning algorithm at hand was capable of dealing with drifts, we have proposed in chapter 7 a criterion which aims at replicating the model learned with infinite memory and showed that it achieved better performances than a simple time stamp criterion. This is mainly the case because the time stamp criterion doesn't get rid of redundant observations which unnecessarily clutter the memory.

8.2.3 Handling uncertainties by abstaining

Another major contribution of this thesis is to stress the importance of allowing the learning algorithm to abstain from prediction in this framework. This is because the drifts can generate a lot of uncertainties and at times, an algorithm might lack the necessary information to accurately predict. In these cases, instead of trying to output a prediction at all cost, we have

showed that it might be better to automatically disconnect the algorithm by allowing it to abstain from prediction. This would ensure that more data are collected before a prediction can be given and could avoid costly mistakes. The challenge here was to accurately spot the observations which might have lead to a wrong prediction. We studied the benefits of this approach in two cases:

In chapter 5, we have studied the regression setting with a constrain on the expected precision of the outputted predictions. We have shown that any state of the art algorithm capable of learning on a drifting data stream could be allowed to abstain if it is given a set of reliability estimators. Each estimator was assessing a particular aspect of the reliability that can be given to a prediction. Their estimates were then combined with a simple majority vote.

In chapter 6 we have studied the general classification setting. We have made use of the information brought by the instance based structure in order to automatically disconnect the Droplets algorithm during periods of time when information where judged insufficient. This would happen in two cases: either when a new part of the feature space would be explored, either when an observation would be received in a part of the feature space where similar past observations with conflicting labels would have been received.

In both cases, the experiments have shown that these 2 strategies were very effective at filtering the tricky observations and therefore that abstaining could significantly improve the prediction performances of the learning algorithms.

8.3 Limitations and perspectives

We now go through some of the identified limits and perspectives of our work.

8.3.1 Instance Based structure

8.3.1.1 Slow learner

IB algorithms require a lot of observations in order to accurately learn the concept which can make them slow learners in some scenarios. For instance, when the range of values taken by the features is $[-\infty, +\infty]$, the algorithm might never be able to accurately learn the concept. Furthermore, in the particular case of highly dimensional input space, they are likely to suffer from the "curse of dimensionality" as the number of observations necessary to accurately learn the concept will exponentially grow with the number of dimensions. A consequence of this, is that they tend to be sensible to irrelevant attributes which unnecessarily increase the dimensionality.

This is in contrast with model based methods which can speed up the learning process by making assumptions about the nature of the concept. When these assumptions prove to be close to the reality, these algorithms will be much faster to learn the concept, will use less computer memory and will also most likely obtain better prediction performances.

In order to deal with these shortfalls, it could be interesting to extend some classical frameworks such as dimensionality reduction or feature selection to the setting of data streams

subject to concept drifts. For instance, in the case of feature selection, this would mean a continuous re-assessment of the selected features as some features might become irrelevant over time.

8.3.1.2 Trade-off between running time and memory consumption

IB algorithms tend to be computationally expensive during the prediction step as they are often required to go through all the observations saved into memory in order to create a local model. When the data are streamed at a high frequency, this could prove to be an issue as they might be too slow. We have seen that it was possible to speed up this process by limiting the number of observations saved into memory. The challenge here is that, the trade-off between running time and memory consumption will impact the predictions performances of the algorithm. Indeed, in general (provided that the algorithm is capable of distinguishing the observations which should be retained into memory) the more observations allowed into memory, the better the prediction performances.

The idea of aggregating informations contained into similar observations through prototypes seems promising (Kuncheva [67]). In particular, Losing et al. [75] proposed to cluster similar observations into a long term memory. It could be interesting to work further in this direction, by maintaining an ensemble of prototypes along with some observations and to automatically recognize the regions of the feature space where the observations could be aggregated and the ones where it is necessary to keep instances. This would lead to the creation of an hybrid algorithm merging the characteristics of the instance based structure with the model based approach.

8.3.1.3 Sensitive to the distance / similarity function used

IB algorithms are also known for being sensitive to the similarity / distance function used. Throughout this thesis, we have sticked to the Euclidean distance but it is unclear whether a single similarity function would remain optimal over time in such an evolving framework. In future works, it could be interesting to compare the evolution in the performances achieved by different functions over time to see whether this is the case or not.

Encoding categorical features to numerical values is also known to be a problem for IB algorithms as it isn't always clear how the similarity between 2 different values should be assessed.

8.3.1.4 Lack of detection mechanism not always optimal

As mentioned earlier, the forgetting mechanism used in this thesis (forget observations which have been proved wrong) will always gradually forget past observations. This mechanism will be slower than a method using a detector as soon as this detector is tracking the right metric (and taking the best suited action once an alarm is triggered) or if there is prior information about the expected type of drifts.

For instance, in the case of an abrupt drift which would make the whole model out of date, learning the new concept could be significantly speed up by forgetting the old model at once. This could also reduce the error rate during the short period of time after the drift has occurred.

The above limit makes it clear that when the goal is to predict on a data stream subject to concept drifts, the most important step is to spend some time understanding the characteristics of the data stream as well as the type of drifts which can be expected. If these informations can be extracted, it will become much easier to decide which algorithm's structure will be the best suited. It could be interesting to learn to automatically estimate the type of drifts in real time and dynamically select the most appropriate adaptation strategy.

This would mean using a two steps learning algorithm. The first step would be to track several metrics at the same time (e.g. the error rate, the distribution of the observations, ...) and create a learning algorithm which would associate a particular type of drift to particular values of these metrics. The second step would be to create a second learning algorithm which would learn the best adaptation method (e.g. local update, forget the old model and retrain from scratch, ...) for the identified type of drift.

8.3.2 Cost of abstaining sometimes unclear

When it comes to decide whether abstaining should be considered or not for a given task, the answer will depend on the costs which are associated with abstaining, a good and a bad prediction. One issue is that these costs might not always be easy to know beforehand and even when they are known, they might change over time. When this is the case, it is unclear whether abstention should be considered or not.

8.3.3 Other future works

8.3.3.1 Use the abstention rate of the droplets as a drift detection mechanism

As shown in figure 6.2, when an abrupt drift occurs, the percentage of observations unclassified by the Droplets algorithm soar. It could be interesting to conduct further research to see whether this indicator could be used as a drift detection mechanism or not. The key point in this case would be to make use of the available prior information that we have. Indeed there are only 2 scenarios where the percentage of unclassified observations can increase for the Droplets algorithm:

1. Either a new part of the feature space is being explored. In this case, it could be interesting to see if it's worth learning a new model locally (not necessarily an instance based one) or to modify the algorithm's parameters to temporarily increase its learning speed.

2. Either a part of the feature space which has already been explored received conflicting informations. In this case, it could be worth speeding up the forgetting mechanism locally and delay further the prediction mechanism in this area in order to accumulate more informations.

8.3.3.2 Automatically select a proper parameter value

One of the most interesting and challenging issue which hasn't been covered in this PhD thesis is the problem of setting proper parameter's values for the adaptive models. Throughout this thesis we have argued that there would be no point in doing some prior optimization of these values as a given parameter value would only be suitable at a given time and for a given concept. However it is very unlikely that a single parameter value would be optimal throughout the lifespan of the stream.

It could be interesting to work further on the problem dynamically setting a parameter's value. This would probably involve a meta-learning algorithm which would be in charge of continuously fine tunning the parameters value.

8.3.3.3 Adaptive machine learning for quantitative trading

Finally, all these researches on adaptive machine learning algorithms could be applied to the quantitative trading universe. Few data streams are more challenging for a learning algorithm than the ones generated by the prices of assets on financial markets. They are indeed very noisy and subject to many different types of unexpected drifts. Creating an adaptive algorithm able to forecast the future value associated with the price of an asset would be an interesting challenge and a good way to test how these methods react in the most extreme real world scenarios.

Chapter 9

Appendix

9.1 Droplets for the regression setting

Here we briefly present how the Droplets algorithm could be extended to the regression setting.

The Droplets algorithm as it was presented in chapter 6 is given in Algorithm 9.1:
In order to comply with the regression setting, a few changes are necessary:

The *Predict* function could be modified to become a simple average of the values associated with the n Droplets the latest observation fell into:

$$\hat{y}_i = \frac{\sum_{j=1}^{n} C_j}{n}$$

with C_j the value associated with the j^{th} Droplet and $\hat{y}_i$ the estimated label of the latest observation.

The *Get Overlapped Droplets with conflicting labels* function should be modified to return all the overlapped droplets with a label value which is different from the real label value of the latest observation y_i.

Finally, the *Update Radii* function should be modified as shown in Algorithm 9.2. A few explanations are given thereafter:

Base computes the highest difference in absolute terms of all the values associated with the labels of the overlapped Droplets.

$Dissimilarity_k \in [0, 1]$ computes how much difference there is between the labels of Droplet k and Droplet i (the latest Droplet) relative to the local base. The higher the number the more Droplets k and i are dissimilar.

$\frac{\Delta_k}{2}$ is the value that will be subtracted of the radius of Droplet k. In particular, when $Dissimilarity_k = 0$, we have $\Delta_k = 0$ which is equal to the value it would have taken in the

Algorithm 9.1 Droplets for the classification setting

Inputs:
$R_{default}$: parameter setting the default value of the radius
N: size of the initializing set
ε: an arbitrarily small positive number

$\overrightarrow{D} \leftarrow Compute\ normalization\ constants\,(x_1, ..., x_N)$
$map_0 \leftarrow \emptyset$ //Initialize the map to empty
Foreach: x_i with $i \in \{1, 2, ...\}$
$x_i^{norm} \leftarrow Normalize\ observation\left(x_i, \overrightarrow{D}\right)$
// **Prediction step**
If $i > N$
$\hat{y}_i \leftarrow Predict\,(map_{i-1}, x_i^{norm})$
End If
// **Map update**
$y_i \leftarrow True\ Label\ of\ x_i$
$indexes \leftarrow Get\ Overlapped\ Droplets\ with\ conflicting\ labels\,(map_{i-1}, x_i^{norm}, y_i)$
$map_i \leftarrow Update\ radii\,(map_{i-1}, indexes, R_{default}, \varepsilon)$ // Algorithm 6.1
End Foreach

Algorithm 9.2 Update of the radii sizes

Inputs:
$indexes$: the identifiers of the Droplets that should be updated in the map
map_{i-1}: holds all the Droplets saved into memory at the previous time step
ε : an arbitrarily small positive number
$R_{default}$: parameter setting the default value of the radius

$Base = max\,\{C_1, ..., C_n\} - min\,\{C_1, ..., C_n\}$ $//\forall j \in \{1, ..., n\},\ C_j \in indexes$ and the set $\{C_1, ..., C_n\}$ includes C_i the value associated with the latest observation.
Foreach:$k \in indexes$
$Dissimilarity_k = \frac{|C_k - C_i|}{Base}$
$\Delta_k \leftarrow \left[\frac{R_k + R_{default} - \|x_k^{norm} - x_i^{norm}\|}{2} + \varepsilon\right].Dissimilarity_k$
$R_k \leftarrow R_k - \frac{\Delta_k}{2}$
End Foreach
$R_i \leftarrow R_{default}$
$map_{i-1} \leftarrow Add\ Droplet\ to\ map\,(x_i^{norm}, y_i, R_i)$
$map_i \leftarrow Remove\ unnecessary\ Droplets\ from\ map\,(map_{i-1}, Criteria : R \leq 0)$
Return map_i

classification setting if both classes where the same and conversely when $Dissimilarity_k = 1$, we have Δ_k equal to the value it would have had in the classification setting if the classes where different.

Bibliography

[1] Charu Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589, 2006.

[2] Charu C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. *Proceedings of the 32nd international conference on Very large data bases*, 12(15):607–618, 2006.

[3] Rakesh Agrawal, Arun Swami, and Tomasz Imielinski. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[4] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6(1):37–66, 1991.

[5] Mohammed Al-Kateb, Byung Suk Lee, and X. Sean Wang. Adaptive-size reservoir sampling over data streams. *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, 2007.

[6] Ezilda Almeida, Carlos Ferreira, and João Gama. Adaptive Model Rules from Data Streams. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013.

[7] Vineeth N. Balasubramanian, Shen-Shyang Ho, Vladimir Vovk, Harry Wechsler, and Fayin Li. *Conformal Prediction for Reliable Machine Learning*. 2014.

[8] Jürgen Beringer and Eyke Hüllermeier. Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11:627–650, 2007.

[9] Albert Bifet and Ricard Gavaldà. Kalman Filters and Adaptive Windows for Learning in Data Streams. In *Proc. of the 9th Int. Conf. on Discovery science (DS)*, volume LNAI 4265, pages 29–40, 2006.

[10] Albert Bifet and Ricard Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. *Conference: Proceedings of the Seventh SIAM International Conference on Data Mining*, 7, 2007.

[11] Albert Bifet and Ricard Gavaldà. Adaptive Learning from Evolving Data Streams. *Proceedings of the 8th International Symposium on Intelligent Data Analysis*, pages 249–260, 2009.

[12] Albert Bifet, Rafael Morales-bueno, Manuel Baena-Garcia, Jose Del Campo-Avila, Raul Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-bueno. Early Drift Detection Method. *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, 6:77–86, 2006.

[13] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, pages 139–148, 2009.

[14] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6321 LNAI, pages 135–150, 2010.

[15] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA Massive Online Analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2011. ISSN 15324435. URL `http://moa.cs.waikato.ac.nz/details/`.

[16] Avrim Blum and Avrim Blum. Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning*, 26(1):64–72, 1995.

[17] Zoran Bosnić and Igor Kononenko. Comparison of approaches for estimating reliability of individual regression predictions. *Data and Knowledge Engineering*, 67(3):504–516, 2008.

[18] Zoran Bosnić and Igor Kononenko. Automatic selection of reliability estimates for individual regression predictions. *Data and Knowledge Engineering*, 25(1):27–47, 2010.

[19] Abdelhamid Bouchachia. Incremental learning with multi-level adaptation. *Neurocomputing*, 74(11):1785–1799, may 2011.

[20] Abdelhamid Bouchachia. Fuzzy classification in dynamic environments. *Soft Computing*, 15(5):1009–1022, 2011.

[21] Abdelhamid Bouchachia and Charlie Vanaret. GT2FC: An online growing interval type-2 self-learning fuzzy classifier. *IEEE Transactions on Fuzzy Systems*, 22(4):999–1018, 2014.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees.* Chapman and Hall, 1984.

[23] Leo Breiman. Bagging predictors. *Machine Learning*, 24(421):123–140, 1996.

[24] Sebastian Briesemeister, Jörg Rahnenführer, and Oliver Kohlbacher. No Longer Confidential: Estimating the Confidence of Individual Regression Predictions. *PLoS ONE*, 7 (11), 2012.

[25] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: the Accuracy Updated Ensemble algorithm. *IEEE transactions on neural networks and learning systems*, 25(1):81–94, jan 2014.

[26] Vitor R Carvalho and William W Cohen. Single-pass online learning: performance, voting schemes and online feature selection. *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 548–553, 2006.

[27] Edith Cohen and Martin J. Strauss. Maintaining time-decaying stream aggregates. *Journal of Algorithms*, 59(1):19–36, 2006.

[28] Luc Devroye, Laszlo Gyorfi, and Gabor Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31. 2014.

[29] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 71–80, 2000.

[30] J Duarte and J Gama. Ensembles of Adaptive Model Rules from High-Speed Data Streams. *Proceedings of the 3rd International Workshop on Bigmine*, pages 1–16, 2014.

[31] Pavlos S. Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

[32] R. Elwell and R. Polikar. Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[33] William J. Faithfull and Ludmila I. Kuncheva. On optimum thresholding of multivariate change detectors. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8621 LNCS:364–373, 2014.

[34] Alan Fern and Robert Givan. Online Ensemble Learning: An Empirical Study. *Machine Learning*, 53:71–109, 2003.

[35] Yoav Freund and Re Robert E Schapire. Experiments with a New Boosting Algorithm. *International Conference on Machine Learning*, pages 148–156, 1996.

[36] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[37] Jerome H. Friedman. Multivariate Adaptive Regression Splines. 19(1):1–141, 1991. URL `http://moa.cs.waikato.ac.nz/details/`.

[38] J. Gama, R. Fernandes, and R. Rocha. Decision Trees for Mining Data Streams. In *Intelligent Data Analysis 10*, pages 23–46, 2006.

[39] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with Drift Detection. In *17th Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.

[40] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, pages 329–338, 2009.

[41] Joao Gama, Albert Bifet, Mykola Pechenizkiy, Abdelhamid Bouchachia, and Indre Zliobaite. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 2014.

[42] Alexander Gammerman and Vladimir Vovk. Hedging predictions in machine learning. *Computer Journal*, 50:151–163, 2007.

[43] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *ACM SIGKDD Explorations Newsletter*, 3(2):1, 2002.

[44] T Hastie and R Tibshirani. Discriminant adaptive nearest neighbor classification. *Pattern Analysis and Machine intelligence*, 18(6), 1996.

[45] E. Hellinger. Neue Begrundung der Theorie quadratischer Formen von unendlichvielen Veranderlichen. *Journal fur die Reine und Angewandte Mathematik*, 1909(136):210–271, 1909.

[46] G. Hulten and P. Domingos. VFML – A toolkit for mining high-speed time-changing data streams, 2003. URL `http://www.cs.washington.edu/dm/vfml/`.

[47] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, 18:97–106, 2001.

[48] Elena Ikonomovska. *Algorithms for Learning Regression Trees and Ensembles on Evolving Data Streams*. PhD thesis, 2012.

[49] Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, 2011.

[50] Ghazal Jaber. *An approach for online learning in the presence of concept changes*. PhD thesis, 2013.

[51] Ghazal Jaber, Antoine Cornuejols, and Philippe Tarroux. A new on-line learning method for coping with recurring concepts: The ADACC system. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8227 LNCS, pages 595–604, 2013.

[52] G James, D Witten, and T Hastie. *An Introduction to Statistical Learning with applications in R.* Springer, 2013.

[53] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis P Vlahavas. An Ensemble of Classifiers for coping with Recurring Contexts in Data Streams. *Ecai*, pages 763–764, 2008.

[54] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Tracking recurring contexts using ensemble classifiers: An application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010.

[55] B Kégl. Robust regression by boosting the median. *Learning Theory and Kernel Machines*, pages 1–16, 2003.

[56] Abbas Khosravi, Saeid Nahavandi, Doug Creighton, and Amir F Atiya. Comprehensive review of neural network-based prediction intervals and new advances. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 22(9):1341–1356, 2011.

[57] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, 11(May 2000):487–494, 2000.

[58] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. *Workshop Notes of the ICML/AAAI-98 Workshop Learning for Text Categorization*, pages 33–40, 1998.

[59] Teuvo Kohonen. *Self-organizing maps*, volume 3rd editio. Springer Series in Information Sciences, 2000.

[60] JZ Kolter and MA Maloof. Dynamic Weighted Majority : An Ensemble Method for Drifting Concepts. *Journal of Machine Learning Research*, 8:2755–2790, 2007.

[61] I. Koychev. Gradual forgetting for adaptation to concept drift. *Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*, pages 101–106, 2000.

[62] Ivan Koychev and Robert Lothian. Tracking drifting concepts by time window optimisation. *Research and Development in Intelligent Systems XXII - Proceedings of AI 2005, the 25th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 46–59, 2006.

[63] Georg Krempl, Myra Spiliopoulou, Jerzy Stefanowski, Indre Žliobaite, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, and Sonja Sievi. Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter*, 16(1):1–10, 2014.

[64] L I Kuncheva. Classifier ensembles for changing environments. *Multiple classifier systems*, pages 1–15, 2004.

[65] Li Kuncheva. Using control charts for detecting concept change in streaming data. Technical report, 2009.

[66] Ludmila I Kuncheva. Classifier Ensembles for Detecting Concept Change in Streaming Data: Overview and Perspectives. *Proceedings of the Second Workshop SUEMA, ECAI 2008*, (July):5–9, 2008.

[67] Ludmila I. Kuncheva and Iain A.D. Gunn. A concept-drift perspective on prototype selection and generation. *Proceedings of the International Joint Conference on Neural Networks*, 2016-Octob:16–23, 2016.

[68] Ludmila I. Kuncheva and Catrin O. Plumpton. Adaptive learning rate for online linear discriminant classifiers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5342 LNCS:510–519, 2008.

[69] Ludmila I. Kuncheva and Indre Zliobaite. On the Window Size for Classification in Changing Environments. 13(6):861–872, 2009.

[70] Yn Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. *Knowledge Discovery in Databases: PKDD 2005*, pages 108–120, 2005.

[71] Mihai M Lazarescu. A Multi-Resolution Learning Approach to Tracking Concept Drift and Recurrent Concepts. *Proc. of PRIS*, page 52, 2005.

[72] Mm Lazarescu, Svetha Venkatesh, and Hh Bui. Using multiple windows to track concept drift. *Intelligent Data Analysis*, 8(1):1–28, 2004.

[73] N. Littlestone and M.K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261, 1994.

[74] Pierre-Xavier Loeffel, Christophe Marsala, and Marcin Detyniecki. Classification with a reject option under Concept Drift: the Droplets Algorithm. In *The IEEE International Conference on Data Science and Advanced Analytics (DSAA'2015)*, 2015.

[75] Viktor Losing, Barbara Hammer, and Heiko Wersing. KNN classifier with self adjusting memory for heterogeneous concept drift. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, volume 1, pages 291–300, 2017.

[76] MA Maloof and Ryszard S Michalski. A Method for Partial-Memory Incremental Learning and its Application to Computer Intrusion Detection. In *7th IEEE Int. Conf. on Tools with Artif. Intell*, pages 392–397, 1995.

[77] T M Mitchell, R Caruana, D Freitag, J McDermott, and D Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, 1994.

[78] KP Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[79] N. Littlestone. Learn quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1987.

[80] Marlon Nunez, Raul Fidalgo, and Rafael Morales. Learning in Environments with Unknown Dynamics: Towards more Robust Concept Learners. *Journal of Machine Learning Research*, 8:2595–2628, 2007.

[81] N.C. Oza. Online bagging and boosting. *2005 IEEE International Conference on Systems, Man and Cybernetics*, 3, 2005.

[82] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100, 1954.

[83] R. Klinkenberg. Learning Drifting Concepts: Example Selection vs. Example Weighting. In *Intelligent Data Analysis 8*, pages 281–300., 2004.

[84] Pedro Pereira Rodrigues, Joao Gama, and Zoran Bosnic. Online reliability estimates for individual predictions in data streams. *Proceedings - IEEE International Conference on Data Mining Workshops, ICDM Workshops 2008*, (February 2016):36–45, 2008.

[85] Juan J Rodriguez and Ludmila I Kuncheva. Combining Online Classification Approaches for Changing Environments. *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings*, pages 520–529, 2008.

[86] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. Exponentially Weighted Moving Average Charts for Detecting Concept Drift. *Pattern Recognition Letters*, 44(0), 2012.

[87] Marcos Salganicoff. Explicit Forgetting Algorithms for Memory Based Learning. Technical report, 1993.

[88] Marcos Salganicoff. Density-Adaptive Learning and Forgetting. *IRCS Technical Reports Series.*, (197), 1993.

[89] Christophe Salperwyck, Marc Boulle, and Vincent Lemaire. Concept drift detection using supervised bivariate grids. *Proceedings of the International Joint Conference on Neural Networks*, 2015-Septe, 2015.

[90] Christophe Salperwyck, Vincent Lemaire, and Carine Hue. Incremental weighted naives bayes classifiers for data stream. *Studies in Classification, Data Analysis, and Knowledge Organization*, 48:179–190, 2015.

[91] C Sammut and GI Webb. *Encyclopedia of machine learning*. Springer, 2011.

[92] Schlimmer J and Granger R. Incremental learning from noisy data. *Machine Learning*, 1:317–354, 1986.

[93] Ammar Shaker and Eyke Hüllermeier. IBLStreams: A system for instance-based classification and regression on data streams. *Evolving Systems*, 3:235–249, 2012.

[94] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, 4:377–382, 2001.

[95] Alexander L. Strehl and Michael L. Littman. Online linear regression and its application to model-based reinforcement learning. *In Advances in Neural Information Processing Systems 2007*, pages 737–744, 2007.

[96] Ahmed Syed, Huan Liu, and Kah Kay Sung. Handling Concept Drifts in Incremental Learning with Support Vector Machines Learning. *KDD '99 Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 317–321, 1999.

[97] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, (Cisda):1–6, 2009.

[98] Marko Toplak, Rok Mocnik, Matija Polajnar, Zoran Bosnic, Lars Carlsson, Catrin Hasselgren, Janez Demsar, Scott Boyer, Blaz Zupan, and Jonna Stalring. Assessment of machine learning reliability methods for quantifying the applicability domain of QSAR regression models. *Journal of Chemical Information and Modeling*, 54(2):431–441, 2014.

[99] Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, 2006:679–684, 2006.

[100] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[101] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining Concept-Drifting Data Streams using Ensemble Classifiers. In *KDD*, pages 226–235, 2003.

[102] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing Concept Drift. *Data Mining and Knowledge Discovery*, 2015.

[103] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[104] Yair Wiener and R El-Yaniv. Pointwise Tracking the Optimal Regression Function. *Neural Information Processing Systems 25*, pages 2051–2059, 2012.

[105] Gil Zeira and Oded Maimon. Change detection in classification models induced from time series data. In *Data Mining in Time Series Databases*, number June, pages 101–125. 2004.

[106] Chicheng Zhang. Beyond Disagreement-based Agnostic Active Learning. *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, 13:442–450, 2014.

[107] I. Zliobaite. *Adaptive training set formation*. PhD thesis, 2010.

[108] Indre Zliobaite and Ludmila I Kuncheva. Determining the training window for small sample size classification with concept drift. In *ICDM Workshops 2009 - IEEE International Conference on Data Mining*, pages 447–452, 2009.

[109] Indre Zliobaite and Ludmila I Kuncheva. Theoretical Window Size for Classification in the Presence of Sudden Concept Drift. Technical report, 2010.