Novel Methods for Mining and Learning from Data Streams

Dissertation

zur Erlangung des Doktorgrades

der Naturwissenschaften

Dr. rer. nat.

der Fakultät für Elektrotechnik, Informatik und Mathematik

der Universität Paderborn

vorgelegt von

Ammar Shaker

Paderborn, Oktober 2016

Abstract

In this thesis we elaborate on knowledge acquisition and learning from non-stationary data streams. A data stream is formed by consecutively arriving data examples, whose data generating process may change in the course of time. Both the cumulative and the non-stationary nature of the data within a stream create a challenge for traditional machine learning methods.

Concentrating on adaptive supervised learning from data streams, we introduce two novel learning methods: IBLStreams and eFPT. IBLStreams is an instance-based learner that shows how instance-based learning approaches, compared to model-based approaches, are naturally incremental besides their inherent ability to adapt upon the occurrence of a concept change.

Evolving fuzzy pattern trees (eFPTs) utilize the potential interpretability of the fuzzy logic concepts in inducing compact trees; the induced trees offer the tradeoff between compact interpretable models and generalization performance. eFPTs attempt to dynamically evolve the induced tree in order to reflect any change in the underlying data generating process.

We also introduce "recovery analysis" as a new type of evaluation for adaptive supervised learners on data streams. It is an experimental protocol to assess the learner's ability to learn and recover after a concept change. The resulting recovery pattern of the learning method can be analyzed both graphically and numerically using recovery measures.

Apart from the full supervision offered in the streams studied in the previous approaches, we also consider streams of events: such a stream contains temporal events emitted from instances under observation. For a given instance, the survival time is the time this instance spends in the study until experiencing the event of interest. This survival time, however, is not always obtainable because some instances become censored by surviving until the end of the study without exhibiting the wanted event. In

this thesis, survival analysis is applied on streams of events by developing an adaptive variant of the Cox proportional hazard model. Using this model, the hazard rate, which depends on covariates associated with each instance, is dynamically modeled such that any change in this dependence is reflected as a change in the estimated hazard.

Zusammenfassung

Die vorliegende Arbeit befasst sich mit dem Erwerb von Wissen durch Lernen aus nichtstationären Datenströmen. Ein Datenstrom besteht aus einer kontinuierlichen Folge von Datenobjekten, wobei sich Eigenschaften des datengenerierenden Prozesses im Laufe der Zeit ändern können. Sowohl die Kontinuität und Dynamik als auch die Nichtstationarität von Datenströmen gehen einher mit neuen Herausforderungen für Methoden des maschinellen Lernens.

Zwei neue Methoden zum überwachten Lernen (Klassifikation und Regression) auf Datenströmen werden in der Arbeit vorgestellt: IBL-Streams und eFPT. IBLStreams ist ein instanzbasiertes Verfahren und als solches besonders gut geeignet, inkrementell zu lernen und sich adaptiv an Veränderungen des datengenerierenden Prozesses anzupassen, vor allem im Vergleich zu modellbasierten Ansätzen. Der zweite Ansatz, *evolving Fuzzy Pattern Trees* (eFPT), kombiniert Konzepte der Fuzzy-Logik mit der Flexibilität nichtlinearer Aggregationsfunktionen und der Ausdrucksstärke hierarchischer Strukturen, um interpretierbare Modelle in Form kompakter Bäume zu induzieren. Für diese sogenannten *fuzzy pattern trees* werden Lernverfahren entwickelt, die es ermöglichen, Bäume inkrementell zu lernen und an Veränderungen des Datenstroms anzupassen.

Ein weiterer Beitrag der Arbeit ist ein experimenteller Ansatz, der darauf abzielt, eine wichtige Eigenschaft von Methoden zum Lernen auf Datenströmen zu untersuchen, nämlich die Fähigkeit, auf einen so genannten *concept change* zu reagieren. Hierunter versteht man eine plötzliche oder graduelle Änderung des datengenerierenden Prozesses, der in der Regel zu einer (temporären) Verschlechterung der Prädiktionsgüte führt. Die hier vorgestellte *Recovery Analysis* ist ein Versuch, das Verhalten von Lernverfahren in solchen Situationen zu erfassen, grafisch darzustellen und zu

quantifizieren, in welchem Ausmaß und wie schnell sich das Verfahren erholt.

Schließlich geht die Arbeit über klassische Ansätze des überwachten Lernens hinaus und betrachtet sogenannte Ereignisdaten bzw. (parallele) Ereignisströme. Daten dieser Form informieren über die Zeitpunkte des Eintretens gewisser Ereignisse bzw. die Verweildauer (Überlebenszeit) bis zum Eintreten des Ereignisses. Ereignisdaten sind häufig zensiert, weil das Ereignis außerhalb des aktuellen Betrachtungshorizonts (z.B. in der Zukunft) liegt. Ein zentrales mathematisches Konzept zur statistischen Analyse von Ereignisdaten ist das der Übergangsrate (Hazardrate). In dieser Arbeit wird eine adaptive Variante des Cox Proportional Hazard Modells entwickelt, die sich zum Lernen auf kontinuierlichen Datenströmen eignet.

To Damascus, my parents and my wife

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Eyke Hüllermeier for his invaluable guidance and endless support throughout this work. Only after having the chance to work under his supervision and observing the tremendous effort he invests in his doctoral students, I realized the reason why the German term for the doctoral adviser is "Doktorvater", in that his support and care resemble that of a father. Thank You, Eyke!

I would also like to thank Prof. Jerzy Stefanowski for being the second reviewer of my thesis; it is a great honor.

I would like to thank all my former colleagues at the Knowledge Engineering & Bioinformatics (KEBI) at the Philipps-Universität Marburg and my current colleagues at the Intelligent Systems (IS) group. Besides Prof. Eyke Hüllermeier, I had the privilege to co-author papers with Prof. Jerzy Stefanowski, Dr. Edwin Lughofer, Dr. Robin Senge, Prof. Myra Spiliopoulou, Prof. Mark Last, Dr. Vincent Lemaire, Dr. Georg Krempl, Dr. Jean-Luc Bouchot, Dr. Dariusz Brzezinski, Dr. Indrė Žliobaitė, Sonja Sievi, Tino Noack, Dr. Jöran Beel, Prof. Bela Gipp and Nick Friedrich.

I appreciate the professional support I always received from Dr. Marc Strickert, the constructive and long discussions about concept change with Dr. Georg Krempl and the mathematical support with a fuzzy sarcastic flavor from Dr. Róbert Busa-Fekete.

I would like to give special thanks to my colleagues and friends Dr. Robin Senge, Dr. Thomas Fober and Dr. Bilal Alsallakh for urging me to finish the thesis, also for telling me how beautiful life would be when I finish. I am grateful to all office mates I had in Marburg and Paderborn Dr. Ali Fallah Tehrani, Patrice Schlegel and Sascha Henzgen.

I appreciate the proficient support from the *Marburger RechenCluster 2* (MaRC2), especially from Dr. Thomas Gebhardt, and also the support

I received from the *Paderborn Center for Parallel Computing* (PC^2) and its technical associate, Axel Keller. I acknowledge the kind administrative assistance from Elisabeth Lengeling and Christina Lange. Many thanks to Ludovic Larger for his support in running some experiments and visualizing their results, and to Rachel Schmidt for proofreading the final version of this dissertation.

Last but not least, I would like to thank my family in Syria. I give my sincere appreciation to my parents; their love and support were the torch that enlightened my way from the moment I decided to study in Germany. I am full of hope of a reunion with them again in peaceful and safe Syria. I owe my wife a lot for her patience, unlimited support and reminding me to sleep every night. To my cousins and friends who lost their lives in the last five years: You will always be in my mind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The information society, in which we are active members, is characterized by the wide spread of technologies that produce immense amounts of data all the time. These data emitting technologies are not only limited to expensive machines but range from cheap sensors and personal smart devices to computer clusters. Artificial intelligence in general and machine learning in particular are the fields that focus on transforming data into knowledge through: *(i)* giving computers the ability to simulate human intelligence and *(ii)* utilizing this intelligence to extract forms of knowledge from data. This learning is achieved by conventional learning techniques that learn from static data of a limited size, called a batch of data. In order to learn from this static data, it is assumed that the data is available as a whole and of a size that can be managed with the available physical resources, i.e., storage, memory and processors. It is also assumed that the data is created by stationary, not changing with time, processes.

The need for more practical solutions that can cope with the increasing amount of generated data has become more present in the last few years. It has been speculated in [168] that in the year 2020 the size of the generated data would be ten times larger than what is generated in 2013, namely reaching 44 trillion gigabytes (44 zettabytes). Learning from such huge amounts of data is clearly not feasible for the conventional learning methods, which require to learn on the entire data at once. Moreover, relaxing both assumptions of availability and stationarity of the data makes conventional learning approaches deficient to learn models that generalize the data well. This deficiency comes from two reasons: *(i)* The non-instantaneous availability of the whole data implies that the data arrives in a continuous manner and forms a stream of data, often of an infinite length. Therefore, the learner no longer has the freedom to decide what examples to learn from and how many iterations to perform. Conversely, learning from infinite data streams should be performed in a one-pass mode.

(ii) When the data generating distribution changes, i.e., learning from non-stationary environments, the learner becomes challenged by the demand of producing a model that generalizes the data well at each point in time.

1.1 Application Example

In the following, an example of adaptive learning in a changing environment is illustrated.

Imagine an intelligent system called *Interesting Topics Only* (ITO); as the name suggests, this system receives news articles and classifies them into interesting and non-interesting from its user's point of view. In order to learn the user's preferences, the system observes the topics read by the user, learns patterns from them and uses these patterns to inform the user about interesting articles in the future.

Bob, a 17 years old secondary school student, receives this software from his father who never had the time to use it. In the following, we depict a hypothetical progress of how this system could learn, adapt and perhaps react to changes in the user's interest.

1. Bob, at this age, reads only news about computer games. Thus, ITO learns to classify gaming articles as interesting and everything else as not interesting.

2. After half a year, Bob starts thinking about his future career and which university to choose after the secondary school; he begins reading articles about the different career paths and the available undergraduate programs, besides of course following the news about anything new in the gaming world. ITO now notices that what it used to predict as interesting remains interesting, whereas many articles become interesting to Bob despite being classified as not interesting. As a result, the system adapts its learned patterns to integrate the new type of articles that have become interesting for Bob. This slight change in Bob's interest represents what is later on called a "concept drift"; more specifically, it represents an incremental change in the studied concept, i.e., Bob's interest.

3. When Bob decides to take a long summer break and stay away from his electronic devices, his father decides to use the ITO system, which has until now learned only from Bob's reading patterns. The father notices that ITO never suggests articles that are interesting for him; he is not interested in gaming

and college programs but in sports and politics. Thus, the father starts reading what he really cares about, which causes ITO to notice a tremendous change in the read topics. This change requires a major revision of the learned patterns through learning new patterns about sports and politics and putting aside previously learned ones. Later on, we call this type of change a concept shift (sudden/abrupt drift), which causes any previously obtained knowledge to become obsolete.

4. Finally, when Bob comes back and decides to continue using ITO, he realizes that the system has completely forgotten about his preferences and has become more personalized to what his father reads. Therefore, Bob starts over by searching and reading the news he used to read. Similarly, the system also recognizes a new change from politics and sports to career, universities and computer games. The recent topics, however, are not ultimately new, but were already learned in the past. Thus, the system reemploys the patterns already learned in the past. The type of change, in which the recent problem resembles some of the learned ones in the past, is referred to as recurrent concept.

1.2 Learning from Data Streams

The learning settings discussed in this thesis differ from the conventional learning settings. The focus here is on learning from *data streams*, which have recently gained an increasing interest by both applied and theoretical computer science, such as information theory, statistical learning, distributed and real time systems. A data stream is a sequence of data items arriving in the course of time [70]. This data is produced by an underlying hidden process that has a high rate of data generation. Such processes include social media and networks, system event logs and sensor networks. The need to develop new knowledge discovery methodologies to learn from data streams emerges from the special properties a data stream might exhibit [14]:

- The data samples arrive with a high throughput.
- The continuous arrival of data samples gives them a temporal order. This order should be respected and taken into consideration, i.e., any shuffling or changing of the order of the elements would corrupt the temporal relation between them.
- Data streams are by definition massive data, if not even of a limitless length. This property makes a full storage of the data stream inconceivable.

- A single arriving data sample remains transient, i.e. it is discarded after being processed by the learner, unless a relatively small fraction of the data is explicitly stored for further inspection.

In order to produce a valid predictive model under such constraints, Domingos and Hulten [57] describe a number of properties that an ideal stream mining system should exhibit:

- The learning system should use only a limited amount of memory for processing each newly incoming data sample, no matter how enormous the already seen data is.
- The time to process a single sample should be short and ideally constant.
- The stream can be observed only once, i.e., multiple scans of the streams cannot be realized.
- The incrementally induced model should be equivalent to the model that would have been obtained through conventional learning (on all data samples seen so far).
- The learning algorithm should react to concept drift (i.e., any change of the underlying data generating process) in a proper way and maintain a model that always reflects the current concept.

1.3 Incremental, Adaptive and Evolving Learning

Learning on data has gained importance in different research areas of artificial intelligence. This has led to its independent appearance under different terminologies based on the nature of the learning methods they emerged in. In the following, we list the different types of learning and the motivation behind their emergence.

Incremental learning: Incremental learning algorithms try to accomplish a learning task in an accumulative way. Giraud-Carrier [79] defines a task whose training examples arrive over time to be an incremental task. A learning algorithm is called incremental if it produces for the sequence of arriving data examples $e_1, \cdots, e_n$ a sequence of models $\mathcal{M}_1 \cdots, \mathcal{M}_n$, such that any model $\mathcal{M}_{i+1}$ is only obtained from the current observation e_i and the previous model $\mathcal{M}_{i-1}$. Hence, incremental learners have the memorylessness property [13]. Motivated by this definition, the decision tree induction algorithm ID5 [169] is an incremental version of the ID3 algorithm [129],

due to the way ID5 updates the newly induced tree, by simply updating the counters kept in the nodes after receiving a new observation.

Adaptive learning: The demand for adaptive learning comes from the need to update the induced models either upon the arrival of a more recent data [138], or because of a change in the underlying data generation process, which triggers an update in the learning strategy as in FLORA [179]. Thus, an adaptive learner, besides being incremental by nature, should also exhibit the awareness to discover any potential change in the concept to be learned. Surprisingly, the perceptron learning algorithm by Rosenblatt [134] would be the first work satisfying the definition of adaptive learners, due to its iterated adaptation of the learned coefficient vector, thus diminishing the importance of old observations and increasing the representation of recent ones. Consequently, any change of the linear decision boundary of a stream data is guaranteed to be fit.

Evolving learner: The exposure to the same type of problems on data streams has led to the emergence of the so-called evolving intelligent systems in the research field of computational intelligence [10, 11]. Soft computing, a paradigm realized in the form of fuzzy systems, and evolutionary algorithms, realized as genetic algorithms [80], and genetic programming [101] have incorporated the aspect of evolving the induced systems with time. The term "evolving" should not be confused with the term "evolutionary". Evolutionary systems [160] imitate the development of a population in which the properties of each generation are crossed over, mutated and passed to the new generation. Only individuals that fit their environment remain to produce the next offspring, in a way that mimics the natural selection and the survival of the fittest [44, 162], which takes place in the real world. Evolving systems, on the other hand, are more concerned about the adaptation of the systems' structure and parameters in a non-stationary environment. This learning paradigm mimics the development of an individual in his environment, by gradually learning from his surrounding. The concept of evolving systems was first made known in [92, 93] for artificial neural networks and in [6, 8] for fuzzy rule-based systems. The notion of evolving fuzzy systems is used in the field of fuzzy research for learners employing the concepts of fuzzy logic [12, 110, 11, 111]. Besides being incremental and adaptive, evolving learners allows for structural adaptation whenever this adaptation improves the performance.

Finally, it is worth mentioning that online learning (OL) is one of the theory-oriented research areas in machine learning; online learning allows the sequence of training data to be generated by a deterministic, stochastic or even by an adversarial process. In the adversarial case, the data generating process is aware of the learners

decisions and chooses the true outcome that makes the learner's prediction incorrect [158], as in the case of the electronic spamming systems that adapt to spam filters.

In this thesis, we mainly adopt the terminology of both adaptive and evolving learning.

1.4 Contribution and Outline of the Thesis

The focus of this thesis is on methods for supervised learning on streaming data. In spite of a significant amount of existing work on this topic, there is arguably a number of "gaps" to be filled and open questions to be addressed.

First, while model-based approaches to machine learning, such as decision tree induction, have been explored quite extensively in the context of data streams, comparatively little attention has been paid to the paradigm of instance-based learning so far. In light of a number of potential advantages of the latter, this is indeed somewhat surprising. In particular, thanks to the "lightweight" nature of instance-based learning and its conceptual simplicity, instance-based concept representations are presumably much more flexible and much easier to adapt to changes of a dynamic environment. Therefore, we develop and implement an instance-based learner called IBLStreams that is applicable to both classification and regression problems.

In search of experimental procedures that could be used to validate our conjecture and to systematically compare model-based and instance-based methods with regard to their ability of adapting to concept drift, we found that existing procedures commonly used in the field are not fully appropriate for this purpose. Therefore, we propose a new type of experimental analysis, called recovery analysis, which is aimed at assessing the ability of a learner to discover a concept change quickly, and to take appropriate actions to maintain the quality and generalization performance of the model.

Fuzzy machine learning takes advantage of tools and techniques from fuzzy logic to develop methods for machine learning in general and learning on data streams in particular. As for the latter, however, the focus has almost been exclusively on the problem of regression so far. In this thesis, we therefore elaborate on the suitability on fuzzy methods for classification on data streams. More specifically, we develop an evolving version of so-called fuzzy pattern tree learning, which has recently been introduced in [82, 146] as a promising alternative to fuzzy rule models.

Finally, we address so-called survival analysis (also known as event history analysis) as another data analysis problem that, despite its great popularity in applied

statistics, has not been considered in the context of data streams so far. Survival analysis is about the analysis of temporal "events" or, more specifically, questions regarding the temporal distribution of (duration between) the occurrence of events and their dependence on covariates of the data sources. To this end, we develop an incremental, adaptive version of survival analysis, namely an adaptive variant of a model that is closely related to the well-known Cox proportional hazard model.

The thesis is outlined in the following way:

- *Chapter 2: Background.* This chapter offers an introduction to supervised learning, with focus on learning from data streams; it also presents an overview of the related work and its development.
- *Chapter 3: Instance-Based Classification and Regression.* This chapter introduces a nonparametric approach for classification and regression tasks on non-stationary streams. This approach is an extension of IBL-DS [17], which introduces three important factors that have to be considered when maintaining a case base of observed examples. These factors are the temporal relevance, the spatial relevance and the consistency aspect of a training example. Parts of this chapter were published in [152, 151].
- *Chapter 4: Evolving Fuzzy Pattern Trees.* This chapter discusses the role of fuzzy logic in data-driven systems on data streams. Here, we present an evolving variant of the fuzzy pattern trees which we introduced in [156, 157].
- *Chapter 5: Survival Analysis on Event Streams.* This chapter introduces a survival analysis method for streams of events. As a proof of concept, we apply the proposed method on two types of streams: the stream of occurring earthquakes and the stream of Twitter[1] data. The work presented in this chapter is published in [150, 153, 155].
- *Chapter 6: Recovery Analysis for Adaptive Learning.* This chapter discusses assessing the learning capability of an adaptive learner; it suggests a new type of performance comparison based on the learner's ability to recover after suffering from a concept drift. Parts of this chapter were published in [154, 149].

The thesis is concluded in Chapter 7, in which we summarize the different methods developed in the thesis. As a supporting material we add the following appendixes:

[1] `http://www.twitter.com`, accessed on October 9, 2015

- *Appendix A: Methods.* This appendix gives a brief introduction to the approaches used for comparison throughout the thesis.
- *Appendix B: MOA.* This appendix introduces the MOA framework, used for performing the majority of our experiments.
- *Appendix C: M-Tree.* Here, we introduce the index structure M-Tree.
- *Appendix D: Data sets.* This appendix explains the utilized real, synthetic and event data streams, in the thesis. It also explains how the different types of concept change can be simulated.
- *Appendix E: Incremental Statistics.* A set of incremental and adaptive descriptive statistical measures are derived in this appendix.

Chapter 2

Background

Learning from data streams extends the research area of data mining and machine learning by forming methods, parallel to the conventional learning methods, that cover the same topics in the streaming setting.

As previously explained in Section 1.2, streams of data prevail when data sources cease to generate small amounts of data that can be handled as a single batch and begin to generate continuous streams of data that are: *(i)* of a high throughput, *(ii)* with an implicit temporal order, *(iii)* possibility of limitless length and *(iv)* exhibiting a potentially changing concept, see [14].

The immense size of the data combined with the changing nature of the learned concepts lead to the emergence and development of new topics invented to tackle various learning tasks under the challenge of streamed data. These advancements lead to the enrichment of machine learning.

Learning from data streams has been considered for many learning tasks such as supervised learning [67, 1, 69, 106], non-supervised learning [49, 5, 122] and frequent itemset mining [36, 25, 36, 37]. Many supervised learning methods have been adapted for the streaming setting such as soft computing [10, 11, 111], active learning [173, 185, 161] and ensemble methods [185, 34, 137, 175].

This thesis mainly develops methods that work in the supervised setting, which makes other settings less relevant to the discussed topics here, except the work introduced in Chapter 5 which adopts a different type of supervision. For this reason, we present an overview in Section 2.2 of the related work, with the focus on the supervised learning methods on data streams. The work in Chapter 5 develops a survival analysis method on data streams; this chapter presents its own statistical background in Section 5.2.

Section 2.1 introduces the definition of a machine learning task and presents the different types of learning problems. Next, the supervised learning model on data

streams is motivated in Section 2.2 and the different types of concept changes are discussed in Section 2.3. In Section 2.4, an overview of change detection methods is presented.

2.1 Machine Learning

A non-formal definition of machine learning by Thomas M. Mitchell [119] declares it as "The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience." In other words, for well-defined problems such as matrix multiplication, one can simply design an algorithm that transforms the two inputs into one output representing the result of the needed product. Different algorithms may compete in their complexity and performance, however, they still deliver the same output by definition.

In contrast to the previous problem, the task of recognizing whether an image depicts an adult male or a female, which is usually simple for humans to solve, is very hard to transform into a computer program that is able to perform this recognition. Machine learning aims at granting computers the ability to learn and find suitable algorithms to solve such problems by only seeing examples and solutions of the studied problems. The learner should extract knowledge from the presented examples, in means of the best suitable model, with the aim of producing correct solutions for similar problems.

For a more formal definition of learning, we use another quote from Thomas M. Mitchell [119]. "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." The direct projection of this definition on the aforementioned example maps the task T to the differentiation between male and female figures in an image. The experience E is the presented example image, annotated with the gender of the appearing person. Finally, the target is to compute predictions, for new images, that are as correct as possible, i.e. to achieve a high classification rate as a performance measure.

Learning tasks

Machine learning focuses mainly on two tasks, which vary depending on the availability of a supervision:

Supervised learning: The learner in the supervised setting is expected to model the relation between an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, based on a set of

observed examples. The input space serves the role of describing and representing the experiences to learn from. The output space contains the different values the prediction can take. A supervised learner tries to model the dependencies between samples drawn from the space $\mathcal{X}$ and their observed target values from $\mathcal{Y}$, i.e. inducing a model $\mathcal{M}\colon \mathcal{X} \to \mathcal{Y}$.

In the previous example, the input space $\mathcal{X}$ can be the space representing the $n \times n$ pixels in a picture, $\mathcal{Y}$ is a dichotomy containing only two values: male and female. This type of problem is also called classification whenever the prediction space contains categorical values; the supervised learning task is called regression when the target space is numerical.

Unsupervised learning: In this setting, the task is to find patterns and relations in the given data without any awareness of an output, neither in the form of a category nor in the form of a numeric target value. The most practiced approach of unsupervised learning is clustering. The aim of clustering is to find groupings of the input data based on different criteria such as their similarity as in k-means [112], density as in DBSCAN [60], etc. The absence of an objective evaluation criterion of the found clusters makes clustering of a subjective nature, as stated by [61]: "clustering is in the eye of the beholder."

The discussed learning scenarios, most of the time, assume the availability of the data before initializing the learning process. This assumption enjoys the benefit of having the whole data set stored on a permanent medium, and fitting it as a whole in the available memory. This assumption grants the learning process the advantage of accessing the data multiple times, probably through a number of scans. The resulting induced model(s) are then validated in order to perform the model selection. Finally, the selected model is deployed and tested.

2.2 Supervised Learning from Data Streams

This section begins by formally introducing the supervised learning in the batch mode. Thereafter, we present how the supervised learning should be modeled in the streaming setting.

Supervised learning in the batch setting

In supervised learning, the data space $\mathcal{Z}$ is composed of the input space $\mathcal{X}$ and the output space $\mathcal{Y}$, i.e., $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. The learning algorithm is then applied to induce the model $\mathcal{M}\colon \mathcal{X} \to \mathcal{Y}$, which captures the dependency between the input space

and output space. The induction process is carried on a set of training examples $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N} \subset \mathcal{Z}$, whose examples are assumed to be independently sampled and identically distributed. A learning problem is a classification problem when the space $\mathcal{Y}$ is a set of categories, e.g., the supervised problem on the output space $\mathcal{Y} = \{c_1, \ldots, c_k\}$ defines the k-class classification problem.

With $\mathcal{H}$ being the set of all possible hypotheses inducible by a learning algorithm, the risk of adopting the hypothesis $h \in \mathcal{H}$ is the expected committed error

$$R(h) = \mathrm{E}[l(h(\boldsymbol{x}), y)] = \int l(h(\boldsymbol{x}), y)\, d\mathbf{P}(\boldsymbol{x}, y) \ ,$$

such that $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function which quantifies the committed failure when predicting $h(\boldsymbol{x})$ instead of y. The expectation, in the expected risk, is taken with respect to an underlying probability measure $\mathbf{P}$ on $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. This probability measure formally specifies the data generating process. A learning algorithm would optimally aim at finding the hypothesis h^* at which the risk is at its minimum, i.e., $h^* = \mathrm{argmin}_{h \in \mathcal{H}} R(h)$.

However, h^* can not be computed mainly because of the unknown probability measure $\mathbf{P}(\boldsymbol{x}, y)$. Therefore, this hypothesis can be approximated by $\hat{h}$, the hypothesis that minimizes the empirical risk (the average loss committed on the training set)

$$R_{emp}(h) = \frac{1}{N} \sum_{i=1}^{N} l(h(\boldsymbol{x}_i), y_i) \ .$$

The empirical risk minimization, as an induction principle, produces

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} R_{emp}(h)$$

as a result of minimizing the loss on the training data. This approach might fail in producing a hypothesis that generalizes well as a result of either overfitting the training data, which becomes more probable for hypothesis spaces of large capacities, or underfitting when $\mathcal{H}$ is very small.

Other induction principles, such as the structural risk minimization (SRM) [170] balances between the empirical risk and the complexity of a hypothesis class. The minimum description length (MDL) principle [132] offers the tradeoff between the empirical risk and the length of the hypothesis.

Practically, a validation set, sampled according to the same probability measure $\mathbf{P}$, can be used to produce a more accurate estimate of the generalization performance; the performance on the validation set can be used for model selection in which the

complexity and the parameters of the model can be chosen. A practical realization of SRM can be achieved using regularization in which both the loss and the complexity of the model are minimized.

The choice of the loss function l depends mainly on the learning problems. The *0-1 loss* is commonly used for classification problems (as long as the classes are balanced and the cost of a misclassification is the same for all classes), and it is defined for the example $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$ as

$$l(h(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } h(\boldsymbol{x}) = y \\ 1 & \text{if } h(\boldsymbol{x}) \neq y \end{cases} \quad . \tag{2.1}$$

When the output space $\mathcal{Y}$ is a totally ordered set, i.e., equipped with the strict total order relation $\prec \subseteq \mathcal{Y}^2$, the problem is then called ordinal classification. This total order relation allows us to write output space as $\mathcal{Y} = \{c_1, \ldots, c_n\}$ where $\forall i, j \in \{1, \ldots, |\mathcal{Y}|\} : c_i \preceq c_j \Leftrightarrow i \leq j$. In the ordinal case, the 0-1 loss can be replaced by the absolute error loss $\ell(c_j, c_i) = |i - j|$ when predicting c_j instead of c_i for the example $(\boldsymbol{x}, c_i) \in \mathcal{X} \times \mathcal{Y}$.

The supervised problem becomes a regression problem when $\mathcal{Y} = \mathbb{R}$; for regression problems, the *squared loss*

$$l(h(\boldsymbol{x}), y) = (h(\boldsymbol{x}) - y)^2 \ , \tag{2.2}$$

is often utilized.

Supervised learning in the streaming setting

This setting considers that an algorithm $\mathcal{A}$ is learning on a time-ordered stream of examples $S = (\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \ldots)$, where each data item $\boldsymbol{z}_t$ is a tuple $(\boldsymbol{x}_t, y_t) \in \mathcal{X} \times \mathcal{Y}$.

At every time point t, the algorithm $\mathcal{A}$ is supposed to offer a predictive model $\mathcal{M}_t : \mathcal{X} \to \mathcal{Y}$ that is learned on the data seen so far, i.e., on the sequence $S_t = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_t)$. Given a query input $\boldsymbol{x} \in \mathcal{X}$, this model is used to produce a prediction

$$\hat{y} = \mathcal{M}_t(\boldsymbol{x}) \in \mathcal{Y} \ ;$$

the committed loss by this prediction is measured by $l(\mathcal{M}_t(\boldsymbol{x}), y)$, where the loss function l is chosen based on the type of the learning problem.

If the algorithm $\mathcal{A}$ is truly incremental, it will produce $\mathcal{M}_t$ solely on the basis of $\mathcal{M}_{t-1}$ and $\boldsymbol{z}_t$, that is, $\mathcal{M}_t = \mathcal{A}(\mathcal{M}_{t-1}, \boldsymbol{z}_t)$. In other words, it does not store the entire sequence of previous observations $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}$. Many algorithms, however, store at least a synopsis of the most recent observed data, which can then be used for model

adaptation. In any case, the number of observations that can be stored is normally assumed to be finite, which excludes the possibility of memorizing the entire stream. On the other hand, a batch learner $\mathcal{A}_B$ would produce the model $\mathcal{M}_t$ on the basis of the complete set of data $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t\}$. Although $\mathcal{A}$ and $\mathcal{A}_B$ have observed the same data, $\mathcal{A}_B$ can exploit this data in a more flexible way. Therefore, the models produced by $\mathcal{A}$ and $\mathcal{A}_B$ will not necessarily be the same.

Next, driven by the aforementioned difficulties while learning from data streams, we show how the supervised learning should be performed. The learning process is optimally defined as a pipeline of modular tasks, in such a way that serves the purpose of the incremental/adaptive learning, see Figure 2.1. The incremental learning process [76] is triggered at the arrival of each new instance; this process is characterized by the following steps:

- **Observation:** Observing a data sample $\boldsymbol{x}$ for which the target value y is not known yet.
- **Prediction:** At this step, the learned model is applied to make the prediction $\hat{y}$, based on the data seen in the past and realized in the learned model's structure and parameters.
- **True outcome:** The true target value y may or may not be observed in the future. Despite the uncertainty of observing y, this observation could arrive after a time delay of a variable length, depending on the task the adaptive system is trying to solve. The existence of the observation $\boldsymbol{x}$ and its corresponding target value y, constitutes a data sample $(\boldsymbol{x}, y)$ for further processing.
- **Loss estimation:** The observed data sample $(\boldsymbol{x}, y)$ is a good candidate to judge the induced model's performance. This is achieved by calculating the committed loss by predicting $\hat{y}$ instead of y. The loss is a measure whose definition varies depending on the learning task.
- **Drift detection:** Ideally, change detection methods try to detect a change in the data generating distribution; this requires the approximation of the entire probability distribution. Instead, many detection methods try to cumulatively estimate statistical moments or some of the assumed data generating distribution's parameters upon observing the stream of $(\boldsymbol{x}, y)$ pairs. A change in the estimated moments, or parameters, over time supports the learning process with an indication of a change in the data generating process. This detection strategy informs the learning process about any change in the observed data.

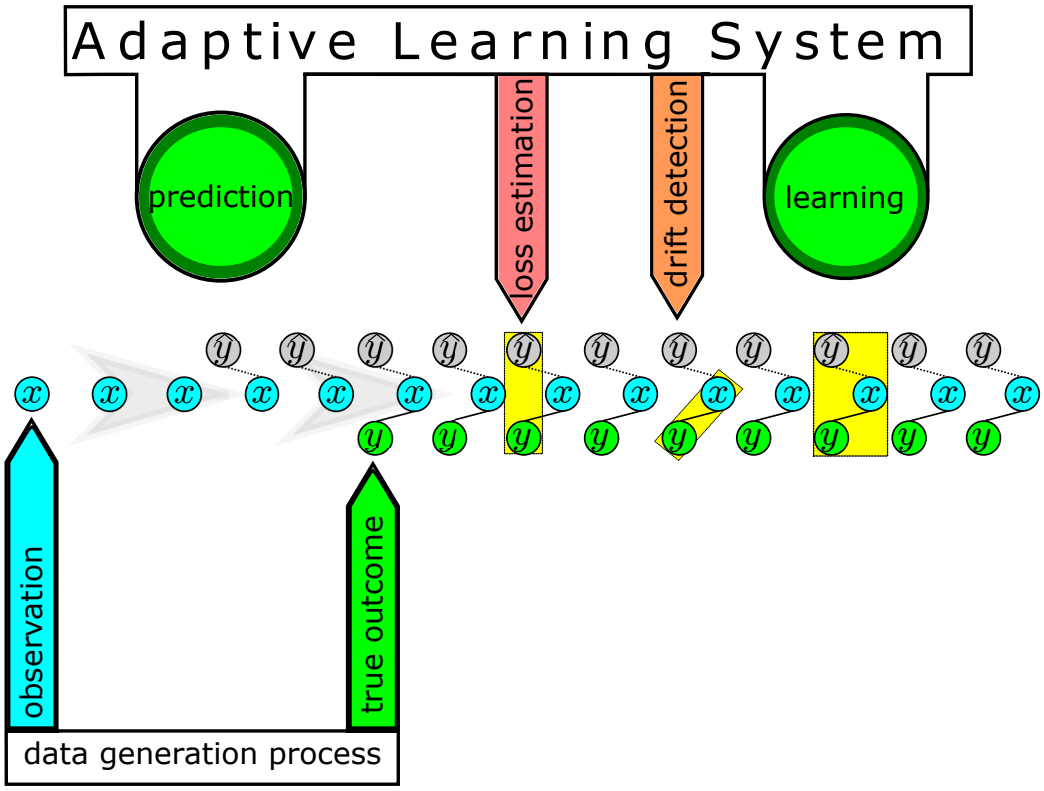


Figure 2.1: The different steps characterizing an adaptive learning system.

- **Learning:** Finally, the learner has the chance to incrementally/adaptively learn from the fully observed new sample $(\boldsymbol{x}, y)$.

The evaluation of an evolving classifier learning from a data stream is clearly a non-trivial issue. When compared to standard batch learning, single-valued performance cannot represent the properties of the learned model in a non-stationary environment. Two different evaluation scenarios can be applied on the previously defined learning process:

- **Holdout evaluation**
  The holdout technique is a generalization of the cross-validation technique commonly used in the batch mode. In this technique, each new example is used either in the testing or the learning phase. The two phases are interleaved as follows: the model is trained incrementally on one block of instances and then evaluated (but no longer adapted) on the next block of instances, then again trained on the next block and tested on the subsequent block, and so forth. The training and the testing blocks do not need to be of the same size. The holdout

error at the time point i is

$$err_i = \frac{1}{M} \sum_{j=i-M}^{i} l(h(\boldsymbol{x}_j), y_j) \ , \tag{2.3}$$

where M is size of the testing block.

- **Test-then-train evaluation**
  While the holdout technique uses an instance either for training or testing, test-then-train utilizes each instance for both: First, the model is evaluated on the instance, and then a single incremental learning step is carried out. This technique can be applied to compute different evaluation measures: the prequential error, the prequential error with a fading factor or the error on a sliding window.

  The prequential measure [48] updates the prediction error cumulatively on each new observation. At the ith time point, the number of committed errors is

$$S_i = \sum_{j=1}^{i} l(h(\boldsymbol{x}_j), y_j) = l(h(\boldsymbol{x}_i), y_i) + S_{i-1} \ . \tag{2.4}$$

  Hence, the average error in $err_i = \frac{1}{i} S_i$. The prequential error using a fading factor (exponential weighting) [74] is

$$err_i = \frac{S_i}{B_i} = \frac{l(h(\boldsymbol{x}_i), y_i) + \alpha S_{i-1}}{1 + \alpha B_{i-1}} \ , \tag{2.5}$$

  where $\alpha \in]0, 1[$ is the forgetting factor and $B_1 = 1$.

  The advantage of applying the test-then-train evaluation scenario is that all instances are employed for both testing and training, without any loss of information that is being kept in the holdout instances as in the previous evaluation method. A holdout block does not only hide vital information, that supports in updating the trained model, but also causes a delay in detecting an occurring concept change during the holdout phase.

The so far discussed evaluation methods assess the predictive performance of the supervised learner. This performance depends only on how well the induced model generalizes and how accurate its predictions are. There are other non-functional properties that need to be considered by the learner, without which learning would not be feasible in the streaming setting. These requirements include the run time required to learn a model, compute predictions and update the learned model. Moreover,

the consumed resources such as the needed memory to learn, read/write operations, network communications and CPU usage (CPU time) are important factors to estimate the cost of each method. Bifet et al. [26], employ the RAM-Hours evaluation measure of the consumed memory by an adaptive learner. In this thesis however, we only focus on the predictive performance of the learners and on their ability to recover their original performance after a concept change. The main reason for not considering other cost measures is that almost all learning methods offer the tradeoff between performance on one hand and runtime and utilized resources on the other hand.

2.3 Concept Change over Time

In the conventional supervised learning setting, the data generating process (the probability measure $\mathbf{P}$ on $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$) is assumed to be stationary; it is also assumed that examples are independently sampled according to $\mathbf{P}$.

Under the assumptions of stationarity and independence, each new observation $\boldsymbol{z}_t$ is generated at random according to $\mathbf{P}$, i.e., the probability to observe a specific $\boldsymbol{z} \in \mathcal{Z}$ is given by[1]

$$\mathbf{P}(\boldsymbol{z}) = \mathbf{P}(\boldsymbol{x}, y) = \mathbf{P}(\boldsymbol{x}) \cdot \mathbf{P}(y \,|\, \boldsymbol{x}) \tag{2.6}$$

$$= \mathbf{P}(y) \cdot \mathbf{P}(\boldsymbol{x} \,|\, y) \ . \tag{2.7}$$

$\mathbf{P}(y)$ represents the probability distribution in the output space, or the so-called prior. The conditional probability $\mathbf{P}(y \,|\, \boldsymbol{x})$ is the posterior probability, which is the probability of observing y after observing $\boldsymbol{x}$.

Giving up the assumption of stationarity (while keeping the one of independence), the probability measure $\mathbf{P}$ generating the next observation may possibly change over time. Formally, we are not dealing with a single measure $\mathbf{P}$, but with a sequence of measures $(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \ldots)$, assuming that $\boldsymbol{z}$ is generated by $\mathbf{P}_t$. One speaks of a *concept change* if these measures are not all equal [94]. Gama et al. [76] present in their survey paper a coherent taxonomy of the different types of concept change and maps them to the change of the underlying distributions. Thus, we distinguish three types of concept changes:

- Real concept change is defined by the change in the posterior $\mathbf{P}(y \,|\, \boldsymbol{x})$[2].

[1] We slightly abuse notation by using the same symbol for the joint probability and its marginals.

[2] This type of change is known as concept shift in [139], despite the fact that recent works preserve the term shift to indicate the rate of change, and is called a conditional change in [77].

- Virtual concept change is the change of the data's probability $\mathbf{P}(\boldsymbol{x})$ in (2.6), i.e., the distribution of the inputs [178]. This change may or may not cause a change in the concept, i.e., the conditional distribution $\mathbf{P}(y \mid \boldsymbol{x})$ [167, 178]. A listing of the different definitions of virtual changes in the literature is presented in [76] as:
  - The term "virtual drift" was initially defined in [178] as a phenomenon caused by the insufficient knowledge about the data distribution and not by a change.
  - A virtual concept change makes the revision of the induced model necessary due to the change in the data distribution, as proposed by [167].
  - A drift is referred to as virtual if its target concept remains unaffected [51].
  - A virtual drift is called a sampling shift in [139], a temporary drift in [104] and a feature change in [77].
- Global and local concept change [167] are properties characterizing the scale in which the change occurs, independent of its nature (real or virtual). Unlike global drifts, local drifts occur in a subspace or a partition of the input space $\mathcal{X}$.

The problem of concept change over time has a second important criteria, namely the rate of change, at which the new concept appears and replaces the previously observed concept. This thesis relies on the terminology defined in [171], which classifies the different types of concept change into categories based on the pattern at which the new concept replaces the old one, as presented in Figure 2.2:

- Concept shift refers to the abrupt/sudden change in the generating process, that is the probability measure $\mathbf{P}_t$ is very different from $\mathbf{P}_{t-1}$. Hence, the new concept has to be learned and any learned concept becomes out of date.
- Gradual drift refers to a progressive change of the data generating process, such as the change from $\mathbf{P}_1$ to $\mathbf{P}_2$. A gradual drift starts at time t_1 and ends at time t_2 when the measures $\mathbf{P}_1$ and $\mathbf{P}_2$ are sampled at the time $t \in [t_1, t_2]$ with probabilities $\lambda(t)$ and $1-\lambda(t)$, respectively. The function $\lambda(t)$ is a monotonically decreasing sample probability, with $\lambda(t) = 1$ for any $t \leq t_1$ and $\lambda(t) = 0$ for any $t \geq t_2$.

  This gradual change occurs through the increase of the rate at which the second measure $\mathbf{P}_2$ is applied, accompanied by the simultaneous decrease of the rate of

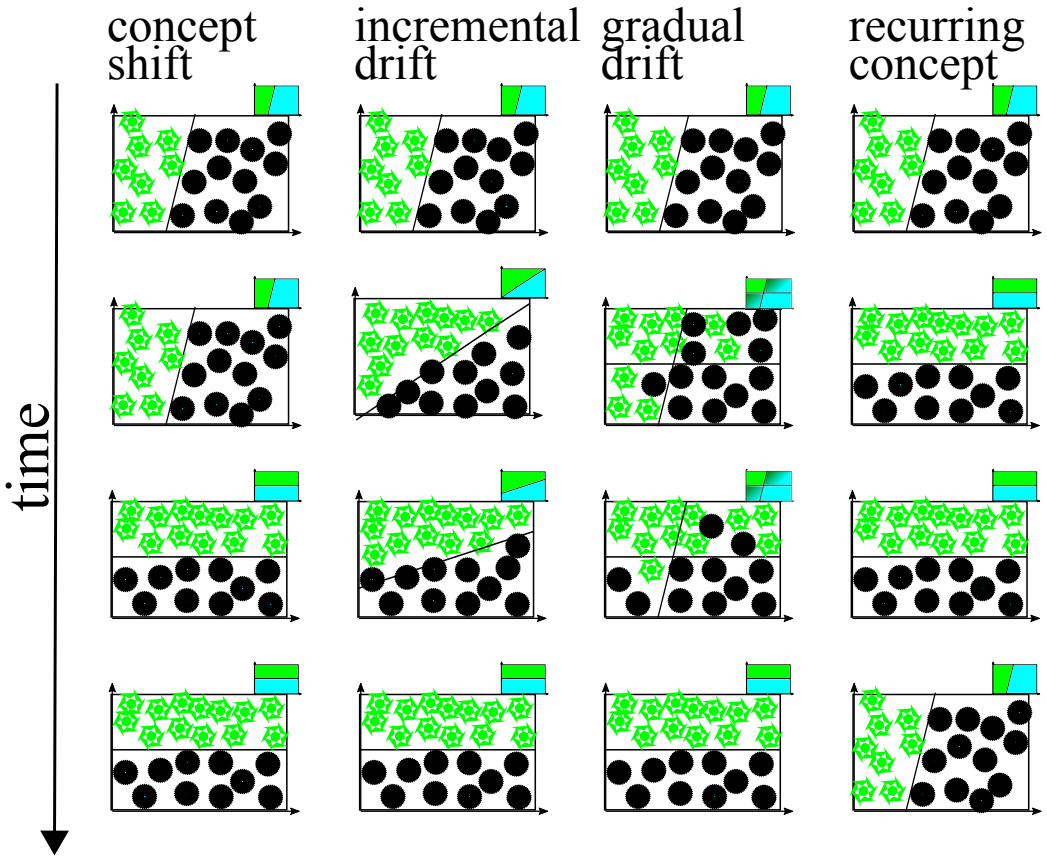


Figure 2.2: The different types of concept change over time.

applying the first measure $\mathbf{P}_1$. As a result, aged examples may remain partially consistent with the current measure. The gradual change, from $\mathbf{P}_1$ to $\mathbf{P}_2$, occurs by having samples from the first measure $\mathbf{P}_1$ with a probability close to 1 at the beginning of the drift, this probability decreases monotonically until it vanishes at the end of the drift, causing the measure $\mathbf{P}_2$ to be the dominant one.

- Incremental drift refers to the smooth transition between two probability measures, e.g., change from $\mathbf{P}_{t_1}$ to $\mathbf{P}_{t_2}$. An incremental drift occurs at time t_1 and ends at time t_2 when the intermediate measures $\mathbf{P}_{t_1+1}, \mathbf{P}_{t_1+2}, \ldots, \mathbf{P}_{t_2-2}, \mathbf{P}_{t_2-1}$ are sampled at the times $t_1+1, t_1+2, \ldots, t_2-2, t_2-1$, such that the distributions $\mathbf{P}_t$ and $\mathbf{P}_{t-1}$ are statistically indifferent. As an example, one could imagine the measures to be the Gaussian distributions $\mathbf{P}_{t_1} \sim \mathcal{N}(\mu_1, \sigma_1^2)$, $\mathbf{P}_{t_2} \sim \mathcal{N}(\mu_2, \sigma_2^2)$. The incremental change occurs when generating the data from intermediate distributions, by shifting the mean slightly from μ_1 to μ_2 and the variance from σ_1^2 to σ_2^2.

- Recurring concept is the concept that occurs at least once after its disappearance from the data.

Finally, it is worth mentioning that Webb et al. [176] propose the first attempt to characterize the different types of concept change in a formal framework.

2.4 Change Detection Methods

In order to meet the requirements of learning from non-stationary data streams, a learning algorithm needs to be aware of any change in the data generating process that could invalidate the learned model. Such an awareness could be achieved by either *(i)* directly inspecting the arriving data and checking them for a change or *(ii)* by observing how the performance of the learned model changes in the course of time and triggering a change whenever this performance significantly deteriorates. In the following, we describe the most applied change detection methods, as reviewed in the surveys [76, 106, 53]:

Classifier-dependent detection methods

This type of detectors uses a change detection strategy that compares the performance of the current model with the best achieved performance up to now, under the assumption that the best performance corresponds with no change in the target concept.

- Statistical process control (SPC) is a family of statistical methods that can be applied to monitor and control processes, such as industrial processes, in order to keep them in an optimal sustainable operation mode. Different variations of this method have been adapted and applied for detecting drifts [97, 72, 15]. Drift detection method (DDM), proposed by Gama et al. [72], is one of the early adaptations of SPC for detecting drifts. For a stream of instances (x_i, y_i) and their assigned predictions $\hat{y}_i$, the zero-one loss function l computes the disagreement between the true class and the prediction, i.e., $l_i = \mathbb{I}(y_i \neq \hat{y}_i)$, where $\mathbb{I}$ is the indicator function. The committed error on one example, with the binary values it takes, forms a Bernoulli trial. As a result, the number of errors committed on a sample of n instances follows the binomial distribution, provided they are independent. For the ith sample, p_i is the probability of being assigned the wrong class with the standard deviation $\sigma_i = \sqrt{p_i(1-p_i)/i}$. These values are incrementally updated on the observed stream. DDM keeps track of the best observed performance by storing the variables p_{min} and σ_{min}. These variables are updated ($p_{min} = p_i$ and $\sigma_{min} = \sigma_i$) whenever $p_i + \sigma_i < p_{min} + \sigma_{min}$ is satisfied after observing the ith example. The confidence interval $p_i \pm z\sigma_i$, such that z depends on the desired confidence level α, helps in defining the following three states for change detection:

  - In-control state is the state at which the prediction performance does not seem to exhibit any change. The system is in this state as long as $p_i + \sigma_i < p_{min} + 2 \cdot \sigma_{min}$.
  - Out-of-control state is the state at which the error has significantly increased, which requires a suitable model adaptation in order to recover the drop in performance. The system is in this state whenever $p_i + \sigma_i \geq p_{min} + 3 \cdot \sigma_{min}$.
  - Warning state is the state at which the error has increased without reaching the critical level. This state occurs when the system's performance lies between the two previous states.

- Early drift detection method (EDDM) [15] builds upon the previously discussed DDM method in order to shorten the temporal gap between the drift and its detection. The problem with the previous method is that the more we see data the more resistant becomes p_i towards slow and gradual changes. EDDM on the other hand considers the number of correct predictions between two

misclassification cases instead of the error rate. This method uses $p_i^{'}$ as the average number of correct predictions between two wrong predictions and $\sigma_i^{'}$ is its standard deviation. Similar to the DDM, the system is in the warning level when $(p_i^{'} + 2 \cdot \sigma_i^{'})/(p_{max}^{'} + 2 \cdot \sigma_{max}^{'}) < \alpha$ and in the drift level when $(p_i^{'} + 2 \cdot \sigma_i^{'})/(p_{max}^{'} + 2 \cdot \sigma_{max}^{'}) < \beta$, such that α and β takes the values 0.95 and 0.90 respectively.

- EWMA for concept drift detection (ECDD) [135] employs an idea similar to SPC for detecting drifts. This is achieved by observing the change in the exponentially weighted moving average (EWMA) [133], which progressively down-weights older observation in order to form a more recent estimate of the average $Z_t = (1 - \lambda)Z_{t-1} + \lambda X_t$, with $X_0, \ldots, X_t, \ldots$ are independent random variables with a known mean μ_0 and standard deviation σ_X. Roberts [133] shows that the mean of Z_t is $\mu_{Z_t} = \mu_0$ and the standard deviation is given by $\sigma_{Z_t} = \sqrt{\frac{\lambda}{2-\lambda}(1 - (1 - \lambda)^{2t})}\sigma_X$. EWMA detects a change in the mean Z_t, from μ_0 to the unknown mean μ_1, whenever the difference between Z_t and μ_0 exceeds a certain threshold, i.e., $Z_t > \mu_0 + L\sigma_{Z_t}$, where L is the control limit which determines how sensitive the detection should be.

  ECDD changes the EWMA method in order to avoid the assumption of knowing μ_0 before the change. It defines the variable $\hat{p}_t = \frac{t-1}{t}\hat{p_{t-1}} + \frac{1}{t}X_t$ for the exact average of all past observations, which weights all observations in the same way. ECCD assumes that the random variables are Bernoulli random variables representing a stream of binary prediction errors. A change is detected in this binary stream whenever $Z_t > \hat{p}_t + L\hat{\sigma}_{Z_t}$, such that $\hat{\sigma}_{Z_t} = \sqrt{\frac{\lambda}{2-\lambda}(1 - (1 - \lambda)^{2t})\hat{p}_t(1 - \hat{p}_t)}$.

- Adaptive windowing (ADWIN) [21, 20] is a drift detection method that, instead of sliding a window over only the recent samples, shrinks the window of observations whenever a change is detected. In this way, the expected value of the observations in the remaining part and the removed part of the window are guaranteed to be different, with probability of $1 - \delta$. ADWIN2 [21, 20] is developed as an efficient alternative to ADWIN; it needs to check only $O(\log(n))$ sub-windows for the shrinkage, where n is the size of the window. ADWIN2 accomplishes this by approximating the window through storing only a variation of exponential histograms [46].

Classifier-independent detection methods

This category of methods is dominated by parametric statistical tests that put assumptions on the population, from which data is sampled. Many statical hypothesis

testing methods can be applied to detect changes in the data generating process. The choice of the most suitable hypothesis test depends on the wanted change criteria reflected in the design of the null hypothesis. In the following, we explain a number of important methods of that kind as reviewed in the survey paper [106], without any claim to completeness:

- The Welch's t-test is a two-samples test used to check whether two normally distributed populations have the same mean. This test differs from Student's t-test in that Welch's t-test allows the population's variances to be unequal. From the two samples X_1, X_2 of different sizes N_1 and N_2, the test statistic is

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \ ,$$

  where $\bar{X}_1, \bar{X}_2$ are the sample means and s_1^2, s_2^2 are the sample variances of X_1, X_2, respectively. The t-distribution, with a degree of freedom based on the sizes and the variances of the two samples, is then applied to test the null hypothesis that the means of the two populations are equal.

- The Kolmogorov-Smirnov test is a two-sample test for the null hypothesis that two samples are drawn from the same distribution. This is achieved by taking the supremum distances between the two empirical cumulative distribution functions. More formally, for the samples X_1 and X_2 of sizes N_1, N_2 respectively, the test statistic becomes

$$d = \sup_x |F_1(x) - F_2(x)| \ .$$

  The null hypothesis is then rejected with confidence α when $d > c(\alpha)\sqrt{\frac{N_1+N_2}{N_1 N_2}}$, where $c(\alpha)$ is found in the Kolmogorov-Smirnov table.

- Sequential probability ratio test (SPRT) [174] is a statistical hypothesis testing method for sequential data. For the sequence $X_n = x_0, \ldots, x_n$ of the independent samples, SPRT tests the null hypothesis that at the sample x_w, with $1 < w < n$, the data generating distribution does not change from p_0 to p_1 . The cumulative variable S_n holds the log ratio of the two likelihoods: the likelihood of $x_w, \ldots, x_n$ being generated by the distribution p_0 over the likelihood

of $x_w, \ldots, x_n$ being generated by the distribution p_1. S_n takes the form

$$\begin{aligned} S_n &= \log \frac{\mathbf{P}(x_w, \ldots, x_n; p_0)}{\mathbf{P}(x_w, \ldots, x_n; p_1)} \\ &= \sum_{i=w}^{n} \log \frac{\mathbf{P}(x_i; p_0)}{\mathbf{P}(x_i; p_1)} \\ &= S_{n-1} + \log \frac{\mathbf{P}(x_n; p_0)}{\mathbf{P}(x_n; p_1)} \ . \end{aligned}$$

The incremental observation of the samples is continued as long as S_n remains in a user-defined interval $[a, b]$. The stopping rule is then activated whenever $S_n \notin [a, b]$; such that H_0 is accepted when $S_n \geq b$ and H_1 is accepted when $S_n \leq a$. The choice of $a < 0 < b < \infty$ depends on the acceptable type I and type II errors.

- The cumulative sum (CUSUM) [125] is a method that triggers a change signal when a parameter of a probability distribution changes. The cumulative variable S_n is defined as

$$S_n = \max(0, S_{n-1} + x_n - w_n) \ ,$$

such that $S_n = 0$ and w_n is the weight for the sample x_n. CUSUM resembles SPRT when w_n is chosen to be the likelihood of x_n. On the other hand, it detects the change only in one direction, in the positive direction in the previous formulas.

- Page-Hinkely test (PH) [125] indicates a change whenever the average of Gaussian random variables significantly changes. This is accomplished through the continuous update of the variables m_n and M_n at the time point n:

$$\begin{aligned} m_n &= \sum_{i=1}^{n} (x_i - \bar{x}_i - \delta) = m_{n-1} + (x_n - \bar{x}_n - \delta) \\ M_n &= \min(m_n, M_{n-1}) \ , \end{aligned}$$

with $\bar{x}_i = \frac{1}{n} \sum_{i=1}^{n} x_i$ and δ represents the tolerance towards the allowed change. The PH test simply monitors the quantity $PH_n = m_n - M_n$. A change of the mean, in the positive direction, is triggered whenever the $PH_n > \lambda$, where λ corresponds to the tolerance towards type I error.

2.5 Adaptive Supervised Learning: Related Work

In this section, a review of the most relevant work to our thesis is presented and categorized into four categories: rule-based, tree-based, instance-based and ensemble methods.

2.5.1 Rule-based learning

Expert knowledge systems often take the form of a set of rules that describe the behavioral properties of an operational system, in means of a reaction/output for an action/input. The tremendous increase of the available data and the changing nature of the data generating processes have led to the need for decision support systems that can learn, evolve and adapt such rules from the available data autonomously. In the following, we present a list of adaptive rule-based systems; these systems are for classification problems, unless otherwise stated. These approaches are given, in chronological order, by:

- STAGGER [142] is the first approach that addresses a solution for concept drifts via incremental concept learning. It operates by finding a symbolic representation of the hidden concept (learning the concept by inspecting the instances with the positive class label). The concept is represented through a set of rules with conjunctive and disjunctive operators between their literals. The search is achieved in a similar way to the search in the version space [120], except that *(i)* the starting point here is the single literals, *(ii)* the generalization is accomplished by adding more disjunctive conditions and *(iii)* the specialization is achieved by adding more conjunctive conditions. Pruning and backtracking through the search process guarantees to reflect any concept drift on the found representations.
- Floating rough approximation 4 (FLORA4) [179] is an approach from the family of rule-based algorithms, which keeps a concept description in means of three types of propositional predicates: *(i)* predicates that cover only positive examples, *(ii)* predicates that cover only negative examples and *(iii)* predicates that cover both types of examples. Predicates of each type are accompanied by their support, the number of examples covered by each predicate. A predicate is moved from one set to the other depending on the change of its purity. FLORA2 applies a window adjusted heuristic (WAH) in order to cope with concept changes in the setting of incremental concept learning from a stream

of objects. This heuristic calls for shrinking the window size of the covered examples whenever a drop in the performance is detected.

- Fast and adaptive classifier by incremental learning (FACIL) [63, 64] introduces an adaptive method for learning a rule-based system incrementally from a data stream. Learned rules are handled based on their purity, the ratio between the number of covered instances belonging to the majority class to the total number of covered examples. On the arrival of new samples, a decision is made based on the following ordered criteria:

  1. If a consistent rule that covers this sample is found, the purity of this rule is increased.
  2. If no covering consistent rule is found, the consistent rule with the minimum generalization cost is chosen and generalized, as long as this cost does not exceed a given threshold.
  3. Otherwise, the purity of inconsistent rules, covering this example, is decreased.
  4. If none of the past criteria is satisfied, a new rule consistent with this sample is created. Rules that have purity lower than a predefined threshold are removed and replaced by less general consistent rules.

- RILL [50] is an adaptive rule-based algorithm that reserves a set of rules and instances. On the arrival of a new instance, which is not covered yet by any of the learned rules, the nearest rule is retrieved and generalized until it covers this instance. The generalized rule is only accepted when this generalization does not drop the purity of the original rule, otherwise it is retracted and the new instance is simply added to the set of rules.

- The field of soft computing has also developed its own incremental data-driven fuzzy rule-based approaches for regression problems, such as FLEXFIS [110] and eTS+ [7]. These two methods learn the so-called Takagi-Sugeno-Kang (TSK) fuzzy system [165], which consists of TKS rules, each of which has a linear function in the consequent part. The rules are learned in an online manner, after the application of incremental clustering. Despite their similarity in the learned models, FLEXFIS and eTS+ technically differ in the way they learn and update the rules' antecedents and consequent.

- In the recent years, adaptive rule learning has witnessed a leap in the complexity of the learned rules. AMRules, for example, is a rule induction method for regression on data streams [4]. Each rule is specified by a conjunction of literals on the input attributes in the premise part, and a (linear) function minimizing the root mean squared error in the consequent. Rules are incrementally added on the basis of Hoeffding's bound [81] and their performance is monitored by the Page-Hinkley (PH) test [121], such that a rule is pruned as soon as its performance drops due to a concept change. AMRules can be seen as an extension to the very fast decision rules (AVFDR) classifier [99] in order to solve regression problems with model rules. Very fast decision rules (VFDR) [71] incrementally induce a compact set of decision rules form a data stream; it is extended by AVFDR to detect and react to changing data by applying SPC, see Subsection 2.4.

In this work we choose to compare our proposed methods with AMRules and FLEXFIS. This choice is based on the following reasons: *(i)* They are considered as the state of the art rule-based evolving methods. *(ii)* The availability of their implementations. For more details, a comprehensive explanation of AMRules and FLEXFIS is added in Appendix A.2 and Appendix A.4, respectively.

2.5.2 Decision trees learning

A decision tree is a tree-shaped hierarchical arrangement of conditions that follows the concept of divide-and-conquer. Each of the tree's internal nodes contains a condition or "test" on one of the attributes describing the data; each outcome of the test is represented by an edge leading either to an internal node or to a leaf node. Each leaf node is assigned a class label, i.e., the decision on how an example should be classified if it falls in that leaf node. Common approaches that induce decision trees such as ID3 [169], CART [31] and C4.5 [130], induce trees by replacing a leaf node with an internal node, whenever this replacement decreases the information entropy, such as Shannon entropy [159].

Each new example traverses from the root to one of the leaf nodes, at the evaluation time. The path this traversal takes depends on the satisfaction of the internal nodes' conditions.

The following list shows decision tree induction methods, that are scalable for the large data sets:

- SLIQ [117] is a scalable decision tree induction method that is designed for very large data sets. It optimizes the split tests using a pre-sorted list that is prepared once for all available data. In addition, SLIQ applies MDL (minimum description length) pruning strategies.
- SPRINT [148] avoids the drawback of SLIQ when dealing with large amounts of data by eliminating the need for centralized memory-resident structures and presenting a parallel classifier as a substitute for the serial execution of the decision trees.
- RainForest [78] proposes modifications over SPRINT's approach when learning the decision tree. The authors replace the sorted list by the statistics of all plausible predicates at each internal node. Consequently, the splitting conditions can be evaluated and tested more efficiently instead of reiterating the whole examples. As a result, large data sets can be used for learning by considering their statistics instead of the sorted list on their attributes as in SPRINT.

The aforementioned approaches may be appropriate for large data sets, however, they suffer from two problems: *(i)* they lack the ability to cope with the continual arrival of infinite streams and *(ii)* they are incompetent to adapt to concept changes. The following depicts a description of the state-of-the-art incremental decision tree induction methods, that are designated for learning on data streams:

- The Hoeffding tree [56] is an incremental decision tree approach, tailored for classification on data streams. It tries to address the first problem using an algorithm that meets the decision of replacing a leaf node by an inner node after seeing an adequate amount of samples, based on statistical hypothesis testing. More specifically, Hoeffding's bound [81] is used to check whether the information gain of an alternative attribute is significantly higher than the gain of the currently chosen attribute.

  Hoeffding's bound states that with probability $1-\delta$, the difference between the empirical mean and the true mean of a random variable r, with $\mathbf{P}[a \leq r \leq b] = 1$ and $R = b - a$, would not exceed

  $$\epsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}} \tag{2.8}$$

  after observing n samples.

Hoeffding tree uses this bound to compare the difference between the information gains of the two best splitting attributes X_a and X_b, assuming that X_a is better than X_b, i.e., $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b) > 0$.

- An adaptive version of the Hoeffding tree (AdpHoef) is presented in [23]. This algorithm maintains a drift detection statistic in each node to judge the compatibility of the current tree/subtree with the latest seen data. For each of these nodes, an alternative tree is maintained and learned on the recent data only. Whenever the drift detector signals a change at a node, the subtree rooted at that node is replaced by the alternative tree. This variant of Hoeffding trees employs the ADWIN [22] technique, a parameter-free method for detecting the rate of change in data streams.

- Rutkowski et al. [136] show that Hoeffding's bound is misapplied in all Hoeffding tree approaches, mainly because the bound is not applied on the mean of the observed samples (the differences between the information gains of two different attributes), but on a function (the information gain) on the sample mean. As an alternative, they propose applying a different bound, derived from the McDiarmid's inequality [116], for inducing a decision tree on data streams. The results of the new approach were not satisfactory, as a result of the wide bound of McDiarmid's inequality, which led to a fewer number of splits, smaller trees, under-fitting the data and causing a poor generalization performance.

In our experiments, the Hoeffding tree and its alternative AdpHoef are used for comparison on classification problems and, therefore, are explained in Appendix A.1. Rutkowski's modifications on Hoeffding tree [136] is not considered for comparison as it never performs better than the Hoeffding tree.

2.5.3 Instance-based learning

Another approach for prediction is to find solutions for new problems based on their similarity to already known ones, without extracting any dependencies between the example's features and the output space. The class of learning methods that follows this concept is called instance-based learning (IBL), which includes case-based learning [163, 140, 98] and k-nearest neighbor [40]. Instance-based learning methods also belong to the lazy learning paradigm [2], mainly because the learner delays the analysis until the prediction phase.

Nearest neighbor (NN) approaches were first applied as non-parametric statistical estimators in the field of pattern recognition by [143, 123]. The *k-nearest neighbor*

(k-NN) is a natural generalization of the nearest neighbor approach, in which the set of the k closest neighbors is consulted for prediction instead of consulting only the closest neighbor. Although k-NN is a consistent estimator [126, 109] for density functions (when k is adopted properly as a function $k(|\mathcal{D}|)$ of the data size), it cannot technically cope with infinite data streams nor can it sustain its consistency when the observed concept changes.

The following is a list of IBL methods that are tailored for adaptive learning on data streams:

- Locally-weighted forgetting (LWF)[139] is an adaptive instance learning algorithm that considers the spatial aspects during the instance accumulation phase, such that examples in the neighborhood of a newly added example are weakened by decreasing their weights. Thereafter, examples with a weight lower than a predefined threshold are discarded.
- Time-weighted forgetting (TWF)[139] is an instance-based learner that lessens the weights of examples based on the temporal aspect. An example is removed when its weight becomes smaller than a predefined threshold.
- Instance-based learning on data streams (IBL-DS)[17] is one of the pioneering IBL approaches, due to its contribution in coining the guidelines an adaptive instance-based learner should consider. They introduce three relevance factors that need to be fulfilled when deciding to keep a new example: *(i)* the spatial relevance, *(ii)* the temporal relevance and *(iii)* the consistency.

Chapter 3 elaborates more on the approaches above and shows a systematic comparison with IBLStreams, our IBL approach for learning on data streams.

2.5.4 Ensemble methods

Ensemble methods train a set of models on the training data set; in this way, a prediction problem can be solved collectively by consulting the set of learned models instead of depending on a single model. Training the ensemble on the same data would lead to an ensemble with clones of the same model, which would make the ensemble redundant and the training cost non-beneficial. Thus, ensemble learning focuses mainly on how to obtain a diverse set of trained models. The second focus concerns the aggregation of the different decisions of the trained models, which is often solved by simply taking the majority vote.

- Bagging [30] is one way of learning a diverse set of models; it maintains a set of learned models trained by the same base learner. Each model is learned on a new replication of the training set $\mathcal{D}$, sampled with replacement. Each replication has the same size $M = |\mathcal{D}|$. In this way, and for a large M, each training example is chosen with probability $1-(1-1/M)^M$ for each replication. Similarly, each replication contains k copies of a training sample with probability of $\binom{M}{k}(\frac{1}{M})^k(1-\frac{1}{M})^{M-k}$. The distribution of k tends to a Poisson distribution $\text{Pois}(\lambda = 1)$ when $M \to \infty$, i.e., $\mathbf{P}(k) = \frac{1}{e \cdot k!}$. Online bagging [124] uses this fact by submitting each new example $k \sim \text{Pois}(\lambda = 1)$ times to each of the adaptive base learners in the ensemble set.

  Saffari et al. [137] apply the online bagging method to learn an online random forest, an ensemble of decision trees. Each training example is presented $k \sim \text{Pois}(\lambda)$ times to each tree in the ensemble. Trees, on the other hand, are trained in an incremental way by allowing the replacement of a leaf node with an internal node whenever *(i)* the number of examples observed at that leaf node exceeds a predefined threshold and *(ii)* the information gain for one of the splitting criteria is at least equal to the threshold β.

- Boosting is introduced by Schapire [141] as generic ensemble method that trains a set of weak learners, by presenting each example sequentially to each learner. After each learning step, the examples are reweighted based on the performance of the previous learners. Misclassified examples by the previous learner become more important and gain a higher weight, whereas correctly classified instances are down-weighted. For each query sample, the predictions of the weak learners are aggregated by weighting each prediction according to the performance of its corresponding learner.

  Learn++ [128] applies an idea that is inspired by AdaBoost [65]. Each time a new block of examples is received, Learn++ trains a set of weak learners on sampled sets from the block. The sampling scheme chooses examples which are wrongly classified, by the current models, with a probability higher than that of the correctly classified ones.

  Oza and Russell [124] also propose an online boosting method that trains a set of adaptive weak learners on a data stream. Similar to the online bagging, each new example is presented to each learner k times such that $k \sim \text{Pois}(\lambda = 1)$. The only difference is that the Poisson's λ parameter is increased whenever the previous model misclassifies it, otherwise λ is decreased.

- Brzezinski and Stefanowski [33] introduce the accuracy updated ensemble (AUE) on a stream of blocks; this approach trains a new classifier on the most recent block and updates the weights of the current classifiers, in the ensemble, based on their performance on this block. Only base classifiers whose performance is better than a dynamic threshold are incrementally updated, the others can still remain in the ensemble for the second round. Accuracy updated ensemble (AUE2) [35] adds a further improvement to AUE by applying a block-based test-then-train evaluation scheme to incrementally evaluate and update base classifiers. Moreover, the ensemble set is pruned whenever the memory consumption exceeds a predefined threshold.

  Different strategies for combining block-based and online-based ensembles are introduced by Brzezinski and Stefanowski [34]. The most prominent one is the online accuracy updated ensemble (OAUE) which couples the block-based ensembles with incremental base learners. Online classifiers are incrementally trained on each newly incoming example, whereas decisions on pruning the worst performing classifier and training a new classifier is always taken after fixed intervals, thus simulating a block-based ensemble method.

Finally, Street and Kim [164] proposed streaming ensemble algorithm (SEA), one of the first ensemble methods on data streams. This work suggests to learn a set of classifiers in an online manner. On each newly arriving block of data samples a new classifier is trained. Thereafter, an old classifier with the worst performance, from the learned ensemble set is replaced with the recently learned classifier if the latter outperforms the former. Their paper also discusses other decision criteria for replacing old classifiers, such as considering the diversity over the performance; it proposes a quality measure which is based not only on the correct predictions, but also on the confidence of each prediction and the error made by the ensemble. For a more detailed description of the online ensemble approaches see [68].

Chapter 3

Instance-Based Classification and Regression

Given the existence of numerous sophisticated and quite complicated methods for learning on data streams, it is surprising that one of the simplest approaches to machine learning, namely the instance-based (case-based) learning paradigm, has received very little attention so far; especially because the core principle of this paradigm, the nearest neighbor principle, is a standard method in machine learning, pattern recognition, and related fields.

This chapter demonstrates the benefit of applying instance-based learning in the streaming setting and introduces our instance-based learning algorithm for two prediction tasks: classification and regression. To this end, the work of [17], which offers a basic classification approach for data streams, is extended and generalized.

The remainder of this chapter is organized as follows: The next section recalls the basic idea of instance-based learning. In Section 3.2, a comparison of model-based with instance-based learning is presented with a discussion on their pros and cons when being used in the streaming setting. Instance-based learning on data streams is motivated in Section 3.3. In Section 3.4 our approach IBLStreams is introduced. Experimental results are presented in Section 3.5, prior to concluding the chapter in Section 3.6.

3.1 Instance-Based Learning

As already explained in Chapter 1, the main goal of machine learning is to extract knowledge, e.g., induced statistical models, from experiences with the aim of applying this knowledge to solve future problems.

One learning paradigm that does not fit a model during the learning phase is the *instance-based learning* (IBL); this paradigm, as the notion suggests, is only concerned about collecting experience in the learning phase. Since IBL methods do not induce a predictive model, they perform the required induction and inference at the prediction phase, hence they are considered as *lazy learning* methods [2]. IBL methods include case-based learning [163, 140, 98], k-nearest neighbor [40] and RBF networks [32], among others.

The *nearest neighbor* (NN) method is an instance-based learning approach that tries to solve new problems in a way similar to what people apply in daily life; it suggests to figure out solutions for new problems based on their resemblance to collected experiences. The first application of the nearest neighbor method, as non-parametric statistical estimator, in the field of pattern recognition, and especially in classification, dates back to the middle of the last century [143, 123]. Moreover, for large samples NN is shown in [40] to have a misclassification rate R that is bound in the interval $[R^*, R^*(2 - mR^*/(m-1)]$, where m is the number of classes and R^* is the *Bayes error*, which is the lowest possible error achievable by any classifier, it is also called the irreducible error [87].

Consider the supervised learning setting (see Section 2.2) where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space; and let d denote the distance function $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that measures the dissimilarity $d(\boldsymbol{x}_1, \boldsymbol{x}_2)$ between two instances $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$.

The only awareness the NN learner has about the learning problem is exhibited through the stored portion of the observed examples so far. The *case base* $\mathcal{M} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ preserves only a subset of the whole training set $\mathcal{D}$. The nearest neighbor principle [45] uses this data whenever a prediction is requested: Upon receiving the query instance $\boldsymbol{x}_q$, it finds the nearest example

$$(\boldsymbol{x}_{NN}, y_{NN}) = \underset{(\boldsymbol{x}_i, y_i) \in \mathcal{M}}{\operatorname{argmin}} \left| d\left(\boldsymbol{x}_q, \boldsymbol{x}_i\right) \right| \quad ,$$

in terms of the distance measure d; NN predicts the output $\hat{y}_q$ of the query instance $\boldsymbol{x}_q$ to be the same as the output of the nearest neighbor, i.e. $\hat{y}_q = y_{NN}$.

Bhatia and Vandana [19] present an extensive survey of nearest neighbor approaches, in which NN methods are classified into two categories: structureless and structure-based techniques. The former methods utilize an additional structure in order to overcome the memory limitation, reduce the complexity and enhance the querying efficiency.

The k-nearest neighbor (k-NN) is a natural generalization of the nearest neighbor approach, in which the set of the k closest neighbors is consulted for prediction. This

method is shown to be a consistent estimator for probability density functions when k is allowed to be adapted properly [126].

For a query instance $\boldsymbol{x}_q$, k-NN retrieves the set of the k nearest neighbors of $\boldsymbol{x}_q$; this set is denoted as

$$\mathcal{N}_k(\boldsymbol{x}_q) = \text{argmin}k_{(\boldsymbol{x}_i, y_i)\in\mathcal{M}} \quad d(\boldsymbol{x}_i, \boldsymbol{x}_q) \ , \tag{3.1}$$

where argmin$k$ returns the k examples that have the smallest distances from $\boldsymbol{x}_q$. Examples are selected at random in the case of ties. In the following, we explain how k-NN makes predictions in both classification and regression scenarios.

3.1.1 Classification

In classification, a prediction is usually determined by the majority vote, i.e. the most common class label in the set of neighbors $\mathcal{N}_k(\boldsymbol{x}_q)$:

$$\hat{y}_q = \underset{c\in\mathcal{Y}}{\text{argmax}} \left|\{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q) \,|\, y_i = c\}\right| \ . \tag{3.2}$$

Noting that $\hat{y}_q$ corresponds to the mode of the distribution on $\mathcal{Y}$ which is obtained by counting the frequencies of class labels in the neighborhood of $\boldsymbol{x}_q$; this prediction can be justified as an empirical risk minimizer of the standard 0/1 loss function.

The estimation (3.1) can be generalized, by weighting examples according to their distance from $\boldsymbol{x}_q$, forming the so-called weighted k-NN (Wk-NN)[16]:

$$\hat{y}_q = \underset{c\in\mathcal{Y}}{\text{argmax}} \sum_{(\boldsymbol{x}_i, y_i)\in\mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}_i) \cdot \mathbb{I}(y_i = c) \ , \tag{3.3}$$

where $\mathbb{I}$ is the indicator function and

$$w(\boldsymbol{x}_i) = \frac{f(d(\boldsymbol{x}_i, \boldsymbol{x}_q))}{\sum_{(\boldsymbol{x}_j, y_j)\in\mathcal{N}_k(\boldsymbol{x}_q)} f(d(\boldsymbol{x}_j, \boldsymbol{x}_q))} \ . \tag{3.4}$$

Here, $f(\cdot)$ is a decreasing function $\mathbb{R}_+ \to \mathbb{R}_+$, which means that the smaller $d(\boldsymbol{x}_i, \boldsymbol{x}_q)$, the higher the weight of y_i. The denominator in (3.4) normalizes the effect of each weight.

In the case of ordinal classification (see Section 2.2), the prediction is taken to be the median after weighting each neighbor in accordance with its distance from $\boldsymbol{x}_q$:

$$\hat{y}_q = \underset{c_s\in\mathcal{Y}}{\text{argmin}} \sum_{(\boldsymbol{x}, c_j)\in\mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}) \cdot |s - j| \ . \tag{3.5}$$

3.1.2 Regression

Like in the case of classification, the basic assumption of NN-based regression approaches is that the dependency to be learned, in the neighborhood of $\boldsymbol{x}_q$, is locally constant, or can at least be approximated sufficiently well by a constant function. More generally, one typically uses the weighted average

$$\hat{y}_q = \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}_i) \cdot y_i \ . \tag{3.6}$$

The choice of the mean of the neighbors' outputs is a direct consequence of applying the least squares method, i.e.,

$$\underset{\hat{y}_q}{\operatorname{argmin}} \sum_{(\boldsymbol{x}_j, y_j) \in \mathcal{N}_k(\boldsymbol{x}_q)} (y_i - \hat{y}_q)^2 \ .$$

Relaxing the assumption from locally constant dependency to locally linear dependency gives rise to the idea of locally weighted linear regression. The linear model takes the form

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{m} \beta_j \cdot x[j] = \boldsymbol{\beta}^\top \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} \ , \tag{3.7}$$

with $x[j]$ the jth entry of the vector $\boldsymbol{x}$ and the model is fitted in the neighborhood of $\boldsymbol{x}_q$. Thus, the vector of coefficients $\boldsymbol{\beta}$ is estimated by

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y} \ , \tag{3.8}$$

where the $k \times (m+1)$ matrix $\boldsymbol{X}$ is composed of the k neighbors $(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)$ (plus the vector of ones modeling the intercept β_0) and $\boldsymbol{Y}$ is the $k \times 1$ vector of corresponding output values y_i. Moreover, $\boldsymbol{W}$ is a diagonal weight matrix $\operatorname{diag}(w_1, \ldots, w_k)$, which is determined by means of a kernel function $f(\cdot)$ centered at $\boldsymbol{x}_q$; thus, the weight w_i of the neighbor $(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)$ is of the form (3.4).

Once (3.8) has been computed, the prediction $\hat{y}_q$ is obtained by evaluating (3.7) with $\boldsymbol{x} = \boldsymbol{x}_q$ and $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}$.

The instance-based prediction strategy, both for classification and regression, is summarized in pseudo-code in Figure 3.1.

3.2 Instance-Based versus Model-Based Learning

The focus of instance-based learning methods lies in the instances as local abstractions instead of an induced model as a global abstraction, this does not necessarily

Procedure Predict

Input: case base $\mathcal{M}$, example e=$\langle \boldsymbol{x}_q, ? \rangle$
k number of considered instances
σ kernel width, used in the case of Gaussian or exponential weighting
kernel kernel function (uniform, inverseDistance, linear, Gaussian, exponential)
regression_method (Locally_Weighted_Linear_Regression, Weighted_Mean)
Output: $\hat{y}_q$

1: $\mathcal{N}_k(\boldsymbol{x}_q) = \text{argmin} k_{(\boldsymbol{x}_i, y_i) \in \mathcal{M}} \quad d(\boldsymbol{x}_i, \boldsymbol{x}_q)$
2: $W = \text{getNormalizedWeightingMatrix}(e, \mathcal{N}_k(\boldsymbol{x}_q), k, \sigma, kernel)$
3: **if** Classification **then**
4: **if** Ordinal Classification **then**
5: $\hat{y}_q = \text{argmin}_{c_s \in \mathcal{Y}} \sum_{(\boldsymbol{x}, c_j) \in \mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}) \cdot |s - j|$ {Equation (3.5)}
6: **else**
7: $\hat{y}_q = \text{argmax}_{c \in \mathcal{Y}} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}_i) \cdot \mathbb{I}(y_i = c)$ {Equation (3.3)}
8: **end if**
9: **else**
10: {Regression}
11: **if** *regression_method* = `Weighted_Mean` **then**
12: {solve it as wKNN}
13: $\hat{y}_q = \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}_i) \cdot y_i$ {Equation (3.6)}
14: **else**
15: {solve it as a locally weighted linear regression}
16: $\boldsymbol{X} = [\boldsymbol{x}_i \quad 1]_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)}$
17: $\boldsymbol{Y} = [y_i]_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)}$
18: $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y}$
19: $\hat{y}_q = [\boldsymbol{x}_q^\top \quad 1] \hat{\boldsymbol{\beta}}$ {Equation (3.7)}
20: **end if**
21: **end if**
22: **return**

Figure 3.1: The instance-based prediction functions for both classification and regression problems.

imply that the examples are simply collected. On the contrary, some instance-based methods learn the similarity (or the distance) function as in metric learning [103], others invest the main effort in evaluating the importance of an instance and then deciding whether it should be stored or not [17].

An instance-based learner as indicated by Aha et al. [3], consists of three main components and each of these can be subject to learning:

- The similarity function: This function measures the similarity between an example and the instances in the concept description (case base).
- The classification function: This function decides, based on the result of the similarity function, how an instance should be classified.
- Concept description (case base) updater: A concept description is the set of examples that are maintained, until now, in the case base. The updater is a process that decides *(i)* whether a new example should be added to the case base or not and *(ii)* whether an example (or a group of examples) should be removed from the case base. These decisions are utterly based on the similarity results, classification results and the current state of the case base.

Model-based learners, on the other hand, invest most of their efforts in learning a model (i.e., abstracting patterns, inducing complex structures, tuning parameters, etc.) from available examples. This model can then be used whenever a prediction is required for a new instance.

For a fair comparison between the two paradigms, while learning from a data stream, we have to analyze the way they perform the learning, update and prediction processes:

- The learning process: It is clear that an IBL approach has a slightly neglected cost for simply storing the seen examples, compared to inducing a model, which usually requires a number of iterations on the data, until some criteria are satisfied, promising for a good generalization performance.
- The update process: An adaptive learner is expected to have the capability to perform two types of update operations: *(i)* An update as a response to observing a new example (i.e., to incrementally learn) and *(ii)* an update as a reaction to a discovered concept change. IBL approaches are inherently incremental [17], mainly because the update process is reduced to the simple addition or removal of an example(s) from the case base.

Model-based approaches, however, often require complex update procedures on the induced model, in order to learn or forget an example. Not to mention that learning methods often need to iterate and evaluate different criteria on the training data during the induction. For this reason, many model-based adaptive methods store either a set of the most recent examples (in sliding window) or try to summarize the recent examples in a form of measures (or counters) to be later used for the update.

– The prediction process: For this process, in contrast to the previous two processes, IBL methods usually require to invest more effort, compared to model-based approaches which directly consult the induced model. Despite the high prediction cost for IBL methods, this cost might seem less dramatic when the predictions are computed only locally (such as finding the k nearest neighbors or fitting a locally weighted linear function).

In a streaming setting, a model-based approach might be preferable when the demand for predictions is high and too frequent or when the concept to be learned is relatively constant, thus making the cost of model updates low. On the other hand, an IBL approach might be advantageous when the data stream suffers from frequent concept changes, or when the demand for predictions is low compared to the number of observed examples, see also [17].

3.3 Instance-Based Learning on Data Streams

Having motivated the intuition behind the IBL methods and having compared them to the model-based ones, now we discuss how an instance-based approach can be effectively used to learn from a non-stationary stream of data.

As we already explained, an instance-based learner maintains a case base that contains the set of collected examples. The capacity of the case base (the number of examples) is of course limited; the maximum allowed size comes often as a part of the application's requirements. Thus, in the streaming setting and while observing the continuously arriving data, the IBL learner has to decide dynamically which examples to collect and which to ignore. The simplest strategy would be to maintain only the recent examples in the case base. To this end, on the arrival of a new example, the oldest example is removed and the new one is added. This strategy is similar to sliding a window of a fixed size (number of examples) over the data stream.

However, an IBL method that applies a selective strategy could wisely choose the useful examples, such as keeping examples from unexplored regions of the input space and avoiding redundant and noisy examples.

In [17], the authors introduce the criteria an adaptive instance-based learner should consider while maintaining the case base. These criteria are:

- Temporal relevance: Recent examples tend to be more important than older ones, due to their ability to reflect the current concept.
- Spatial relevance: A balanced coverage of the whole instance space is preferred over leaving unoccupied regions of the instance space and oversampling from other regions. In other words, examples in underrepresented regions in the case base are more relevant than those belonging to over-represented regions.
- Consistency: A data example should only be preserved as long as it is consistent with the current concept.

As discussed in [17], most of the IBL approaches do not consider all of the aforementioned aspects. Locally-weighted forgetting (LWF) [139] applies only the spatial relevance, time-weighted forgetting (TWF) [139] considers only the temporal aspect, and IB3 [3] checks for the consistency before cumulatively adding new examples. IBL-DS [17], however, applies all the three suggested indicators. In the following, we explain the main IBL approaches that learn from a data stream.

Locally-Weighted Forgetting (LWF)

LWF [139] is an adaptive instance-based learning algorithm that concentrates on the spatial aspects during the learning phase. In this approach, examples in the neighborhood of a newly added example $(\boldsymbol{x}_{new}, y_{new})$ are weakened by decreasing their weights. An instance $(\boldsymbol{x}_i, y_i)$, that is the ith of the k nearest examples to the newly added example $(\boldsymbol{x}_{new}, y_{new})$, suffers a decrease in its weight by a factor $\gamma = \tau + (1-\tau)\frac{d_i^2}{d_k^2}$, where d_i is the distance between the ith nearest neighbor example and the new example $(\boldsymbol{x}_{new}, y_{new})$. A neighboring example is discarded after reaching a weight smaller than the threshold θ. It is worth mentioning that LWF employs an adaptive k, depending on the current size of the case base. The neighborhood is determined by $k = \lceil \beta L \rceil$, where $0 < \beta \leq 1$ and L is the current size of the case base. Figure 3.2(a) shows how LWF decreases the weights of examples in the neighborhood of a newly arriving example.

Time-Weighted Forgetting (TWF)

TWF [139] lessens the weight of each example in the case base by considering its temporal relevance. To this end, the weight of each example is decreased by the factor $\gamma \in]0,1[$; an example is removed when its weight recedes the threshold θ. As a result, the TWF approach resembles a queue buffer or a sliding window of length $\ell = \frac{\log \theta}{\log \gamma}$. Figure 3.2(b) shows an example case base, in which TWF down-weights each example whenever a new example arrives.

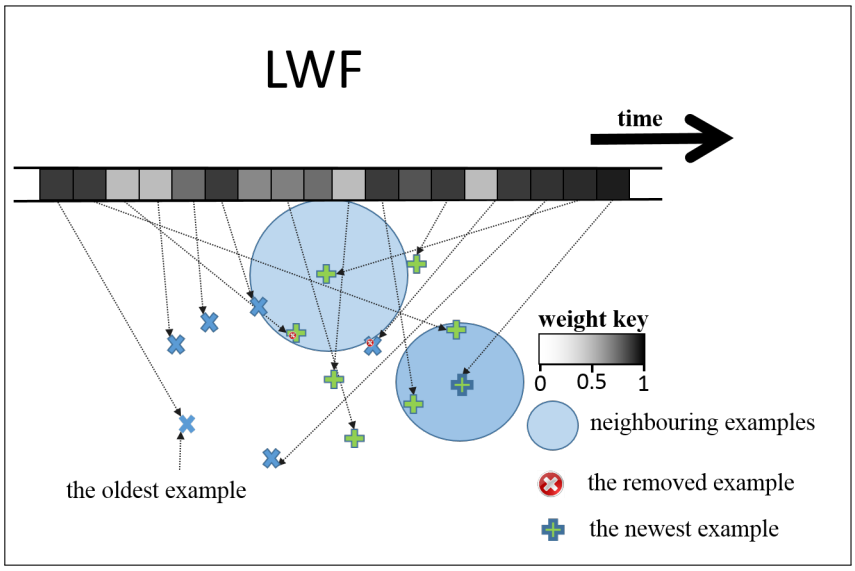
Instance-Based Learning on Data Streams (IBL-DS)

IBL-DS [17] applies all relevance indicators in the streaming setting for classification. IBL-DS is the fundamental method and the corner stone on which we build our approach IBLStreams, introduced in the next section.

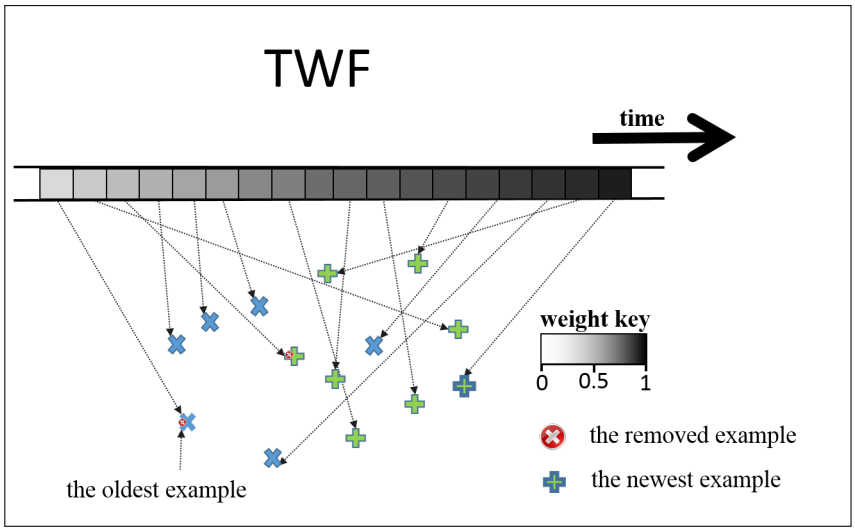
On the arrival of a new example $\boldsymbol{z}_t = (\boldsymbol{x}_t, y_t)$, it is at first added to the case base. Thereafter, IBL-DS tries to make the neighborhood $\mathcal{N}(\boldsymbol{x}_t)$, whose size is chosen to be $|\mathcal{N}(\boldsymbol{x}_t)| = (k_{cand})^2 + k_{cand}$, pure by removing incoherent and noisy examples. To this end, an example $(\boldsymbol{x}_i, y_i) \in \mathcal{N}(\boldsymbol{x}_t)$ is removed from the case base if *(i)* its class y_i differs from y_{maj} (the most frequent class in the neighborhood) and *(ii)* $(\boldsymbol{x}_i, y_i)$ is not one of the k_{cand} most recent examples. In this way, incoherent examples are only tolerated if they were recent, as they might be the beginning of concept change. Abrupt concept changes are detected using the statistical process control method [72, 73], see Section 2.4. Figure 3.2(c) shows how IBL-DS removes an old inconsistent example and retains a recent inconsistent one in the neighborhood of a newly added example; in the depicted illustration we set $k_{cand} = 2$.

3.4 IBLStreams

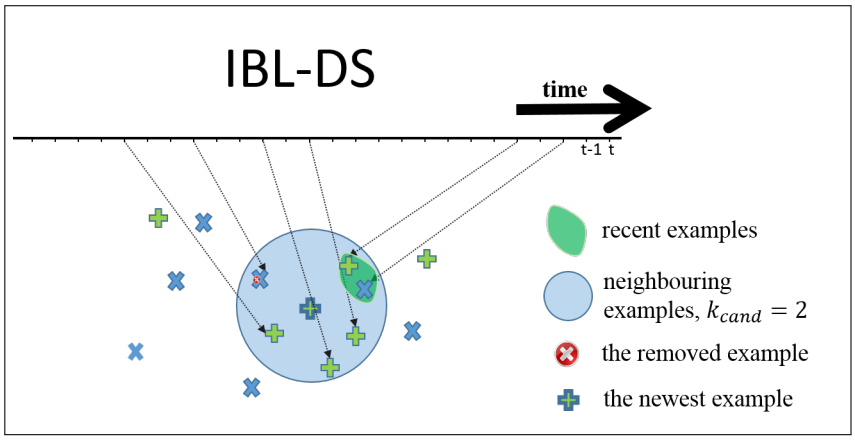
In this section, we introduce our instance-based learning approach on data streams, IBLStreams, that considers all the aforementioned relevance factors; it also exhibits the properties an ideal stream mining system should have. IBLStreams imposes an upper limit L_{max} on the size of the case base. In the following, we explain how IBLStreams maintains the case base and makes predictions when being applied for both classification and regression problems; case base maintenance strategies are depicted in Figure 3.3 and concept detection strategies are presented in Figure 3.4.



(a)



(b)



(c)

Figure 3.2: The IBL approaches that learn from data streams: (a) LWF, (b) TWF and (c) IBL-DS.

3.4.1 Classification

For classification problems, IBLStreams computes the predictions as a weighted voting (3.3). It maintains the case base in the classification scenario as follows:

- Every incoming example $\boldsymbol{z}_t = (\boldsymbol{x}_t, y_t)$ is at first added to the case base, then its neighborhood is checked for any incoherencies. Preserving the coherency in the neighborhood helps maintaining a case base that is consistent with the current concept to be learned from the data stream. We retrieve the neighborhood $\mathcal{N}(\boldsymbol{x}_t)$, which contains $|\mathcal{N}(\boldsymbol{x}_t)| = 2 \cdot k$ examples. Let $\mathcal{A} \subset \mathcal{N}(\boldsymbol{x}_t)$ be the set of the k closest examples, and let $\mathcal{B} \subset \mathcal{N}(\boldsymbol{x}_t)$ be the set of the k most recent examples.

  A match between y_t and the most frequent class y_{maj} in A tells about a local regularity, i.e., $y_t = y_{maj}$; this, however, does not mean that all examples have the same class. One way to purify this neighborhood is to remove an example $\boldsymbol{z}' = (x', y') \in \mathcal{A}$ if *(i)* y' differs from the majority class y_t and *(ii)* it is not a recent example (i.e., $\boldsymbol{z}' \notin \mathcal{B}$). Although this solution increases the consistency in the neighborhood, it is too rigid as it handles all examples, even the ones at the border of the neighborhood, in the same way. One way to increase the tolerance towards border examples, is to avoid removing an example if its distance is greater than the 80th percentile p_{80} of the distances of the examples in $\mathcal{A}$, where p_{80} is the 80th percentile of $\{d(\boldsymbol{x}_i, \boldsymbol{x}_t) | (\boldsymbol{x}_i, y_i) \in \mathcal{A}\}$.

- Like in IBL-DS, we also impose the upper limit restriction on the size of the case base, except that we apply a more complex approach than just simply removing the oldest example in the case base. Our idea is to remove one of the oldest examples by preferring examples from denser regions over the ones in sparser regions. Let $\mathcal{V} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T\} \subset \mathcal{M}$ be the set of the T oldest examples in the case base associated with the set $\mathcal{U} = \{\bar{d_{\boldsymbol{z}_1}}, \ldots, \bar{d_{\boldsymbol{z}_T}}\}$, such that $\bar{d_{\boldsymbol{z}_i}} = \frac{1}{k} \sum_{(\boldsymbol{x}_j, y_j) \in \mathcal{N}_k(\boldsymbol{x}_i)} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, i.e., each element in $\mathcal{U}$ holds the mean distance $\bar{d_{\boldsymbol{z}_i}}$ between an example $\boldsymbol{z}_i$ and its k nearest neighbors. Having computed the set $\mathcal{U}$, we choose the example $\boldsymbol{z}$ with the densest region ($\boldsymbol{z} = \operatorname{argmin}_{\boldsymbol{z} \in \mathcal{V}} \bar{d_{\boldsymbol{z}}}$) to be removed. In this way, we do not only respect the restrictions on the maximum size of the case base, but also both the spatial and the temporal relevance factors; we refer to this aspect as the *spatio-temporal* (ST) aspect.

- Although the first discussed procedure tries to keep local regions consistent against faulty examples, it cannot cope with concept drifts because it removes

only old inconsistent examples in almost pure regions. But how can we adapt to a changing concept, if it can only be observed through irregularities in impure regions?

IBLStreams handles concept changes similarly to IBL-DS. To this end, let p denote the prediction error (when applying the zero-one loss), which we incrementally maintain on a sliding window of N examples. In this way, p and N are the misclassification probability and the number of trials of a binomial distribution which can be approximated by the normal distribution[1] $\mathcal{N}(p, s^2)$, with $s = \sqrt{\frac{p(1-p)}{N}}$. The smallest achieved error rate p_{min} (along the stream) and the associated standard deviation s_{min} are preserved and updated whenever $p_{min} + s_{min} > p + s$. As in IBL-DS, a concept change is detected whenever the current error rate p significantly exceeds p_{min}, that is $p + s > p_{min} + T_{1-\alpha} s_{min}$, with the significance level $\alpha = 0.05$ for the (one-sided) Student's t-test with $N - 1$ degrees of freedom.

When a change is detected, a fraction of the case base is removed or "forgotten". The portion of the forgotten examples should be proportional to the rate at which the drift occurs. As a rule of thumb, we forget a percentage equal to $\min(p - p_{min}, 0.5)$ of the currently preserved examples.

We choose the examples to be forgotten at random with a tendency to remove older examples rather than recent ones; with t an exponentially distributed random variable (i.e., $t \sim \text{Exp}(\lambda)$, with $\lambda = 1.5$), the tth oldest example (in the case base) is chosen for removal.

3.4.2 Regression

Regression in IBLStreams is performed by applying the idea of locally weighted linear regression as presented in (3.7). In the following, we summarize how IBLStreams maintains the case base in the regression scenario:

- As in the classification case, a similar coherency strategy is adopted for the regression scenario. Here we also retrieve the neighborhood $\mathcal{N}(\boldsymbol{x}_t)$ which contains $|\mathcal{N}(\boldsymbol{x}_t)| = 2 \cdot k$ examples, upon adding the new training example $\boldsymbol{z}_t = (\boldsymbol{x}_t, y_t)$ to the case base. From $\mathcal{N}(\boldsymbol{x}_t)$, we define the two sets: $\mathcal{A} \subset \mathcal{N}(\boldsymbol{x}_t)$, the set of the k closest examples and $\mathcal{B} \subset \mathcal{N}(\boldsymbol{x}_t)$, the set of the k most recent examples.

[1] This approximation is valid as N is large enough, so that $p \cdot N > 5$ and $(1 - p) \cdot N > 5$.

Procedure UpdateCaseBase

Input: case base $\mathcal{M}$, example $\boldsymbol{z}_t = \langle \boldsymbol{x}_t, y_t \rangle$
Output: case base $\mathcal{M}$

$\mathcal{N}_{2k}(\boldsymbol{x}_t) = \text{argmin}2k_{(\boldsymbol{x}_i, y_i) \in \mathcal{M}} \quad d(\boldsymbol{x}_i, \boldsymbol{x}_t)$
{$\mathcal{A}$ contains the k nearest neighbors, s.t $\mathcal{A} \subset \mathcal{N}_{2k}(\boldsymbol{x}_t)$}
$\mathcal{A} = \text{argmin}k_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_{2k}(\boldsymbol{x}_t)} \quad d(\boldsymbol{x}_i, \boldsymbol{x}_t)$
{$\mathcal{B}$ contains the k recent neighbors, s.t $\mathcal{B} \subset \mathcal{N}_{2k}(\boldsymbol{x}_t)$}
$\mathcal{B} = \text{argmax}k_{\boldsymbol{z}_i \in \mathcal{N}_{2k}(\boldsymbol{x}_t)} \quad time(\boldsymbol{z}_i)$
if Classification **then**
    $y_{maj} = \text{argmax}_{c \in \mathcal{Y}} |\{(\boldsymbol{x}_i, y_i) \in \mathcal{A} \,|\, y_i = c\}|$
    if $y_t = y_{maj}$ **then**
        p_{80} = the 80th percentile of $\{d(\boldsymbol{x}_i, \boldsymbol{x}_t) | (\boldsymbol{x}_i, y_i) \in \mathcal{A}\}$
        remove $\boldsymbol{z}' = (\boldsymbol{x}', y') \in \mathcal{A}$ s.t. $(\boldsymbol{z}' \notin \mathcal{B}) \wedge (y' \neq y_{maj}) \wedge (d(\boldsymbol{x}', \boldsymbol{x}_t) < p_{80})$
    end if
else
    {Regression}
    $\bar{y} = \frac{1}{|\mathcal{A}|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{A}} y_i$
    $CI_{output} = [\bar{y} - T_{\frac{\alpha}{2}} \frac{s}{\sqrt{k}}, \bar{y} + T_{1-\frac{\alpha}{2}} \frac{s}{\sqrt{k}}]$
    if $y_t \in CI_{output}$ **then**
        p_{80} = the 80th percentile of $\{d(\boldsymbol{x}_i, \boldsymbol{x}_t) | (\boldsymbol{x}_i, y_i) \in \mathcal{A}\}$
        remove $\boldsymbol{z}' = (\boldsymbol{x}', y') \in \mathcal{A}$ s.t. $(\boldsymbol{z}' \notin \mathcal{B}) \wedge (y' \notin CI_{output}) \wedge (d(\boldsymbol{x}_i, \boldsymbol{x}_t) < p_{80})$
    end if
end if
$\mathcal{M} = \mathcal{M} \cup \{(\boldsymbol{x}_t, y_t)\}$
{enforce the upper limit L_{max} on the size of the case base}
if $|\mathcal{M}| > L_{max}$ **then**
    {$\mathcal{V}$ contains the T oldest examples}
    $\mathcal{V} = \text{argmin}T_{(\boldsymbol{x}_i, y_i) \in \mathcal{M}} \quad time(\boldsymbol{x}_i)$
    remove $\boldsymbol{z}' = (\boldsymbol{x}', y') \in \mathcal{V}$ s.t. $\boldsymbol{z}' = \text{argmin}_{\boldsymbol{z}' \in \mathcal{V}} \frac{1}{k} \sum_{(\boldsymbol{x}_j, y_j) \in \mathcal{N}_k(\boldsymbol{x}')} d(\boldsymbol{x}', \boldsymbol{x}_j)$
end if
return
{$time \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{N}$ is a function that returns the timestamp of an example in the case base $\mathcal{M}$.}

Figure 3.3: The algorithm for updating the case base in both classification and regression scenarios.

Procedure ConceptDriftDetection

Input: case base $\mathcal{M}$, example $\boldsymbol{z}_t = \langle \boldsymbol{x}_t, y_t \rangle$
Output: case base $\mathcal{M}$

if Classification **then**
  {the predicted class for the new example $\boldsymbol{z}_t$ given by equation (3.3)}
  $\hat{y}_q = \text{argmax}_{c \in \mathcal{Y}} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_q)} w(\boldsymbol{x}_i) \cdot \mathbb{I}(y_i = c)$
  {update the mean error p_t and the standard deviation s_t on a sliding window of W examples, based on the loss l_t caused by predicting $\hat{y}_t$ instead of y_t}
  $p_t = p_{t-1} + (-l_{t-W} + l_t)/W$
  $s_t = \sqrt{\frac{p_t(1-p_t)}{W}}$.
  if $p_t + s_t > p_{min} + T_{1-\alpha} s_{min}$ **then**
    $\tau = min(p_t - p_{min}, 0.5)$
    delete $min(\tau|\mathcal{M}|, |\mathcal{M}| - 100)$ examples, s.t. an example $\boldsymbol{z}'$ is chosen for removal with $(time(\boldsymbol{z}') - \text{argmin}_{\boldsymbol{z}_i \in \mathcal{M}} time(\boldsymbol{z}_i)) \sim \text{Exp}(\lambda = 1.5)$
  end if
else
  {Regression}
  {the predicted output for the new example $\boldsymbol{z}_t$ given by Equation (3.6) or by (3.7)}
  $\hat{y}_t = \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{N}_k(\boldsymbol{x}_t)} w(\boldsymbol{x}_i) \cdot y_i$ or $\hat{y}_t = \left[\begin{array}{cc} \boldsymbol{x}_t^\top & 1 \end{array} \right] \cdot \boldsymbol{\beta}$
  $p_t = p_{t-1} + (-l_{t-W} + l_t)/W$
  $s_t = \sqrt{s_{t-1}^2 + \frac{W \cdot p_{t-1}^2 - W \cdot p_t^2 - l_{t-W}^2 + l_t^2}{W-1}}$.
  if $p_t + s_t > p_{min} + T_{1-\alpha} s_{min}$ **then**
    $\tau = min(\frac{p_t - p_{min}}{p_{min}}, 0.5)$
    delete $min(\tau|\mathcal{M}|, |\mathcal{M}| - 100)$ examples, s.t. an example $\boldsymbol{z}'$ is chosen for removal with $(time(\boldsymbol{z}') - \text{argmin}_{\boldsymbol{z}_i \in \mathcal{M}} time(\boldsymbol{z}_i)) \sim \text{Exp}(\lambda = 1.5)$
  end if
end if
if $p_t + s_t < p_{min} + s_{min}$ **then**
  $p_{min} = p_t$
  $s_{min} = s_t$
end if
return
{$time$: $\mathcal{X} \times \mathcal{Y} \to \mathbb{N}$ is a function that returns the timestamp of an example in the case base $\mathcal{M}$.}

Figure 3.4: The algorithm for checking and handling concept drifts in both classification and regression scenarios.

For the agreement test between y_t and its neighborhood, it is obvious that the mode of the distribution of the target attribute y in the neighborhood $\mathcal{N}(\boldsymbol{x}_t)$ is impractical for the real-valued output. Instead, we use the outputs of the examples in $\mathcal{A}$ to determine a two-sided confidence interval $CI_{output} = [\bar{y} - T_{\frac{\alpha}{2}} \frac{s}{\sqrt{k}}, \bar{y} + T_{1-\frac{\alpha}{2}} \frac{s}{\sqrt{k}}]$, where $\bar{y}$ is the average output for the examples in $\mathcal{A}$ and s is the associated standard deviation; $\alpha = 0.05$ is the significance level for a two-sided t-distribution with $k-1$ degrees of freedom. Now, an agreement between y_t and its neighborhood can be tested by checking whether $y_t \in CI_{output}$. Only in the case of agreement, we try to purify the neighborhood by removing examples in $\mathcal{A}$ from the case base using the same strategy applied in the classification scenario.

- We also use the same spatio-temporal aspect, as introduced in the classification case, to impose the upper limit restriction on the size of the case base.
- The drift detection test in the regression scenario is conducted with the mean absolute error instead of the classification rate, and the percentage of examples to be removed is determined by the relative increase of this error.

3.4.3 Parameter adaptation in IBLStreams

Although instance-based learning does not induce a global model, its performance still depends on several parameters, such as the size of the neighborhood k. Given its application in an evolving environment, some sort of adaptivity would clearly be desirable in this regard. In IBLStreams, two approaches for parameter adaptation are implemented, see Figure 3.5.

In the first approach, we adapt the size k of the neighborhood. To this end, we continuously check whether it appears beneficial to increase or decrease the current value by 1. In order to make this decision, we monitor the mean error on a window formed by the last 100 instances, not only for the current IBLStreams version with k neighbors, but also the variants with $k-1$ and $k+1$ neighbors. Whenever one of these two variants performs better in terms of the mean error, the current k is adapted correspondingly, see lines 1-10 in the pseudo-code in Figure 3.5.

The second strategy controls the size of the neighborhood indirectly via the weighting function or, more specifically, the corresponding kernel width; this adaptation strategy can only be used in combination with the Gaussian or the exponential kernel. Like in the previous case, three variants of IBLStreams are compared in terms of their mean error on the last 100 instances, namely the current variant, the variant

Procedure UpdateClassifier

Input: case base $\mathcal{M}$, example $\boldsymbol{z}_t = \langle \boldsymbol{x}_t, y_t \rangle$
k number of considered nearest neighbors
σ the kernel function's width, used in the case of Gaussian and exponential kernels
kernel the kernel function (uniform, inverseDistance, linear, Gaussian, exponential)
Output: k, σ
Constants: $\delta = 0.05$

1: **if** Adaptive k **then**
2: {p : mean error for the last 100 examples}
3: update p_0 by Predict $(\boldsymbol{z}_t, k-1, \sigma, kernel)$
4: update p_1 by Predict $(\boldsymbol{z}_t, k, \sigma, kernel)$
5: update p_2 by Predict $(\boldsymbol{z}_t, k+1, \sigma, kernel)$
6: **if** $p_2 < p_1$ **then**
7: $k = k + 1$
8: **else if** $p_0 < p_1$ **then**
9: $k = k - 1$
10: **end if**
11: **else if** Adaptive σ **then**
12: {p : the mean absolute error for the last 100 examples }
13: update p_0 by Predict $(\boldsymbol{z}_t, k, \sigma(1-\delta), kernel)$
14: update p_1 by Predict $(\boldsymbol{z}_t, k, \sigma, kernel)$
15: update p_2 by Predict $(\boldsymbol{z}_t, k, \sigma(1+\delta), kernel)$
16: **if** $p_2 < p_1$ **then**
17: $\sigma = \sigma(1+\delta)$
18: **else if** $p_0 < p_1$ **then**
19: $\sigma = \sigma(1-\delta)$
20: **end if**
21: **end if**
22: **return**

Figure 3.5: The algorithm for updating the parameters of IBLStreams.

with a kernel width increased by 5%, and the variant with a kernel width decreased by 5%, see lines 11-21 in the pseudo-code in Figure 3.5.

3.4.4 Implementation issues

IBLStreams is implemented as an extension[2] for the MOA[3] (Massive Online Analysis) [24] framework, an open source software for mining and analyzing large data sets in a stream-like manner, see Appendix B.

The simple value difference metric (SVDM) is used as a distance function, and the index structure M-Tree [38, 39] is used for indexing and retrieving the instances in the case base, as suggested in [17], see Appendix C.1 and Appendix C. M-Tree makes use of the triangle inequality, satisfied by the metric distance[4], in order to maintain the instances in hierarchical hyperspheres in the metric space.

Although the previous works of IBLStreams [151] and IBL-DS [17] were utilizing the *query processing library* XXL [52], IBLStreams in this thesis is shifted to a simpler open source implementation of M-tree[5], which is hosted in the web-based Git repository hosting service GitHub[6].

Finally, in the locally weighted linear regression case, a solution might not be derivable when $\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$ is singular, i.e., it is not invertible; hence, the weighted average is used for prediction instead. Situations of singularity or close-to-singularity may also occur if the main diagonal of $\boldsymbol{W}$ is strongly dominated by a single entry; such situations lead to the problem of numerical instability. To avoid such a problem, we prevent the kernel width in the exponential or Gaussian weighting to become too small.

3.5 Experiments

We investigate in our experiments the performance of IBLStreams from three different points of view:

- In Subsection 3.5.1, we compare the performance of IBLStreams with the widely used adaptive instance-based approaches.
- In Subsection 3.5.2, we evaluate the different parameter adaptation strategies used in IBLStreams.

[2] `www.uni-marburg.de/fb12/kebi/research/software/iblstreams`, accessed on October 13, 2015

[3] `http://moa.cms.waikato.ac.nz`, accessed on October 8, 2015

[4] The metric distance used by the M-Tree is not necessarily the same as the distance function d utilized by the IBL method.

[5] `https://github.com/erdavila/M-Tree`, accessed on July 13, 2015

[6] `https://github.com`, accessed on July 13, 2015

- In Subsection 3.5.3, IBLStreams is compared with the state-of-the-art adaptive model-based methods.

In the following experiments, we use both synthetic and real data streams. The used real data sets are standard benchmarks taken from the UCI repository[7] [107], whereas the synthetic data streams are generated using the MOA framework, see Appendix B.

For each pure synthetic data stream, a generative model is randomly generated and fixed. Thereafter, 10 streams (repetitions using different seeds) are generated from the fixed model. In this way, we guarantee that the underlying model of the different repetitions is identical. More specifically, let the fixed data generating process be characterized by the probability measure $\mathbf{P}_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ is the parameter vector for that process, and let $\boldsymbol{z} = (\boldsymbol{x}, y)$ be a generated training example. The examples $\boldsymbol{z}_{(i,1)}, \boldsymbol{z}_{(i,2)}, \ldots, \boldsymbol{z}_{(i,|\mathcal{S}_i|)}$ in the pure data stream $\mathcal{S}_i$ and the examples in all pure streams $\mathcal{S}_1, \ldots, \mathcal{S}_{10}$ are i.i.d., i.e., $\boldsymbol{z}_{(i,j)} \sim \mathbf{P}_{\theta}$. Streams with a concept drift, on the other hand, are generated by processes that are time-dependent such that $\mathbf{P}_{\theta}$ is replaced by $\mathbf{P}_{\theta(t)}$. As a result, only the tth examples of the different repetitions are identically distributed, i.e., $\boldsymbol{z}_{(i,t)} \sim \mathbf{P}_{\theta(t)}$. Appendix D is dedicated to give an overview of the used data sets. Finally, all experiments are executed using the test-then-train scenario, see Appendix B.2.

3.5.1 IBLStreams versus other instance-based methods

In the following, we compare the IBLStreams with the other discussed IBL approaches:

- IBL-DS
- LWF (As suggested by the authors, we let this method choose the number of nearest neighbors k dynamically during the training time)
- TWF
- Win5k (The standard k-NN approach, in which the case base is restricted to contain only the examples in a fixed size sliding window)
- Win5kST (The same as Win5k equipped with the proposed spatio-temporal relevance strategy)

[7] `http://archive.ics.uci.edu/ml/`, accessed on October 8, 2015

- IBLStreams no ST (As introduced in this chapter without the spatio-temporal relevance strategy)
- IBLStreams (As introduced in this chapter with the spatio-temporal relevance strategy)

We compare IBLStreams with the discussed NN approaches on synthetic data streams. For all methods, we fix the upper limit L_{max} of the case base to $L_{max} = 5{,}000$ and the number of neighbors to $k = 5$. Only LWF is allowed to choose k freely during the training phase as proposed by its authors; k is defined as $k = \lceil \beta \cdot L \rceil$, where L is the current size of the case base and $\beta = 0.04$. As proposed by the authors, IBL-DS is used with the default parameters, LWF is used with $\gamma = 0.966$ and $\theta = 0.33$. TWF is used with $\gamma = 0.8$ and $\theta = 0.33$. IBLStreams is used with $T = 100$, $k = 5$ and with the equal weight kernel, which simplifies Wk-NN to the standard k-NN case. We apply this disadvantageous restriction on IBLStreams in order to remove any potential benefit that could be gained from the kernel weighting. We also choose to compare with the basic sliding window approach (Win5k), which is often used as a baseline approach; we also compare with Win5kST, which combines Win5k with the spatio-temporal relevance strategy.

The following comparison uses 4 synthetic data sets, presented in Table 3.1, for four main scenarios: *(i)* pure data, *(ii)* data with a concept drift, *(iii)* data with a sampling drift (virtual drift) and *(iv)* data with both a concept drift and a sampling drift. Methods to simulate concept drifts and sampling drifts are implemented in MOA, explained in Appendix D.

We present the average and the standard deviation of the accuracy for the four different scenarios in Tables 3.2-3.5. For each data set, methods with the best results are highlighted in bold. For the two IBLStreams settings, a setting is marked with $\bullet$ whenever its result is significantly better than all IBL approaches (excluding the other IBLStreams setting). Other IBL approaches are marked with $*$ to indicate that their performance is significantly better than the rest. The test of significance is conducted by applying the Wilcoxon signed-rank test for the null hypothesis that the median difference between the pairs of results is zero at the 1‰ significance level.

Table 3.2 shows that IBLStreams, with and without the spatial relevance, wins significantly on the majority of the pure data streams and comes second on the random trees data stream. This experiment also shows how the simple sliding window is often underestimated and/or used for comparison in a misleading way. IBL-DS, on the other hand, shows superiority only on the hyperplane and the SEA data streams.

For the data with a sampling drift, the completely temporal approach Win5k seems to be an effective solution when the data sampling distribution shifts in the input space, but this performance becomes significantly better when accompanying the sliding window with the proposed spatio-temporal aspect, as shown in Table 3.3, where both the spatial Win5kST and IBLStreams accuracies are the highest. The straightforward explanation for this result is that even when abandoned regions become neglected, keeping some examples from these regions is advantageous, for the time when these regions become active again.

Table 3.4 presents the results when applying a concept drift on the generated data sets. Again IBLStreams, supported with the different relevance factors, wins on 5 out of 10 data sets with margin of 3-7% compared to the second best approach, and comes second on the rest of the data sets with a margin less than 1%.

Finally, the sliding window with the spatio-temporal aspect, Win5kST, shows the best performance for the data that contains the two types of drifts, see Table 3.5.

In conclusion, IBLStreams shows a superior performance in most of cases, except for the sampling shift cases where the simple sliding window approach, supported with our spatio-temporal aspect, achieves the best results.

3.5.2 Evaluating the parameter adaptation schemes

To show the advantage of the proposed parameter adaptation approaches, we design new experiments on low dimensional data sets, in order to show how the adaptation strategies affect the decision boundaries of the learned models.

Two synthetic data streams are used for this evaluation, each with two dimensions; the length of each stream is 125k examples. We compare IBLStreams in four different settings *(i)* adaptive k, *(ii)* adaptive σ, *(iii)* nearest neighbor (i.e., $k = 1$) and *(iv)* fixed k and fixed σ. In the beginning, k and σ are initialized to $k = 5$, $\sigma = 0.05$ and $L_{max} = 5{,}000$.

For each experiment we present four main results with respect to change over time: the accuracy, the number of neighbors k, the kernel width σ and how the decision boundary looks like at three specific key points in time (at 25k, 75k and 125k examples).

At first, we use the RBF data (see Appendix D.1.4) with 30 kernels and 4 classes. Figure 3.6 shows how the adaptive k approach achieves the highest accuracy along the whole stream, with k varying in the range $[4, 25]$, whereas the adaptive σ and the fixed (k and σ) approaches seem to have a similar accuracy.

	#classes	#attributes	stream 1 seed	stream 2 seed	params
hyperplane (HP)					
pure	binary	10	111		
concept drift (CD)	binary	10	111	154	$t_0 = 75$ k $w = 10$ k
sampling drift (SD)	binary	10	111		$\delta = 0.1233123$
CD & SD	binary	10	111	154	$t_0 = 75$ k $w = 10$ k $\delta = 0.1233123$
random trees (RT)					
pure	2-5	8	111		$depth = 15$
concept drift (CD)	2-5	8	111	154	$depth = 15$ $t_0 = 75$ k $w = 10$ k
sampling drift (SD)	2-5	8	111		$depth = 15$ $\delta = 0.1233123$
CD & SD	2-5	8	111	154	$depth = 15$ $t_0 = 75$ k $w = 10$ k $\delta = 0.1233123$
RBF					
pure	2-5	20	111		$kernels = 100$
concept drift (CD)	2-5	20	111	154	$kernels_1 = 200$ $kernels_2 = 250$ $t_0 = 75$ k $w = 10$ k
SEA					
pure	binary	3	-		$function = 1$
concept drift (CD)	binary	3	-	-	$function_1 = 1$ $function_2 = 2$ $t_0 = 75$ k $w = 10$ k

Table 3.1: The used data sets with their corresponding parameters for the experiments presented in Tables 3.2-3.5.

	IBL-DS	LWF	TWF	Win5k	Win5k ST	IBLStreams no ST	IBLStreams
HP	**.8685***	.8596	.8346	.8671	.8672	.8657	.8664
binary	(.0011)	(.0010)	(.0013)	(.0008)	(.0008)	(.0009)	(.0007)
RBF	.9432	.8248	.8687	.9819	.9793	**.9865**$^{\bullet}$	.9856$^{\bullet}$
binary	(.0024)	(.0025)	(.0015)	(.0002)	(.0002)	(.0002)	(.0002)
RBF	.9209	.7258	.7844	.9699	.9654	**.9789**$^{\bullet}$	.9774$^{\bullet}$
3 classes	(.0014)	(.0039)	(.0013)	(.0004)	(.0005)	(.0002)	(.0002)
RBF	.9314	.7367	.7991	.9721	.9683	**.9795**$^{\bullet}$	.9785$^{\bullet}$
4 classes	(.0028)	(.0039)	(.0013)	(.0003)	(.0004)	(.0003)	(.0002)
RBF	.9291	.7037	.7700	.9675	.9631	**.9766**$^{\bullet}$	.9753$^{\bullet}$
5 classes	(.0053)	(.0023)	(.0015)	(.0004)	(.0004)	(.0003)	(.0002)
RT	.6447	.6438	.6288	**.6501**	.6493	.6485	.6481
binary	(.0018)	(.0012)	(.0014)	(.0013)	(.0015)	(.0012)	(.0013)
RT	.4778	.4814	.4540	**.4872**	.4863	.4871	.4862
3 classes	(.0017)	(.0011)	(.0017)	(.0011)	(.0012)	(.0012)	(.0012)
RT	.4102	.4102	.3821	**.4212**	.4196	.4210	.4194
4 classes	(.0027)	(.0019)	(.0018)	(.0013)	(.0021)	(.0013)	(.0022)
RT	.3334	.3426	.3093	**.3474**	.3459	**.3474**	.3460
5 classes	(.0016)	(.0013)	(.0012)	(.0018)	(.0017)	(.0019)	(.0016)
SEA	**.9739***	.9586	.9520	.9703	.9719	.9720	.9734
binary	(.0004)	(.0005)	(.0003)	(.0003)	(.0004)	(.0005)	(.0004)

Table 3.2: Comparing IBLStreams with other IBL approaches on pure data streams.

	IBL-DS	LWF	TWF	Win5k	Win5k ST	IBLStreams no ST	IBLStreams
HP SD	.8790	.8730	.8703	.8834	.8863	.8855	**.8877**$^{\bullet}$
binary	(.0189)	(.0202)	(.0177)	(.01567)	(.0160)	(.0171)	(.01719)
RT SD	.6616	.6416	.6613	.6737	**.6743***	.6693	.6699
binary	(.0197)	(.0184)	(.0190)	(.01806)	(.0180)	(.0177)	(.01765)
RT SD	.5280	.5037	.5298	.5475	**.5483***	.5457	.5467
3 classes	(.0097)	(.0101)	(.0106)	(.01009)	(.0100)	(.0103)	(.01038)
RT SD	.4609	.4304	.4612	.4794	**.4804***	.4780	.4790
4 classes	(.0215)	(.0206)	(.0202)	(.01965)	(.0196)	(.0192)	(.01926)
RT SD	.4151	.3852	.4159	.4363	**.4372***	.4354	.4364
5 classes	(.0198)	(.0179)	(.0197)	(.01902)	(.0189)	(.0189)	(.01887)

Table 3.3: Comparing IBLStreams with other IBL approaches on streams with a simulated sampling drift.

	IBL-DS	LWF	TWF	Win5k	Win5k ST	IBLStreams no ST	IBLStreams
HP CD	**.8667**	.8584	.8335	.8654	.8656	.8641	.8648
binary	(.0009)	(.0010)	(.0015)	(.0010)	(.0007)	(.0008)	(.0012)
RBF CD	.8698	.6683	.6917	.9124	.8997	**.9313**$^{\bullet}$	.9259$^{\bullet}$
binary	(.0055)	(.0033)	(.0009)	(.0005)	(.0006)	(.0008)	(.0005)
RBF CD	.8367	.5597	.5892	.8778	.8610	**.9011**$^{\bullet}$	.8947$^{\bullet}$
3 classes	(.0036)	(.0032)	(.0018)	(.0006)	(.0009)	(.0013)	(.0009)
RBF CD	.8360	.5085	.5461	.8694	.8509	**.8937**$^{\bullet}$	.8876$^{\bullet}$
4 classes	(.0032)	(.0043)	(.0020)	(.0007)	(.0009)	(.0008)	(.0010)
RBF CD	.8263	.4684	.5041	.8556	.8349	**.8816**$^{\bullet}$	.8749$^{\bullet}$
5 classes	(.0036)	(.0048)	(.0027)	(.0008)	(.0009)	(.0008)	(.0008)
RT CD	.6305	.6319	.6126	**.6370**	.6362	.6357	.6353
binary	(.0011)	(.0015)	(.0011)	(.0013)	(.0011)	(.0010)	(.0012)
RT CD	.4823	.4863	.4566	**.4911**	.4900	.4907	.4899
3 classes	(.0021)	(.0014)	(.0014)	(.0016)	(.0015)	(.0016)	(.0015)
RT CD	.4084	.4104	.3796	**.4185**	.4172	.4183	.4171
4 classes	(.0036)	(.0021)	(.0021)	(.0019)	(.0019)	(.0019)	(.0019)
RT CD	.3598	.3645	.3321	**.3681**	.3672	**.3681**	.3672
5 classes	(.0020)	(.0012)	(.0016)	(.0015)	(.0017)	(.0016)	(.0018)
SEA CD	**.9709**	.9552	.9485	.9673	.9691	.9690	.9703
binary	(.0003)	(.0006)	(.0004)	(.0003)	(.0004)	(.0003)	(.0004)

Table 3.4: Comparing IBLStreams with other IBL approaches on streams with a simulated concept drift.

	IBL-DS	LWF	TWF	Win5k	Win5k ST	IBLStreams no ST	IBLStreams
HP	.8772	.8717	.8682	.8825	.8851	.8842	**.8865**$^{\bullet}$
binary	(.0145)	(.0155)	(.0139)	(.0118)	(.0122)	(.0128)	(.0128)
RT	.6679	.6487	.6674	.6807	**.6814***	.6765	.6775
binary	(.0151)	(.0152)	(.0153)	(.0146)	(.0146)	(.0145)	(.0144)
RT	.5444	.5186	.5442	.5629	**.5636***	.5607	.5614
3 classes	(.0175)	(.0176)	(.0174)	(.0160)	(.0159)	(.0160)	(.0159)
RT	.4787	.4484	.4775	.4970	**.4980***	.4954	.4965
4 classes	(.0176)	(.0177)	(.0173)	(.0166)	(.0168)	(.0166)	(.0165)
RT	.4460	.4157	.4434	.4639	**.4650***	.4624	.4635
5 classes	(.0202)	(.0183)	(.0202)	(.0187)	(.0187)	(.0185)	(.0185)

Table 3.5: Comparing IBLStreams with other IBL approaches on streams with both simulated drifts: a concept drift and a sampling drift.

A careful look at the adaptive σ and adaptive k approaches reveals that both σ and k tend to be decreasing and increasing in an analogous way. Although this behavior is not in a perfect match, it shows that when the adaptive k scheme decides to consider more data examples to give a better prediction, the adaptive σ scheme also tries to do the same by expanding the kernel width aiming to give a greater weight for more distant neighbors; the adaptive σ, however, gets stuck by encountering the fixed number of the instances k. Figure 3.7 shows how the decision boundary appears at three different points in time. It is apparent that the adaptive k has the most regular decision boundaries followed by the adaptive σ and the fixed σ, k. On the contrary, the nearest neighbor approach achieves lowest accuracy accompanied by the most irregular boundaries.

The second data stream uses the hyperplane data with two dimensions, see Appendix D.1.1. A slight concept drift in the middle of the stream is added, which makes the data simulate a slight hyperplane rotation, and a percentage of noise equal to 15% is also added to this stream. Similar to what has been observed in the previous experiment, we observe the same behavior for the different settings; we also observe that the adaptive k approach shows superiority during and after the concept drift, this fact is supported by the higher accuracy in Figure 3.8 and the regular decision boundary in Figure 3.9.

One may conclude that, despite the analogous performance of both variants, adaptive k is superior to adaptive σ. This result is justified as increasing k allows a larger number of neighboring instances to participate in the decision. Increasing σ, on the other hand, could just lead to increasing the effect of the neighbors but not the area of effect, especially when k is fixed.

3.5.3 IBLStreams versus state-of-the-art model-based methods

In this experiment, we compare IBLStreams with state-of-the-art learners, namely the Hoeffding tree [56] and the adaptive Hoeffding tree [23] for classification, which are explained in Appendix A.1. For regression tasks, we compare IBLStreams with AMRules [4], FIMTDD [86] and FLEXFIS [110], explained in Appendix A.2, Appendix A.3 and Appendix A.4, respectively.

Experiments are conducted with both real and synthetic data streams. Table 3.6 gives a brief overview of the data sets (and their corresponding parameters) used in the forthcoming experiments. Performance curves are averaged over 10 folds for

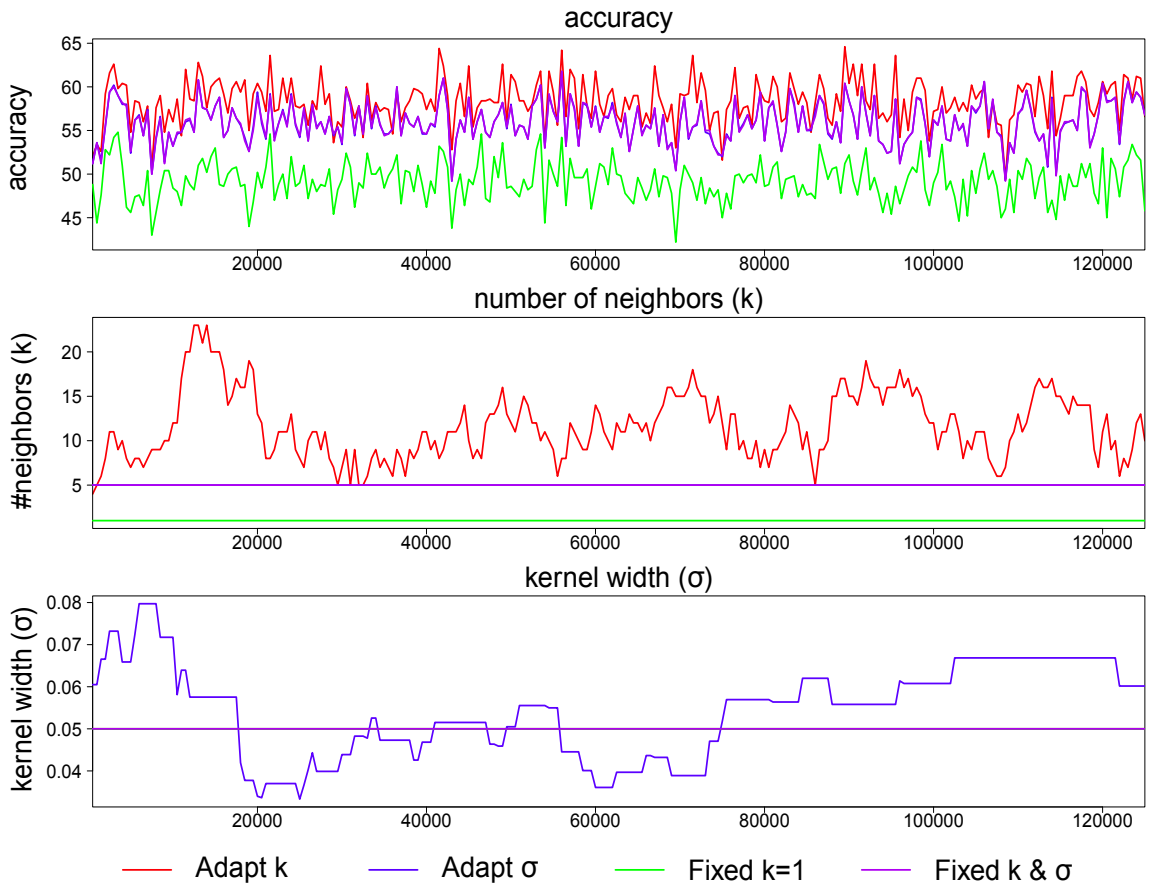


Figure 3.6: The change in performance, number of neighbors (k) and kernel width (σ) when IBLStreams is trained using the different adaptive strategies on the RBF data.

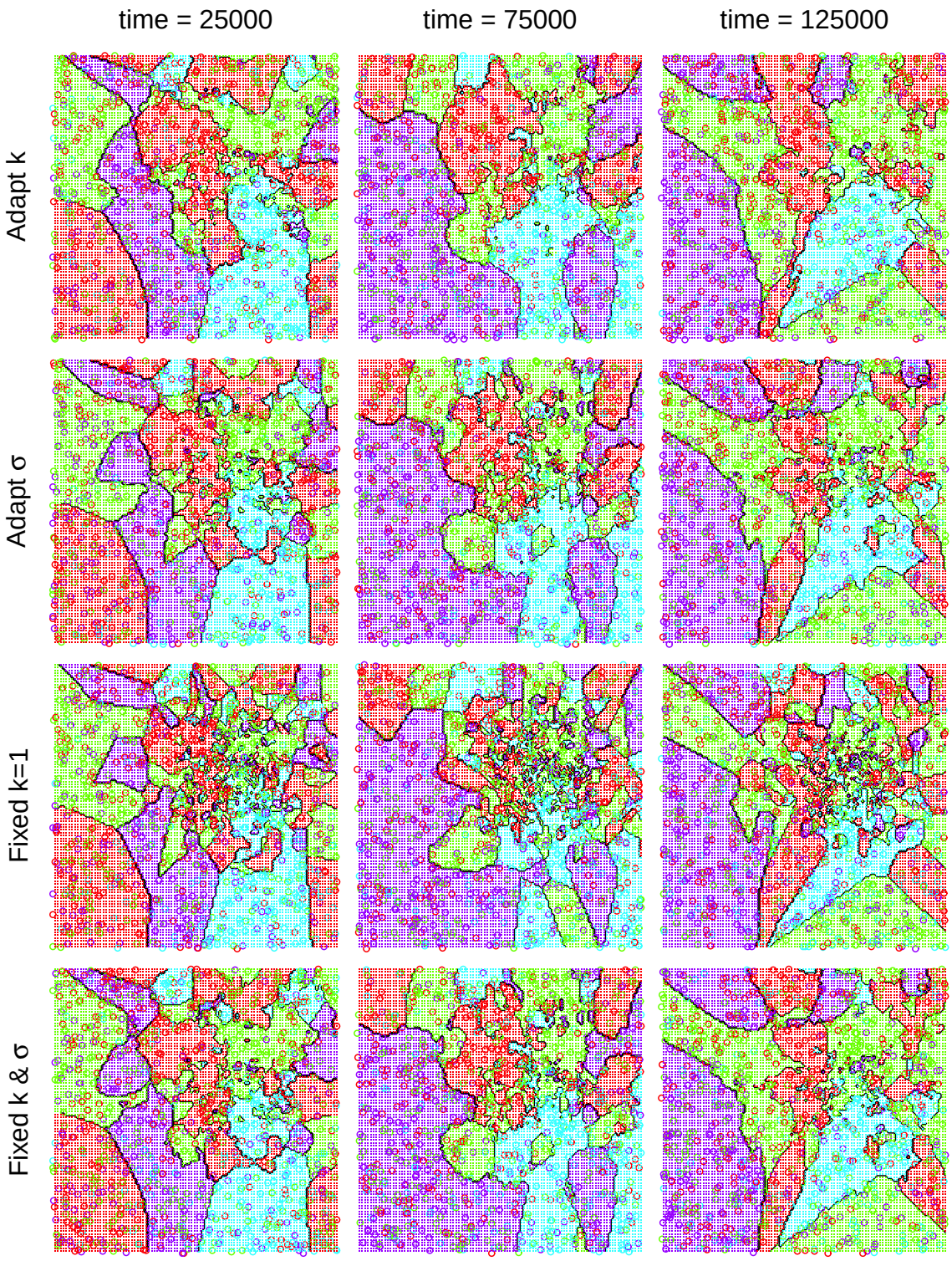


Figure 3.7: The decision boundaries of the different IBLStreams's adaptive strategies on the RBF data.

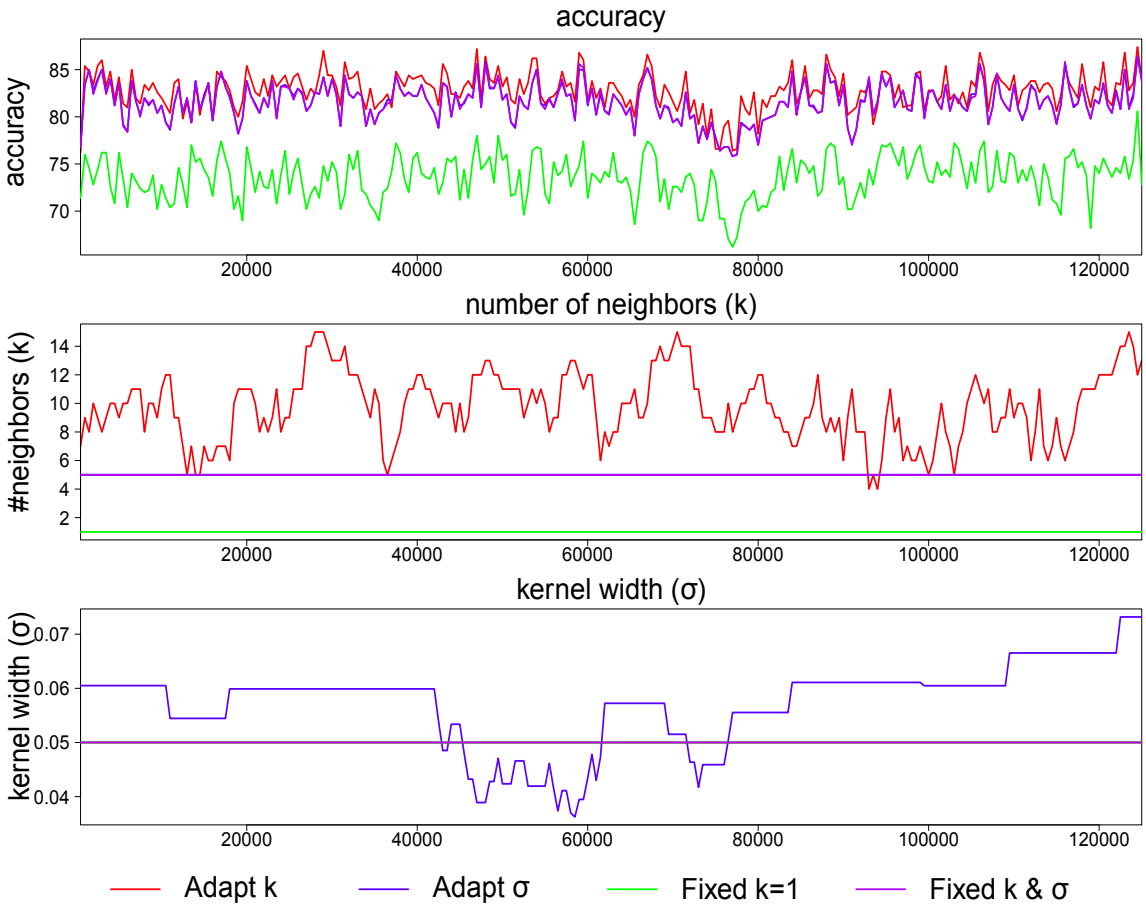


Figure 3.8: The change in performance, number of neighbors (k) and kernel width (σ) when IBLStreams is trained using the different adaptive strategies on the hyperplane data, with a concept drift.

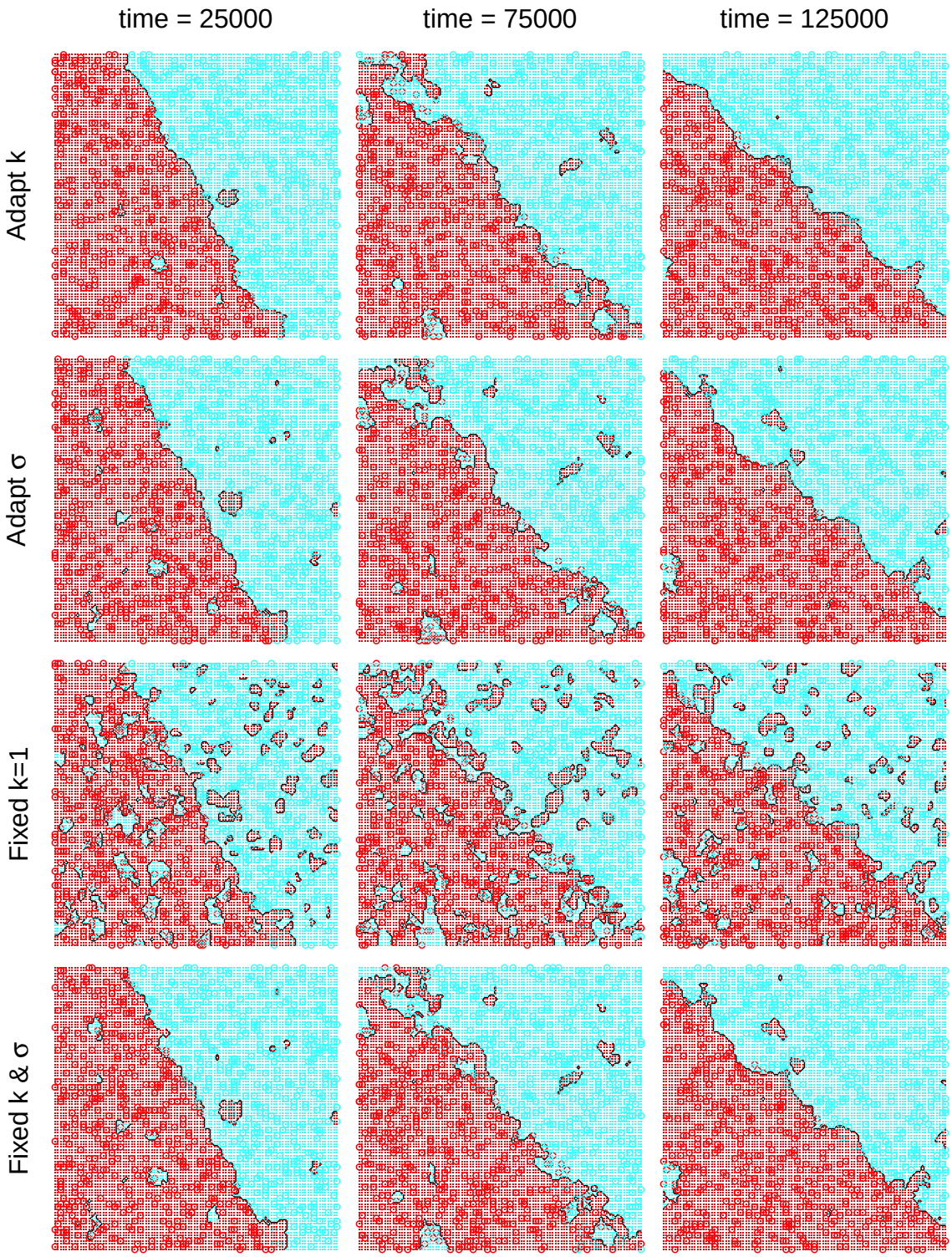


Figure 3.9: The decision boundaries of the different IBLStreams's adaptive strategies on the hyperplane data, with a concept drift.

synthetic data streams, and over 10 randomly shuffled versions of the data for real data sets.

3.5.3.1 Classification

Classification experiments involve the creation of the accuracy curves. In the following experiments, we used both Hoeffding trees and the adaptive Hoeffding trees in the default parameter setting[8]. IBLStreams is used in the Wk-NN mode for classification, for which we set the initial $k = 16$, the initial kernel width (for exponential and Gaussian kernels) $\sigma = 0.50$ and the maximum case base size $L_{max} = 5{,}000$. IBLStreams is applied in three variants:

C1: IBLStreams adaptive kernel width σ, with the Gaussian kernel

C2: IBLStreams adaptive number of neighbors k, with the equal weighting of neighbors

C3: IBLStreams with no adaptivity, i.e. by fixing k and σ, with the equal weighting of neighbors

Synthetic Data

We use two synthetic data streams, each of which is used twice, once as a pure stream and the second time with a simulated concept drift in the middle of the stream; the concept drift is simulated using MOA's ConceptDriftStream procedure, see Appendix D.2.1.

The size of each stream is set to be 125k instances, with a sliding window evaluation of the model's performance on the last 500 instances plotted every 500 instances. For the simulated drift experiments, we locate the drift at the center of the stream by setting $t_0 = 75k$ and $w = 10k$.

The first data stream uses the RBF data (see Appendix D.1.4) in four different difficulties: binary, 3-class, 4-class and 5-class classification problems. On the pure streams, as expected, Figure 3.10 shows that IBLStreams, in all its variations, is superior to both variations of the Hoeffding trees. Similarly, when simulating a drift on the RBF data, the adaptive variations of IBLStreams C1 and C2 have the best performance with almost 100% accuracy along the streams, with a small drop in

[8]gracePeriod $g = 200$, splitConfidence $c = 0$, tieThreshold $t = 0.05$, numericEstimator n=GAUSS10 and leafpreiction l= NBAdaptive

	#classes	length	#atts.	stream 1 seed	stream 2 seed	params
RBF						
pure	2-5	125k	20	111		$kernels = 100$
concept drift	2-5	125k	20	111	154	$kernels_1 = 200$ $kernels_2 = 250$ $t_0 = 75$ k $w = 10$ k
random trees						
pure	2-5	125k	8	111		$depth = 15$
concept drift	2-5	125k	8	111	154	$depth = 15$ $t_0 = 75$ k $w = 10$ k
dis. hyper.						
pure	distance squared cubed	125k 125k 125k	10	111		
concept drift	distance squared cubed&125k	125k 125k	10	111	154	$t_0 = 75$ k $w = 10$ k
cover type	7-classes	581,012	12	-	-	
mushroom	binary	8,124	21	-	-	
page blocks	5-classes	5,473	10	-	-	
StatLog	7-classes	58,000	9	-	-	
skin seg.	binary	245,057	3	-	-	
MAGIC	binary	19,020	10	-	-	
Parkinson's						
motor UPDRS	regression	5,875	18	-	-	
total UPDRS	regression	5,875	18	-	-	
slice loc.	regression	53,500	384	-	-	

Table 3.6: The used data sets with their corresponding parameters for the experiments presented in Figures 3.10-3.18.

performance at the center of the drift; the variations C1 and C2 have a better performance and a smooth adaptation pattern compared to the none-adaptive variation C3. Hoeffding trees, on the other hand, barely manage to learn from this data set, especially from the non-binary cases. Moreover, Hoeffding trees' drop in performance, during the drift, is more pronounced compared to C1 and C2. Finally, the adaptive Hoeffding tree is performing sightly better than the incremental Hoeffding tree, when it comes to learning and adapting to concept drifts, see Figure 3.11.

The second data stream uses the random trees data (see Appendix D.1.3) whose underlying model is a randomly constructed decision tree with class labels randomly assigned to the leaf nodes.

As depicted in Figure 3.12, the Hoeffding trees are now able to compete with IBLStreams when learning from pure streams. They reach an accuracy close to 60-80%, which is not unexpected given that Hoeffding trees are ideally tailored for this kind of data. Once again, Hoeffding trees are more affected by the concept drift than all variations of IBLStreams. The three variants of IBLStreams do not show any drastic decrease in terms of classification rate. In contrast, they continue to improve the performance during the drift, whereas both the Hoeffding tree and the adaptive Hoeffding tree lose up to 20% of their accuracy, with a very slow recovery pattern, see Figure 3.13.

Hence, Hoeffding trees are more affected by the concept drift; this can be observed by the pronounced *valley* in the performance curve when the drift occurs and by the long time they take to recover. IBLStreams recognizes and adapts to the concept drift quite early, and recovers to its original performance as soon as the drift is over.

Real Data

We use six real data sets for classification: cover type, mushroom, page blocks, StatLog, skin segmentation and MAGIC gamma telescope data, explained in Appendix D.3.

The results in Figure 3.14 and Figure 3.15 show that IBLStreams adaptive kernel width variant C1 is superior on all real streams, followed by the adaptive k variant C2. Hoeffding trees, on the other hand, either obtain a performance lower than that of C3, as in the page blocks data, or manage to reach the performance of C2 after at least seeing more than one third of the stream, as in the cover type, mushroom, StatLog and skin segmentation data. The adaptive Hoeffding tree manages to overcome the performance achieved by C1 only on the MAGIC gamma telescope data, close to the end of the data stream.

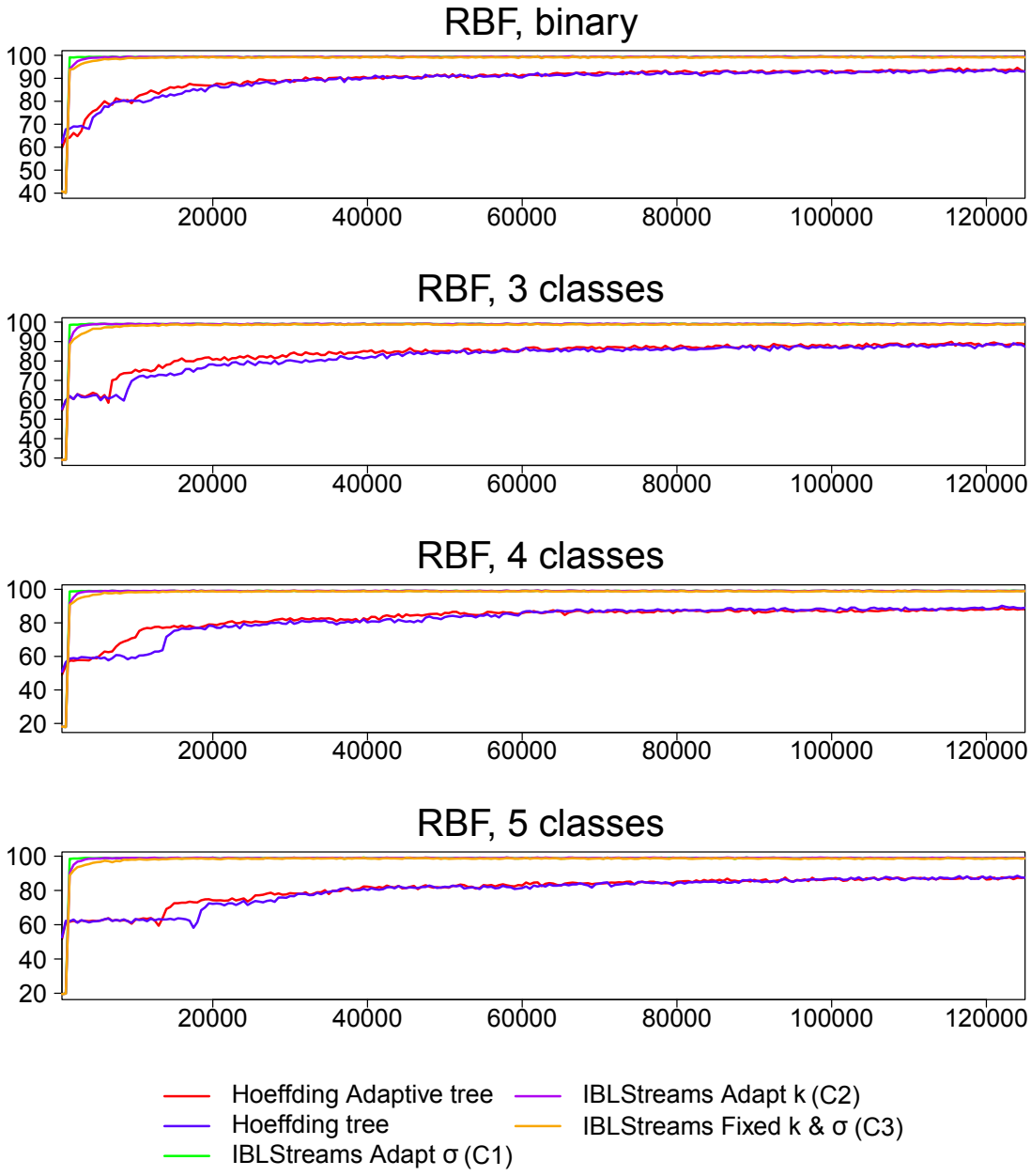


Figure 3.10: Classification rate on the pure RBF data set, 2, 3, 4 and 5 classes.

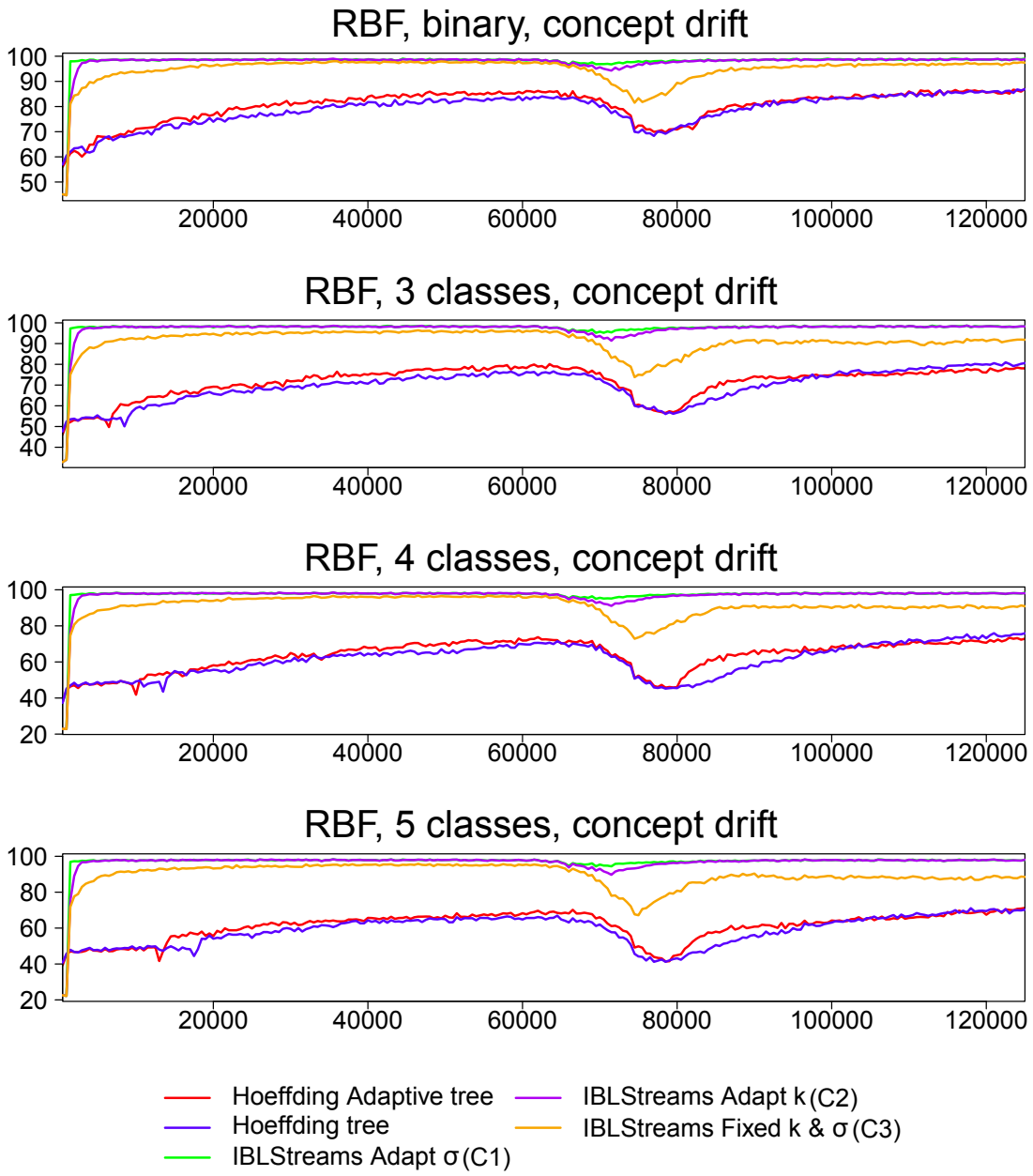


Figure 3.11: Classification rate on the RBF data set, 2, 3, 4 and 5 classes, with a concept drift.

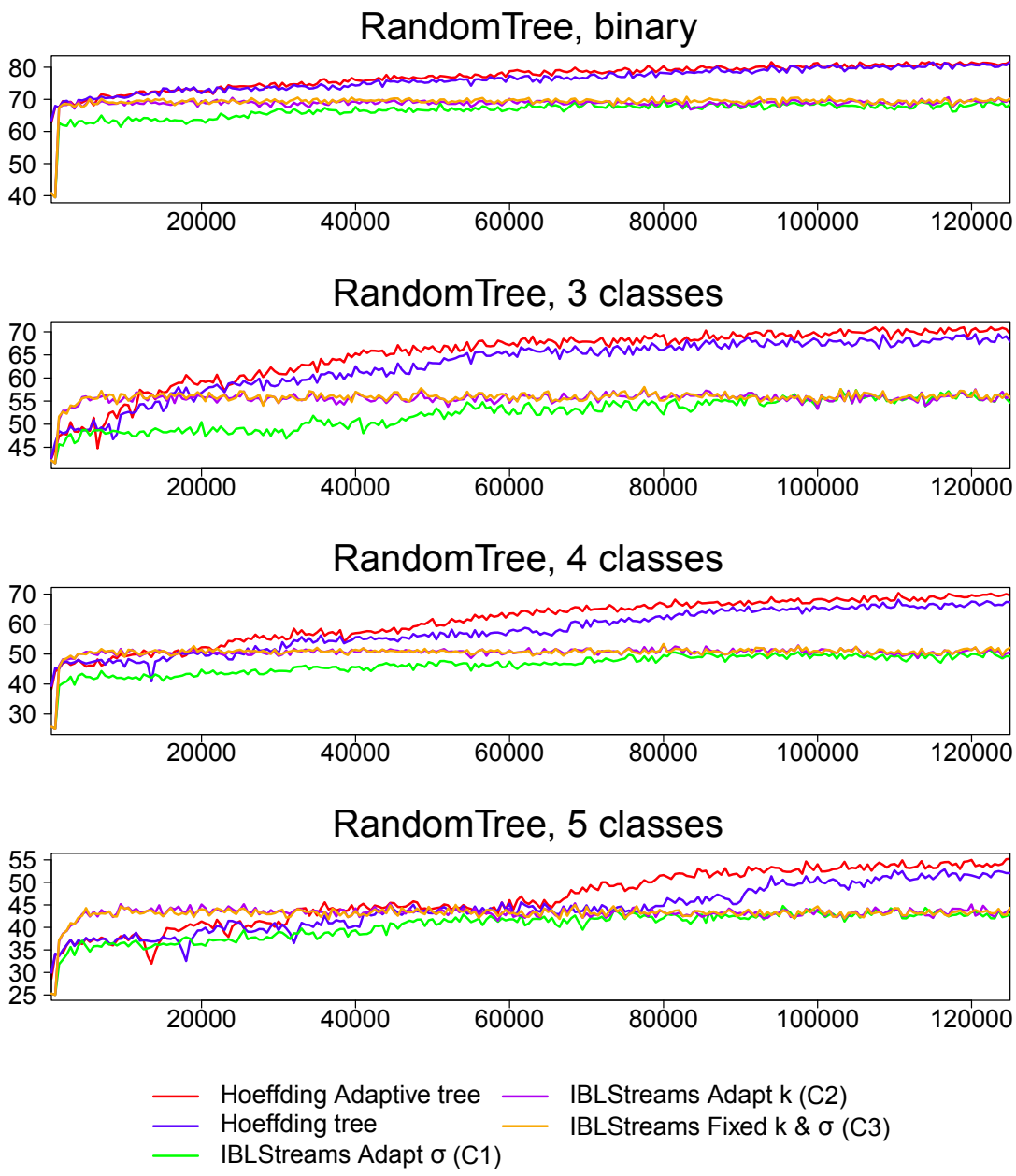


Figure 3.12: Classification rate on the pure random trees data set, 2, 3, 4 and 5 classes.

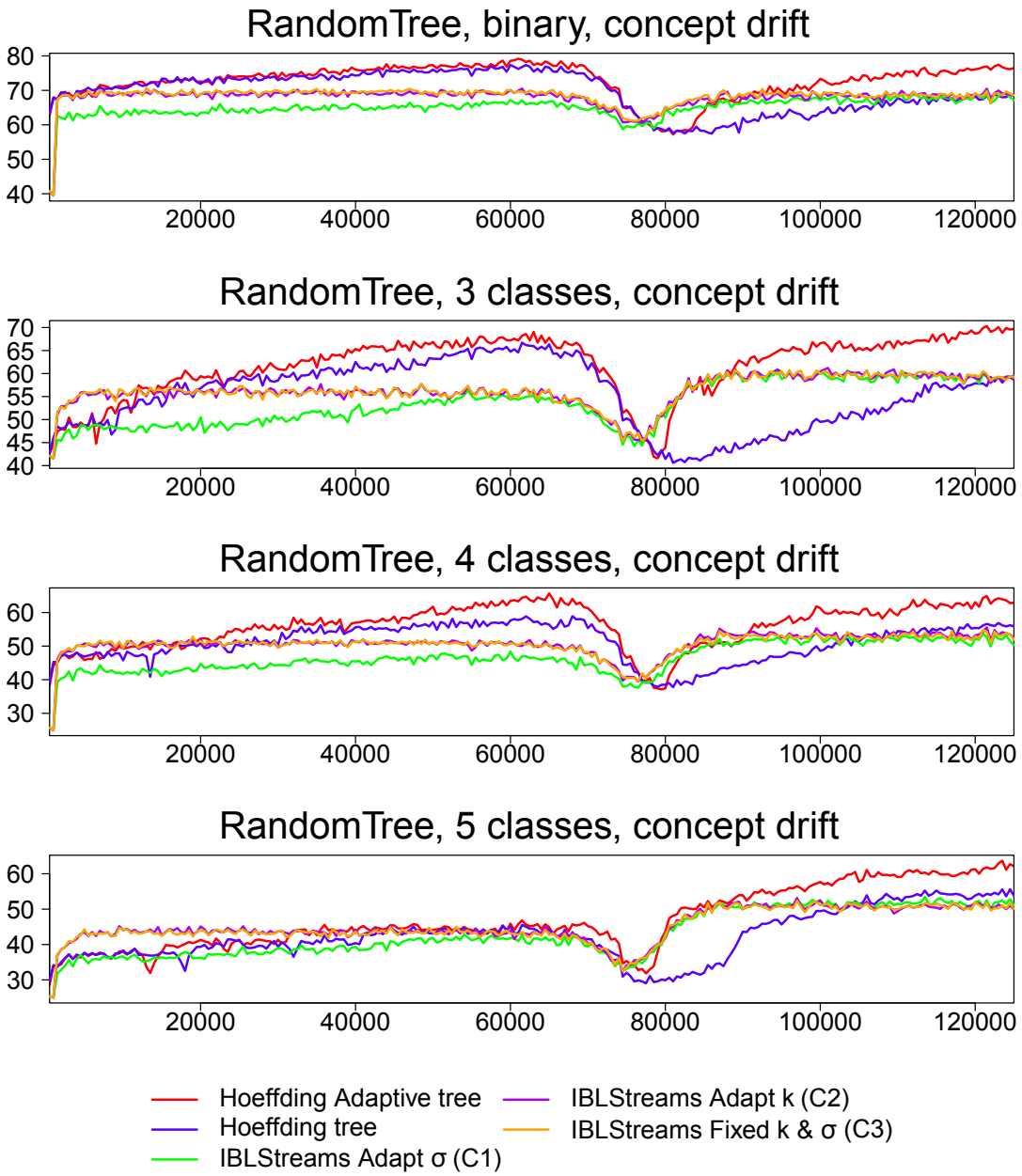


Figure 3.13: Classification rate on the pure random trees data set, 2, 3, 4 and 5 classes, with a concept drift.

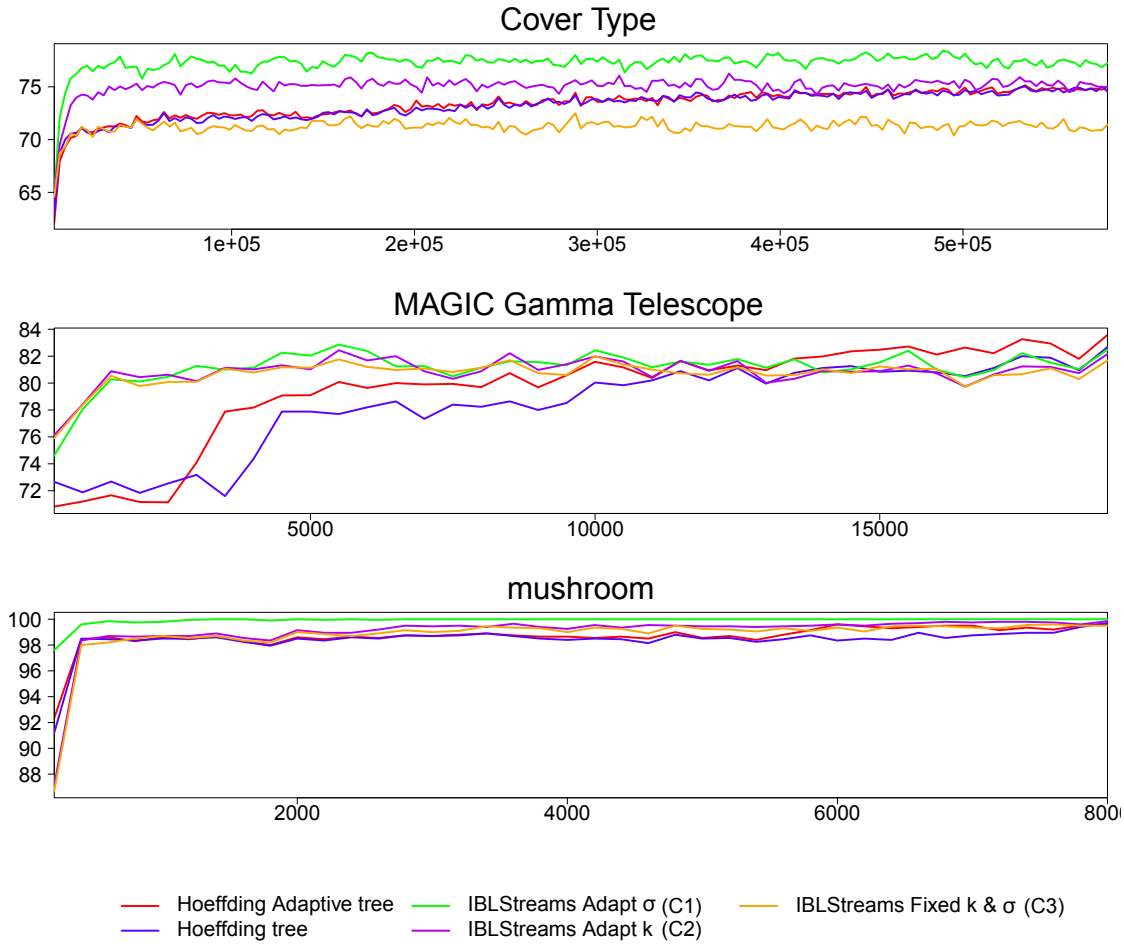


Figure 3.14: Classification rate on the real data sets: covertype, MAGIC gamma telescope and mushroom.

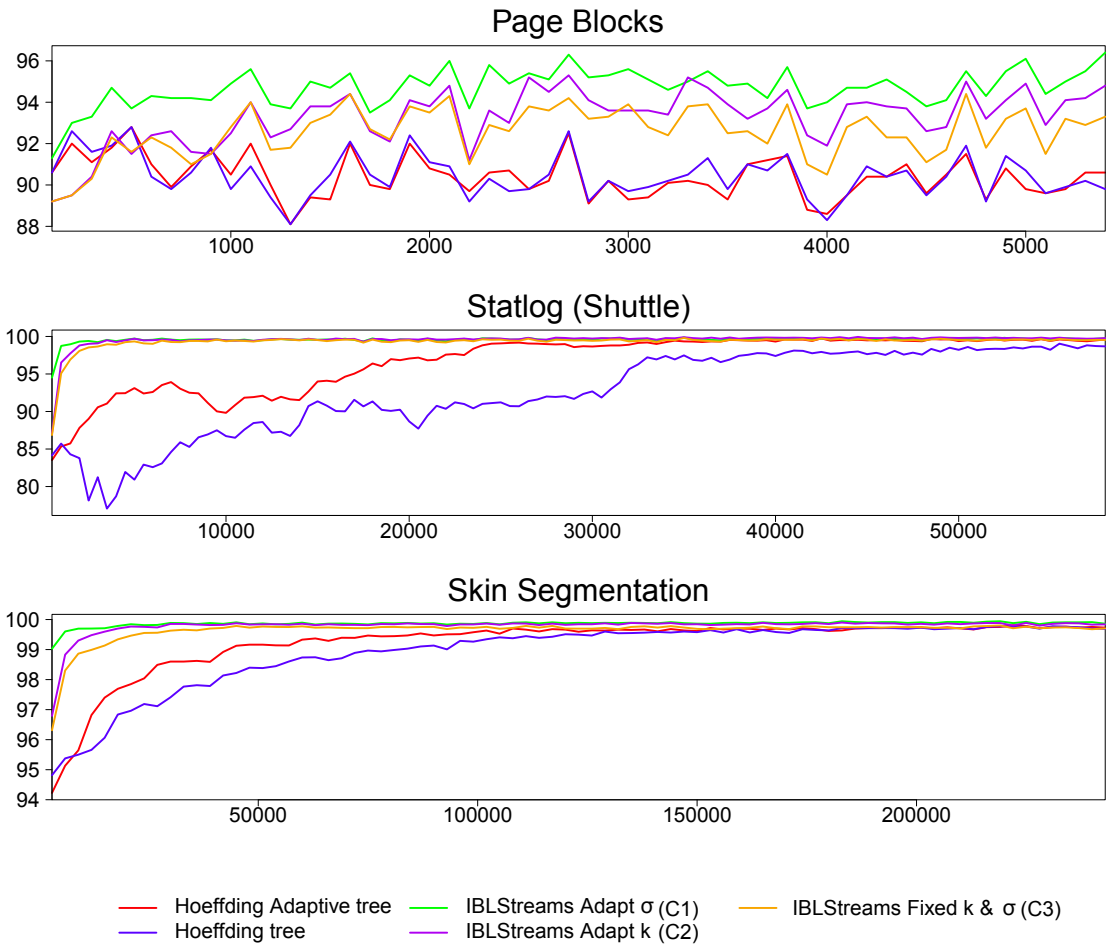


Figure 3.15: Classification rate on the real data sets: page blocks, StatLog (shuttle) and skin segmentation.

3.5.3.2 Regression

Regression experiments involve the production of error curves, in terms of the root mean square error (RMSE), when comparing the performance of IBLStreams with FLEXFIS, AMRules and FIMTDD.

FLEXFIS is implemented in Matlab and offers a function for finding optimal parameter values. We used this function to tune all parameters except the so-called "forgetting parameter", for which we manually found the value 0.999 to perform best; we also enable the pruning option. AMRules[9] and FIMTDD[10] were applied with their default parameters.

For the regression experiments, IBLStreams makes predictions using the locally weighted linear regression in three variants:

R1: IBLStreams adaptive kernel width σ, with the Gaussian kernel

R2: IBLStreams adaptive number of neighbors k, with the equal weighting of neighbors

R3: IBLStreams with no adaptivity, i.e. by fixing k and σ, with the equal weighting of neighbors

Synthetic Data

In this experiment, we use the distance to hyperplane generator in MOA, explained in Appendix D.1.2, which considers the distance to a hyperplane as a target value of the prediction task. As alternative to the simple distance, squared and cubed distance can also be considered. The stream size is set to be 125k instances, with a sliding window-evaluation of the model's performance on the last 500 instances plotted every 500 instances.

For IBLStreams, we set the initial $k = 16$, initial kernel width (for exponential and Gaussian kernels) $\sigma = 0.50$ and the maximum case base size $L_{max} = 5{,}000$.

Figure 3.16 shows the results when learning from the distance, squared and cubed distance to hyperplane. It is clear that the two variants of IBLStreams R1 and R2 have the smallest error on all streams, regardless of their difficulties. The R3 variation, however, has a slightly greater error compared to R1 and R2 on the cubed distance, which is clearly a more challenging task than the simple distance.

[9] predictionFunctionOption p=Adaptative, PageHinckleyAlpha a=0.005, tieThreshold t=0.05, splitConfidence c=1.0E-7 and learningRatio l=0.02

[10] splitCriterion s=VarianceReductionSplitCriterion, PageHinckleyAlpha a=0.005, tieThreshold t=0.05, splitConfidence c=1.0E-7 and learningRatio l=0.02

FLEXFIS and AMRules show a constant error along the streams which is comparatively higher than the error committed by IBLStreams' variations. FIMTDD starts with a relatively high error, which rapidly decreases to an acceptable performance, close to what is achieved by AMRules. Thereafter, its error decreases monotonically showing that it is still improving, yet it remains worse than IBLStreams.

Again, a simulated concept drift is used by mixing two synthetic streams. For the simulated drift experiments, we locate the drift at the center of the stream and we set $t_0 = 75k$ and $w = 10k$.

Figure 3.17 shows the results when applying the simulated drift on the distance, squared and cubed distance to hyperplane. The results show that all learners suffer from a decrease in their performance (an increase in the root mean square error). However, only IBLStreams restores its good performance on these problems by recovering to the same error level it reached before the drift. FLEXFIS also manages to restore its initial performance, which was not good in comparison to the other methods.

In these various examples, FLEXFIS, FIMTDD and AMRules are significantly outperformed by the different versions of IBLStreams. In fact, the RMSE is clearly lower for IBLStreams, not only under the normal conditions but also in cases of a concept drift.

Real Data

In this experiment we used two real data sets: the slice localization data and the Parkinson's telemonitoring, see Appendix D.3.10 and Appendix D.3.9. The latter is used twice, the first time by considering "motor UPDRS" as the target output and the second time by considering the "total UPDRS" attribute.

Figure 3.18 restates that both IBLStreams variants R1 and R2 have competitive performance, which was overcome by AMRules and FIMTDD only after seeing half of the stream on the Parkinson's data set. FLEXFIS, on the other hand, shows the worst performance on all real data experiments.

3.6 Discussion and Conclusion

This chapter presented our instance-based learner on data streams, IBLStreams, for tackling the tasks of classification and regression. IBLStreams is, to some extent, a continuation of IBL-DS; IBLStreams does not only exhibit the desirable properties of an adaptive system proposed by [57], but it also respects all the relevance factors

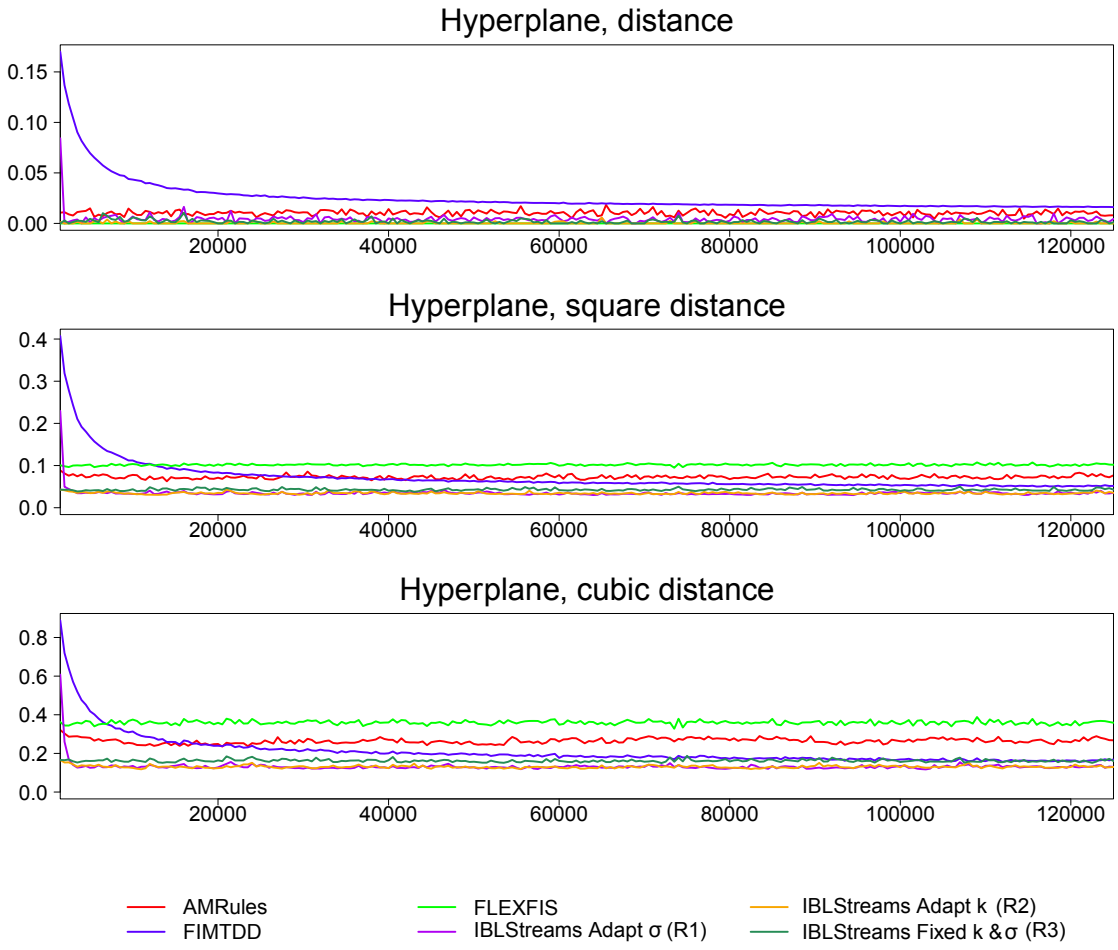


Figure 3.16: RMSE for the pure distance to hyperplane data (distance, squared and cubed distance).

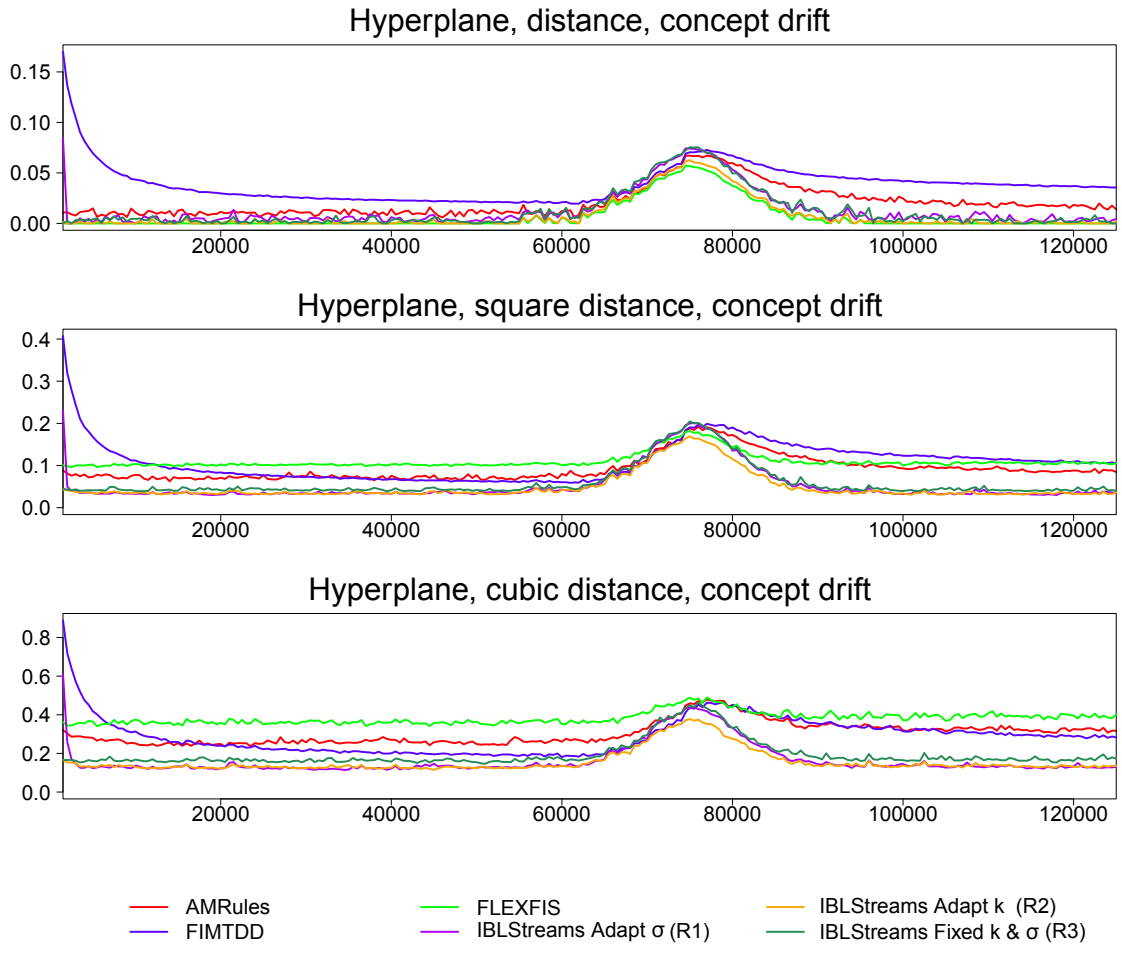


Figure 3.17: RMSE for the distance to hyperplane data (distance, squared and cubed distance), with a concept drift.

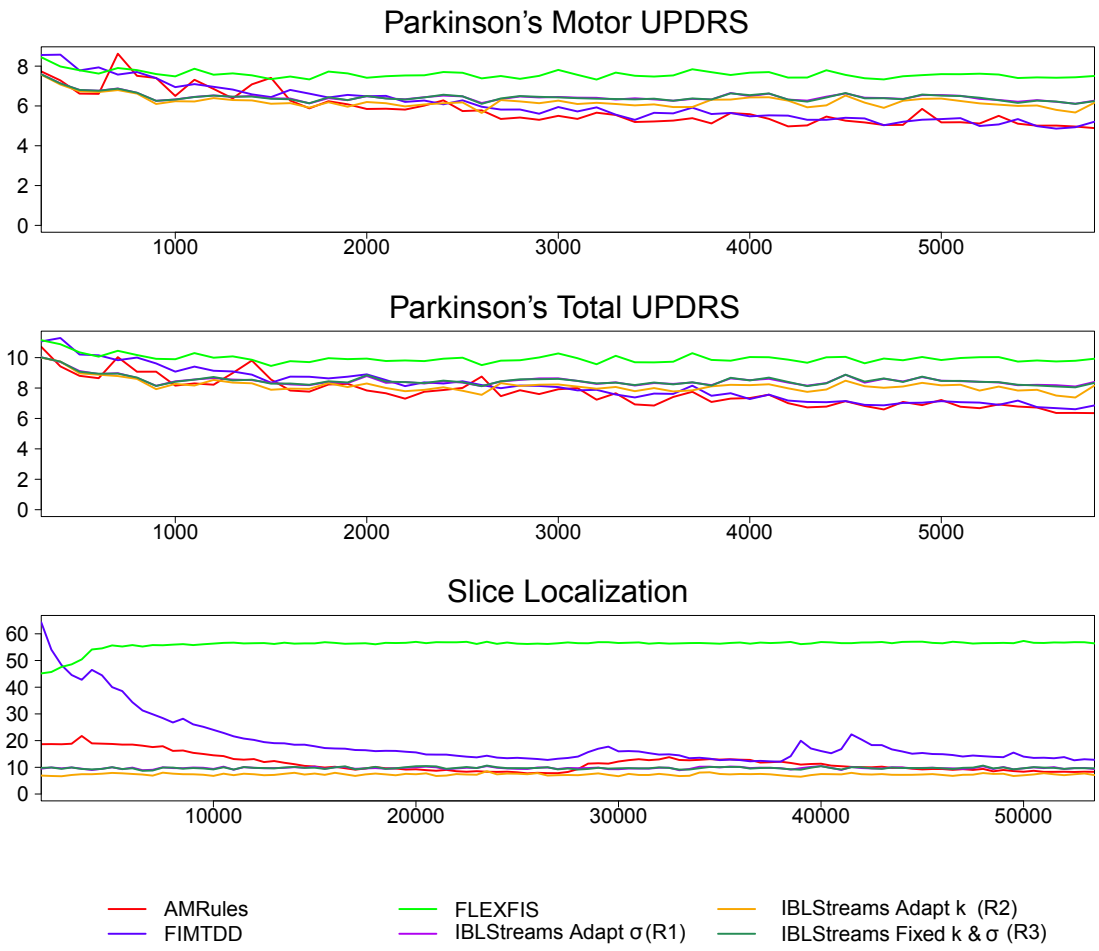


Figure 3.18: RMSE for the real data sets: Parkinson's motor UPDRS, Parkinson's total UPDRS and slice localization.

introduced by [17] for an IBL approach while maintaining a case base. In addition, parameter adaptation strategies are suggested for a dynamic fit to the current concept.

The experiments presented here suggest that IBLStreams competes with the state-of-the-art instance-based and model-based learners on data streams. Indeed, IBLStreams seems to be less "inert" when a concept drift occurs and, moreover, recovers its original performance more quickly when the drift comes to an end. This is arguably due to the advantage of not having to adapt a possibly complex model. Additionally, IBLStreams seems to quickly reach a high performance compared to the other learners, this is seen as a learning curve that rapidly reaches the saturation level. For these reasons, IBLStreams is comparable, if not superior, to the state-of-the-art instance-based and model-based learners on data streams

Chapter 4

Evolving Fuzzy Pattern Trees

This thesis starts by introducing the aspects of learning and the need to develop statistical solutions for transforming data to knowledge; it also shows how learning becomes challenging when the data becomes immense and continuous as in the streaming settings.

Chapter 3 shows an example of how one of the widely used machine learning techniques, namely the simple nearest neighbor approach, can be adapted to make the learning from non-stationary environments possible.

In this chapter, we present a different learning technique that draws its elements from the theory of fuzzy sets [184]. Fuzzy logic is a multivalued logic in which truth values go beyond the binary set $\{true, false\}$ or even the many-valued sets. In this type of logic, truth values are taken from the unit interval, with the ability to employ linguistic terms characterizing the space of underlying variables.

Models that utilize the theory of fuzzy sets are capable of expressing more realistic representation of world's problems than two-valued logic. Fuzzy logic allows propositions to be satisfied, unsatisfied or even partially satisfied; even more, satisfaction is quantified through the notion of membership degree for an element in a set, or the satisfaction degree of a proposition.

The advantage of fuzzy modeling becomes more obvious when considering fuzzy rule-based systems, which allow a fuzzy representation of the data; a fuzzy rules-based model allows rules to become partially satisfied. Because fuzzy logic allows sets to be identified with linguistic terms, the set of rules representing a concept becomes a generalized representation of the concept that is easier to interpret and to understand due to its expressibility in the natural language.

Hüllermeier [83] refers to the advantage of extending machine learning and data mining methods with fuzzy concepts. This extension leads to models that are more comprehensible and less complex; however, it is unlikely that the fuzzy extension

would lead to major improvements in the generalization performance, especially because these fields have reached a mature state.

Motivated by these developments, we propose an extended version of the fuzzy pattern trees suitable for learning from data streams. More specifically, by building on the (batch learning) algorithm for pattern tree induction as proposed in [146], we develop an evolving variant for the problem of binary classification.

This chapter is organized as follows: By way of background, Section 4.1 recalls some basic information about the theory of fuzzy sets. Section 4.2 presents a few data-driven approaches that utilize the aspects of fuzzy logic. Section 4.3 introduces the fuzzy pattern tree and its main induction methods, which we extend to the streaming setting in Section 4.4. Experimental results are presented in Section 4.5, prior to concluding the chapter in Section 4.6.

4.1 Introduction to Fuzzy Sets

Proposed as an extension to the set theory, fuzzy sets theory relaxes the crisp definition of the set membership "$\in$". This extension is motivated by the natural way we represent the continuity of our knowledge and belief, which suffers from information loss when discretized. Thus, an element now belongs to a set to some degree and is characterized by the notion of membership. The characteristic function of a subset A of a reference set Ω is defined as follows:

$$A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} , \tag{4.1}$$

whereas, a fuzzy set [184] is defined by a membership function A that assumes values in the unit interval:

$$A : \Omega \to [0, 1] \ .$$

A large number of fuzzy membership functions have been proposed in the literature [127], such as triangular function, β-function, S-function, trapezoidal, and Gaussian, among others. The *triangular functions* take the form of a triangle with the mode at b and the support at $[a, c]$:

$$A(x) = \begin{cases} \frac{x-b}{c-a} & \text{if } x \in [a, b] \\ \frac{c-x}{c-a} & \text{if } x \in [b, c] \\ 0 & \text{if } x \notin [b, c] \end{cases} . \tag{4.2}$$

4.1.1 Operations on Fuzzy Sets

Fuzzy sets require new definitions of three main set operations, intersection, union and complement in order to fit their multivalued nature. These definitions can be achieved based on the generalization of the logical operators. *Triangular norms* were formally defined as generalization of the triangular inequality in probability metric spaces [118]. Subsequently, triangular norms [96] were used as a substitute for the conventional conjunction and disjunction operations as shown in the following two definitions.

A *t-norm* is the generalization of the logical conjunction and it is a function $\top : [0,1] \times [0,1] \to [0,1]$ that needs to satisfy the following conditions:

- Commutativity: $\top(a,b) = \top(b,a)$
- Associativity: $\top(a, \top(b,c)) = \top(\top(a,b), c)$
- Monotonicity: if $a \leq c$ and $b \leq d$, then $\top(a,b) \leq \top(c,d)$
- Identity element: $\top(a,1) = a$

A *t-conorm* is the generalization of the logical disjunction and it is a function $\perp : [0,1] \times [0,1] \to [0,1]$ that needs to satisfy the following conditions:

- Commutativity: $\perp(a,b) = \perp(b,a)$
- Associativity: $\perp(a, \perp(b,c)) = \perp(\perp(a,b), c)$
- Monotonicity: if $a \leq c$ and $b \leq d$, then $\perp(a,b) \leq \perp(c,d)$
- Identity element: $\perp(a,0) = a$

Each t-*norm* has a dual t-*conorm* for which

$$\perp(a,b) = 1 - \top(1-a, 1-b) \ ,$$

or equivalently

$$\top(a,b) = 1 - \perp(1-a, 1-b) \ .$$

Table 4.1 depicts a group of the most popular triangular norms and conorms.

4.1.2 Aggregation Operations on Fuzzy Sets

The rich representation of fuzzy sets allows for a class of operators that aggregate multiple fuzzy sets into a single set. A fuzzy aggregation operator [127] is an n-ary $\psi : [0,1]^n \times [0,1] \to [0,1]$ operator for which the following holds:

- Monotonicity: $\psi(a_1, \ldots, a_n) \geq \psi(b_1, \ldots, b_n)$ if $a_i \geq b_i$, $i = 1, \ldots, n$
- Boundary conditions $\psi(0, \ldots, 0) = 0$ and $\psi(1, \ldots, 1) = 1$

Obviously, the set of fuzzy aggregation operators contains the set of triangular norms and conorms. The aggregation operators include: the compensatory operators [186], symmetric sums [58], averaging operators [59] and the ordered weighted averaging [181]; many data-driven approaches focus and utilize the last two.

The *weighted average* (WA) operator is an n-ary function $\texttt{WA} : [0,1]^n \to [0,1]$ identified by the vector $w = (w_1, \ldots, w_n) \in [0,1]^n$ with $\sum_{i=1}^n w_i = 1$ such that

$$\texttt{WA}(a_1, \ldots, a_n) = \sum_{i=1}^{n} w_i a_i \ .$$

Similarly, the *ordered weighted average* (OWA) [181] is an n-ary function $\texttt{OWA} : [0,1]^n \to [0,1]$ that takes the weighted average of the n arguments after sorting them; it is identified by the vector $w = (w_1, \ldots, w_n) \in [0,1]^n$ with $\sum_{i=1}^n w_i = 1$ such that

$$\texttt{OWA}(a_1, \ldots, a_n) = \sum_{i=1}^{n} w_i f(i, a_1, \ldots, a_n) \ ,$$

where the value $f(i, a_1, \ldots, a_n)$ is the ith smallest value in the vector $(a_1, \ldots, a_n)$. The OWA operator exhibits the property of generalizing other operators:

- The arithmetic mean: if $w = (1/n, \ldots, 1/n)$ then $\texttt{OWA}(a_1, \ldots, a_n) = \frac{1}{n} \sum_{i=1}^n a_i$
- The minimum operator : if $w = (1, \ldots, 0)$ then $\texttt{OWA}(a_1, \ldots, a_n) = \min(a_1, \ldots, a_n)$
- The maximum operator : if $w = (0, \ldots, 1)$ then $\texttt{OWA}(a_1, \ldots, a_n) = \max(a_1, \ldots, a_n)$

Notably, the $\texttt{t} - \texttt{DRA}$ is the smallest t-norm and the MIN is the largest, whereas t-conorms are bounded by the MAX and the $\texttt{co} - \texttt{DRA}$. The averaging operators WA and OWA take values in the wide spectrum of operators between the least strict t-norm and the most strict t-conorm:

$$\begin{gathered} \texttt{t} - \texttt{DRA} \leq \texttt{t} - \texttt{EIN} \leq \texttt{t} - \texttt{LUK} \leq \texttt{t} - \texttt{ALG} \leq \texttt{MIN} \\ \leq \texttt{WA}, \texttt{OWA} \leq \\ \texttt{MAX} \leq \texttt{co} - \texttt{ALG} \leq \texttt{co} - \texttt{LUK} \leq \texttt{co} - \texttt{EIN} \leq \texttt{co} - \texttt{DRA} \ . \end{gathered}$$

Operator	t-*norm*	t-*conorm*
Gödel	$\texttt{MIN}(a,b) = \min\{a,b\}$	$\texttt{MAX}(a,b) = \max\{a,b\}$
algebraic	$\texttt{t}-\texttt{ALG}(a,b) = ab$	$\texttt{co}-\texttt{ALG}(a,b) = a+b-ab$
Łukasiewicz	$\texttt{t}-\texttt{LUK}(a,b) = \max\{a+b-1,0\}$	$\texttt{co}-\texttt{LUK}(a,b) = \min\{a+b,1\}$
Einstein	$\texttt{t}-\texttt{EIN}(a,b) = \frac{ab}{2-(a+b-ab)}$	$\texttt{co}-\texttt{EIN}(a,b) = \frac{a+b}{1+ab}$
drastic	$\texttt{t}-\texttt{DRA}(a,b) = \begin{cases} b & \text{if } a=1 \\ a & \text{if } b=1 \\ 0 & \text{otherwise} \end{cases}$	$\texttt{co}-\texttt{DRA}(a,b) = \begin{cases} b & \text{if } a=0 \\ a & \text{if } b=0 \\ 1 & \text{otherwise} \end{cases}$

Table 4.1: Fuzzy triangular operators.

4.2 Data-Driven Fuzzy Modeling

The flexibility enjoyed by fuzzy sets made their introduction to engineering processes reasonable and beneficial; these fuzzy concepts helped many researchers in designing control systems that are easier to understand and interpret. Fuzzy logic did not remain restricted to engineering fields, but it has become a good candidate to transfer the expert's knowledge and experience into an expert system with a minimum information loss during the knowledge transfer. Such expert systems were mainly focusing on fuzzy rule-based systems, such as Mamdani controller [114] and Takagi-Sugeno-Kang Controllers (TSK) [165].

Artificial intelligence, on the other hand, does not only focus on representing the expert's knowledge as an intelligent system, but it is also concerned with discovery of this knowledge from observations, i.e., in a data-driven way, which is the main target of data mining and machine learning. Over the last three decades, there have been plenty of approaches that try to harvest interpretable models from data using fuzzy logic. Based on the supervised learning setting (see Section 2.2), we introduce in this section the main approaches that led to the idea of fuzzy pattern trees as introduced in [82].

4.2.1 Fuzzy Subsethood-Based Algorithm

The subsethood $S(A,B)$ [100] is a measure that tells to which degree the fuzzy set A belongs to the fuzzy set B

$$S(A,B) = \frac{\sum_{x\in U} \min(\mu_A(x), \mu_B(x))}{\sum_{x\in U} \mu_A(x)} \ . \tag{4.3}$$

Subsethood-based algorithm (SBA) is a rule-based framework that induces rules modeling the training data; it employs the subsethood measure to learn these rules.

For a supervised learning problem with m dimensions and l classes $C_1, \ldots, C_l$, SBA creates the set of rules:

$$\begin{aligned}
\text{Rule}_1 \quad \text{IF} \quad & A_1 \text{ IS } (A_{11} \text{ OR } \ldots \text{ OR } A_{1m}) \text{ AND } \ldots \text{ AND} \\
& A_n \text{ IS } (A_{n1} \text{ OR } \ldots \text{ OR } A_{nm}) \\
\text{THEN} \quad & \text{predict class } \quad C_1 \\
& \vdots \\
\text{Rule}_l \quad \text{IF} \quad & A_1 \text{ IS } (A_{11} \text{ OR } \ldots \text{ OR } A_{1m}) \text{ AND } \ldots \text{ AND} \\
& A_n \text{ IS } (A_{n1} \text{ OR } \ldots \text{ OR } A_{nm}) \\
\text{THEN} \quad & \text{predict class } \quad C_l \ ,
\end{aligned} \tag{4.4}$$

such that $A_{j1}, \ldots, A_{jm}$ are the fuzzy terms for the variable A_j, the AND and the OR logical operators are replaced by the t-norm (`MIN`) and the t-conorm (`MAX`) operators. Thereafter, SBA evaluates the similarity between each fuzzy term A_{ji} and each class C_v according to (4.3); only fuzzy terms A_{ji} whose subsethood $S(A_{ji}, C_u) > \alpha \in [0, 1]$ in the rule Rule_u are kept and the others are removed.

The SBA model (4.4) lacks the ability to take into account the relative contribution of each term of each variable towards the consequence part. Rasmani and Shen [131] introduce a weighted SBA (WSBA) to solve this problem by weighting each linguistic term with its respective contribution. The relative weight for the linguistic term B_i of the fuzzy variable[1] B with respect to the fuzzy set A is given by

$$w(A, B_i) = \frac{S(A, B_i)}{\max_{j=1\ldots m} S(A, B_j)} \ . \tag{4.5}$$

The resulting default fuzzy rules take the form

$$\begin{aligned}
\text{Rule}_u \quad \text{IF} \quad & A_u \text{ IS } (w(C_u, A_{u1})A_{u1} \text{ OR } \ldots \text{ OR } w(C_u, A_{um})A_{um}) \\
& \text{AND } \ldots \text{ AND} \\
& A_n \text{ IS } (w(C_u, A_{n1})A_{n1} \text{ OR } \ldots \text{ OR } w(C_u, A_{nm})A_{nm}) \\
\text{THEN} \quad & \text{predict class } \quad C_u \ ,
\end{aligned} \tag{4.6}$$

which are learned similar to those learned in the SBA approach.

4.2.2 Fuzzy Decision Trees

Fuzzy decision trees are the fuzzy variation of the well-known decision trees [129], which are induced in a recursive manner by replacing a leaf node with an internal

[1] A linguistic variable is a variable whose values are words (linguistic terms) instead of numbers, and each linguistic term is characterized by a fuzzy set.

node, labeled by an attribute, and number of branches leading to child (leaf) nodes. Each of these leaf nodes can be reached from the internal node after satisfying a logical predicate assigned to its corresponding branch; the set of logical predicates (labeling all edges) are mutually exclusive, such that for each value of the corresponding attribute, there is only one branch whose predicate is satisfied.

Fuzzy decision trees while maintaining a tree structure similar to that of decision trees, differ from decision trees in four important issues: *(i)* The logical predicates assigned to the tree's branches are extended to become fuzzy predicates, i.e., a predicate can be satisfied to some degree $u \in [0, 1]$. *(ii)* Allowing paths to be partially satisfied leads to the relaxation of the mutual exclusion condition, which means that multiple paths can be simultaneously active with different degrees. *(iii)* Decisions concluded at the leaf nodes of all active paths need to be aggregated with respect to their degrees of satisfaction/activation. *(iv)* The information gain measured in the decision tree's induction methods has to be extended to a measure that considers both the attributes in their fuzzy representation and the graded activation of the multiple paths. Yuan and Shaw [183] and Janikow [88] propose two different methods for inducing fuzzy decision trees; these approaches differ mainly in addressing these four discussed issues.

4.3 Fuzzy Pattern Trees

Fuzzy pattern trees (FPTs) are introduced in [82] as tree-like structures induced to solve supervised learning problems. This model relaxes the restrictions imposed in the previously motivated rule-based systems, in which only a conjunction between the different attributes is allowed. Fuzzy decision trees suffer from the same restriction because each path from a leaf node to the root is formed as a chain of conjunction operators, and the different active paths are then distinctively aggregated.

Fuzzy pattern trees, on the contrary, allow the application of arbitrary operators on the fuzzy representation of the object's attributes. In this way, a more comprehensive space of rules is explored which promises to find a better fit to the training data.

An FPT is a binary tree that represents one class in the output domain, each of the tree's internal nodes contains one fuzzy aggregation operator from the categories: t-norms, t-conorms, and averaging operators. These operators aggregate the scores realized in the left and the right sub-trees of an internal node and then propagate the result to the parent node. Each leaf node contains a fuzzy term on one of the

input attributes. In this way, a learning example is observed at all leaf nodes, which then propagate the fuzzy membership degrees upwards; internal nodes recursively aggregate their inputs until the final aggregation at the root node. As a result, each pattern tree forms a hierarchical logical description of the represented class, and its compact representation offers the tradeoff between the correctness and the interpretability of the induced model.

For a multiclass learning problem, a standard reduction scheme, such as one-vs-rest decomposition, can be applied to transform the problem into a set of binary problems and then solve them using a set of FPTs.

Fuzzy pattern trees allow t-norms and t-conorms to be chosen from the different types of operators shown in Table 4.1, except the drastic norm. By allowing the internal nodes to choose averaging operators (`WA` and `OWA`) instead of t-/co-norms, a flexible aggregation of the node's operands is facilitated through constituting the possible convex combinations of fuzzy terms.

An alternative to the original pattern trees' induction algorithm [82] is proposed and developed by Senge and Hüllermeier in [146]; they also introduce an FPT variant for regression problems in [145]. Moreover, Senge proves in his thesis [144] that the fuzzy pattern trees are universal approximators with an infinite VC dimension, i.e., for any real-valued function f there is an FPT that approximates it.

Finally, it is worth mentioning that independent of the fuzzy pattern trees, the same type of fuzzy model structure was introduced in [182] under the name "fuzzy operator tree".

In the following, we show the basic concepts of fuzzy pattern trees as presented in [146, 145]. For a binary classification problem, an instance is a vector $\boldsymbol{x} \in \mathcal{X}^m$, and each domain $\mathcal{X}_i$ is discretized through fuzzy partitioning into n_i fuzzy sets $F_{i,j} : \mathcal{X}_i \to [0, 1]$. Each training example is defined as $(\boldsymbol{x}, y) \in \mathcal{X}^m \times \mathcal{Y}$, where $y \in \mathcal{Y} = \{\ominus, \oplus\}$ is the class label.

Leaf nodes are labeled by the fuzzy sets $F_{i,j}$, which is the jth fuzzy set of the ith attribute. An example $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$ is fuzzified into the vector

$$(f_{1,1}, \ldots, f_{1,n_1}, \ldots, f_{m,1}, \ldots, f_{m,n_m}) \ ,$$

such that the $f_{i,j}$ is membership degree of the attribute x_i in the fuzzy set $F_{i,j}$. Each internal node contains one operation θ from the set of the allowed operations

$$\Psi = \{\texttt{MIN}, \texttt{t} - \texttt{ALG}, \texttt{t} - \texttt{LUK}, \texttt{t} - \texttt{EIN}, \texttt{MAX}, \texttt{co} - \texttt{ALG}, \texttt{co} - \texttt{LUK}, \texttt{co} - \texttt{EIN}, \texttt{WA}, \texttt{OWA}\} \ .$$

With fuzzy sets residing at the leaf nodes, membership degrees are propagated to parent nodes, which propagate the results on their turn after aggregating them depending on the operation they hold. The consecutive recursive propagation of internal results leads the final prediction to reach the root node. This result represents the belief for an instance to belong to the modeled class. Predictions for binary classification problems are then determined after thresholding. Figure 4.1 presents an example of fuzzy pattern trees.

4.3.1 Bottom-Up Induction of Fuzzy Pattern Trees

The original proposal of FPT [82] is accompanied with two algorithms that induce the FPT in a bottom-up manner. The two methods start by the definition of the following sets: The set of primitive trees $\mathbf{P} = \{F_{i,j}\}$, such that a primitive tree is a one-node tree labeled by a fuzzy term. The second set is the set of candidate trees $\mathbf{C}^t$ at iteration t. The initial set of candidates $\mathbf{C}^0$ contains the primitive trees that are most similar to the class to be learned. For a given data set, the performance of an FPT is measured by the similarity between the tree's outputs and the true outputs using the Jaccard measure.

The bottom-up induction approach follows one of two algorithms:

- The first approach induces the so-called "simple pattern trees". It restricts the candidate set $\mathbf{C}^t$ to contain only the best performing current tree. The simple FPT is generated by extending the current tree in $\mathbf{C}^{t-1}$ through aggregating it using all available operators $\theta \in \Psi$ with all primitive trees $S \in \mathbf{P}$. Only when the new tree improves the performance, i.e., increases the similarity between its predictions and the target class, the current tree is discarded and the new tree is adopted for the next iteration. This process ends when further extensions do not lead to any improvements. Notice that each iteration increases the tree's depth by one. As a result, the induced tree is an unbalanced binary tree because each internal node has at least one leaf node as a child.

- The second approach induces "general pattern trees". It allows the set of candidate trees $\mathbf{C}^t$ to contain the best L trees instead of a single candidate tree. In each iteration, an aggregation is attempted between each candidate tree $C \in \mathbf{C}^{t-1}$ and each tree in the "slave set" $\mathbf{S}^{t-1}$ using all available operators $\theta \in \Psi$. The slave set $\mathbf{S}^{t-1}$ contains in addition to primitive fuzzy terms the M best performing trees from $\mathbf{S}^{t-2}$ and $\mathbf{C}^{t-1}$.

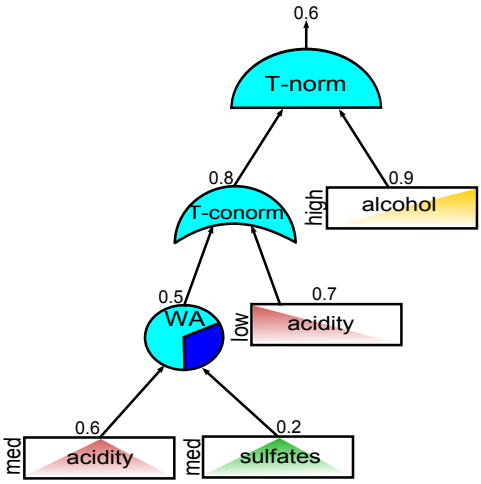


Figure 4.1: An example of a fuzzy pattern tree, modeling the quality of a red wine based on its chemical properties, see [145].

4.3.2 Top-Down Induction of Fuzzy Pattern Trees

This top-down induction method is introduced in [146] as an alternative to the original algorithm [82]. This section explains in more detail how the top-down method works, and how nominal and numerical attributes can be discretized and fuzzified, as suggested in [146].

This top-down induction method (depicted in Figure 4.2) utilizes two sets of trees, similar to the bottom-up approach. The first set is the set of primitive trees $\mathbf{P}$, each of which is a one-node tree labeled by a fuzzy term. The second set is the set of candidate trees $\mathbf{C}^t$ at iteration t. The initial set of candidates $\mathbf{C}^0$ contains the best B primitive trees. The parameter B is set by default to $B = 5$.

The algorithm iterates over all quadruples

$$(C, l, \theta, P) \in \left(\mathbf{C}^{t-1} \times leafs\,(C) \times \Psi \times \mathbf{P}\right) \quad ,$$

each of which corresponds to one possible extension $_{l}\swarrow^{\theta}\searrow_{P}$. These quadruples cover the space of trees resulted from replacing each leaf node l, in the set of leaf nodes $leafs\,(C)$ of each candidate tree C in the current candidate set $\mathbf{C}^{t-1}$, with an internal node that aggregates the leaf l with the primitive tree $P \in \mathbf{P}$ by means of the operation $\theta \in \Psi$.

After evaluating all possible extensions, only the best B candidates are preserved for the next iteration. The iterations terminate when the maximum number of iterations t_{max} is reached or when further extensions cannot improve the performance by more than $\epsilon\%$ with $\epsilon = 0.0025$.

Procedure Top-DownBatchFPTInduction

1: $\mathbf{P} = \{F_{ij}\}, i = 1, ..., m; j = 1, ..., n_i$
2: $\mathbf{C}^0 = \arg\max_{P \in \mathbf{P}} \text{B}[Sim(P, \Upsilon)]$
3: **for** $t = 1$ **to** t_{max} **do**
4: $\quad \mathbf{C}^t = \mathbf{C}^{t-1}$
5: $\quad$ **for all** $(C, l, \theta, P) \in (\mathbf{C}^{t-1} \times leafs\,(C) \times \Psi \times \mathbf{P})$ **do**
6: $\quad\quad \mathbf{C}^t = \mathbf{C}^t \cup ExtendLeafInTree(C, l, \theta, P)$
7: $\quad$ **end for**
8: $\quad \mathbf{C}^t = \arg\max_{C \in \mathbf{C}^t} \text{B}[Performance(C, \Upsilon)]$
9: $\quad$ **if** $\max_{C \in \mathbf{C}^t}(Performance(C, \Upsilon)) < (1 + \epsilon) \cdot \max_{C \in \mathbf{C}^{t-1}}(Performance(C, \Upsilon))$ **then**
10: $\quad\quad$ **break**
11: $\quad$ **end if**
12: **end for**
13: **return** $\arg\max_{C \in \mathbf{C}^t} \text{B}[Performance(C, \Upsilon)]$

Figure 4.2: Top-down induction algorithm for learning fuzzy pattern trees, as introduced in [146].

The performance evaluation of each candidate tree PT is performed based on the similarity between its predictions and the true outputs of the training data. A fuzzy pattern tree can be seen as a fuzzy subset, since its output lies in the unit interval; this tree is then compared with the subset of the training examples Υ, which contains only examples from the modeled class. For an FPT modeling the positive class $\oplus$ and for a training example $(\boldsymbol{x}, y) \in \mathcal{D}$, the subset Υ is defined as

$$\Upsilon(x) = \begin{cases} 1 & \text{if } y = \oplus \\ 0 & \text{otherwise} \end{cases} \quad . \tag{4.7}$$

To evaluate the performance of a pattern tree PT, its predictions on the examples $(\boldsymbol{x}_i, y_i)$ are compared to Υ. The performance measure is given by the additive inverse of the root mean squared distance, which is shown to yield a reasonable fit [146]:

$$Performance(\text{PT}, \Upsilon) = 1 - \sqrt{\frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (\text{PT}(\boldsymbol{x}_i) - \Upsilon(\boldsymbol{x}_i))^2} \quad . \tag{4.8}$$

Finally, the tree from the candidate set that achieves the maximum performance, Equation (4.8), is returned as a result of the induction process.

Before the induction, attribute values go through two main operations before playing any role in the induction and prediction processes. The first operation is the discretization process in order to limit the number of primitive trees per attribute. The second operation is the fuzzification in which attribute values become membership values. The top-down induction process simply discretizes each attribute into three fuzzy sets associated with the linguistic terms "low", "medium" and "high", see [146]. The "medium" fuzzy set $F_{i,medium}$ is a triangular function (4.2) with the parameter a, b and c found by maximizing the absolute Pearson correlation between $F_{i,medium}$, on the ith attribute, and the subset Υ (4.7). The other two sets "low" and "high" are defined for the ith attribute on the domain $X_i = [a, b]$ as $F_{i,low}(x) = \min(\max(\frac{b-x}{b-a}, 0), 1)$ and $F_{i,high}(x) = \min(\max(\frac{x-a}{b-a}, 0), 1)$, respectively.

Attributes with nominal domains are modeled by characteristic functions, degenerated fuzzy sets, one for each value v of that attribute. The characteristic function $A_{i,v}(x)$ for the value v of the ith attribute takes the form:

$$A_{i,v}(x) = \begin{cases} 1 & \text{if } x = v \\ 0 & \text{otherwise} \end{cases} . \tag{4.9}$$

The top-down induction process is mainly motivated by the advantage of making small adaptations on the current tree, thus guaranteeing a better coverage and an extensive exploration of the space of pattern trees. This behavior is especially advantageous when compared to the bottom-up approach, which merges two candidate trees with an operator, leading to an arbitrary jump in the search space.

It is worth mentioning that Senge and Hüllermeier [147] propose, in a recent publication, a modification for the pattern tree's induction procedure in order to accelerate the learning process. This acceleration is based on the application of the Hoeffding race [115] and heuristics similar to our potential refinement, introduced in Section 4.4.3.

4.4 Evolving Fuzzy Pattern Trees

To meet the requirements of learning from a data stream, we develop an evolving version of fuzzy pattern tree learning, in which model adaptation is realized by anticipating possible local changes of the current model, and confirming these changes through statistical hypothesis testing.

The basic idea of our evolving version of fuzzy pattern tree learning (eFPT) is to maintain an ensemble of pattern trees, consisting of the current (active) model and a set of neighbor models. The current model is used to make predictions, while

the neighbor models can be seen as anticipated adaptations: they are kept ready to replace the current model in case of a drop in performance, caused, for example, by a change in the concept to be learned as explained in Section 2.3. More generally, the current model is replaced, i.e., the anticipated adaptation is realized, whenever its performance appears to be significantly worse than the performance of one of the neighbor models; in this case, the set of neighbors is also revised.

More specifically, the set of neighbor models is always defined by the set of trees that are "close" to the current model. Hence the term "neighbor" refers to the tree derivable from this model by means of a single "edit operation", namely an expansion or a pruning step; a detailed explanation of how the neighbor trees are generated is given by the algorithm *GenerateNeighborTrees* shown in Figure 4.3. Like in batch algorithm, an expansion replaces a leaf l of the current tree by a three-node pattern tree ${}_l\swarrow^{\theta}\searrow_R$. A pruning step is essentially undoing an expansion. More precisely, each inner node, except the root, can be replaced by one of its sibling nodes, i.e., the subtree rooted by this node is lifted by one level, while the subtree rooted by the other sibling is pruned.

Looking at the neighbor trees as the local neighborhood of the current model in the space of pattern trees, the algorithm performs an adaptive local search in this space and, therefore, is somewhat comparable to a discrete variant of a swarm-based search procedure. The collective movement of the active model and its "surrounding" neighbor models in the search space is similar, for example, to the flocking of a group of birds. Moreover, the pruning step does not necessarily lead to a tree that was already observed in the induction process, thus allowing the search to be more flexible.

4.4.1 Performance Monitoring and Hypothesis Testing

At each time step t, the error rate of the current model PT and, likewise, of all neighbors is calculated on a sliding window consisting of the last n training examples $\{(\boldsymbol{x}_{t-i}, y_{t-i})\}_{i=0}^{n-1}$:

$$\tau_t = \frac{1}{n} \sum_{i=0}^{n-1} \left(y_{t-i} - \hat{y}_{t-i}\right)^2 \quad , \tag{4.10}$$

where $\hat{y}_i$ is the prediction of y_i. The length of the sliding window, n, is a parameter of the method; as a default value, we use $n = 100$, which is large enough from the point of view of statistical hypothesis testing (see below) and small enough to enable a fast reaction to changes of the data generating process.

Storing the predictions and the true class labels, τ_{t+1} can easily be updated in an incremental way:

$$\tau_{t+1} \leftarrow \tau_t + \frac{1}{n}\left((y_{t+1} - \hat{y}_{t+1})^2 - (y_{t-n+1} - \hat{y}_{t-n+1})^2\right) \quad , \tag{4.11}$$

where y_{t+1} and y_{t-n+1} are the true class labels of the most recent and the oldest observations in the current window, respectively.

In order to decide whether or not one of the neighbor trees is superior to the current model, each update of the error rates is followed by a statistical hypothesis test. Let $\tau^{(0)}$ and $\tau^{(1)}$ denote, respectively, the error rate of the current model and a neighbor tree. We then test the null hypothesis $H_0 : \tau^{(0)} \leq \tau^{(1)}$ against the alternative hypothesis $H_1 : \tau^{(0)} > \tau^{(1)}$. A suitable test statistic for doing so is

$$t = \frac{\tau^{(0)} - \tau^{(1)}}{SE}$$
$$SE = \sqrt{\frac{(s^{(0)})^2 + (s^{(1)})^2}{n}} \quad ,$$

based on one-tailed Welch's t-test for two samples with equal size and unequal variance, where n is the sample size (window length). The statistics $s^{(0)}, s^{(1)}$ are the standard deviations of the error rates $\tau^{(0)}, \tau^{(1)}$, respectively. Standard deviations are also updated incrementally on the sliding window, as presented in Appendix E. The test statistic t follows Student's t-distribution with

$$\text{d.f.} = \frac{\left(\left(s^{(0)}\right)^2 + \left(s^{(1)}\right)^2\right)^2 (n-1)}{\left(s^{(0)}\right)^4 + \left(s^{(1)}\right)^4}$$

degrees of freedom. The null hypothesis is rejected if t exceeds a critical threshold $\mathcal{T}_{d.f.,\alpha}$; note that α controls the proneness of the algorithm toward changes of the model: The smaller α, the less often the model will be changed (by default, we use $\alpha = 0.01$).

The above test is conducted for each neighbor tree; if H_0 is rejected in at least one of these tests, the current model is replaced by the alternative tree for which the test statistic was the highest. In this case, the fuzzy partitions of the numerical attributes are recomputed and the refinements in Subsection 4.4.3 are applied based on the data of the current window to the newly selected tree.

A Bonferroni-corrected significance level can also be applied here, however, we tend to avoid applying such a correction as the multiple tests, associated with the neighbor trees, are positively correlated. This correlation occurs because each pair of the neighbor trees differ only with at most two edit operations; thus, such a correction would increase the Type II error and subsequently decrease the power of the test.

4.4.2 Summary of the Algorithm

The algorithm for evolving fuzzy pattern tree (eFPT) learning on data streams is summarized in Figure 4.4. The main steps of this algorithm are as follows:

1. In the initialization phase, the first pattern tree is learned by applying the top-down batch induction algorithm *Top-DownBatchFPTInduction* on a small set of training examples. The current model is initialized with this tree as shown in Figure 4.2.

2. The set of neighbor trees is generated for the current model using the *GenerateNeighborTrees* procedure depicted in Figure 4.3.

3. Upon the arrival of a new example, the sliding window is shifted and the error rates for the current model and all neighbors are updated, see lines 7 to 12 in Figure 4.4.

4. The error rates of the neighbors of the current model are compared, see line 14 in Figure 4.4.

5. If a neighbor is significantly better than the current model, the latter is replaced by the former; in this case,

   (a) the primitive pattern trees are reinitialized,

   (b) the operators used in the pattern trees are optimized (e.g., by searching for more fitting triangular norms and conorms, as in the case when only a representative set of norms is considered),

   (c) the set of neighbor trees is again recomputed, see Figure 4.3.

6. Loop at step 3

4.4.3 Refinements on the Neighbor Trees Generation

The computational complexity of our eFPT algorithm critically depends on the size of the model ensemble, i.e., the number of neighbor trees. While monitoring the performance of a single tree can be done quite efficiently, the overall costs may become high due to the potentially large number of trees that have to be monitored and compared to the current model. More specifically, the number of neighbor trees resulting from one extension step is thus $O(|\Psi| \cdot |\mathbf{P}| \cdot |leafs\,(C)\,|)$, and the number of

Procedure GenerateNeighborTrees(C)

1: {Initialization}
{Every primitive pattern tree is labeled by a Fuzzy subset $F_{i,j}$ associated with attribute A_i}
2: $\mathbf{P} = \{F_{ij}\}, i = 1, ..., m; j = 1, ..., n_i$
3: $\mathbf{N} = Null$
4: {Creating the neighbor extension trees}
5: {Loop on each leaf of the current tree,
on each available operator and on each primitive pattern tree}
6: **for all** $(l, \theta, P) \in (leafs(C) \times \Psi \times \mathbf{P} \backslash \{l\})$ **do**
7: $\quad \mathbf{N} = \mathbf{N} \cup ExtendLeafInTree(C, l, \theta, P)$
8: **end for**
9: {Creating the neighbor pruning trees}
10: {Loop on each internal node of the current tree}
11: **for all** $node \in internalNodes(C)$ **do**
12: $\quad$ {Replacing the chosen node by its children nodes}
13: $\quad \mathbf{N} = \mathbf{N} \cup ReplaceNode(C, node, child_1)$
14: $\quad \mathbf{N} = \mathbf{N} \cup ReplaceNode(C, node, child_2)$
15: **end for**
16: **return** N

Figure 4.3: Algorithm for generating neighbor trees.

trees resulting from one pruning step is $2 \cdot |internalNodes\,(C)\,|$. Additional costs are caused by the re-computation of the neighbor models, which becomes necessary after the replacement of the current model.

In the following, we propose two refinements of the above algorithm, both of which are meant to reduce the computational complexity by reducing the number of neighbor models. Because this number mainly depends on two factors, namely the number of leaf nodes of the current model and the number of operators, an obvious solution is to reduce either of these factors.

Selecting Leaf Nodes

The eFPT induction algorithm constructs a neighbor tree by either expanding or pruning a leaf node of the current model. Here, we try to reduce the complexity by allowing these edit operations only for a subset of promising candidates. In order to select this subset, we apply a heuristic that estimates the potential influence of a leaf on the tree's output. More specifically, this heuristic tries to give an approximate answer to the following question: Provided we allow a leaf node L in a pattern tree

Evolving Fuzzy Pattern Tree

{Initialization}
$C = Top{-}DownBatchFPTInduction()$
$\mathbf{N} = GenerateNeighborTrees(C)$
{New instance from the stream is present}
while incoming instance t **do**
  {Update the error rate for the current tree}
  $\tau_t^{(current)} = \tau_{t-1}^{(current)} + \frac{1}{n}L(y_t, \hat{y}_t^{(current)}) - \frac{1}{n}L_(y_{t-n}, \hat{y}_{t-n}^{(current)})$
  {Loop on each neighbor tree}
  for all $N_k \in \mathbf{N}$ **do**
    {Update the error rate for each neighbor tree}
    $\tau_t^{(k)} = \tau_{t-1}^{(k)} + \frac{1}{n}L(y_t, \hat{y}_t^{(k)}) - \frac{1}{n}L_(y_{t-n}, \hat{y}_{t-n}^{(k)})$
  end for
  {Testing the null hypothesis that the current error rate is lower than that of all neighbor trees}
  if $\exists N_k \in \mathbf{N}$: Reject $H_0(\tau_t^{(current)} < \tau_t^{(k)})$ **then**
    {A neighbor tree with a lower error rate is found}
    $C = N_k$
    {Recompute all primitive pattern trees}
    $\mathbf{P} = \{A_{ij}\}, i = 1, ..., m; j = 1, ..., n_i$
    $OptimizeUsedOperator(C)$
    $\mathbf{N} = GenerateNeighborTrees(C)$
  end if
end while

Figure 4.4: The induction algorithm of the evolving fuzzy pattern trees.

PT to be expanded, i.e., to replace L by a subtree $N = {}_L\swarrow^\theta\searrow_R$, what improvement can be expected from this modification?

An optimistic answer to this question can be given by assuming that N will produce optimal outputs, namely $N(\boldsymbol{x}) = 1$ for positive and $N(\boldsymbol{x}) = 0$ for negative examples. Based on this assumption, the *potential* of a leaf node L is defined in terms of its *average relative* improvement:

$$\mathrm{POT}(L) = \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \begin{cases} \frac{\mathrm{PT}'(\boldsymbol{x})-\mathrm{PT}(\boldsymbol{x})}{1-L(\boldsymbol{x})} & \text{if } y = \oplus \text{ and } L(\boldsymbol{x}) \neq 1 \\ \frac{\mathrm{PT}(\boldsymbol{x})-\mathrm{PT}'(\boldsymbol{x})}{L(\boldsymbol{x})} & \text{if } y = \ominus \text{ and } L(\boldsymbol{x}) \neq 0 \\ 0 & \text{otherwise} \end{cases},$$

where PT' is the pattern tree after the expansion of L. Based on this conception of the potential of a leaf, we modify our algorithm by considering only the p leaf nodes

with highest potential; p is a parameter that has to be defined by the user (our default value is $p = 3$). As a result, the number of neighbor trees resulting from one extension step will drop to $O(|\Psi| \cdot |\mathbf{P}| \cdot p)$, which becomes a constant number of neighbor trees along the stream, independent of the size of the current model.

The same heuristic has also been applied in [144] in order to accelerate the induction process of the fuzzy pattern tree, albeit its application in a batch mode.

Retaining Operators

Another idea to reduce the number of expansions is to restrict the set of operators θ. More specifically, we provisionally retain some operators: Instead of trying all logical operators right away, we only try the largest (least extreme) t-norm `MIN` and the smallest t-conorm `MAX` (in addition to the two averaging operators). Only in case `MIN` is selected as an optimal operator, we also try the other (more extreme) t-norms; likewise, if `MAX` is selected, the other t-conorms are tried, and the best one is adopted. The basic assumption underlying this procedure is that, if any of the t-norms (t-conorms) is the most appropriate operator, the algorithm will select `MIN` (`MAX`) in the first step, because this is the "closest" among the available operators.

4.5 Empirical Evaluation

In this section, we compare our evolving fuzzy pattern trees with IBLStreams, described in Chapter 3, and with the Hoeffding tree (see Appendix A.1) in terms of performance, model size, and the handling of concept drift. Both variations of the Hoeffding tree are used, the incremental [56] and the adaptive version [23].

The empirical evaluation is performed using the MOA framework (see Appendix B), for which we offer an implementation of both eFPT and IBLStreams. MOA includes data stream generators, different methods for classifier evaluation and also offers several classifiers, including an implementation for both versions of Hoeffding trees.

eFPT is used with the default settings (the size of the sliding window for the statistical hypothesis testing $n = 100$ and the number of leaf nodes with highest potential $p = 3$), the initial tree is learned in a batch mode on a window of length $eFPT_{init} = 500$, unless stated differently in Table 4.2. IBLStreams is used with the adaptive k settings with a base size of $IBL_{size} = 5000$ instances, unless stated differently in Table 4.2. Hoeffding trees are used with their default settings.

Our experimental evaluation has three main targets: *(i)* comparing the performance of eFPT with other evolving learners, *(ii)* comparing the size of the pattern

	#classes	length	#atts.	stream 1 seed	stream 2 seed	params
RBF						
pure	2-5	125k	20	111		$kernels = 100$
concept drift	2-5	125k	20	111	154	$kernels_1 = 200$ $kernels_2 = 250$ $t_0 = 75$ k $w = 10$ k
random trees						
pure	2-5	125k	8	111		$depth = 15$
concept drift	2-5	125k	8	111	154	$depth = 15$ $t_0 = 75$ k $w = 10$ k
dis. hyper.						
pure	distance	125k	10	111		
	squared	125k				
	cubed	125k				
concept drift	distance	125k	10	111	154	$t_0 = 75$ k
	squared	125k				$w = 10$ k
	cubed	125k				
mushroom	binary	8,124	21	-	-	$eFPT_{init} = 200$ $IBL_{size} = 200$
skin seg.	binary	245,057	3	-	-	$eFPT_{init} = 3000$ $IBL_{size} = 3000$
MAGIC	binary	19,020	10	-	-	$eFPT_{init} = 500$ $IBL_{size} = 500$

Table 4.2: The used data sets with their corresponding parameters for the experiments presented in Figures 4.5- 4.13.

tree with that of the induced Hoeffding trees and *(iii)* studying the effect of the different parametrization and refinements on the performance of eFPT.

4.5.1 Performance Comparison

In these experiments, the Test-then-train evaluation (see Appendix B.2) procedure for measuring the prediction accuracy is employed. The performed experiments are not only conducted with real data sets, but also with synthetic data.

For the performance comparison, we use Łukasiewicz operators as an initial set of triangular norms and conorms when operator retraining is enabled. A loose significance level $\alpha = 0.25$ is used, which is a justified decision as shown in the following subsections.

4.5.1.1 Synthetic Data

We use the three data generators offered by MOA: hyperplane, RBF and random trees, see Appendixes D.1.1, D.1.3 and D.1.4. Similar to the evaluations performed in Section 3.5, all synthetic experiments are performed by randomly generating the model underlying each data set and fixing it, thereafter results are averaged over 10 folds. The sampled instances in each of these folds are randomly generated using different seeds, hence, we generate folds with independent and identically distributed data samples.

In the first part of the experiments, we use these data generators in their pure form, without any simulated change, as seen in Figure 4.5.

The first experiment uses data taken from a hyperplane generator. eFPT, as well as the other approaches, manages to learn the concept behind the hyperplane data with a relatively good accuracy measure.

RBF data, on the other hand, seems to be difficult to fit, not only for eFPT but also for the competitive methods. This is not a surprising result because such a data can be best fit by a generative model or by a local approach such as the nearest neighbor, as confirmed by the good performance of IBLStreams. In the third synthetic experiment, we use a random tree generator, which constructs a random decision tree by making random splits on attribute values, to produce examples. Obviously, this generator is favorable for the Hoeffding tree, which is confirmed by the better performance of Hoeffding trees compared to that of eFPT and IBLStreams.

In the second part of the experiments, we use the same synthetic data generators with a simulated concept drift (see Appendix D.2.1) using the ConceptDriftStream

procedure offered by MOA. As depicted in Figure 4.6, although eFPT does not have the best fit on the three simulated drifts, except for the hyperplane data, it has often a smaller drop in performance and a better recovery pattern. Despite the visible drop in performance at the beginning of the concept drift, eFPT is able to recover quite quickly and reaches the same performance, as before, after a short while. The Hoeffding tree, on the other hand, needs quite a long time to learn the concept and is more strongly affected by the drift. IBLStreams seems to have a good fit, minor drop in performance and perfect recovery patten.

eFPT, compared to the other approaches, shows a relatively good performance; this performance, however, is not superior mainly because of restricting the search space with a coarse-grained fuzzification (only three fuzzy sets are defined for each input attribute). That said, the proposed extension to pattern trees assists them with the ability to discover a concept change and to recover appropriately.

4.5.1.2 Real Data

In this part of the evaluation, we use three binary real data sets, namely mushroom (Appendix D.3.2), skin segmentation (Appendix D.3.6) and the MAGIC (Appendix D.3.7) data sets. Performance evaluations are measured on windows of a fixed-size, which is chosen based on the size of the data set as shown in Table 4.2. The real data sets are standard benchmarks taken from the UCI repository [107]. Because they do not have an inherent temporal order, we generate data streams by randomly sampling, without replacement, instances from each data set. Performance is computed by taking the average of 10 performance curves of randomly shuffled versions of each of these data sets.

Figure 4.7 shows the results of the three experiments. As a proof of concept, eFPT is clearly capable of fitting these static data sets well, after observing a number of training examples less than what the Hoeffding tree requires. The Hoeffding tree reaches the performance of eFPT on the MAGIC data set only after observing least 5k examples.

4.5.2 Model Size

Apart from comparing the performance of the methods, we also compare the size of the learned models. The size of eFPT, in its four different variations, is compared with that of the incremental and the adaptive Hoeffding trees. eFPT is used with and without retaining operators (see Subsection 4.4.3) for both significance levels

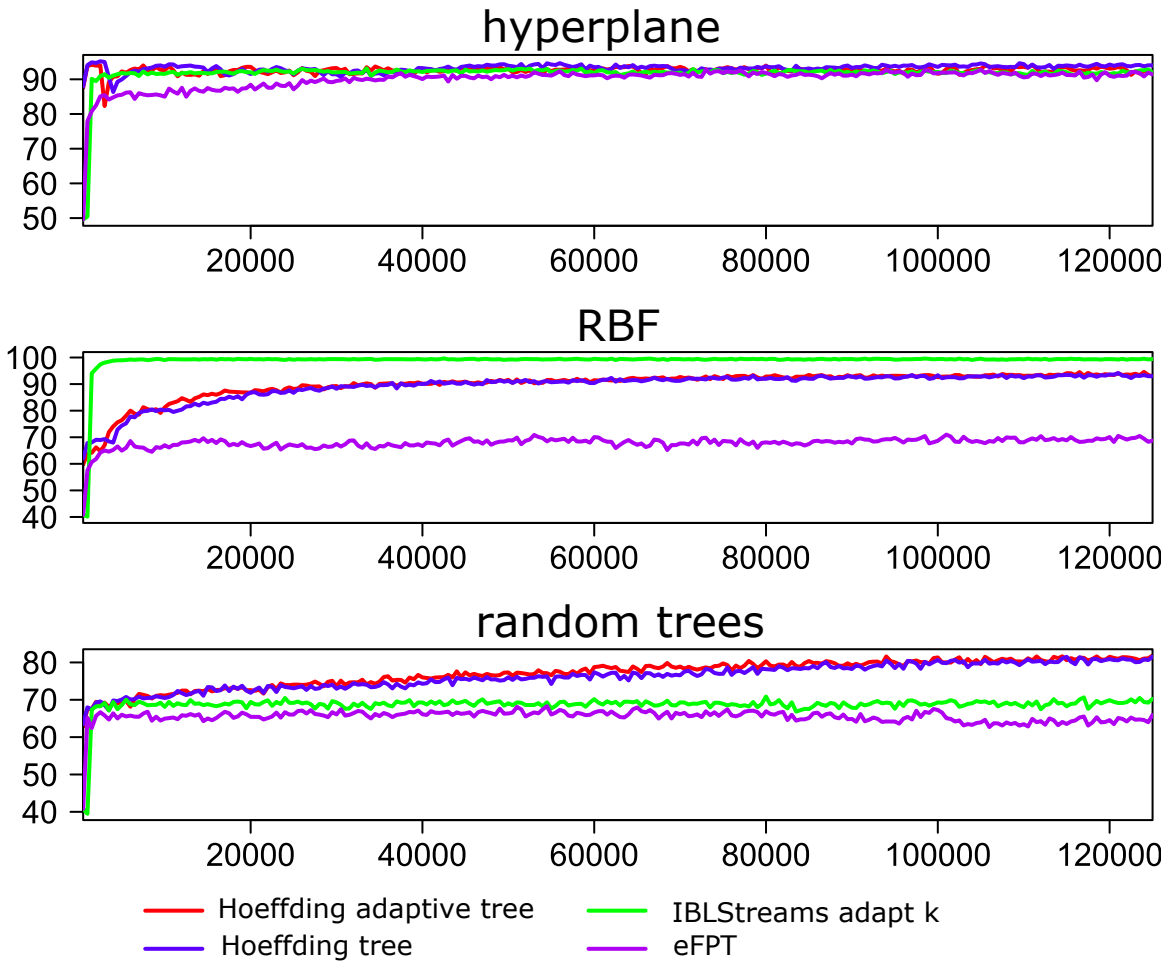


Figure 4.5: Performance comparison between eFPT, Hoeffding trees and IBLStreams when learning from synthetic data streams.

$\alpha = 0.1$ and $\alpha = 0.25$. The initial set of triangular norms and conorms applied here contains only the Łukasiewicz operators. Using the same synthetic data sets employed for the performance comparison in the previous subsection, we present the size of the compared approaches in Figures 4.8 and 4.9. As a result, one can observe the following:

- Obviously, a larger significance level corresponds to a less conservative hypothesis test, which leads to larger trees. Trees, built using the significance level $\alpha = 0.25$, are almost twice the size of those built with the significance level $\alpha = 0.1$.

- For the same significance level, refining the operators often leads to smaller trees. This is because the non-refined induction procedures try to force-fit some

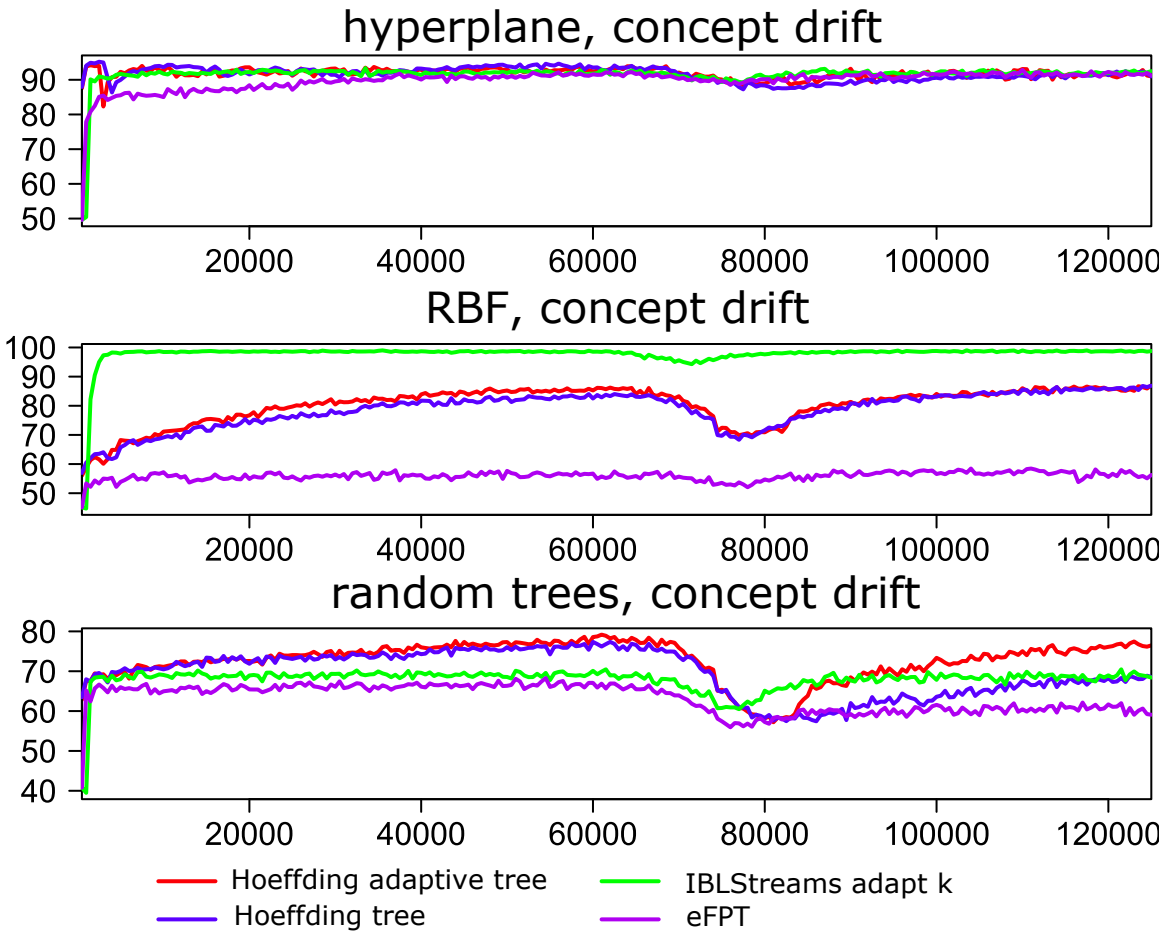


Figure 4.6: Performance comparison between eFPT, Hoeffding trees and IBLStreams when learning from synthetic data streams with simulated concept drifts.

functions through recursive application of a single type of t-norms/t-conorms, instead of choosing the t-norm/t-conorm that leads to a better fit.

- Hoeffding trees, in its incremental versions, outnumber all variations of eFPT in size. The adaptive version, on the other hand, shows a continuous increase in the number of nodes in a linear way, independently whether the stream contains a concept change or not.

- eFPT does not show a monotone increase in its tree size. On the contrary, some concept changes are better fit by first pruning and then extending the current model.

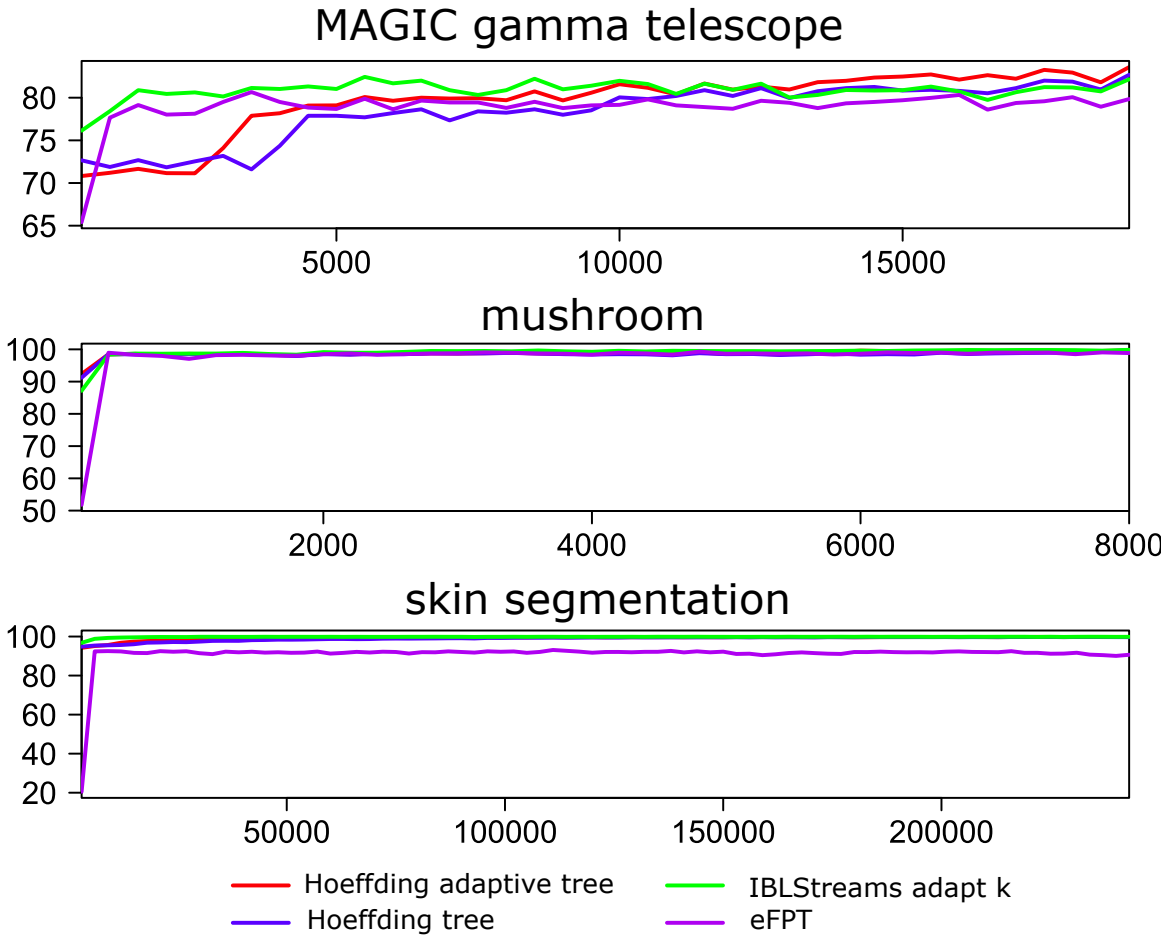


Figure 4.7: Performance comparison between eFPT, Hoeffding trees and IBLStreams when learning from real data streams.

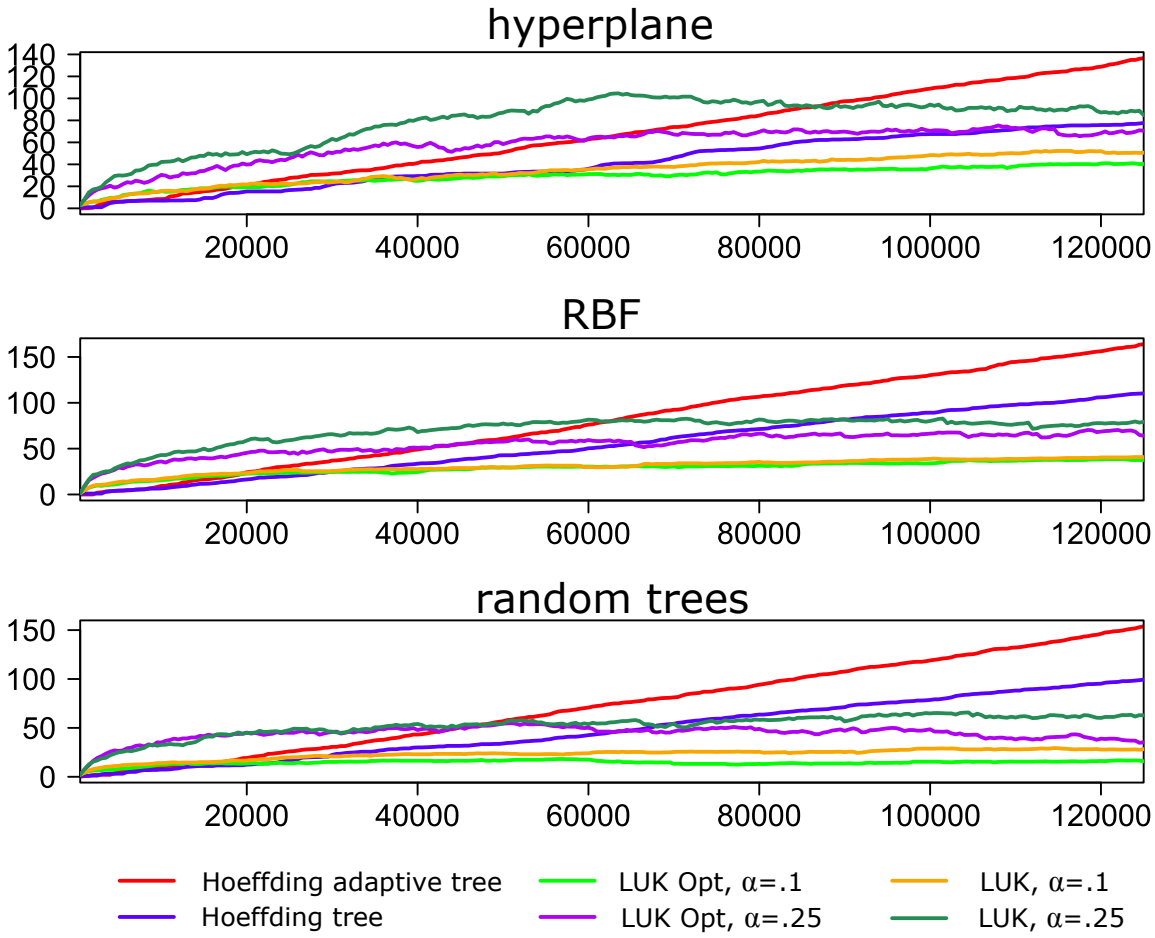


Figure 4.8: Tree size of eFPT and Hoeffding trees when learning from synthetic data streams.

4.5.3 Sensitivity Towards Significance Levels and Operators Retraining

As confirmed in the previous subsection, one can see that a larger significance level leads to larger trees which correspond with a better fit to target concept. In this subsection, the effect of the initial operator set of triangular norms and conorms on the tree's performance and size is investigated. To this end, we compare the accuracy and the number of retraining steps during the induction of an eFPT in its four different variations. eFPT is used with two initial sets for retaining operators (see Subsection 4.4.3), Gödel and Łukasiewicz, for both significance levels $\alpha = 0.1$ and $\alpha = 0.25$.

Employing the same synthetic data sets used in the previous subsections, we present the performance in Figures 4.10 and 4.11, and the number of times an operator was retrained in Figures 4.12 and 4.13. As a result, one can observe the following:

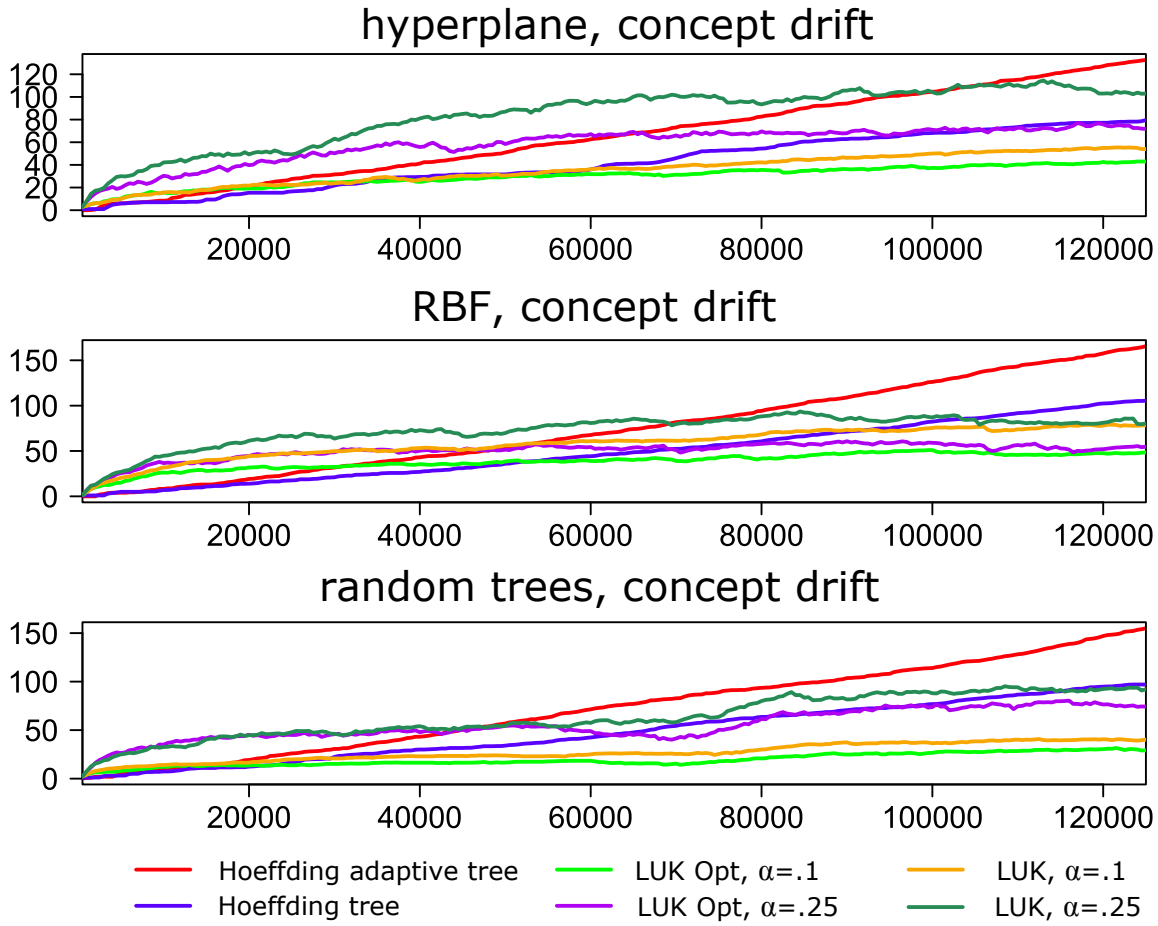


Figure 4.9: Tree size of eFPT and Hoeffding trees when learning from synthetic data streams with simulated concept drifts.

- As expected, a larger significance level corresponds to larger trees (see Figures 4.10 and 4.11) and thus a better performance, provided the complexity and stability of the target concept to be learned.
- A small significance level leads to fewer changes and a smaller number of successful operator retrainings.
- From Figures 4.10 and 4.11, one can see that the set of Gödel operators seems to be outperformed by the other type of operators.
- For the significance level $\alpha = 0.25$, Gödel operators are more prone to be outperformed by other operators, and thus being replaced. This phenomena can be explained by the fact that Gödel operators just consider the `MIN`/`MAX` values, which ignore any possible interactions between the operands.

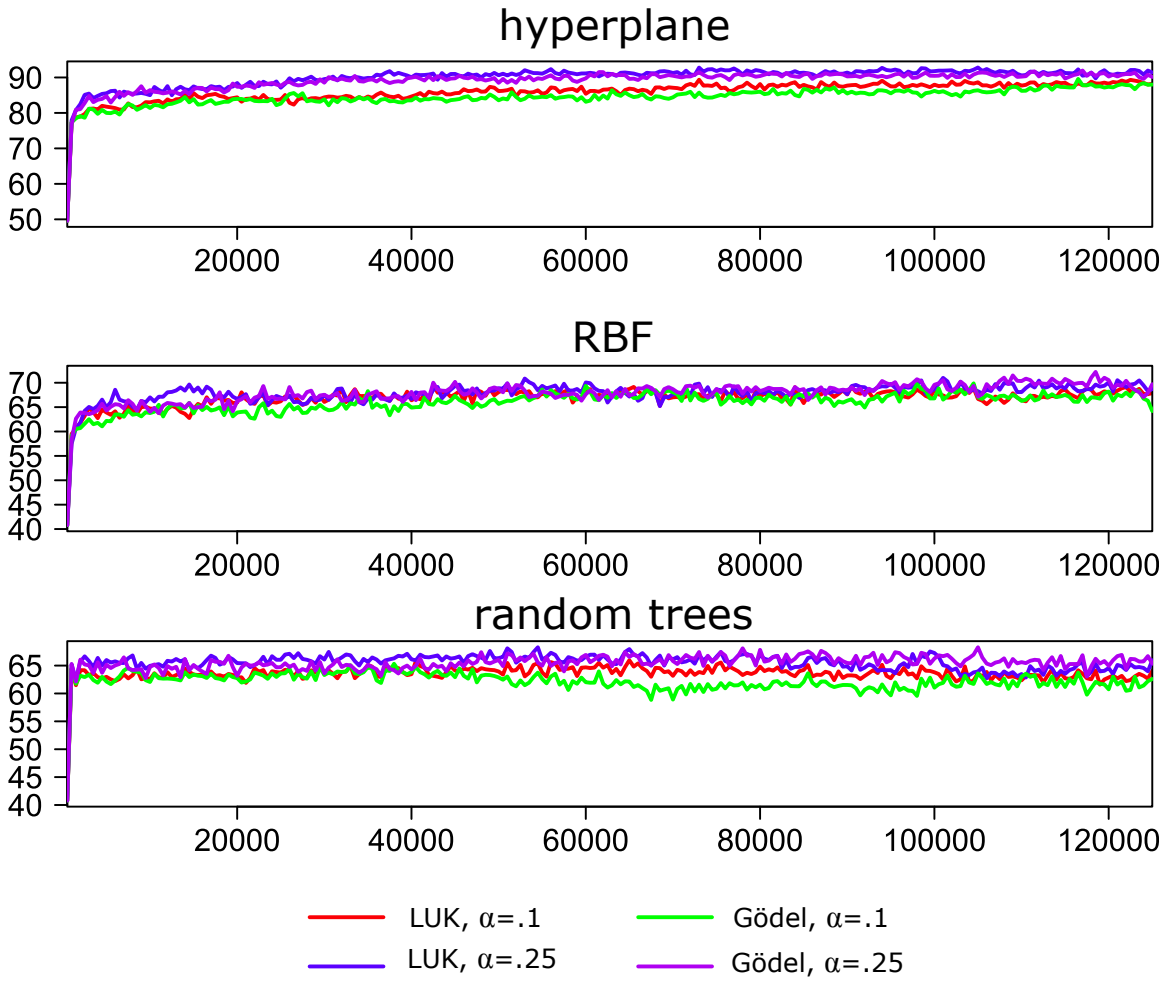


Figure 4.10: Performance comparison between different eFPT parametrizations (significance level and retaining operators) when learning from synthetic data streams.

4.6 Summary and Conclusion

In this chapter, an evolving version of the fuzzy pattern tree classifier is proposed; this eFPT meets the requirements of adaptive learning on data streams. The key idea of eFPT is to maintain the current model and a set of neighbor trees that can replace the current model if the performance of the latter is no longer optimal. Thus, a modification of the current model is realized implicitly in the form of a replacement by an alternative tree. A replacement decision is made on the basis of the performance of all models, which is monitored continuously on a sliding window of fixed length.

Fuzzy pattern trees form an attractive model class of interpretable representation, besides the fact that they are universal approximators [144].

In an experimental study, we compared eFPT with the two versions of the Hoeffding trees and with IBLStreams on real and synthetic data. The obtained results

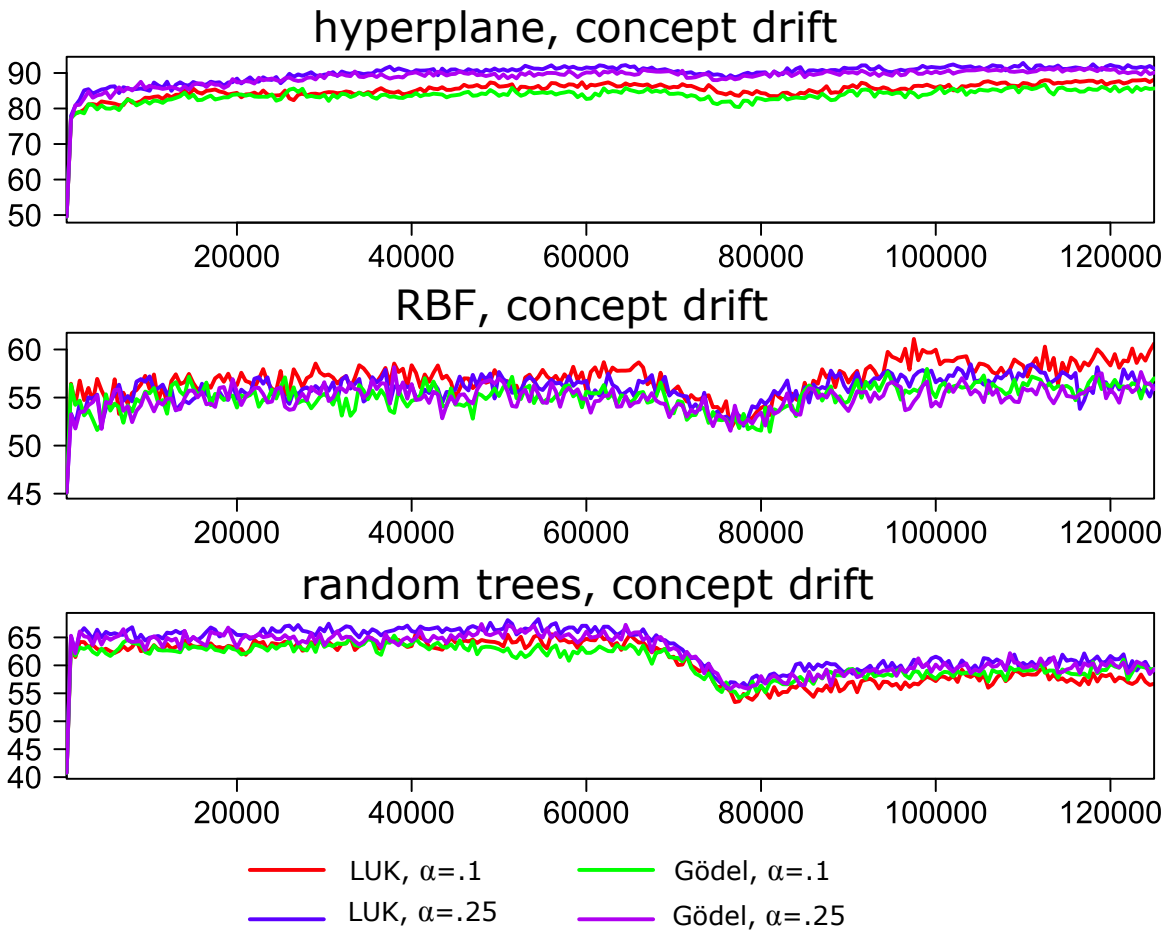


Figure 4.11: Performance comparison between different eFPT parametrizations (significance level and retaining operators) when learning from synthetic data streams with simulated concept drifts.

are quite promising, despite the failure to learn on the RBF data. They suggest that eFPT is competitive in terms of accuracy, while being less affected by concept drift and producing smaller, more compact models. These criteria are of course interrelated: The smaller a model is, the more easily and quickly it can be adapted in the case of a concept drift; besides, compactness of a model is of course desirable from an understandability point of view. On the other hand, producing large models can be advantageous in cases where the target concept to be learned is complex and the data generating process sufficiently stable; in our experiments, Hoeffding trees and IBLStreams performed comparatively well, especially in these cases.

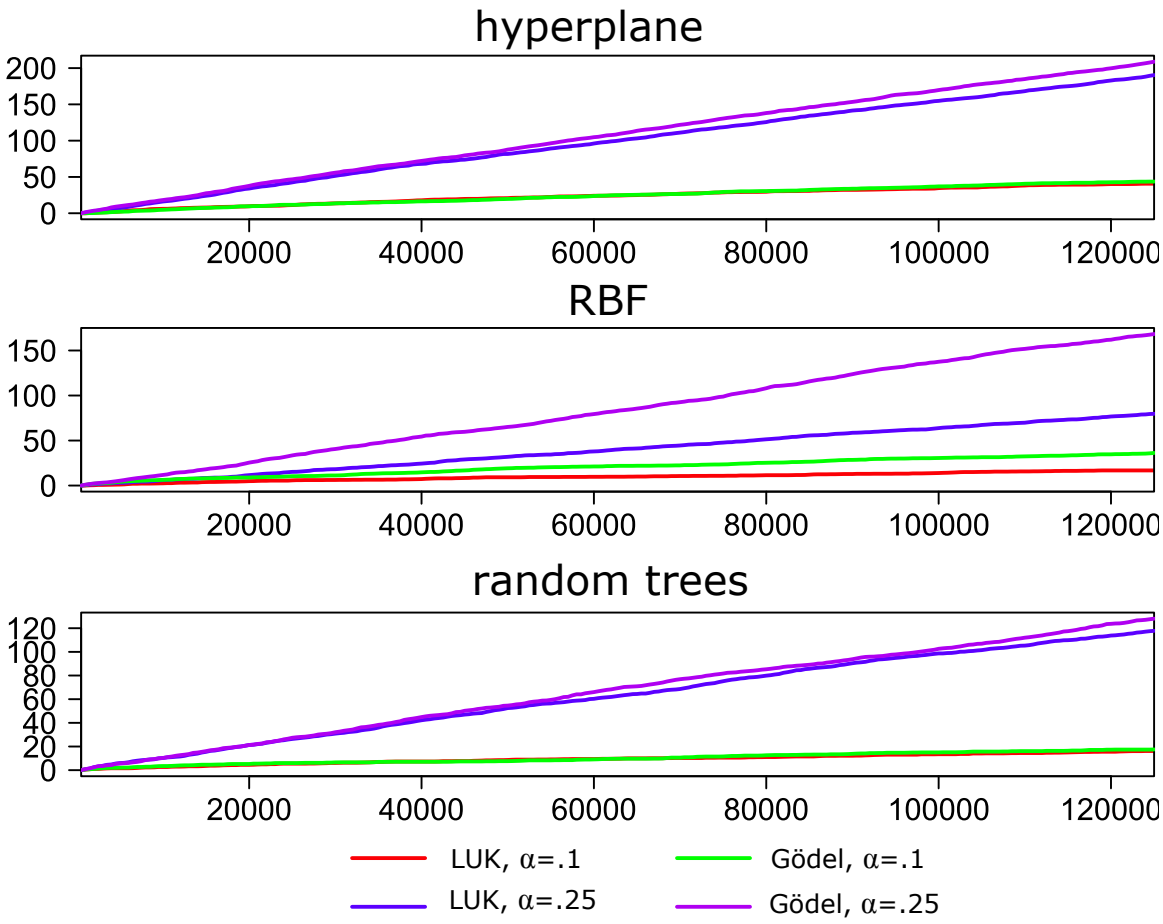


Figure 4.12: Number of retrained operators for the different eFPT parametrizations (significance level and retaining operators) when learning from synthetic data streams.

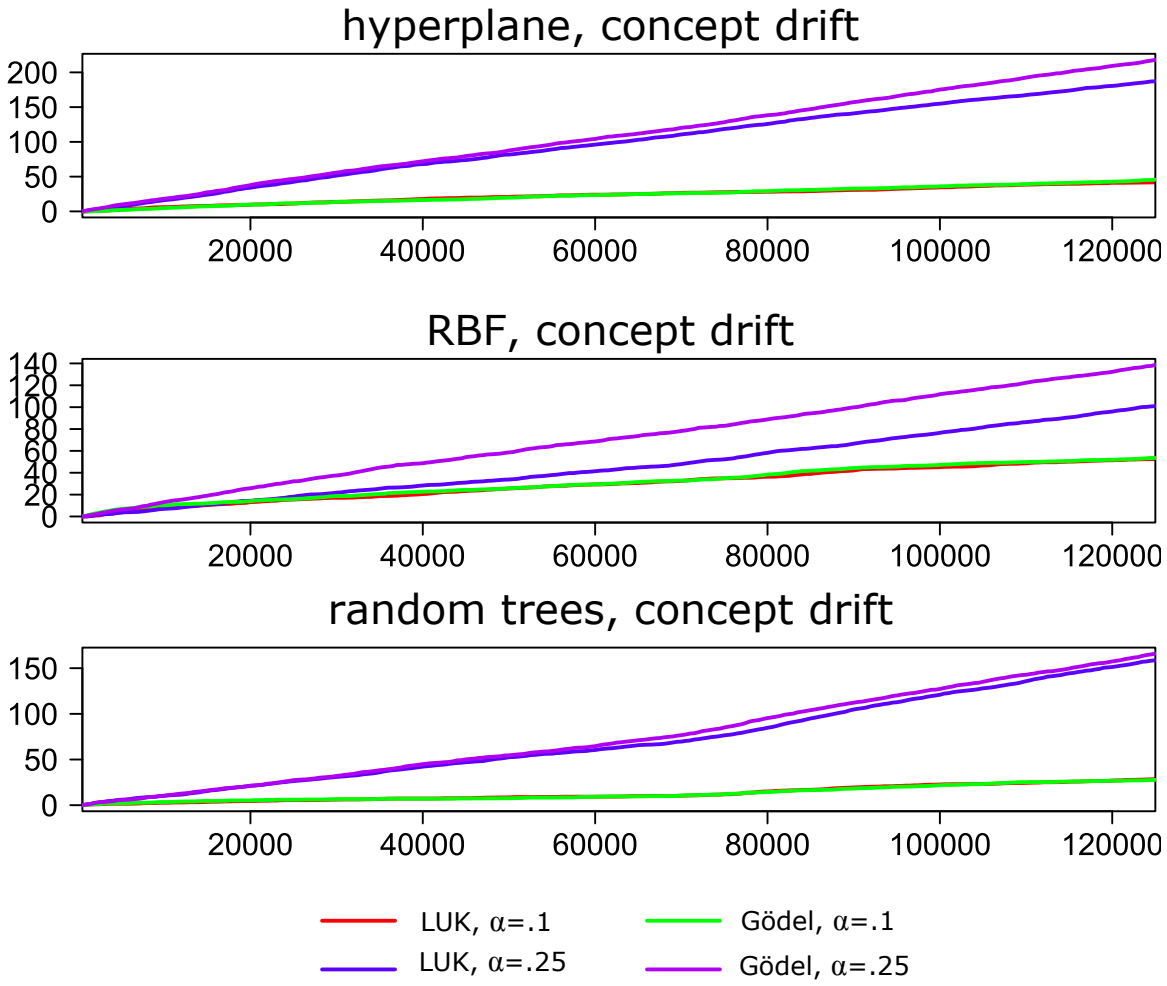


Figure 4.13: Number of retrained operators for the different eFPT parametrizations (significance level and retaining operators) when learning from synthetic data streams with simulated concept drifts.

Chapter 5

Survival Analysis on Event Streams

This chapter introduces a method for survival analysis on data streams; survival analysis is an established statistical method for the study of temporal *events* or, more specifically, questions regarding the temporal distribution of the occurrence of events and their dependence on the features of the data sources.

To the best of our knowledge, survival analysis has not yet been considered in the stream setting so far. This is arguably surprising for several reasons. Most notably, the temporal nature of event data naturally fits the data stream model and *event data* is naturally produced by many data sources. Moreover, survival analysis is widely applicable and routinely employed in many application fields. Survival analysis, a term commonly used in medical studies, is also referred to as event history analysis in sociology, reliability analysis in engineering and duration analysis in economics.

5.1 Introduction

The introduced learning methods so far focus on the supervised learning from examples, observed in a stream of data, by inducing models that capture the conditional dependency between the examples' features and target values. The supervision, granted by the data, allows the learner to observe the properties and the target value of each example. This supervision is weakened when the target value becomes partially known, such as only knowing an interval containing that target value. Survival data is one example of weakly supervised data: *(i)* It is supervised because an individual, in clinical studies, (with his/her characteristics) forms a learning example and the time he/she survives (before experiencing a specific event) is the target value.

(ii) It is weakly supervised because an individual may leave the study or get lost before experiencing the target event; this makes his/her survival time only partially known, by knowing that the smallest value it could take is equal to the length of the period spent in the study. Such individuals are called censored as explained later in Subsection 5.2.1.

Survival analysis (SA) is a statistical method for modeling and analyzing the temporal distribution of events in the course of time or the duration before the occurrence of an event of interest. The notion of an *event* is completely generic and may indicate important information such as the failure of an electrical device. The event of interest is usually associated with a special cause, such as the death of a patient caused by a specific infection.

One learning task that could benefit from such data is the task of learning the dependencies between the life span and the features of these examples. The life span of each example is realized by the time interval $[t_{start}, t_{event}]$, where t_{start} is the time point when the study becomes aware of the example and t_{event} is the time when the event occurs and the example leaves the study.

The motivation behind this work is to bring both aspects closer together: the well-established statistical methods of survival analysis on event data and the learning from data streams. In particular, applying survival analysis becomes challenging when the event data ceases to remain small and easy to handle and starts to become immense and continuously arriving, i.e., forming an *event stream*. This event stream resembles the previously studied data streams by their immensity and time invariability, due to possible changes in a dynamic environment.

To make survival analysis applicable in the setting of data streams, an adaptive (online) variant of a model that is closely related to the well-known proportional hazard model proposed by Cox [42] is developed. In this model, the hazard rate may depend on one or more covariates associated with a statistical entity; more specifically, the effect of an increase of a covariate by one unit is multiplicative with respect to the hazard rate.

The proposed approach adopts the sliding window approach, which is a common technique in data stream analysis; in order to estimate the influence of the covariates, the hazard rate is assumed to be constant on the current window. The estimate then depends on the frequency and temporal distribution of events falling inside the window, and sliding the window calls for adapting the estimate in an incremental (and as efficient as possible) manner.

The remainder of this chapter is organized as follows: By way of background, Section 5.2 recalls some basic information about survival analysis. Section 5.3 is devoted to our extension of survival analysis and describes the main adaptations realized to make this method applicable in a streaming setting. Finally, to evaluate our approach and as a proof of principle, two case studies are presented in Sections 5.4 and 5.5. In both studies, our method is used for a specific type of spatio-temporal data analysis, namely the analysis of earthquake data (Section 5.4) and of Twitter data (Section 5.5). In an attempt to explain the frequency of events by the spatial location of the data source, both studies use the location as covariates of the sources.

5.2 Survival Analysis

Survival analysis, as the name suggests, originates from medical researches, where survival data is derived from clinical and epidemiological researches of humans and laboratory studies of animals. Nonetheless, survival analysis includes a broader scope of studies, such as the lifetime of electrical products in reliability engineering or the duration of a marriage in event history analysis in sociology.

In a clinical study, the death of a patient under observation can be the event of interest and the *survival time*, or the *time to event*, is the time duration $s = t_{event} - t_{start}$ between t_{start}, the time of the patient entering the study, and t_{event}, time at which the event occurred.

The motivation behind analyzing survival data, instead of simply applying regression models with the survival time as a target value, is that the target events of some objects are not observed. These objects are called *censored data*. Censoring occurs when an object is lost before the planned end of the study, as in the case when a patient decides to leave the study for personal reasons. An object is also censored when the event occurs caused by a reason different from the targeted one, such as the death of a patient by a car accident in a study on leukemia, instead of dying as a result of leukemia.

Although the survival times for the censored objects are not known, the minimum time they spent in the study before being censored gives a lower bound of their survival times. This makes the survival data less applicable to regression models, unless the censored data is ignored, thus valuable information is lost.

Since a statistical entity is not always a person as indicated by the term "individual", the more neutral term "instance" is subsequently used. Suppose such an instance can be described in terms of the feature vector

$$\boldsymbol{x} = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n, \tag{5.1}$$

where x_i is the value of the ith property of the instance (for example, the age of a patient in a medical study).

5.2.1 Censored data

Censored data can be categorized into three main types [105]:

- Type I censoring: When the experimenter fixes a predetermined time T at which he plans to end the study, instances that remain in the study past this time without experiencing the target event are considered type I censored. For these instances, it is known that their survival time is at least T. Figure 5.1(a) shows how the Instances 2 and 4 become type I censored after surviving until time T without experiencing the target event.
- Type II censoring: Starting the study with N instances, the experimenter might decide to end the study after observing a fixed percentage of events r/N. In this case, animals that experienced the events at times $t_1 \leq \cdots \leq t_r$ have clear survival times, whereas cases that are still alive have a survival time that is at least t_r. Figure 5.1(b) shows how the Instances 2 and 4 become type II censored only after Instance 5 has experienced the event, by which the fixed percentage 3/5 is reached.
- Type III censoring: In more realistic clinical studies, instances do not enter the study at the same time. Some remain until experiencing the event and others get lost or survive until the end of the study. Instances of the last two cases are considered censored. Figure 5.1(c) shows the type III censored Instances 2 and 4; Instance 4 is either lost or left the study before its planned end. Instance 2, besides being type III censored, is also referred to as right censored.
- Left censoring: This type of censoring results when an instance experiences the targeted event even before entering the study, at an unknown time point. This type of censoring is only relevant when multiple events, for the same instance, are allowed to take place; as will become clear in Section 5.3. Figure 5.1(d) shows the left censored Instances 2 and 5, as they experience the event at times before the starting time T_0 of the experiment.

Type I and type II are also referred to as *right censoring*, whereas type III is also known as *random censoring*. Data sets with no censored instances, as in the case when event times are known for all studied instances, are called *complete data*.

5.2.2 Survival Functions

The survival time is a random variable, whose distribution can be described by three functions: *(i)* the probability density function, *(ii)* the survival function and *(iii)* the hazard function. By determining one of these functions, the other two functions can be derived.

Consider the time for an event to occur as a real-valued random variable T with probability density function $f(\cdot)$, which models the instantaneous probability for an event to take place in the infinitesimal interval $[t, t + \Delta t]$, and defined as

$$f(t) = \lim_{\Delta t \to 0} \frac{\mathbf{P}\left\{t < T \leq t + \Delta t\right\}}{\Delta t} \ . \tag{5.2}$$

The cumulative distribution function $F(\cdot)$,

$$F(t) \ = \ \mathbf{P}\left\{T \leq t\right\} = \int_0^t f(x)\, dx \ , \tag{5.3}$$

is the probability of an event to occur before the time t. The *survival function* $S(\cdot)$ is then defined as

$$S(t) \ = \ \mathbf{P}\left\{T > t\right\} = 1 - F(t) = \int_t^\infty f(x)\, dx \ , \tag{5.4}$$

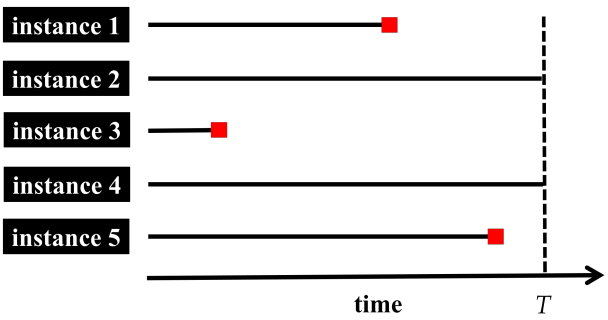
where $S(t)$ is the probability that an instance survives at least until the time point t. The survival function can also model the probability of an instance to be right censored, i.e., $S(t = T_{end})$, where T_{end} is the end time of the study. Unlike the cumulative function $F(\cdot)$, the survival function is a decreasing function with $S(0) = 1$ and $\lim_{t\to\infty} S(t) = 0$.

Finally, the *hazard function* or hazard rate $h(\cdot)$ is defined as follows:

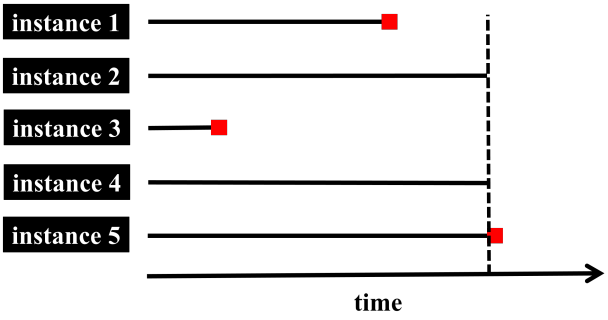
$$h(t) = \lim_{\Delta t \to 0} \frac{\mathbf{P}\left\{t < T \leq t + \Delta t \,|\, T > t\right\}}{\Delta t} \tag{5.5}$$

$$= \frac{f(t)}{S(t)} \ . \tag{5.6}$$

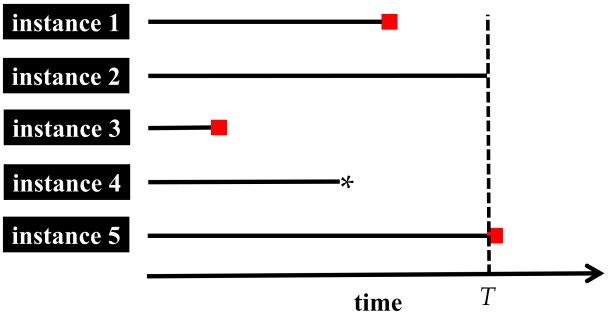
Generally, $h(t)$ is the conditional probability that the event will occur within a small time interval after t, given that it has not occurred until t. More specifically, $h(t)$ is the limit of this probability when the length of the time interval tends to 0. Mathematically, it is hence a kind of density (and not a probability) function, which means



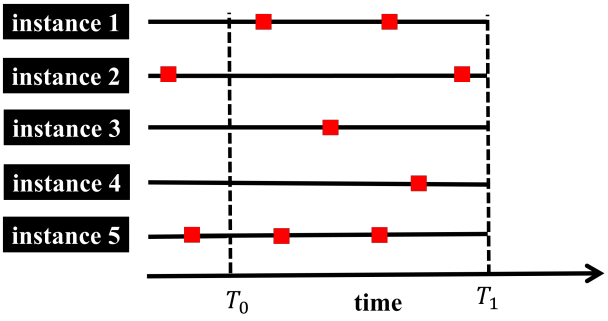
(a) Type I censoring



(b) Type II censoring



(c) Type III censoring



(d) Left censoring

Figure 5.1: The different types of censoring.

that it may thoroughly assume values larger than 1. The hazard function is also known as the conditional failure rate, which makes the density function to be the unconditional failure rate.

By knowing one of the three functions, the other two can be easily derived. The density function $f(\cdot)$ can be derived from the survival function $S(\cdot)$ in (5.4) as

$$f(t) = \frac{d}{dt}[1 - S(t)] = -S'(t) \ . \tag{5.7}$$

The hazard function $h(\cdot)$ can also be derived from (5.6) and (5.7) as

$$h(t) = \frac{f(t)}{S(t)} = \frac{-S'(t)}{S(t)} \ . \tag{5.8}$$

5.2.3 Estimating the Survival Function

As previously explained, analyzing survival data faces the challenge of the missing event times for the censored instances. However, the censored instances can still contribute to the study through the minimum time they survive. The product limit approach for estimating the survival function $S(\cdot)$, derived by Kaplan and Meier [91], estimates the survival function using both *(i)* the survival times for instances that experienced the target event and *(ii)* the minimum time spent by censored instances in the experiment.

The *Kaplan-Meier* method is one of the most commonly applied non-parametric methods because it can be easily computed for a moderate size data set and it is supported by a graphical interpenetration. Imagine a set of m instances under observation whose event/censoring times are $t_1 \leq t_2 \leq \cdots \leq t_m$. The index set I contains the indices of event times only, i.e., $I = \{i : t_i \text{ is an event time}\}$. The *Kaplan-Meier* survival function is:

$$\hat{S}(t) = \prod_{t_i < t \wedge i \in I} \frac{m - i}{m - i + 1} \ , \tag{5.9}$$

where the product considers only the event times $\{t_i : i \in I\}$, not the censoring times. The denominator represents the count of instances in the *risk set* $R(t_i)$, the instances that survived until time t_i, regardless of whether they will become censored or experience the event in the future. In the complete data set case, when censoring does not occur, the estimation in (5.9) becomes simply

$$\hat{S}(t_i) = \frac{m - i}{m} \ . \tag{5.10}$$

As an alternative to the Kaplan-Meier method, the *life table* [18, 43] methods can also be applied on the survival data. Life table methods are suited for a large number of observations, whose survival times are grouped into intervals.

Supported by physical explanations, failures (events) occur at distinct times following some probability distribution. Hence, parametric approaches for survival analysis try to fit well-known distributions to the survival data. These distributions have most of the time an analytical solution for the maximized likelihood of both instances, censored and failed ones. It is shown in [47] that the failure times of radar components follow the exponential distributions. Weibull distribution is shown to model the failures in electron tubes by [90]. Considering the many applications in biology and economics, the lognormal distribution is shown by [62] to closely fit the survival times of chronic leukemia patients, for more examples see [105].

5.2.4 Prognostic Factors for Survival

Although the main concern of survival analysis lies in estimating the survival functions for a group of individuals under study and probably the expected survival time or the expected time to failure, another important concern seeks the identification of the prognostic factors (variables and attributes) and their relation to the expected survival [105].

The simplest way of identifying the prognostic variables can be achieved by applying a *nonparametric approach* (such as the Kaplan-Meier method) on the survival data. This application can be either repetitively applied on univariate problems by investigating one variable at a time, each time the observed instances are grouped based on different breakdowns of that variable. Thereafter, survival functions of each group of each variable can be estimated by applying the Kaplan-Meier method, which can then be statistically compared to identify the prognostic factors. Alternately, the nonparametric approaches can consider multiple variables simultaneously by stratification, which defines multiple strata each of which contains a group of instances that share similar values for the considered variables. The univariate and the multivariate approaches require the construction of a large number of problems ($O(n)$ for the univariate and up to $O(2^n)$ for the multivariate where n is the number of covariates/dimensions) that need to be solved in order to find the prognostic variables. In the following we describe the semi-parametric and the parametric approaches, which fit a single regression model, independently of the survival data's dimensionality.

The multivariate semi-parametric approach

Parametric regression approaches attempt to model the relation between the independent prognostic variables and the survival times. The semi-parametric approaches, on the other hand, do not try to fit the whole model. Instead, they assume the proportionality of hazards, that is the *hazard ratio*

$$HR = h(t|\boldsymbol{x}_1)/h(t|\boldsymbol{x}_2)$$

between the instances $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is constant. This leads to writing the hazard function of an instance $\boldsymbol{x}$ in the form

$$h(t|\boldsymbol{x}) \;=\; h_0(t) \cdot r(\boldsymbol{x}) \;\;, \tag{5.11}$$

where $r(\boldsymbol{x})$ is a time-independent function that depends only on $\boldsymbol{x}$. $h_0(t)$ is the baseline hazard that depends only on the time t; it is also the remaining hazard for an instance when $r(\boldsymbol{x}) = 1$, i.e., the baseline hazard is the hazard when all variables are set to zero [105].

Equation (5.11) defines the so-called *proportional hazard* (PH) assumption; this assumption is made in the Cox proportional hazard model [42], in which the hazard rate is modeled as a log-linear function of the covariates x_i:

$$h(t|\boldsymbol{x}) = \; h_0(t) \cdot \exp\left(\beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n\right) \tag{5.12}$$

$$= \; h_0(t) \cdot \exp\left(\sum_{i=1}^{n} \beta_i \cdot x_i\right) \;\;. \tag{5.13}$$

In the Cox model, the effect of an increase of an independent variable by one unit is multiplicative with respect to the hazard rate; i.e., the hazard rate is proportional to each variable, therefore increasing x_i by one unit increases $h(t|\boldsymbol{x})$ by a factor of $\alpha_i = \exp(\beta_i)$.

The coefficients of the Cox model can be estimated by maximizing the likelihood function, whose formulation requires knowing the distribution of the study's outcomes (the time to event). However, the Cox model does not assume any distribution for the dependent variable; it only assumes the proportionality of hazards. Therefore, the likelihood of the Cox model depends on the ordering of events instead of their joint probabilities. This can be realized by applying the *partial likelihood* (PL) estimation, which considers the conditional probability of the event only for the instances that experience the event, given the risk set. For the m distinct event times $t_1 < t_2 < \cdots <$

t_m, the probability for an instance $\boldsymbol{x}_o$ to have an event at the time t_o, conditionally on the risk set $R(t_o)$, is

$$L_o(\boldsymbol{\beta}) = \frac{\exp\left(\sum_{i=1}^{n} \beta_i \cdot x_{oi}\right)}{\sum_{l \in R(t_o)} \exp\left(\sum_{i=1}^{n} \beta_i \cdot x_{li}\right)} \ . \tag{5.14}$$

The partial likelihood considers a censored instance to be in the risk set $R(t_o)$ only if it was not censored yet by the time t_o. Finally, the partial likelihood function is written as the product of the probabilities of the m events:

$$PL(\boldsymbol{\beta}) = \prod_{o=1}^{m} L_o(\boldsymbol{\beta}) \ . \tag{5.15}$$

Maximizing the partial likelihood can be obtained by maximizing its log-likelihood

$$LPL(\boldsymbol{\beta}) = \sum_{o=1}^{m} \sum_{i=1}^{n} \beta_i \cdot x_{oi} - \sum_{o=1}^{m} \ln \left(\sum_{l \in R(t_o)} \exp \left(\sum_{i=1}^{n} \beta_i \cdot x_{li} \right) \right) \ ; \tag{5.16}$$

this maximum can be numerically approximated using the iterative Newton-Raphson method [28].

The semi-parametric property of the Cox model formulation comes from the fact that the baseline hazard remains unspecified. This property makes the PH models more favorable in comparison to fully parametric models that require the hazard functions to be specified. Factors for the Cox model's popularity are: *(i)* it requires minimum assumptions, *(ii)* it is robust in that it closely approximates the correct parametric model as in the case when the parametric model follows the exponential or the Weibull distribution and *(iii)* the estimated hazard will always be non-negative due to the exponential part in (5.12)[95].

The approximated coefficients of the Cox model can be further employed to estimate a survival curve that is adjusted with the found coefficients. This survival function is called the *adjusted survival curve* [113], motivated by the need to make the visual representation of survival curves more consisted with the induced semi-parametric models; it is written as

$$S(t|\boldsymbol{x}) = \left[S_0(t)\right]^{\exp\left(\sum_{i=1}^{n} \beta_i \cdot x_i\right)} \ , \tag{5.17}$$

with $S_0(\cdot)$ the baseline survival function, which has a closed form solution [89].

Statistical methods for survival analysis, such as Cox regression [41], provide estimates of the model parameters β_i and, therefore, of the hazard rate itself (given that the baseline hazard rate is known or can be estimated). The latter can be used, for

example, for prediction purposes. Given an estimate of the hazard rate, one can predict the time span until the next event will occur, both in terms of point predictions, e.g., the expected survival time of a patient, and confidence sets, e.g., a confidence interval for the survival time. The estimations of the parameters β_i are as interesting as the hazard rate itself; they inform about the influence of different covariates on the hazard rate. For example, if $\beta_i = \log(2)$ is the parameter modeling the influence of the covariate *smoking* (a binary attribute with value 1 if the patient is a smoker and 0 otherwise) in a medical study, it means that—under the model (5.12) and *ceterus paribus*, i.e., all other covariates being equal—smoking doubles the hazard rate, thus cutting the expected survival time[1] in half.

The multivariate parametric approach

Some survival data tends to follow some known distributions and, therefore, allows us to put some model assumptions on the outcome (the survival time). Parametric survival models make use of such assumptions, that the survival times follow a given distribution, and find the distribution's parameters that make the current data most likely. The commonly assumed survival distributions are the exponential, the Weibull, the log-logistic and the log-normal.

Some survival models satisfy the PH property, such as the exponential and the Weibull distributions. Other parametric models are *accelerated failure time* (AFT) models instead [177], such as the log-normal and the gamma distributions.

The AFT property is a useful property for comparing survival times, unlike the PH property which is used for comparing hazards. Consider a study claiming that the people who smoke develop health problems $\alpha > 1$ times faster than nonsmokers, i.e., a (t)-years-old smoking person $\boldsymbol{x}_1$ would develop health problems as much as an older $(\alpha \cdot t)$-years-old nonsmoking person $\boldsymbol{x}_2$. For this example, the AFT assumption states that $S(t|\boldsymbol{x}_1) = S(\alpha t|\boldsymbol{x}_2)$ for $t \geq 0$, with α the acceleration factor, which describes how the survival time stretches or contracts as a function of the independent variables [95]. Distributions such as the log-normal and the gamma distributions can only be used in AFT models, whereas the exponential and the Weibull distributions can be used in both PH and AFT models [177]. The next section introduces our adaptive approach to survival analysis, in which we assume an exponential hazard model; hence, accommodating both the PH and the AFT properties.

[1]This is true only in the case when the baseline hazard $h_0(\cdot)$ is a time-independent constant; i.e., the survival times follow the exponential distribution.

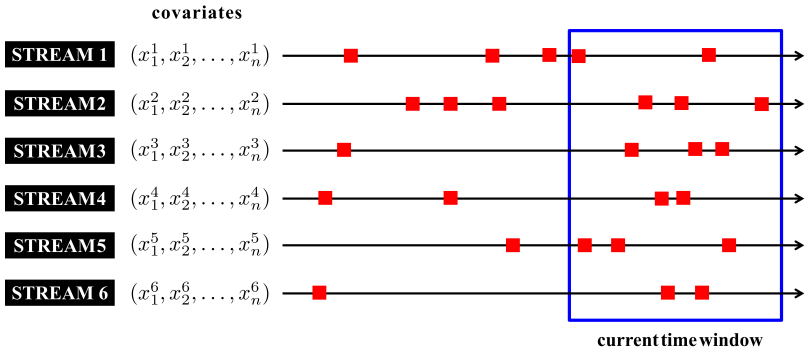


Figure 5.2: Illustration of our setting consisting of a set of J (here $J = 6$) parallel data streams: Every stream corresponds to a statistical entity characterized in terms of a vector of covariates. Moreover, each stream produces a sequence of temporal events (marked by solid squares). A sliding window (indicated by the grey box) is masking outdated events that occurred in the past.

5.3 Survival Analysis on Data Streams

The survival analysis setting here assumes a fixed set of J data streams to be given, each of which corresponds to an instance $\boldsymbol{x}$ characterized in terms of a vector of covariates $(x_1, \ldots, x_n)^\top$. Moreover, each stream produces a sequence of temporal events, i.e., events that are associated with a unique time of occurrence; see Figure 5.2 for an illustration. For simplicity, we assume the underlying time scale to be discrete, i.e., time progresses in discrete steps (such as seconds or minutes).

Imagine a case where each stream corresponds to a book offered by an online book store and the covariates are properties of the book (price, genre, etc.). In this example, an "event" occurs whenever a client purchases a book. The hazard rate associated with a book can then be interpreted as a measure of the propensity of people to buy this book. Obviously, this propensity will change over time and for each book; therefore it is interesting to monitor the evolution of its hazard rate. Apart from that, it is interesting to determine the influence of the covariates on the buying behavior of the clients and, perhaps even more importantly, how this influence changes over time. One may expect, for example, that the price of a book will become more important in times of an economic crisis and will hence have a stronger influence on the hazard rates of all books.

The previous example has made clear that, when looking at a single data stream, we are interested in events that can occur repeatedly (for the same instance $\boldsymbol{x}$) in the course of time. Such events are called *recurrent events* and need to be distinguished

from events that can occur at most once (like the death of a patient in a medical study). More specifically, we are interested in the time duration between the occurrence of two events. Assuming that the hazard rates for the set of streams can be modeled as a Cox proportional hazard model (5.13) with a constant base line hazard $h_0(t)$, the hazard rate for a fixed instance (data stream) $\boldsymbol{x}$ becomes

$$\begin{aligned} h(t|\boldsymbol{x}) &= h_0(t) \cdot \exp\left(\sum_{i=1}^{n} \beta_i x_i\right) \\ &= \exp\left(\sum_{i=0}^{n} \beta_i x_i\right) = h_{\boldsymbol{x}} \ , \end{aligned} \tag{5.18}$$

by extending the covariate of $\boldsymbol{x}$ in (5.1) by a constant entry $x_0 \equiv 1$, leading to a compact hazard of the form $h_{\boldsymbol{x}} = \exp\left(\boldsymbol{x}^\top \cdot \boldsymbol{\beta}\right)$, and let $t_1 < t_2 < \ldots < t_k$ denote the time points at which an event has been observed for this instance; moreover, let $a = t_0 < t_1$ and $b = t_{k+1} > t_k$ denote the start and the end of the observation interval $[a, b]$. The probability of the observation sequence $\mathcal{T}(\boldsymbol{x}) = \{t_\tau\}_{\tau=1}^{k}$ is then given by

$$\begin{aligned} \mathbf{P}(\mathcal{T}(\boldsymbol{x})) \ &= \ \left(\prod_{\tau=1}^{k} f_{\boldsymbol{x}}(t_{\tau-1}, t_\tau)\right) \cdot S_{\boldsymbol{x}}(t_k, t_{k+1}) \\ &= h_{\boldsymbol{x}}^k \cdot \prod_{\tau=1}^{k+1} \exp\big(-h_{\boldsymbol{x}} \cdot (t_\tau - t_{\tau-1})\big) \end{aligned} \tag{5.19}$$

where

$$\begin{aligned} f_{\boldsymbol{x}}(t', t) &= h_{\boldsymbol{x}} \cdot S_{\boldsymbol{x}}(t', t) \\ &= h_{\boldsymbol{x}} \cdot \exp\big(-h_{\boldsymbol{x}} \cdot (t - t')\big) \end{aligned} \tag{5.20}$$

is the probability that an event occurs usually at time t for an instance $\boldsymbol{x}$; having survived at least until time t'.

Notably, the non-constant hazard functions $h(t|\boldsymbol{x})$, in which the rate does not only depend on covariates $\boldsymbol{x}$ but also changes with time t, have been studied extensively in the statistical literature and many parameterized families of functions have been proposed for modeling the influence of time on the rate [42]. However, we later explain how the constant model $h_{\boldsymbol{x}}$ is sufficient for our purpose, or at least provides a sufficiently good approximation. This is due to the use of a sliding window approach: The assumption of a constant rate does not refer to a data stream as a whole but only to the current time window. Therefore, by sliding the window, the hazard rate may

actually vary in the course of time, too. Overall, our model becomes very flexible, especially given time-dependence is modeled in a non-parametric way.[2]

5.3.1 Left Censoring

Suppose that for an instance $\boldsymbol{x}$ surviving until time t_1, the first event occurred at an unobserved time t prior to t_0, the time at which the sliding window starts; this is a *left censoring* situation faced when applying the sliding window approach. The probability to observe the duration from t_0 to t_1 is then given by the conditional probability of the event at time t_1 given survival until t_0, i.e., by the expression

$$\begin{aligned}\frac{f_{\boldsymbol{x}}(t, t_1)}{S_{\boldsymbol{x}}(t, t_0)} &= \frac{h_{\boldsymbol{x}} \cdot \exp\big(-h_{\boldsymbol{x}} \cdot (t_1 - t)\big)}{\exp\big(-h_{\boldsymbol{x}} \cdot (t_0 - t)\big)} \\ &= h_{\boldsymbol{x}} \cdot \exp\big(-h_{\boldsymbol{x}} \cdot (t_1 - t_0)\big) = f_{\boldsymbol{x}}(t_0, t_1) \ .\end{aligned}$$

Thus, we eventually obtain the same expression (5.20). This is due to the fact that a process with a constant hazard rate is "memoryless".

5.3.2 Parallel Event Sequences

In a streaming setting, we assume to observe a sequence of recurrent events $\mathcal{T}(\boldsymbol{x}) = \{t_\tau\}_{\tau=1}^{k}$ not only for a single instance $\boldsymbol{x}$ but for a fixed set of J instances $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_J\}$, with $\boldsymbol{x}_j = (x_{j1}, \ldots, x_{jn})^\top$. Thus, the data relevant to a time window $[a, b]$ is given in the form of J parallel event sequences

$$\begin{aligned}\mathcal{D} &= \Big(\mathcal{T}(\boldsymbol{x}_1), \ldots, \mathcal{T}(\boldsymbol{x}_J)\Big) \\ &= \Big(\{t_{1\tau}\}_{\tau=1}^{k_1}, \ldots, \{t_{J\tau}\}_{\tau=1}^{k_J}\Big) \ ,\end{aligned} \tag{5.21}$$

where k_j is the number of events for $\boldsymbol{x}_j$ and $\{t_{j_\tau}\}_{\tau=1}^{k_j}$ the corresponding time points. Assuming independence, the probability of $\mathcal{D}$ is

$$\begin{aligned}\mathbf{P}(\mathcal{D}) &= \prod_{j=1}^{J} \mathbf{P}(\mathcal{T}(\boldsymbol{x}_j)) \\ &= \prod_{j=1}^{J} \left[(h_{\boldsymbol{x}_j})^{k_j} \cdot \prod_{\tau=1}^{k_j+1} \exp\Big(-h_{\boldsymbol{x}_j} \cdot (t_{j\tau} - t_{j\tau-1})\Big) \right] \\ &= \prod_{j=1}^{J} \Big[(h_{\boldsymbol{x}_j})^{k_j} \cdot \exp\Big(-h_{\boldsymbol{x}_j} \cdot (b - a)\Big) \Big] \ ,\end{aligned}$$

[2] To some extent, this is comparable with statistical methods like kernel density estimation or locally weighted linear regression.

and the logarithm of this probability is

$$\log\left(\prod_{j=1}^{J}\mathbf{P}(\mathcal{T}(\boldsymbol{x}_j))\right) = \sum_{j=1}^{J}\left[k_j \log\left(h_{\boldsymbol{x}_j}\right) - \sum_{\tau=1}^{k_j+1} h_{\boldsymbol{x}_j}\cdot(t_{j\tau} - t_{j\tau-1})\right]$$
$$= \sum_{j=1}^{J}\left[k_j \log\left(h_{\boldsymbol{x}_j}\right) - h_{\boldsymbol{x}_j}\cdot(b-a)\right]. \tag{5.22}$$

Notice that this likelihood differs from the partial likelihood (5.15), the likelihood used to estimate the parameters of the Cox model. Recall that in the streaming setting, events are assumed to be recurrent and instances never leave the risk set, thus the risk set in the denominator of (5.15) becomes constant for all events. Hence, the partial likelihood fails to maximize the probability it was originally designed to maximize, namely the probability of the observed order of events.

For the model (5.18), the expression in (5.22) yields the following log-likelihood function for the parameter vector $\boldsymbol{\beta}$:

$$LL(\boldsymbol{\beta}) = \sum_{j=1}^{J}\left[k_j\left(\sum_{i=0}^{n}\beta_i x_{ji}\right) - \sum_{\tau=1}^{k_j+1}\exp\left(\sum_{i=0}^{n}\beta_i x_{ji}\right)(t_{j\tau} - t_{j\tau-1})\right]$$
$$= \sum_{j=1}^{J}\left[k_j\left(\sum_{i=0}^{n}\beta_i x_{ji}\right) - \exp\left(\sum_{i=0}^{n}\beta_i x_{ji}\right)(b-a)\right]. \tag{5.23}$$

5.3.3 Adaptive ML Estimation

Parameter estimation on a time window $[a, b]$ can now be done by means of maximum likelihood estimation (MLE), i.e., by finding the maximizer of the above likelihood function:

$$\boldsymbol{\beta}^* = (\beta_0^*, \beta_1^*, \ldots, \beta_n^*) \;=\; \underset{\boldsymbol{\beta}}{\operatorname{argmax}}\, \ell(\boldsymbol{\beta}) \tag{5.24}$$

Unfortunately, there is no analytical expression for $\boldsymbol{\beta}^*$, so that the estimator needs to be found by means of numerical optimization procedures. Nevertheless, because the log-likelihood function $LL(\boldsymbol{\beta})$ is concave (which is proven at the end of this subsection by showing that the corresponding conditions on the second derivatives are satisfied), simple gradient-based optimization techniques and online versions of gradient descent [29] can be applied and turned out to work rather well.

The use of local optimization techniques is also reasonable as it can be transformed quite naturally into an incremental learning algorithm, applicable in our streaming

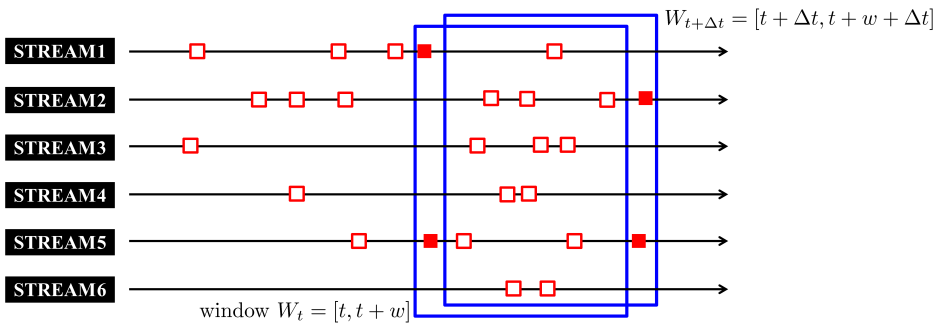


Figure 5.3: Illustration of the shift of the time window: The current window $W_t = [t, t+w]$ is replaced by the new one $W_{t+\Delta t} = [t+\Delta t, t+w+\Delta t]$. While the status of some of the events changes (filled boxes), the status of the others (non-filled boxes) remains the same (either outdated or active).

setting. Recall that we slide a time window of fixed length w along the time axis. More specifically, the window is repeatedly moved in discrete steps, each time replacing the current window $W_t = [t, t+w]$ by the shifted one $W_{t+\Delta t} = [t+\Delta t, t+w+\Delta t]$. A shift of this kind also changes the parallel event sequences (5.21) associated with the current time window and, therefore, demands a re-estimation of the parameter vector $\boldsymbol{\beta}$. Typically, the event sequences $\mathcal{T}(\boldsymbol{x}_j)$ will change slightly because most of the current events $t_{j\tau}$ will remain inside the window—only those close to the lower boundary t will fall out (namely those with $t \leq t_{j\tau} < t + \Delta t$), while new events observed between $t + w$ and $t + w + \Delta t$ will be added, see Figure 5.3 for an illustration. In any case, the new ML estimate of $\boldsymbol{\beta}$ will normally be found in close proximity to the old one. Therefore, the current estimate $\boldsymbol{\beta}_t^*$, i.e., the ML estimate for the current time window $W_t = [t, t+w]$, will provide a good initial solution for the re-estimation problem to be solved by our gradient-based optimizer. Indeed, in practical experiments, we found that only a few adaptation steps are generally needed to reach the new ML estimate $\boldsymbol{\beta}_{t+\Delta t}^*$ (with sufficient accuracy).

This adaptive estimation procedure eventually produces a sequence of parameter estimates that (implicitly) represents the evolution of both the parameter $\boldsymbol{\beta}$ and the hazard rates $h_{\boldsymbol{x}_j} = h(t|\boldsymbol{x}_j)$ over time. More specifically, for a fixed time point τ, let $\mathcal{W}_\tau$ denote the set of all time windows covering this time point:

$$\mathcal{W}_\tau = \{ W_t \,|\, \tau \in [t, t + \Delta t] \}$$

Moreover, let $\boldsymbol{\beta}_t^*$ denote the ML estimation of $\boldsymbol{\beta}$ on W_t. Parameter $\boldsymbol{\beta}$ at time τ is

then defined by averaging:

$$\boldsymbol{\beta}(\tau) = \frac{1}{|\mathcal{W}_\tau|} \sum_{W_t \in \mathcal{W}_\tau} \boldsymbol{\beta}_t^* \tag{5.25}$$

Correspondingly, the hazard rate $h(\tau|\boldsymbol{x})$ for the instance $\boldsymbol{x}$ at time τ is given by

$$h(\tau|\boldsymbol{x}) = \exp\left(\boldsymbol{x}^\top \cdot \boldsymbol{\beta}(\tau)\right) \ . \tag{5.26}$$

Finally, one may wonder whether a simple count of events on a sliding window would conclude the same observations and results. This is a justified question; especially given that the only parameter of the exponential distribution, the rate λ, is found to be the inverse of the mean time between the events (by maximizing the likelihood of the data), i.e., for the k events at times $t_1, \ldots, t_k$ the rate is $\lambda = \frac{1}{\sum_{i=2}^{k}(t_i - t_{i-1})}$. This is true when events occur either to a single instance or to identical copies of the same instance, which makes all covariates superficial and redundant. However, in our setting, instances differ greatly based on their covariates; from these covariates, we discover the prognostic covariates and build a model that captures the dependencies in local subspaces.

Claim. *The log-likelihood function $LL(\boldsymbol{\beta})$ (5.23) is a concave function.*

Proof. We show that the log-likelihood function $LL(\boldsymbol{\beta})$ is concave by showing that its Hessian matrix $\mathbf{H}$, the matrix of the second-order partial derivatives s.t. $\mathbf{H}_{v,u} = \frac{\partial^2 LL(\boldsymbol{\beta})}{\partial \beta_v \partial \beta_u}$, is negative-semidefinite, i.e., $\forall z \in \mathbb{R}^n : z^* \mathbf{H} z \leq 0$, where z^* is the conjugate transpose of z. Each entry $\mathbf{H}_{v,u}$ of the Hessian matrix can be written as

$$\mathbf{H}_{v,u} = \sum_{j=1}^{J} \left[-x_{ju} x_{jv} \exp\left(\sum_{i=0}^{n} \beta_i x_{ji} \right) (b-a) \right] \tag{5.27}$$

$$= \sum_{j=1}^{J} \left[-x_{ju} x_{jv} Q_j \right] \ , \tag{5.28}$$

such that $Q_j = \exp\left(\sum_{i=0}^{n} \beta_i x_{ji}\right)(b-a) \geq 0$. As a result, the Hessian matrix can be

written as a sum of J matrices $\mathbf{H}_j$:

$$\mathbf{H} = \begin{bmatrix} \sum_{j=1}^{J} \left[-x_{j1}^2 Q_j\right] & \cdots & \sum_{j=1}^{J} \left[-x_{j1}x_{jn} Q_j\right] \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^{J} \left[-x_{jn}x_{j1} Q_j\right] & \cdots & \sum_{j=1}^{J} \left[-x_{jn}^2 Q_j\right] \end{bmatrix} \tag{5.29}$$

$$= -\sum_{j=1}^{J} Q_j \cdot \begin{bmatrix} x_{j1}^2 & \cdots & x_{j1}x_{jn} \\ \vdots & \ddots & \vdots \\ x_{jn}x_{j1} & \cdots & x_{j,n}^2 \end{bmatrix} \tag{5.30}$$

$$= -\sum_{j=1}^{J} Q_j \cdot \mathbf{H}_j \tag{5.31}$$

The matrix H_j is clearly positive-semidefinite because it is the Gram matrix for the vector $\boldsymbol{x}_j$:

$$\mathbf{H}_j = \begin{bmatrix} x_{j1}^2 & \cdots & x_{j1}x_{jn} \\ \vdots & \ddots & \vdots \\ x_{jn}x_{j1} & \cdots & x_{j,n}^2 \end{bmatrix} = \boldsymbol{x}_j \boldsymbol{x}_j^\top \tag{5.32}$$

Consequently, $Q_j \cdot \mathbf{H}_j$ is positive-semidefinite in that $Q_j \geq 0$; the matrix $\sum_{j=1}^{J} Q_j \cdot \mathbf{H}_j$ is also positive-semidefinite as it is a sum of positive-semidefinite matrices. Thus, the matrix $-\sum_{j=1}^{J} Q_j \cdot \mathbf{H}_j$ is negative-semidefinite, which leads to the conclusion that the function $LL(\boldsymbol{\beta})$ is concave. □

5.4 Case Study: Earthquake Analysis

We conduct two case studies, for a proof of concept, in which our streaming version of survival analysis is used for spatio-temporal data analysis. While the temporal aspect is naturally captured by the hazard rate model, the spatial aspect is incorporated through the use of spatial information as covariates of the data streams. This suggests that the vector (5.1) of covariates describes the spatial location of a data source.

In the first study, our method is applied to the analysis of earthquake data. The data is collected from the USGS[3] (United States Geological Survey), specifically from the catalog of NEIC[4] (National Earthquake Information Center). The mission of

[3] `http://www.usgs.gov`, accessed on October 8, 2015

[4] `http://earthquake.usgs.gov/contactus/golden/neic.php`, accessed on October 8, 2015

these organizations is to quickly discover the most recent destructive earthquakes in terms of location and magnitude and then broadcast this information to international agencies and scientists.

5.4.1 Data Generation

The earthquake data was collected in the time period between January 1, 2000 and March 2, 2012. Because entries in the USGS/NEIC catalog can be added or modified at any time, only the data in the catalog at the time of exportation is used, namely April 12, 2012. Table 5.1 presents an example of earthquake data, in which a list of five earthquakes with their occurrence time and attributes is shown; these earthquakes are the first to occur on January 1, 2012.

The online catalog of USGS/NEIC retains only significant earthquakes with a magnitude bigger than 2.5, though very few micro-earthquakes (with a magnitude less than 1) could be found. There are even a few earthquakes with missing magnitudes. In total, we collected the data of 319,884 earthquakes throughout the globe in the given time period.

Year	Month	Day	UTC Time hhmmss.mm	Latitude	Longitude	Mag.	Depth	Catalog
2012	01	01	003008.77	12.008	143.487	5.1	35	PDE-W
2012	01	01	003725.28	63.337	-147.516	3.0	65	PDE-W
2012	01	01	004342.77	12.014	143.536	4.4	35	PDE-W
2012	01	01	005008.04	-11.366	166.218	5.3	67	PDE-W
2012	01	01	012207.66	-6.747	130.007	4.2	145	PDE-W

Table 5.1: A sample earthquake data containing five earthquakes occurred on the first day of 2012.

Every earthquake is identified by its geographic coordinates, the exact time of occurrence (up to the second), the magnitude and the depth. Figure 5.4(a) depicts a plot of the collected earthquakes, each of which is represented as a point at the place of its geographic location.

Recall that in the setting introduced in Section 5.3, we assume to observe event sequences for a fixed set of instances. In order to define these instances, we discretize the globe, both in terms of latitude and longitude, and associate one instance with each intersection point. More specifically, with $\phi \in \{-90, -89, \ldots, 90\}$ for latitude and with $\lambda \in \{-180, -179, \ldots, 180\}$ for longitude, the total number of instances becomes $181 \times 361 = 65,341$. The regions produced are obviously not equal in

area because longitudes are not parallel lines like latitudes; therefore, areas near the equator are larger than those closer to the poles.

Furthermore, recall that each instance is described in terms of features (covariates) x_i, which, according to (5.18), have a proportional effect on the hazard rate. In order to account for possibly nonlinear dependencies between spatial coordinates and the risk of an earthquake, we define these features in terms of a fuzzy partition; a partition defined in terms of fuzzy sets [184]. In contrast to a standard partition defined in terms of intervals, this allows for a smooth transition between spatial regions. More specifically, we discretize both latitude and longitude by means of triangular fuzzy sets as shown in Figure 5.4(b). A two-dimensional (fuzzy) discretization of the globe is defined in terms of the Cartesian product of these two one-dimensional discretizations, using the minimum operator for fuzzy set intersection. The covariates of an instance $\boldsymbol{x}$ associated with coordinates (ϕ, λ) are then simply given by the membership degrees in all these two-dimensional fuzzy sets, i.e., the covariates are of the form

$$x_{i,j} \;=\; \min\Big(A_i(\phi), B_j(\lambda)\Big) \;\;,$$

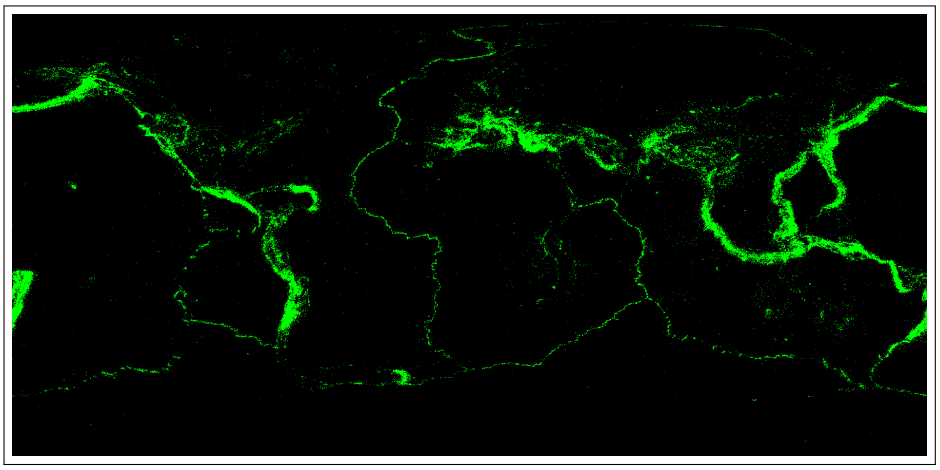
where A_i is one of the 10 fuzzy sets for latitude and B_j one of the 12 fuzzy sets for longitude; thus, each instance is of the form

$$\boldsymbol{x} = \big(x_{1,1}, x_{1,2}, \ldots, x_{1,12}, \ldots, x_{10,12}\big) \in [0,1]^{120} \;\;.$$

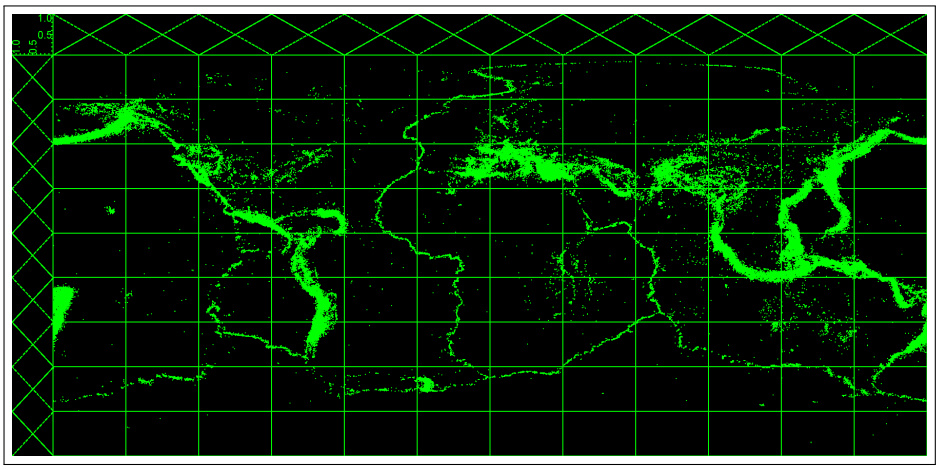
The Mercator projection, used to project both coordinates in Figure 5.4(b), is meant to preserve angles and the shapes of small objects. As a result, distances of objects are distorted and lines meeting at the poles become parallel. For this reason, we attempt to keep the fuzzy partition as coherent as possible, i.e., fuzzy sets defined on the longitudes should have the same width, independent of their latitude. This can be realized by applying the haversine formula to preserve the distances on the Earth's surface (approximated as a sphere with a radius of 6371 km), as opposed to applying the Euclidean distance between the geometric coordinates. The vertical fuzzy set at the longitude $\lambda = 0$ is shown in 5.4(c), projected with the Mercator projection.

The distance d between two points, with the coordinates (ϕ_1, λ_1) and (ϕ_2, λ_2), on the globe with radius r can be derived from the haversine formula and is given by

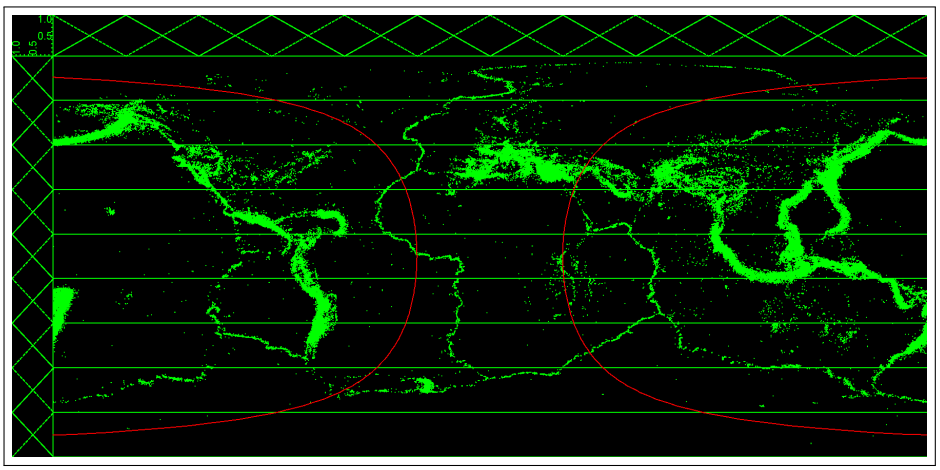
$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \;\;. \qquad (5.33)$$



(a)



(b)



(c)

Figure 5.4: The collected data set of earthquakes, plotted by their geographic coordinates. The data contains earthquakes between the January 1, 2000 until midnight March 27, 2012. (a) earthquakes only; (b) with fuzzy partitions on the two coordinates; (c) the center longitude fuzzy set after correction with the haversine formula. The two red lines represent the Mercator projection of the center latitude fuzzy set.

5.4.2 Results

Given the data produced in this way and after sorting all earthquakes by their time of occurrence, we are able to apply our method as outlined in Section 5.3. We set the length of the time window to three months and the shift parameter Δ to one week. The results we obtain in terms of time-dependent estimates of the parameters $\beta_{i,j}$, each of which is associated with a covariate $x_{i,j}$ and hence with a spatial (fuzzy) region $A_i \times B_j$, appear to be quite plausible. Several interesting observations can be made for data from the last decade. We focus on three of the most significant earthquakes that occurred in 2008 and 2011:

- The May 2008 Great Sichuan earthquake[5] occurred on Monday, May 12, 2008 at 06:28:01 UTC. At a magnitude of 7.9 (M_w) and an epicenter 30.986° N, 103.364° E. This event can be assigned to the nearest instance whose sparse feature vector has the following nonzero entries $\boldsymbol{x}^\top = [x_{7,10} = 0.63, x_{7,11} = 0.51, x_{8,10} = 0.05, x_{8,11} = 0.05]$.

- The February 2011 Christchurch earthquake[6] occurred on Monday, February 21, 2011 at 23:51:42 UTC. At a magnitude of 6.1 (M_w) and an epicenter 43.583° S, 172.680° E. The instance whose sparse feature vector has the following nonzero entries $\boldsymbol{x}^\top = [x_{3,1} = 0.65, x_{3,2} = 0.07, x_{3,12} = 0.46, x_{4,1} = 0.35, x_{4,2} = 0.07, x_{4,12} = 0.35]$ is the nearest to the epicenter.

- The March 2011 earthquake[7] off the Pacific coast of Tōhoku occurred on Friday, March 11, 2011 at 05:46:24 UTC. At a magnitude of 9.0 (M_w) and an epicenter 38.297° N, 142.372° E. The instance whose sparse feature vector has the following nonzero entries $\boldsymbol{x}^\top = [x_{7,1} = 0.002, x_{7,11} = 0.42, x_{7,12} = 0.6, x_{8,1} = 0.002, x_{8,11} = 0.4, x_{8,12} = 0.4]$ is the nearest to the epicenter.

As can be seen in Figure 5.5, the occurrence of the three earthquakes is accompanied with a significant increase in the coefficients of the fuzzy sets covering these areas. The higher the fuzzy membership of an instance in a given two-dimensional fuzzy set, the more relevant the coefficient, associated with the fuzzy set, to the overall hazard. For this reason, we present only the relevant coefficients in Figure 5.5. Notably, the

[5] `http://earthquake.usgs.gov/earthquakes/eqinthenews/2008/us2008ryan/`, accessed on October 9, 2015

[6] `http://earthquake.usgs.gov/earthquakes/eqinthenews/2008/us2008ryan/`, accessed on October 9, 2015

[7] `http://earthquake.usgs.gov/earthquakes/eqinthenews/2008/us2008ryan/`, accessed on October 9, 2015

coefficients as given by (5.18) are logarithmically inversely proportional, indicating that, an increase in one coefficient is calibrated by a decrease in other coefficients (without changing the estimated hazard). Although the different coefficients can be used as prognostic factors, the real change of the hazard can be better observed in the estimated hazard (5.18), which is shown in Figure 5.6 as hazard curves for the three studied areas. The figure reveals how the hazard rate significantly increases even before the occurrences of these earthquakes.

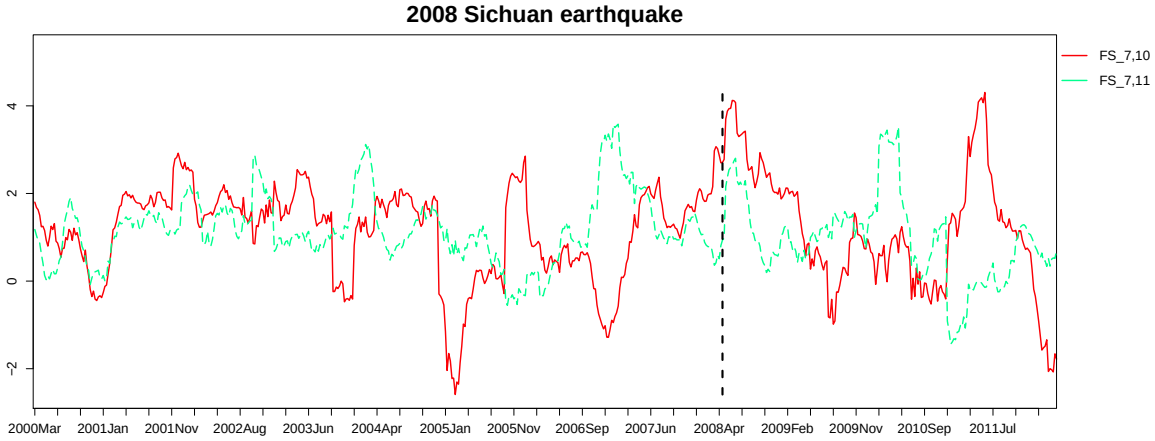
5.5 Case Study: Twitter Data

Our second case study is based on data collected from Twitter[8], which is an online microblogging web site. Twitter is a service that allows users to send short messages of up to 140 characters known as *tweets*. Every tweet is attributed by some meta data, including the ID of the user who wrote it and the time the tweet was sent. Further attributes can be extracted from the tweet with the permission of the user. Those attributes indicate the user's geolocation when the tweet was posted; this is supported by a GPS (Global Positioning System) functionality embedded in a mobile device or a tablet PC. The geolocation is represented as a tuple (`lat`, `long`) with an entry for the latitude and for the longitude. Table 5.2 gives an example of Twitter data, written in Json[9] format. The table contains two Twitter messages, after removing unimportant attributes, whereas important ones are written in bold. The shown messages are artificially created with no real user information.

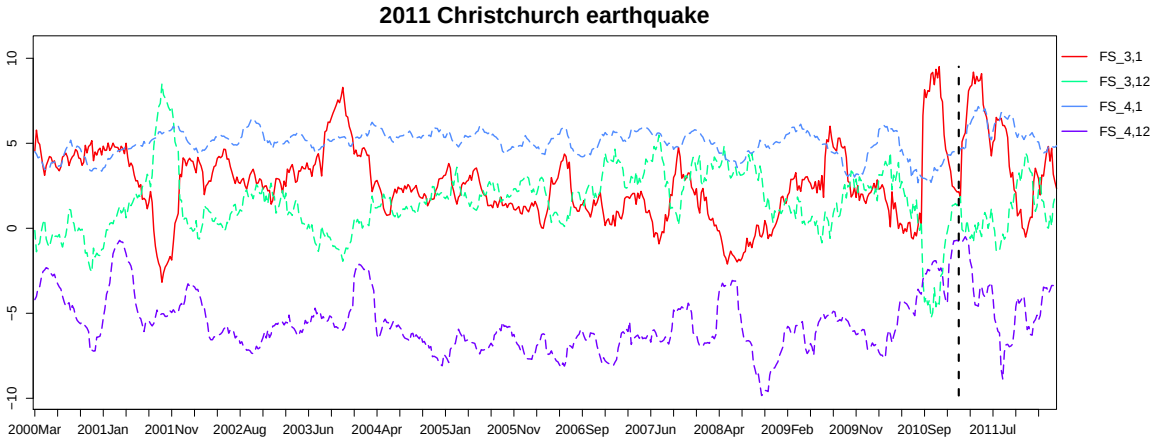
We collected tweets generated inside the bounding box of Germany, which is determined by the corner points (`lat`, `long`) = (47°16′N, 5°52′E) and (55°03′N, 15°02′E). This data was collected during a two months period; from March 20, 2012 until May 27, 2012. In total, we collected about 4.9 million tweets originating from Germany and its surrounding countries (Denmark, Poland, Czech Republic, Austria, Switzerland, France, Belgium and the Netherlands). Germany accounted for only 1.8 million of these tweets.

Similar to the previous study on earthquakes, we apply a discretization on the area of Germany, considering every intersection point of the two coordinates with $\phi \in \{47.1, 47.2, \ldots, 55.1\}$ for latitude and $\lambda \in \{5.5, 5.6, \ldots, 15.2\}$ for longitude, provided the intersection lies inside the borders of Germany. As a result, we maintain $5,013$ intersection points $p_j = (\phi_j, \lambda_j)$.

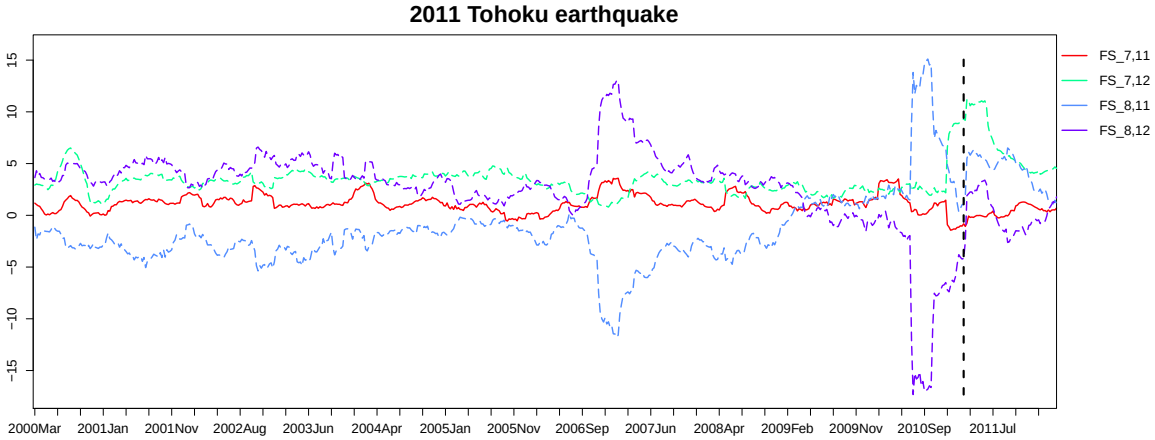
[8] `http://www.twitter.com`, accessed on October 9, 2015
[9] `http://json.org`, accessed on October 9, 2015



(a)

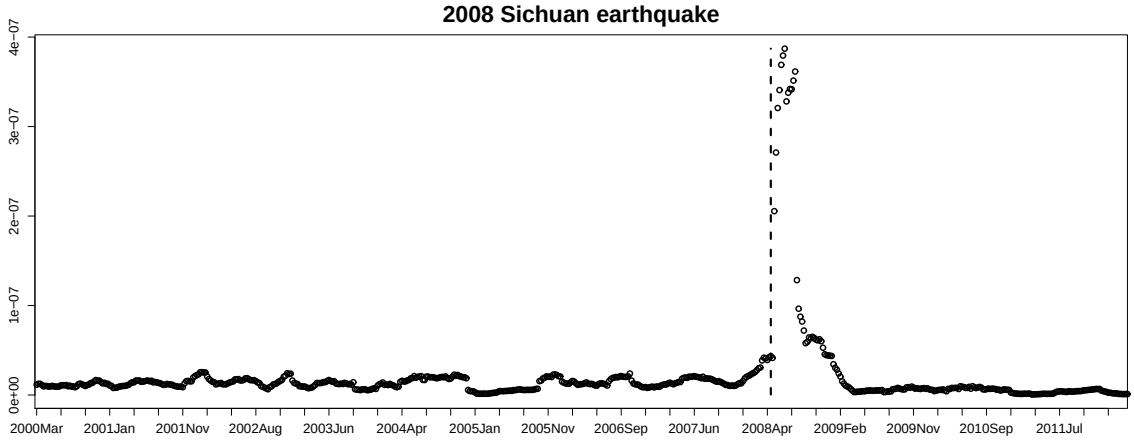


(b)

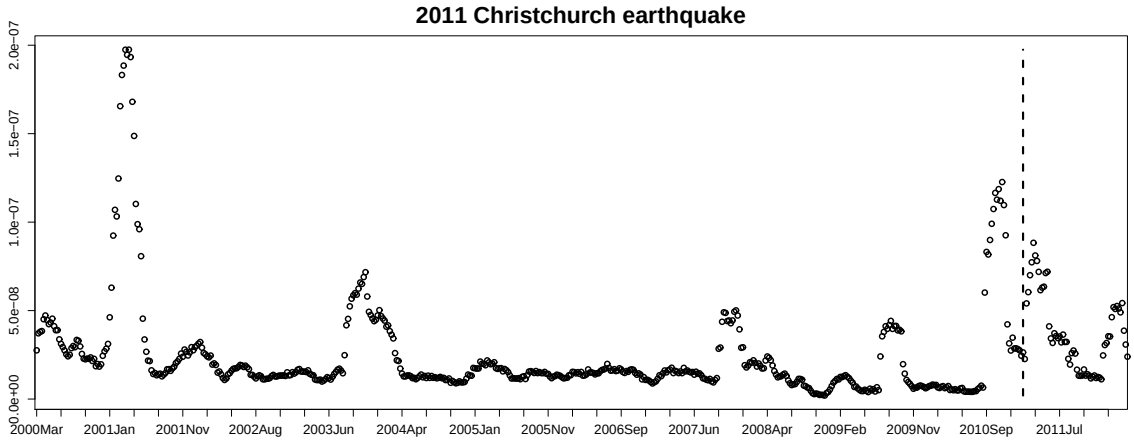


(c)

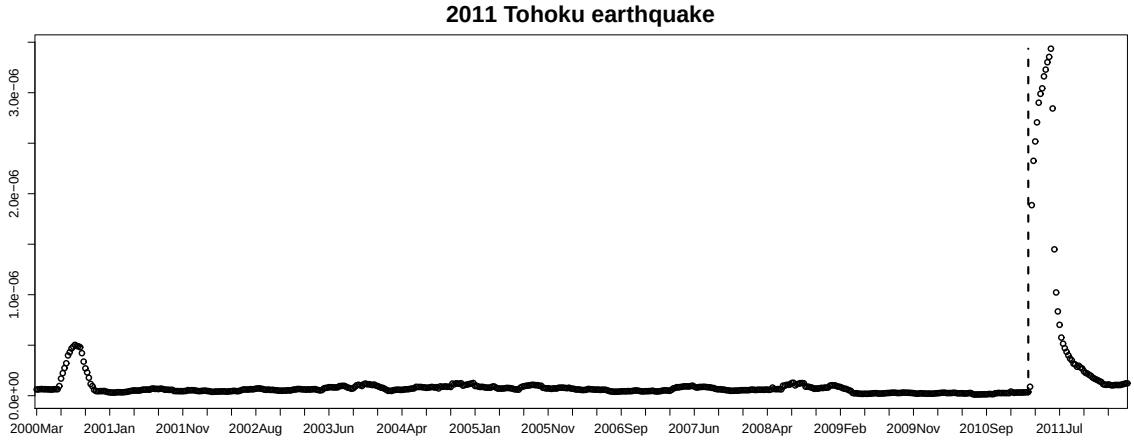
Figure 5.5: Coefficients for the areas with significant earthquakes in 2008 and 2011. The exact date of each earthquake is marked as a dashed vertical line.



(a)



(b)



(c)

Figure 5.6: The hazard values for the areas with significant earthquakes in 2008 and 2011. The exact date of each earthquake is marked as a dashed vertical line.

favorited:false, **text**:'Stau: A8 München Richtung Stuttgart 6 km zur Ausfahrt im Schneckentempo..', truncated:false, **created_at**:Fri Feb 10 10:38:47 +0000 2012, retweeted:false, retweet_count:0, coordinates:type:Point, **coordinates**:[9.55755, 48.6333], ..., entities:user_mentions:[], urls:[], hashtags:[], geo:type:Point, coordinates:[48.6333, 9.55755], ..., place:bounding_box:type:Polygon, coordinates:[[[9.534815, 48.616779], [9.594667, 48.616779], [9.594667, 48.640891], [9.534815, 48.640891]]], place_type:city, ..., country_code:DE, attributes:, full_name:Aichelberg, Göppingen, name:Aichelberg, id:29ef9f01a553e601, country:Germany, ..., id_str:###, user:default_profile:true, notifications:null, ..., time_zone:Berlin, created_at:Fri Sep 03 14:25:38 +0000 2010, verified:false, geo_enabled:true..., favourites_count:0, lang:de, ..., followers_count:335, ..., location:Karlsruhe, ..., name:###, ..., listed_count:21, following:null, screen_name:###, id:###, ..., statuses_count:10935, utc_offset:3600, friends_count:0, ..., id:###, ...
text:'top atmosphere in Weserstadion today, a very good match...', ..., **created_at**:Tue Apr 10 21:37:28 +0000 2012, place:bounding_box:type:Polygon, coordinates:[[[8.481599, 53.011035], [8.990593, 53.011035], [8.990593, 53.228969], [8.481599, 53.228969]]], country:Germany, attributes:, full_name:Bremen, Bremen, .., country_code:DE, name:Bremen, id:9467fbdc3cdbd2ef, place_type:city, coordinates:type:Point, **coordinates**:[8.837596, 53.06693] , retweeted:false, in_reply_to_status_id:null, ..., truncated:false, contributors:null, possibly_sensitive:false, in_reply_to_screen_name:null, favorited:false, user:default_profile:false, follow_request_sent:null, lang:de, friends_count:200, ..., is_translator:false, created_at:Sat May 23 13:01:45 +0000 2009, id_str:###, ..., url:null, following:null, verified:false, ..., location:Germany, ..., statuses_count:4537, ..., time_zone:Berlin, .., utc_offset:3600, followers_count:432, ..., id:###, retweet_count:0

Table 5.2: A sample Twitter data containing two Twitter messages.

The next step is to find a proper representation of the intersection points in terms of covariates; we describe every instance (data streams) $\boldsymbol{x}_j$ by the normalized vector of Mahalanobis distances to the center of each of the 16 German states. Thus, each instance is represented in terms of a vector

$$\boldsymbol{x}_j = (x_{j,1}, \ldots, x_{j,16})^\top \in \mathbb{R}^{16} \ ,$$

where $x_{j,i}$ is the distance of the intersection points p_j from the geometric center of the ith German state. By sorting the tweets according to their creation times, considering only those originating from Germany and assigning every tweet to the closest instance $\boldsymbol{x}_j$, we obtain a parallel stream of events that can again be processed by our method described in Section 5.3. The reason for applying the Mahalanobis distance

$$d_{Mahalanobis}(\boldsymbol{x}, \boldsymbol{c}) = \sqrt{(\boldsymbol{x} - \boldsymbol{c})^T S^{-1} (\boldsymbol{x} - \boldsymbol{c})}$$

is that states differ in the size of the area they occupy and how they are spread along the two coordinates, causing points further from the center of a large state to gain a smaller weight compared to points that are close but not in small states. This influence of smaller states is weakened by the covariance matrix S in the denominator. The vector $\boldsymbol{c}$ is the geometric center of the state and S is the covariance matrix, which describes the spread of the state along the coordinates.

We fix the window size to three days and the shift parameter Δt to one day. As a result, we again obtain time-dependent estimates (5.25) of the parameters $\beta_1, \ldots, \beta_{16}$ associated with the 16 German states. Figure 5.7 shows how the estimated parameters change over time, compared with the base line hazard α_0 that is also plotted in each subfigure. An increasing parameter β_i can be interpreted as follows: The closer a location $\boldsymbol{x}_j$ is to the corresponding state, the higher the hazard becomes, or in this case the propensity of users to send a tweet from that location. Conclusively, users within that state or nearby are more active in the sense of sending more tweets.

In Figures 5.7(a) and (d), the parameter for the state of Berlin is increasing in the time between May 2 and May 5, while the parameter for the state Brandenburg is decreasing. In search for an explanation for this observation, we found that the conference re:publica[10] took place during that time. This is a conference for bloggers from Germany and all around the world. Consequently, one can expect that more bloggers were in Berlin and less in the surrounding areas, including the state of Brandenburg.

[10] `http://re-publica`, accessed on October 5, 2012

The opposite can be said about Saxony-Anhalt, which was seen as a gateway for travelers, so its parameter was also increasing during that time. The parameters associated with the mentioned states are marked by the '$*$' symbol in Figure 5.7. Similarly, Figure 5.7(d) shows how the hazard associated with the state Schleswig-Holstein, marked by the '+' symbol, has increased on April 28. This was supposedly a direct effect of hosting a conference for the "Piratenpartei", a political party in Germany.

In a second experiment with the same stream of events, our aim was to observe changes between the city and the countryside. This was done by considering only instances located inside the states of Bremen and Lower Saxony[11]. Instances are now described only by a single binary covariate, indicating whether an instance is located in Bremen or not. Figure 5.8 shows how the corresponding parameter changes on a weekly basis. Interesting patterns can be observed especially for the weekends. First, there are normal weekends where people move from the condensed area of Bremen to the surrounding state, causing a decrease in the hazard (less tweets sent from inside Bremen and more from outside); this pattern is marked by the '+' symbol. Second, the weekends on which the local soccer club (Werder Bremen) has hosted a soccer match in the German soccer league (Bundesliga), causing an increase in the hazard; this group is marked by '$*$' symbol.

5.6 Conclusion

In this chapter, an adaptive approach to survival analysis on data streams is introduced. To this end, we adopt a sliding window approach and propose an adaptive (online) variant of a model that is closely related to the well-known Cox proportional hazard model. In this approach, maximum likelihood estimation of the model parameters is performed repeatedly, adapting the estimates whenever the time window has been shifted.

The assumption of a constant hazard rate (5.18) has led to an exponential model which exhibits both properties: the proportional hazard and the accelerated failure time, explained in Section 5.2.4. The model's coefficients vector $\boldsymbol{\beta}^*$ (5.24) is estimated by means of maximizing the likelihood of the distribution of events and the time spent between events; unlike the partial likelihood that maximizes the likelihood of the order of events. As a result, this coefficients vector also includes the baseline hazard h_0.

[11]Bremen is the smallest state in Germany, containing only two cities. It is surrounded by the larger state of Lower Saxony.

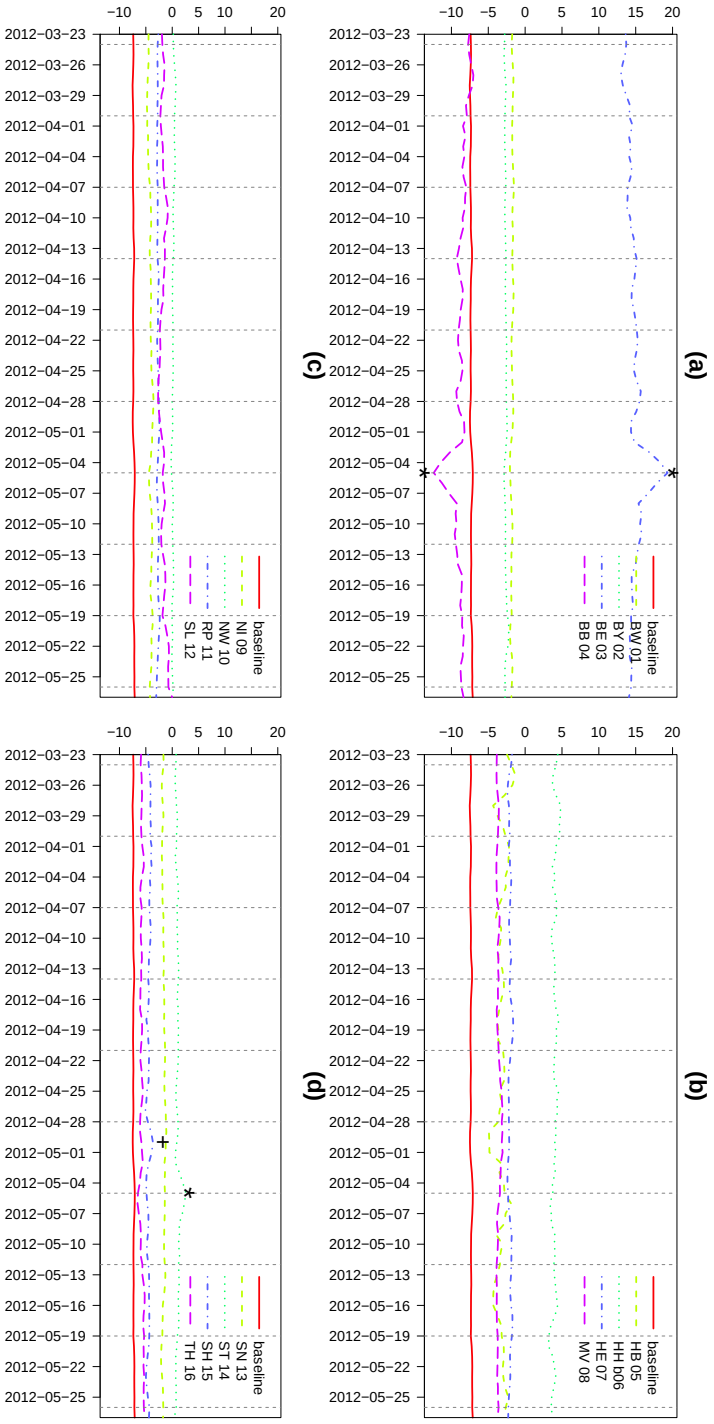


Figure 5.7: Parameters for the 16 German states together with the base line hazard α_0. BW: Baden-Württemberg, BY: Bavaria, BE: Berlin, BB: Brandenburg, HB: Bremen, HH: Hamburg, HE: Hesse, MV: Mecklenburg-Vorpommern, NI: Lower Saxony, NW: North Rhine-Westphalia, RP: Rhineland-Palatinate, SL: Saarland, SN: Saxony, ST: Saxony-Anhalt, SH: Schleswig-Holstein, TH: Thuringia.

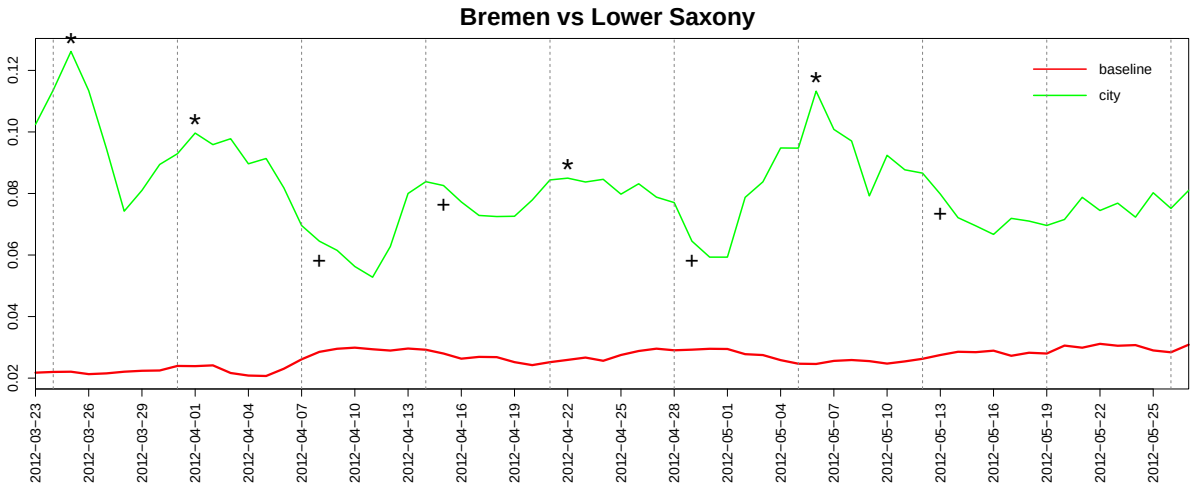


Figure 5.8: Baseline hazard and parameter distinguishing the city of Bremen from the surrounding state of Lower Saxony.

As a first proof of concept, we used our method for studying the occurrence of significant earthquakes during the last decade. Here, an event is an earthquake and a statistical entity is a two-dimensional region on the globe characterized by its spatial coordinates; more specifically, we make use of fuzzy discretization techniques in order to capture the influence of the spatial location on the hazard rate in a flexible way. The results we obtain are plausible and agree with expectation. For a region such as Tohoku, Japan, one can observe a significant increase in the hazard rate prior to the disastrous earthquake in 2011. Similar observations can be made for other significant earthquakes such as Sichuan's in 2008 and Christchurch's in 2011. Plausible results could also be obtained in a second study using streams of almost 5 million Twitter messages. Interesting patterns or irregularities in the time-dependent parameter estimations of our hazard model could be explained by massive events, such as conferences or soccer matches.

Chapter 6

Recovery Analysis for Adaptive Learning

Chapters 2, 3 and 4 present examples of how machine learning methods can be extended to learn adaptively from data streams in non-stationary environments.

Adaptive learners are often compared based on their empirical generalization performance, with the focus on their ability to properly react to concept changes. In this chapter, we develop a new type of experimental analysis called recovery analysis, which aims at assessing the ability of a learner to quickly discover a concept change and to take the appropriate actions to maintain the quality and generalization performance of the model. We develop recovery analysis for two types of supervised learning problems: classification and regression. As a practical application, we employ the recovery analysis in order to assess the recovery pattern of the state-of-the-art adaptive methods, with focus on comparing model-based and instance-based approaches.

6.1 Introduction

As explained in Chapter 1, learning from data streams has been a topic of active research in the recent past [66, 70]. Motivated by the idea of building a system that learns incrementally on a continuous and endless stream of data, this system should also be able to cope with changes in the data generating process.

Applying standard machine learning approaches in a data stream setting faces several challenges. Above all, the learner should react to a concept change in a proper way and should maintain a model that always reflects the current concept. Consequently, adapting to concept changes is often emphasized as a key feature of learning algorithms, because non-stationarity is arguably the most important difference between static and dynamic environments. While the idea of incremental learning is

crucial in the setting of data streams, it is not an entirely new problem and has been studied for learning from static data [142]. The ability of a learner to maintain the quality and generalization performance of the model in the presence of a concept drift is a property that becomes truly important when learning under changing environmental conditions.

In this chapter we propose *recovery analysis*; with the help of this analysis, we aim to assess a learner's ability to maintain its generalization performance in the presence of a concept drift. Roughly speaking, recovery analysis suggests a specific experimental protocol and a graphical presentation of the learner's performance that provides an idea of how quickly a drift is recognized, to what extent it affects the prediction performance, and how quickly the learner manages to adapt its model to the new condition. Our method utilizes real data in a modified and specifically prepared form, which is a main prerequisite for conducting controlled experiments under suitable conditions; therefore, it could be considered as a "semi-synthetic" approach.

Another contribution of the chapter is an experimental study, which illustrates the usefulness of recovery analysis by comparing different types of learning methods with regard to their ability to handle concept drift. In particular, we focus on the comparison of instance-based and model-based approaches for learning on data streams.

The remainder of the chapter is organized into the following sections: the next section recalls some important aspects of concept change. Our method of recovery analysis is introduced in Section 6.3. In Section 6.4, we contrast model-based with instance-based learning approaches and motivate their comparison, prior to describing the experiments and results in Section 6.5. The chapter closes with some concluding remarks in Section 6.6.

6.2 Learning under concept drift

In Chapter 2, we define a learning algorithm $\mathcal{A}$ (learning on a time-ordered stream of data $S = (\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \ldots)$, where $\boldsymbol{z}_t = (\boldsymbol{x}_t, y_t) \in \mathbb{X} \times \mathbb{Y}$) to be incremental, if it produces the predictive model $\mathcal{M}_t : \mathcal{X} \to \mathcal{Y}$ solely on the basis of $\mathcal{M}_{t-1}$ and $\boldsymbol{z}_t$, that is, $\mathcal{M}_t = \mathcal{A}(\mathcal{M}_{t-1}, \boldsymbol{z}_t)$. For reference, we briefly recall some of the main ideas discussed in Section 2.3.

The data generating process, according to which the sequence S is generated, is characterized by the probability measure $\mathbf{P}$ on $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. Under the stationarity and

independence assumptions, each new observation $\boldsymbol{z}_t$ is generated at random according to $\mathbf{P}$, i.e., the probability to observe a specific $\boldsymbol{z} \in \mathbb{Z}$ is given by $\mathbf{P}(\boldsymbol{z}) = \mathbf{P}(\boldsymbol{x}, y) = \mathbf{P}(\boldsymbol{x}) \cdot \mathbf{P}(y \mid \boldsymbol{x})$.

Giving up the assumption of stationarity (while keeping the one of independence), the probability measure $\mathbf{P}$ generating the next observation may possibly change over time. Thus, instead of a single measure $\mathbf{P}$, there is now a sequence of measures $(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \ldots)$, assuming that $\boldsymbol{z}_t$ is generated by $\mathbf{P}_t$. This is considered a concept change if these measures are not all equal [94].

Concept change [76] can be categorized into three types of change: *(i)* real concept change caused by the change of the conditional distribution $\mathbf{P}(y \mid \boldsymbol{x})$, *(ii)* virtual concept change caused by the change in the data's probability $\mathbf{P}(x)$ and *(iii)* global and local concept change, independent of the change's nature (real or virtual). The other important aspect when analyzing concept changes is the rate of change, which can also be categorized into *(i)* a concept shift caused by the abrupt change in the data generating process which makes the concept to be learned ($\mathbf{P}_t$) very different from the learned concept $\mathbf{P}_{t-1}$, *(ii)* a gradual change occurs when two different data generating processes, $\mathbf{P}_1$ and $\mathbf{P}_2$, are active at the same time, and the rate of their activation changes over time from favoring $\mathbf{P}_1$ to favoring $\mathbf{P}_2$ [171], *(iii)* an incremental change and *(iv)* the recurring concept.

Learning algorithms can handle concept change in an active or passive way. Passive approaches try to continuously utilize the most recent examples in updating the learned model, regardless of whether a concept drift has occurred or not. Active approaches, on the other hand, apply change detection techniques [76] as an indicator for concept drifts, see Section 2.4. Upon discovering a change, the learner decides on how the model should be updated; an update may often be drastic in order to forget the old concept and rapidly learn the emerging one.

6.3 Recovery Analysis

In practical studies, data streams are never truly infinite. Instead, the term "stream" indicates a large data set in the form of a long, yet finite sequence $S = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_T)$. In experimental studies, such streams are commonly used to produce a performance curve showing the generalization performance of a model sequence $(\mathcal{M}_t)_{t=1}^{T}$ over time. Although many of these studies are interested in analyzing the ability of a learner to deal with concept drift, such an analysis is inhibited by at least two problems:

- Ignorance about drift: For a real data stream S, it is normally not known whether it contains any concept drift, let alone when such a drift occurs.

- Missing baseline: Even if a concept drift is known to occur, it is often difficult to assess the performance of a learner or to judge how well it recovers after the drift, simply because a proper *baseline* is missing: The performance that could theoretically be reached, or at least be expected, is not known.

These problems are less of an issue if data is generated synthetically; for synthetic data, the "ground truth" is always known. Moreover, synthetic data has the big advantage of enabling controlled experiments. For example, one may be interested in how an algorithm reacts to a drift depending on certain characteristics of the drift, such as its strength and duration. While real data will contain a single drift, at most, these characteristics can easily be varied in experiments with synthetic data.

On the other hand, purely synthetic data begs the danger of being unrealistic or overly idealized. Therefore, in our approach to recovery analysis, we attempt to find a reasonable compromise by using a setting that qualifies as "semi-synthetic". The following section details how we use real data in a "manipulated" form that circumvents the above problems and allows us to conduct controlled experiments.

6.3.1 Main idea and experimental protocol

Instead of using a single data stream, our idea is to employ three streams in parallel, two "pure streams" and one "mixture". The pure streams

$$S_A = (\boldsymbol{z}_1^a, \boldsymbol{z}_2^a, \ldots, \boldsymbol{z}_T^a)$$
$$S_B = (\boldsymbol{z}_1^b, \boldsymbol{z}_2^b, \ldots, \boldsymbol{z}_T^b)$$

are supposed to be stationary and generated, respectively, according to distributions $\mathbf{P}_A$ and $\mathbf{P}_B$; in the case of real data, stationarity of a stream can be guaranteed, for example, by permuting the original stream at random.[1] These two streams must also be compatible, in the sense they are sharing a common data space $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. The mixture stream $S_C = (\boldsymbol{z}_1^c, \boldsymbol{z}_2^c, \ldots, \boldsymbol{z}_T^c)$ is produced by randomly sampling from the two pure streams:

$$\boldsymbol{z}_t^c = \begin{cases} \boldsymbol{z}_t^a & \text{with probability } \lambda(t) \\ \boldsymbol{z}_t^b & \text{with probability } 1 - \lambda(t) \end{cases} \tag{6.1}$$

[1] Permutation of data streams is also used in [172], albeit in a different way and for a different purpose.

For example, a concept drift can then be modeled by specifying the (time-dependent) sample probability $\lambda(t)$ as a sigmoidal function:

$$\lambda(t) = \left(1 + \exp\left(\frac{4(t - t_0)}{w}\right)\right)^{-1} . \tag{6.2}$$

This function has two parameters: t_0 is the mid point of the change process, while w is the length of this change. The length of the drift is related to the tangent of the sigmoid function at the center of the drift by $\tan\theta = \frac{1}{w}$. Using this transition function, the stream S_C is obviously drifting "from S_A to S_B": In the beginning, it is essentially identical to S_A, in a certain time window around t_0, it moves away from S_A toward S_B. In the end, it is essentially identical to S_B. Thus, we have created a gradual concept drift with a rate of change controlled by w.

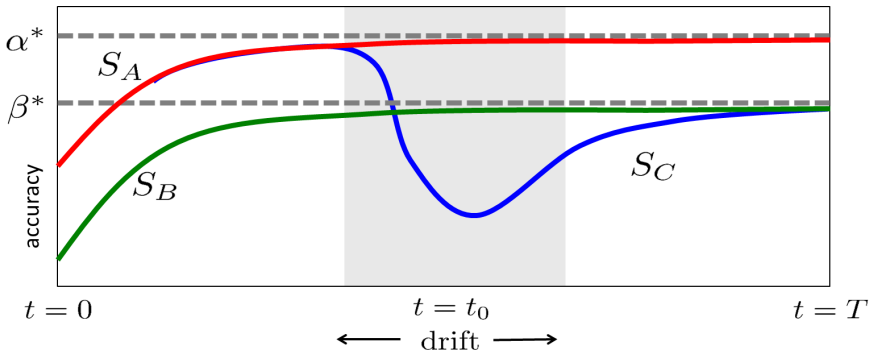


Figure 6.1: Schematic illustration of a recovery analysis: The three performance curves are produced by training models on the pure streams S_A and S_B, as well as on the mixed stream S_C, each time using the same learner $\mathcal{A}$. The region shaded in grey indicates the time window in which the concept drift (mainly) takes place. While the concept is drifting, the performance on S_C will typically drop to some extent. This can be seen by the drop in the classification accuracy.

Now, suppose the same learning algorithm $\mathcal{A}$ is applied to all three streams S_A, S_B and S_C. Since the first two streams are stationary, we expect to see a standard learning curve when plotting the generalization performance (for example, the classification accuracy) as a function of time. In the following, we denote the performance curves for S_A and S_B by $\alpha(t)$ and $\beta(t)$, respectively. These curves are normally concave, showing a significant increase in the beginning before reaching a certain saturation level later on; see Figure 6.1 for an illustration. The corresponding saturation levels α^* and β^* provide important information about the best performance that can be expected by the learner $\mathcal{A}$ on the pure streams S_A and S_B, respectively.

The performance curve $\gamma(t)$ is interesting for this analysis because it is a result of learning from the stream S_C, which exhibits a concept drift. In the beginning,

this curve will be essentially identical to the curve for S_A, so that the learner $\mathcal{A}$ should reach the level α^*. Upon the beginning of the concept drift, the performance is expected to drop and this decrease is supposed to continue until the drift ends and the learner $\mathcal{A}$ starts to recover. Eventually, $\mathcal{A}$ may or may not reach the level β^*. This level is indeed an upper bound on the asymptotic performance, since $\mathcal{A}$ cannot do better even when being trained on S_B from the very beginning. Thus, reaching this level indicates an optimal recovery.

Obviously, the performance curve for S_C provides vital information about the ability of $\mathcal{A}$ to deal with concept drift. In particular, the minimum of this curve indicates how strongly $\mathcal{A}$ is affected by the concept drift. Moreover, the curve provides information about how quickly the performance deteriorates (giving an idea of how sensitive $\mathcal{A}$ is). It also reveals how much time $\mathcal{A}$ needs to recover and whether or not it manages to recover optimally.

6.3.2 Bounding the optimal generalization performance

As explained above, the performance curve produced by a learner $\mathcal{A}$ on the stream S_C is expected to decrease while this stream is drifting from S_A to S_B. In order to judge the drop in performance, not only relatively in comparison to other learners but also absolutely, it would be desirable to have a kind of reference performance as a baseline. This leads to an interesting question: Is it possible to quantify our expectations regarding the drop in performance? More specifically, what is the optimal generalization performance

$$\gamma^*(t) \;=\; \sup_{\mathcal{M}\in\mathbf{M}} \gamma_{\mathcal{M}}(t) \tag{6.3}$$

we can expect on the stream S_C at time t? Here $\mathbf{M}$ is the underlying model class (i.e., the class of models that $\mathcal{A}$ can choose from), and $\gamma_{\mathcal{M}}(t)$ denotes the generalization performance of a model $\mathcal{M} \in \mathbf{M}$ on the mixture distribution (6.1), i.e.,

$$\mathbf{P}_C(t) = \lambda(t)\mathbf{P}_A + (1-\lambda(t))\mathbf{P}_B \ .$$

Our experimental setup allows for answering this question by exploiting knowledge about the performance levels $\alpha(t)$ and $\beta(t)$ that can be reached on S_A and S_B, respectively. Thus, there are models $\mathcal{M}_A, \mathcal{M}_B \in \mathbf{M}$ whose performance is $\alpha_{\mathcal{M}_A}(t) = \alpha(t)$ and $\beta_{\mathcal{M}_B}(t) = \beta(t)$. Now, suppose we were to apply the model $\mathcal{M}_A$ on the stream S_C. What is the expected generalization performance? Consider the case of classification, with the classification rate as a performance measure, which assumes values in the unit interval, with 0 and 1 indicating the worst and best performance, respectively.

If an example $(\boldsymbol{x}, y)$ on S_C is generated according to $\mathbf{P}_A$, the generalization performance of $\mathcal{M}_A$ on this example is the same as on S_A, namely $\alpha_{\mathcal{M}_A}(t)$. Otherwise, if the example is generated according to $\mathbf{P}_B$, nothing can be said about the performance of $\mathcal{M}_A$; thus, we can only assume the worst case performance of 0. Because the first case occurs with a probability of $\lambda(t)$ and the second one with a probability of $1 - \lambda(t)$, the overall expected performance of $\mathcal{M}_A$ is given by

$$\lambda(t) \cdot \alpha_{\mathcal{M}_A}(t) + (1 - \lambda(t)) \cdot 0 = \lambda(t) \cdot \alpha_{\mathcal{M}_A}(t) \ .$$

Following this same line of reasoning, the performance of the model $\mathcal{M}_B$ on the stream S_C is given by $(1-\lambda(t))\beta_{\mathcal{M}_B}(t)$. Choosing optimally from the two candidate models, M_A and M_B, can at least guarantee the performance

$$\gamma^{\bullet}(t) = \max\left\{\lambda(t) \cdot \alpha_{\mathcal{M}_A}(t), (1 - \lambda(t)) \cdot \beta_{\mathcal{M}_B}(t)\right\} \ . \tag{6.4}$$

Because the supremum in (6.3) is not only considered over $\{\mathcal{M}_A, \mathcal{M}_B\}$ but over the entire model class $\mathbf{M}$, $\gamma^{\bullet}(t)$ is only a lower bound on the optimal performance $\gamma^*(t)$, that is, $\gamma^{\bullet}(t) \leq \gamma^*(t)$. If the performance levels $\alpha(t)$ and $\beta(t)$ are already close enough to the optimal levels α^* and β^*, respectively, then (6.4) can be written more simply as

$$\gamma^{\bullet}(t) = \max\left\{\lambda(t) \cdot \alpha^*, (1 - \lambda(t)) \cdot \beta^*\right\} \ . \tag{6.5}$$

Strictly speaking, this estimation is not correct, since α^* and β^* are only limit values that will not necessarily be attained; however, this is of no importance, because we have to work with estimations of these values anyway.

The above estimation can be generalized to other performance measures or loss functions in a straightforward way, provided these measures assume values in a bounded range (not necessarily $[0, 1]$). As soon as the range is bounded, the worst performance can easily be derived. For example, consider the case of regression with the root mean squared error as a loss function and assume that $\mathbb{Y} = [y_{min}, y_{max}]$. The bound on the worst performance of an optimal model, analogous to (6.5), is

$$\begin{aligned}\gamma^{\bullet}(t) = \min\Bigg\{&\sqrt{(\alpha^*)^2 \cdot (1 - \lambda\,(t)) + l^2 \cdot \lambda\,(t)}, \\ &\sqrt{(\beta^*)^2 \cdot (\lambda\,(t)) + l^2 \cdot (1 - \lambda\,(t))}\Bigg\} \ ,\end{aligned} \tag{6.6}$$

where $l = |y_{min} - y_{max}|$, $(\alpha^*)^2$ is the mean squared error of $\mathcal{M}_A$ on S_A and $(\beta^*)^2$ the mean squared error of $\mathcal{M}_B$ on S_B. In contrast to the (the-higher-the-better)

classification rate, we are now estimating a (the-less-the-better) loss. Therefore, (6.6) is an upper bound on the optimal performance: $\gamma^*(t) \leq \gamma^\bullet(t)$.

Practically, the above estimate is not unproblematic. First, the output variable in regression does normally not have natural bounds y_{min} and y_{max}. Even if such limits can be established, the bound (6.6) will be very loose. A more realistic bound can be obtained by estimating the true performance of $\mathcal{M}_A$ on S_B and the true performance of $\mathcal{M}_B$ on S_A. In our experimental setting, this can be done by computing the average performance $\tilde{\alpha}$ of the model $\mathcal{M}_A$ (once it has stabilized and reached the performance α^* on S_A) on the data S_B. Likewise, the average performance $\tilde{\beta}$ of $\mathcal{M}_B$ can be obtained on S_A. Replacing l^2 in (6.6) by these estimates then yields

$$\gamma^*(t) \leq \min\left\{\sqrt{(\alpha^*)^2 \cdot (1-\lambda(t)) + (\tilde{\alpha})^2 \cdot \lambda(t)}, \sqrt{(\beta^*)^2 \cdot (\lambda(t)) + (\tilde{\beta})^2 \cdot (1-\lambda(t))}\right\} \quad . \tag{6.7}$$

6.3.3 Recovery measures

The major goal of our recovery analysis is to provide insight into how an algorithm behaves in the presence of a concept drift. This behavior is most clearly represented by the recovery curves that are produced as a graphical output, see Figure 6.1. Yet, in some cases, it may also be desirable to have a more quantitative summary of the algorithms' recovery, similar to the use of metrics for performance evaluation [75], even if a quantification in terms of a scalar measure will necessarily come along with a certain loss of information. In this section, we propose concrete examples of these types of measures.

The *duration* measures the (relative) length of the recovery phase or, more specifically, the suboptimal performance of the algorithm. It is defined as

$$\frac{t_2 - t_1}{T} \in [0,1] \ ,$$

where t_1 is the time at which the curve S_C drops below 95% of the performance curve S_A, t_2 the time at which S_C recovers up to 95% of the performance of S_B and T the length of the entire stream.

The *maximum performance loss* measures the maximal drop in performance. In the case of the classification rate, it compares S_C with the pointwise minimum

$$S(t) = \min\{S_A(t), S_B(t)\}$$

as a baseline and derives the maximum relative performance loss

$$\max_{t \in T} \frac{S(t) - S_C(t)}{S(t)} \tag{6.8}$$

as compared to this baseline. In the case of regression, the measure is defined analogously as

$$\max_{t \in T} \frac{S_C(t) - S(t)}{S(t)} = \max_{t \in T} \frac{S_C(t) - \max\{S_A(t), S_B(t)\}}{\max\{S_A(t), S_B(t)\}} \quad .$$

6.3.4 Defining pure streams

The two previously constructed streams S_A and S_B have to be compatible in the sense of sharing a common data space $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. An important practical question is: where are these streams coming from? Ideally, two separate data sets of this kind are directly available. An example is the wine data from the UCI repository [107], with input attributes describing a wine in terms of physicochemical properties and the output, a quality level between 1 and 10. This data comes in two variants, one for red wine and one for white wine. Thus, using the first data set for S_A and the second one for S_B, one can simulate a transition from rating red wine to rating white wine.

If ideal data of that kind is not available, it needs to be produced in one way or another, preferably on the basis of a single source data stream S. In the following, we suggest a few possibilities for such a construction. However, we would not like to prescribe one specific way of data generation. Even in reality, there is not only one type of concept drift or a single source for a drift. Instead, concept drift can occur for many reasons, which one may attempt to mimic.

In the example of wine data, one can also imagine a single data set in which the type of wine (red or white) is added as a binary attribute. Conversely, instead of merging two data sets into a single one, one can split a data set S on the basis of a binary attribute X, using those examples with the first value for S_A and those with the second value for S_B (and removing X itself from both data sets). Again, the idea is to have a hidden variable that defines the context. Obviously, this approach can be generalized from binary to any type of attributes, simply by using appropriate splitting rules.

In some data sets S, the role of the attributes as either input (predictor) variable or output (response) variable is not predetermined. If the data contains two attributes A and B that are both of the same kind and can both be used as outputs, then S_A and S_B can be obtained, respectively, by using these attributes as a target for prediction.

There are many other ways of "manipulating" a data set S in order to simulate a concept change, such as changing the order of some of the input attributes. In later experiments, we also apply another simple idea: S_A is given by the original data S, while S_B is constructed by copying S and reversing or shifting the output attribute.

6.3.5 Further practical issues

Our discussion of recovery analysis so far has left open some important practical issues that need to be addressed when implementing the above experimental protocol. An obvious question is: how to determine the generalization performance of a model $\mathcal{M}_t$ (induced by the learner $\mathcal{A}$) at time t, which is needed to plot the performance curve? First of all, it is clear that this generalization performance can only be estimated on the basis of the data given, just like in the case of batch learning from static data. In Section 2.2, we show how an evolving classifier can be evaluated using one of the procedures: *(i)* the holdout approach and *(ii)* the test-then-train approach. The test-then-train procedure has clear advantages over the holdout approach. It makes better use of the data, because each example is used for both training and testing. More importantly, it avoids "gaps" in the learning process. In the holdout approach, $\mathcal{A}$ only learns on the training blocks but stops adaptation on the evaluation blocks in-between. Such gaps are especially undesirable in the presence of a concept drift, because they may bias the assessment of the learner's reaction to the drift. For this reason, we prefer the test-then-train procedure for our implementation of recovery analysis.

The second practical issue concerns the length of the data streams. To implement recovery analysis in a proper way, the streams should be long enough in order to ensure that the learner $\mathcal{A}$ will saturate on all streams. First, it should reach the saturation levels α^* and β^* on S_A and S_B, respectively. Moreover, the streams should not end while $\mathcal{A}$ is still recovering on S_C; otherwise one cannot decide whether or not an optimal recovery (reaching β^*) is accomplished.

Finally, to obtain smooth performance curves, we recommend repeating the same experiment with many random permutations S_A and S_B of the original streams and averaging the produced curves. In this case, averaging is legitimate because the results are produced for the same data generating processes (specified by the distributions $\mathbf{P}_A$, $\mathbf{P}_B$ and their mixture $\mathbf{P}_C$). This can be easily performed for the experiments on synthetic data. For a given synthetic data generator, we first generate a random model R_A and then fix it, i.e., R_A is the data generating process that produces the

pure stream S_A. Random permutations of S_A can be obtained by draining different sequences from the fixed model R_A, each time with a different seed.

6.4 A comparison of algorithms

The remainder of the chapter is devoted to a case study, in which we compare a number of different learning algorithms with respect to their ability to handle concept drift. A main goal of this study is to determine whether there are important differences between these methods and whether recovery analysis helps uncover them. In particular, we focus on the comparison between model-based and instance-based approaches to learning on data streams.

While a model-based approach focuses on inducing a model that fits the training data well, an instance-based approach focuses on the instances as local abstractions. In Section 3.2, we compare instance-based and model-based approaches in terms of their ability to learn and adapt in a streaming context.

IBL methods are inherently incremental [17]; their adaption is simply achieved by the addition or removal of examples. On the contrary to IBL methods, model-based approaches require a more difficult and careful adaptation strategy depending on the type of the induced models and their learned structures and parameters.

The most important aspect of adaptive learning is the adaptation of the learning process to any change in the learned concept in the course of time. Here, IBL approaches have the advantage of applying a simple adaptation strategy, which is basically reduced to the forgetting of old examples either globally or in some local regions after discovering a change. Model-based methods, however, might not have the ability to unlearn outdated examples, such as forgetting the influence of an example after learning a set of rules or after updating the weights in a neural network.

Table 6.1 provides a summary of the methods that we include in our study, as well as their main properties. Our selection was based on the following considerations:

- Because one of our main goals is to compare model-based and instance-based learning, we include both types of algorithms. Because the former clearly prevails in the literature, there are five model-based and only one instance-based approach.

- The algorithms should be representative and reflect the state-of-the-art. Many of the methods are therefore based on tree induction or rule learning.

	Learning problem			Model type			Approach		Change detection	
	Binary Classification	Multiclass Classification	Regression	Tree Structure	Rule-based	Instance-based	Fuzzy	Hoeffding Bound	Page-Hinkley	Stat. Hypot. Testing
eFPT	✓			✓			✓			✓
IBLStreams	✓	✓	✓			✓				✓
FLEXFIS			✓		✓		✓			
AdpHoef	✓	✓		✓				✓	✓	
AMRules			✓		✓			✓	✓	
FIMTDD			✓	✓				✓	✓	

Table 6.1: Summary of the learning algorithms and their main characteristics.

- The idea of adaptive learning in dynamical environments has not only received attention in machine learning but also in computational intelligence, where it is intensively studied under the notion of "evolving fuzzy systems" [9]. Therefore, we also include methods for the adaptive incremental learning of fuzzy systems on data streams.

The following is a brief description of each of the algorithms, for more technical details see Appendix A:

- Adaptive Hoeffding trees (AdpHoef) [22]: This method is an adaptive version of the Hoeffding Tree (an incremental decision tree approach). It mainly differs from the incremental version by maintaining drift detection indicators in each node in order to judge the compatibility of the current tree/subtree with the data, see Appendix A.1.

- Adaptive model rules (AMRules) [4]: This approach is a rule-based induction method for regression on data streams, see Appendix A.2.

- Fast incremental model trees with drift detection (FIMTDD) [86]: This tree-based approach induces model trees for regression on data streams, see Appendix A.3.

- Flexible fuzzy inference systems (FLEXFIS) [110]: This system learns TSK fuzzy rules [165] for modeling regression tasks on data streams, see Appendix A.4.
- Instance-based learner on data streams (IBLStreams) [151]: This method is our instance-based learner, which we introduce in Chapter 3.
- Evolving fuzzy pattern trees (eFPT) [157]: This method is an evolving variant of fuzzy pattern trees for binary classification problems, as introduced in Chapter 4.

Lastly, the availability of implementations is an important criterion as well. All approaches, except FLEXFIS, are implemented and executed under MOA [24], see also Appendix B. IBLStreams and eFPT can be downloaded as extensions of MOA, while the other methods can be acquired from the 2013.11 release of MOA.

Regarding the detection and the reaction to concept drifts, most of the used approaches apply drift detection techniques (see Section 2.4) which help the learner in becoming "drift-aware". Two main change detection methods are employed: the Page-Hinkley (PH) test [121] and the statistical hypothesis testing.

6.5 Experiments and results

All real data sets are collected from the UCI[2] repository [107] unless otherwise stated, as in the case of two data sets acquired from the DELVE[3] repository. Synthetically generated data, on the other hand, is produced using MOA's data stream generators, see Appendix B.1. Table 6.2 provides a summary of the data sets and their main properties (attributes, size, nature and source). Appendix D is devoted for the detailed description of the used real and synthetic data sets.

We conduct experiments with three different drift settings. The speed of change is varied by modifying the width parameter w in the sigmoid function (6.2). More specifically, we control the angle θ of the tangent of this function at $t = t_0$. Figure 6.2 depicts the three drift velocities:

- $\theta = \frac{\pi}{75}$ for a slow concept drift,
- $\theta = \frac{\pi}{30}$ for concept drift with a modest speed,
- $\theta = \frac{\pi}{2}$ for a sudden concept change (concept shift).

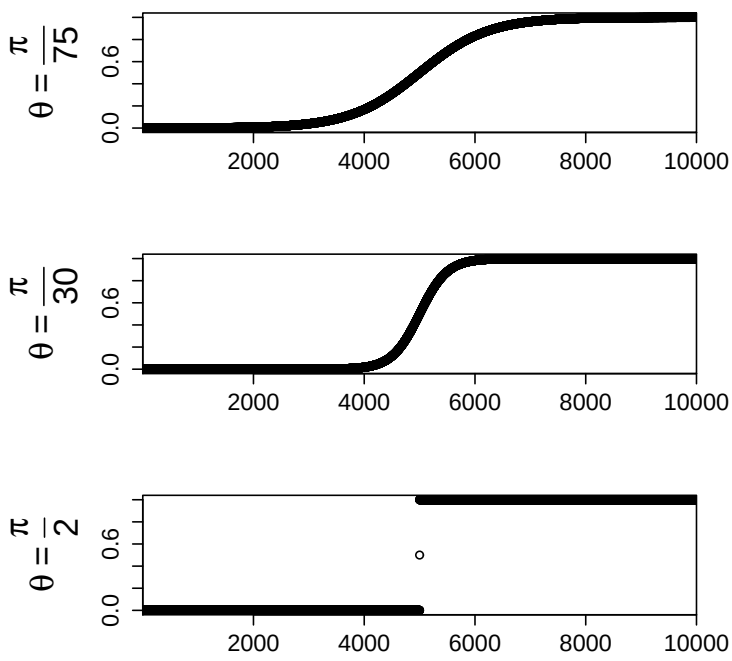


Figure 6.2: Sigmoid transition function modeling different types of concept drift: slow drift (top), moderate drift (middle), sudden drift (bottom).

Table 6.2 also contains the size of the evaluation window, which is always adapted to the size of the data set. Because the real data sets do not share the same size, we define the length of the window as $w = \frac{\ell}{100 \tan(\theta)}$, where ℓ is the size of the data set. Given the drift angle θ, the proportion of the entire stream that is subject to drift is the same for all data sets.

The results are generated by plotting the average evaluation, accuracy for classification and the root mean squared error for regression, on each data chunk; this is achieved through utilizing the test-then-train approach, which works in a sample by sample way. For generating a concept drift, MOA's drift simulation procedure is applied as illustrated in Appendix D.2.1.

6.5.1 Binary classification

Random trees This is a synthetic data set offered by MOA (see Appendix D.1.3), for which we use 4 numerical attributes to describe each instance. Different streams

[2] `http://lib.stat.cmu.edu/`, accessed on October 8, 2015

[3] `http://www.cs.utoronto.ca/~delve/data/datasets.html`, accessed on October 8, 2015

	Learning Problem			Properties				Source	
	Binary Cl.	Multiclass Cl.	Regression	#Attributes	#Instances	#Eval. Window	Target Attribute	Origin	**R**eal/**A**rtif./**S**im.
mushroom	✓			22	8,124	100	2-Class	UCI	R
breast	✓			9	699	25	2-Class	UCI	R
page blocks		✓		10	5,473	100	5-Class	UCI	R
letter		✓		16	20,000	200	26-Class	UCI	R
bank32h			✓	32	8,192	100	[0,0.819665]	DELVE	S
house8L			✓	8	22,784	200	[0,500001]	DELVE	R
random trees									
binary	✓			4	125,000	500	2-Class	MOA	A
5-classes		✓		4	125,000	500	2-Class	MOA	A
dis. hyper.									
distance			✓	4	125,000	500	[0.0388,1.7016]	MOA	A
cubed distance			✓	4	125,000	500	[0.0001,4.9271]	MOA	A

Table 6.2: Summary of the data sets used in the experiments.

of this data are produced using different random seeds. These streams define our pure streams S_A and S_B.

Figure 6.3 shows the recovery curves for different drift velocities. The different approaches reach different saturation levels α^* and β^* on the pure streams. Despite showing different "recovery patterns", they all manage to recover from level α^* to level β^*. For example, eFPT exhibits a rather smooth recovery with almost no drop in performance, whereas AdpHoef deteriorates quite significantly. Notably, all approaches seem to perform better than the estimated lower bound.

Mushroom This real data set represents a binary classification problem with the objective of predicting whether a given mushroom is edible or poisonous, see Appendix D.3.2. The original data is used as a first pure stream S_A and an "inverted copy" as a pure stream S_B; for the stream S_B, we simply invert the target attribute. Thus, the problem on the mixture stream S_C changes from predicting whether a mushroom is edible to predicting whether it is poisonous.

Figure 6.4 shows that both IBLStreams and AdpHoef recover quickly and very well to the optimal performance curve. Nonetheless, AdpHoef shows a drastic drop in performance during the drift. This could be explained by the cost for repairing the model: The original tree becomes invalid and needs to be transformed into a valid one through successive replacements of internal nodes or complete subtrees. For eFPT, on the other hand, it seems that the more drastic the change, the better the recovery. This may be due to the use of statistical tests for discovering changes: the more obvious the change, the easier it can be detected.

Breast cancer Wisconsin This is another real data set with the aim of classifying clinical reports as benign or malignant, see Appendix D.3.8. Like for the mushroom data, we produce a stream S_B by "inversion" of the original stream.

This problem appears to be quite difficult. Figure 6.5 depicts that only IBLStreams recovers well, but even this learner shows a significant drop in performance (below the estimated bound) during the drift. The tree-based methods eFPT and AdpHoef never manage to recover. Apparently, the data stream is too short to accomplish the complex process of tree reconstruction.

6.5.2 Multiclass classification

5-class random trees This data is used in the same way as for binary classification, but now with five classes; two versions of the stream are created using different random

seeds.

Figure 6.6 shows that both IBLStreams and AdpHoef recover well, although AdpHoef is a bit slower. It seems that the quicker the drift, the deeper the performance drop for IBLStreams. One explanation for this observation is that, in the case of a (detected) drift, IBLStreams removes an amount of data from the case base proportional to the error rate. However, this removal can harm its predictive performance during the drift, especially when the case base becomes almost empty.

Page blocks The task in this data set is to classify blocks in an image into one of five classes: text, horizontal line, picture, vertical line or graphic, see Appendix D.3.3. We simulate a drift by means of a cyclic shift of the class labels, replacing label i by $1 + i \bmod 5$.

Figure 6.7 shows that only IBLStreams recovers, whereas AdpHoef remains on a very low level of performance after the drift occurred.

Letter recognition The task in this data set is to recognize the 26 English letters, see Appendix D.3.4. We simulate a drift by shifting the class labels in a circular manner.

The result in Figure 6.8 appears to be qualitatively similar to the results on the 5-class random trees data. In particular, only IBLStreams is able to recover.

6.5.3 Regression

Distance to hyperplane data This is another synthetic data set that we produced by modifying the *HyperplaneGenerator* (for classification data) in MOA, see Appendix D.1.2. The output for an instance $\boldsymbol{x}$ is determined by the distance $y = f_1(\boldsymbol{x}) = |\boldsymbol{w}^\top \boldsymbol{x}|$ from the hyperplane (defined by the normal vector $\boldsymbol{w}$). Different output values can also be computed by this generator, such as the cubed distance $y = f_3(\boldsymbol{x}) = |\boldsymbol{w}^\top \boldsymbol{x}|^3$.

In a first experiment, we generated S_A using f_1 and S_B using f_3; thus, the drift is from the simpler to the more difficult problem. Figure 6.9 shows the recovery curves of the learning methods. Both IBLStreams and FLEXFIS have a relatively small error on the first problem, compared to AMRules and FIMTDD. During the drift, both suffer from a high drop in performance, which is visible as a bell-shaped peak. Nonetheless, IBLStreams recovers quite well, whereas FLEXFIS fails to do so. AMRuless and FIMTDD, on the other hand, show very similar performance curves, and both manage to recover in a comparable way.

In a second experiment, we change the order of the problems: S_A was generated using f_3 and S_B using f_1; thus, the drift is now accompanied with a change in the problem's difficulty, from the more difficult to the less difficult one. As shown in Figure 6.10, IBLStreams and FLEXFIS suffer from the same drop in performance. However, in this experiment they both succeed to recover. AMRules and FIMTDD share the same performance, with the exception that FIMTDD smoothly and perfectly recovers without any loss in the end.

Bank32h This is a simulation data from DELVE repository, see Appendix D.3.11. It represents how customers select and reject banks. The second pure stream S_B is created by "inverting" the target values: For an example $(\boldsymbol{x}, y)$, the original output y is replaced by $y_{max} + y_{min} - y$, where y_{min} and y_{max} are the smallest and largest target values in the data set.

Figure 6.11 shows that all methods manage to recover quite well on this data, despite a visible drop in performance in the middle of the drift region.

Census-house This data is also from the DELVE repository with the task of predicting the median price of houses in different regions, see Appendix D.3.12. The second stream S_B was created by inverting the original outputs in the same way as in the previous data set.

Again, Figure 6.12 shows that all approaches recover quite well, except for FIMTDD which has a comparatively long recovery phase.

6.5.4 Recovery measures

For the above experiments, the quantitative recovery measures are computed. Figures 6.13, 6.14 and 6.15 plot the duration against the maximum performance loss for the different learning algorithms on the different data sets. IBLStreams is often better than the other methods in terms of both the duration and the maximum performance loss measures, at least for classification problems.

6.5.5 Summary of the experiments

Although our experimental study is quite comprehensive, it is neither complete nor fully conclusive. The main goal of this study is not to "prove" the superiority of one method over another, but rather to illustrate the potential of our recovery analysis. Nevertheless, from the results we obtained, we can extract some trends and draw some preliminary conclusions, which are summarized in the following observations:

- For classification problems, eFPT, IBLStreams and Hoeffding trees recover quite well on the synthetic data sets. On the real data sets, eFPT achieves a partial recovery on the mushroom data set and does not manage to recover on the cancer data. This problem is caused by the slow discovery of the change till it is statistically significant after observing enough amounts of data. Similarly, Hoeffding trees recover less quickly and tend to require a larger amount of data. While they successfully recover on the large streams (see Figure 6.4), they do not completely recover on relatively short ones, see Figures 6.5, 6.7 and 6.8; it is also observed that the larger the number of classes in the problem, the more data the Hoeffding trees need to recover. This is exemplified by comparing the good recovery on the binary data (8K instances) in Figure 6.4 with the incomplete recovery on the 26 classes problem (20K instances) in Figure 6.8.

- For regression problems, FLEXFIS and IBLStreams are quite strong in terms of absolute accuracy, compared to the other methods (FIMTDD and AMRules). Nevertheless, they tend to have slightly higher peaks (maximum performance loss) in the area of drifts.

- In terms of recovery, IBLStreams appears to be the strongest method and it recovers well in all experiments.

- FLEXFIS tends to have difficulties with adapting to problems with increasing hardness: When the second stream is more complex than the first one, it often fails to recover, see Figure 6.9.

- Overall, FIMTDD and AMRules perform quite similarly, regardless of the problem and the type of drift (slow or sudden). This is expected in that both methods are quite comparable in terms of their model structure (trees and rules are closely related). Moreover, both are using the Hoeffding bound for model adaptation and PH for drift detection, see Figures 6.9–6.12.

- Notably, FIMTDD recovers especially smoothly and with almost no drop in performance when drifting from a difficult concept to a simpler one, see Figure 6.11.

6.6 Conclusion

We have introduced *recovery analysis* as a new type of experimental analysis in the context of learning from data streams. The goal of recovery analysis is to provide an

idea of a learner's ability to discover a concept drift quickly and to take appropriate actions to maintain the quality and generalization performance of the model.

To demonstrate the usefulness of this type of analysis, we have presented an experimental study, in which we analyzed different types of learning methods on classification as well as regression problems.

Our results clearly reveal some qualitative differences in how these methods react to concept drift, how much they are affected and how well they recover their original performance. The results affirm some important factors that seem to be responsible for these differences, such as the number of classes (in classification problems) and whether the drift is from a simpler to a more difficult problem or the other way around.

Overall, our results also provide evidence in favor of our conjecture that instance-based approaches to learning on data streams are not only competitive to model-based approaches in terms of performance, but also advantageous with regard to the handling of concept drift. This is arguably due to their "lightweight" structure; removing some outdated examples is more simple than completely reconstructing a possibly complex model.

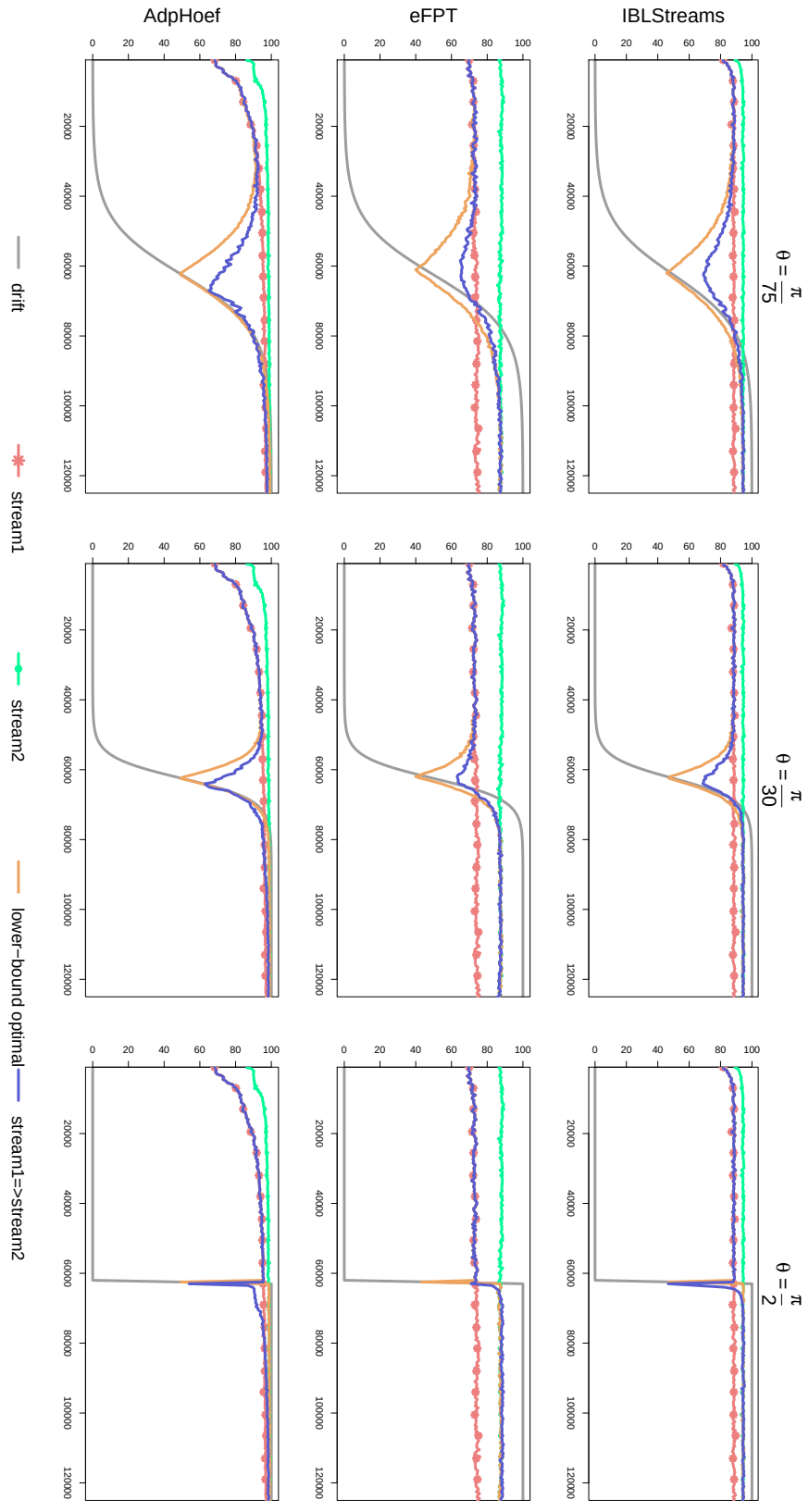


Figure 6.3: Performance curves (accuracy) on the random trees data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

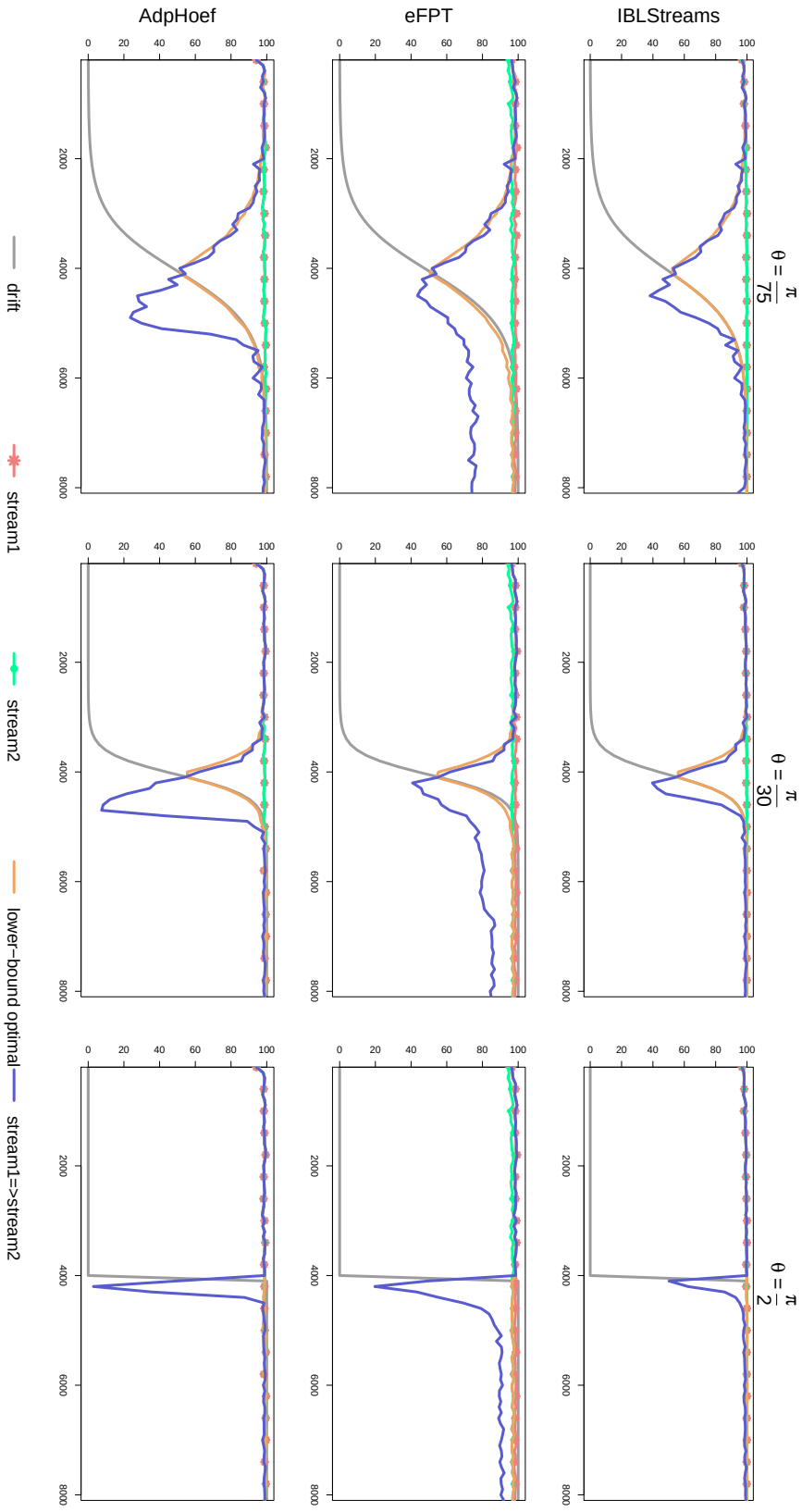


Figure 6.4: Performance curves (accuracy) on the mushroom data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

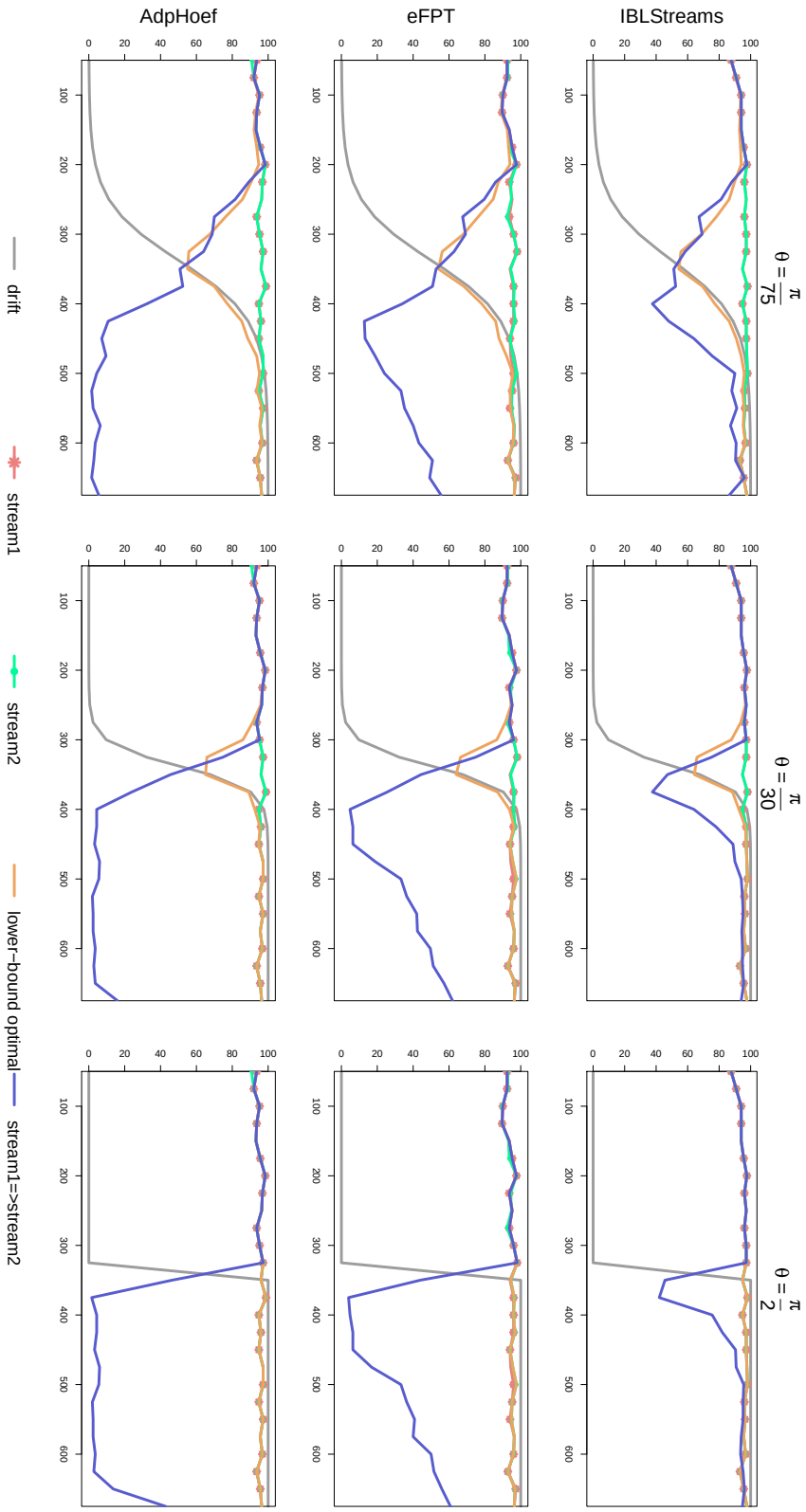


Figure 6.5: Performance curves (accuracy) on the breast cancer Wisconsin data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

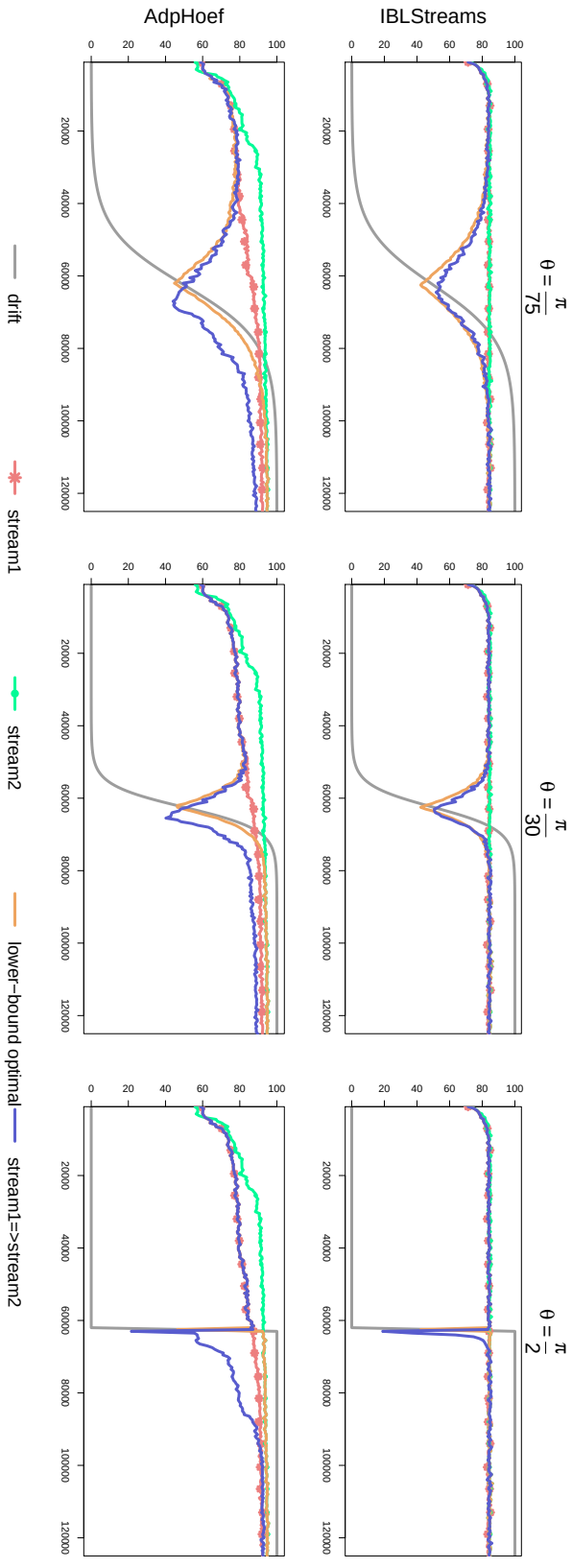


Figure 6.6: Performance curves (accuracy) on the random trees 5-classes data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

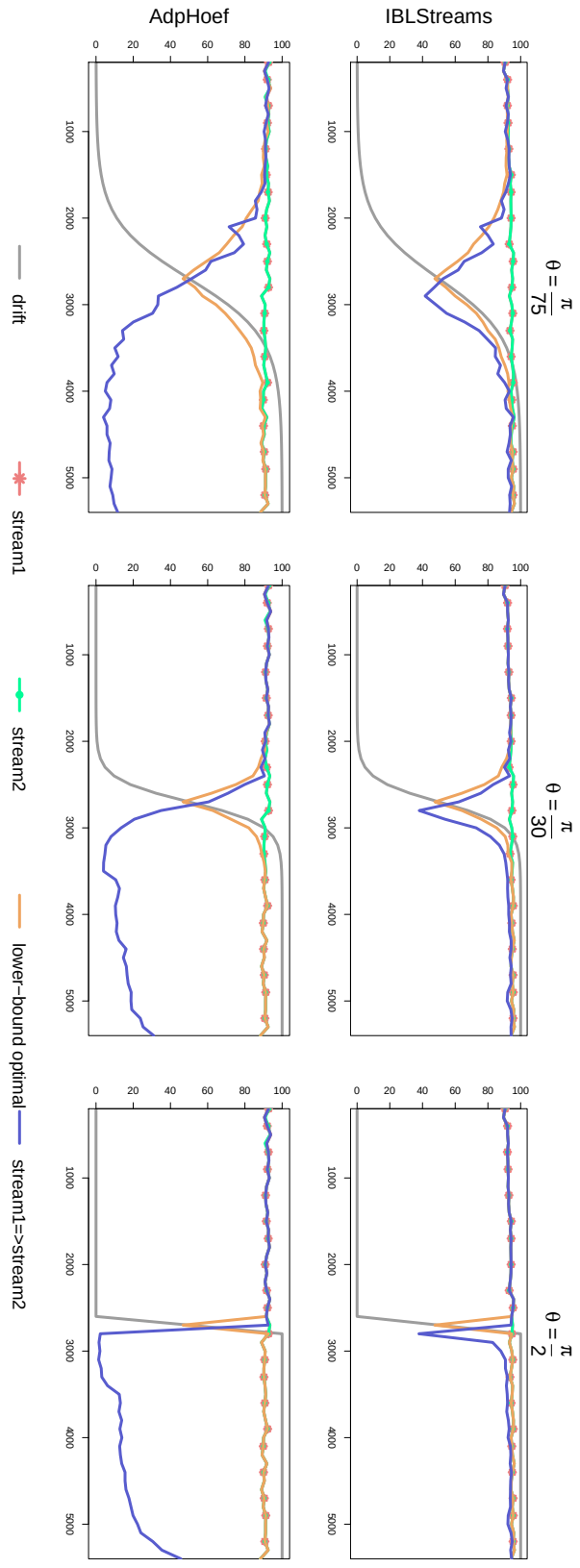


Figure 6.7: Performance curves (accuracy) on the page blocks data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

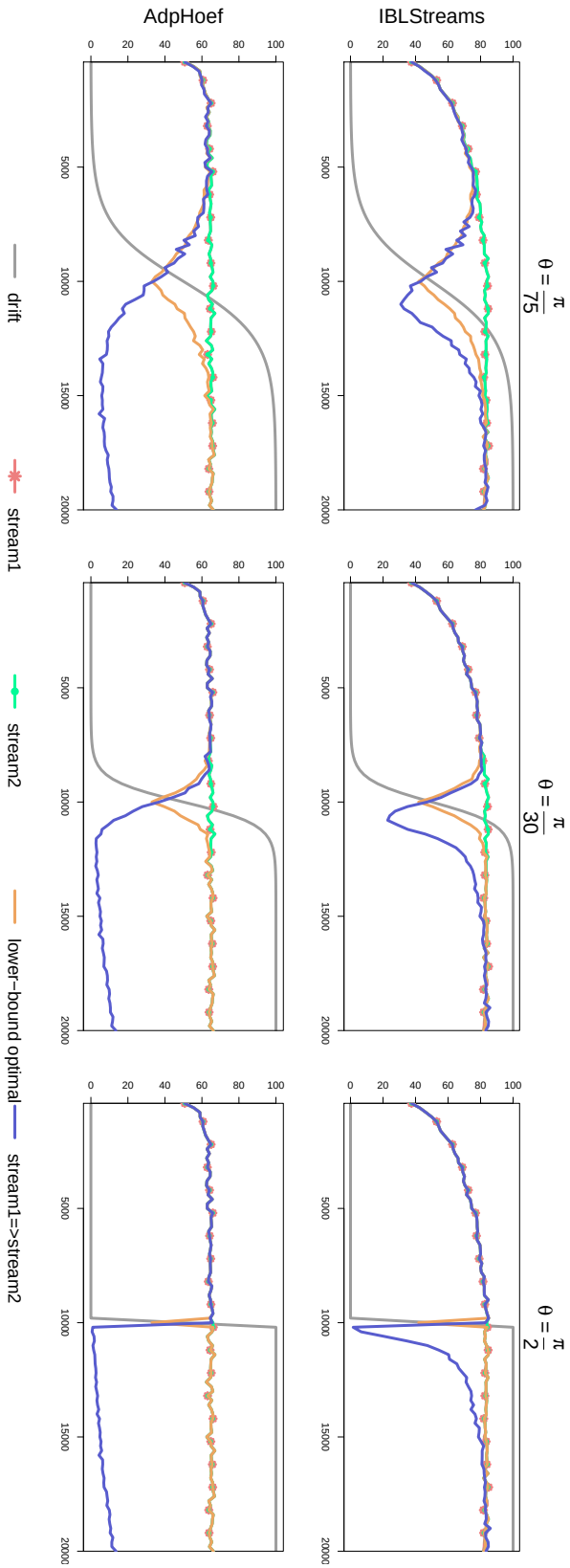


Figure 6.8: Performance curves (accuracy) on the letter recognition data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

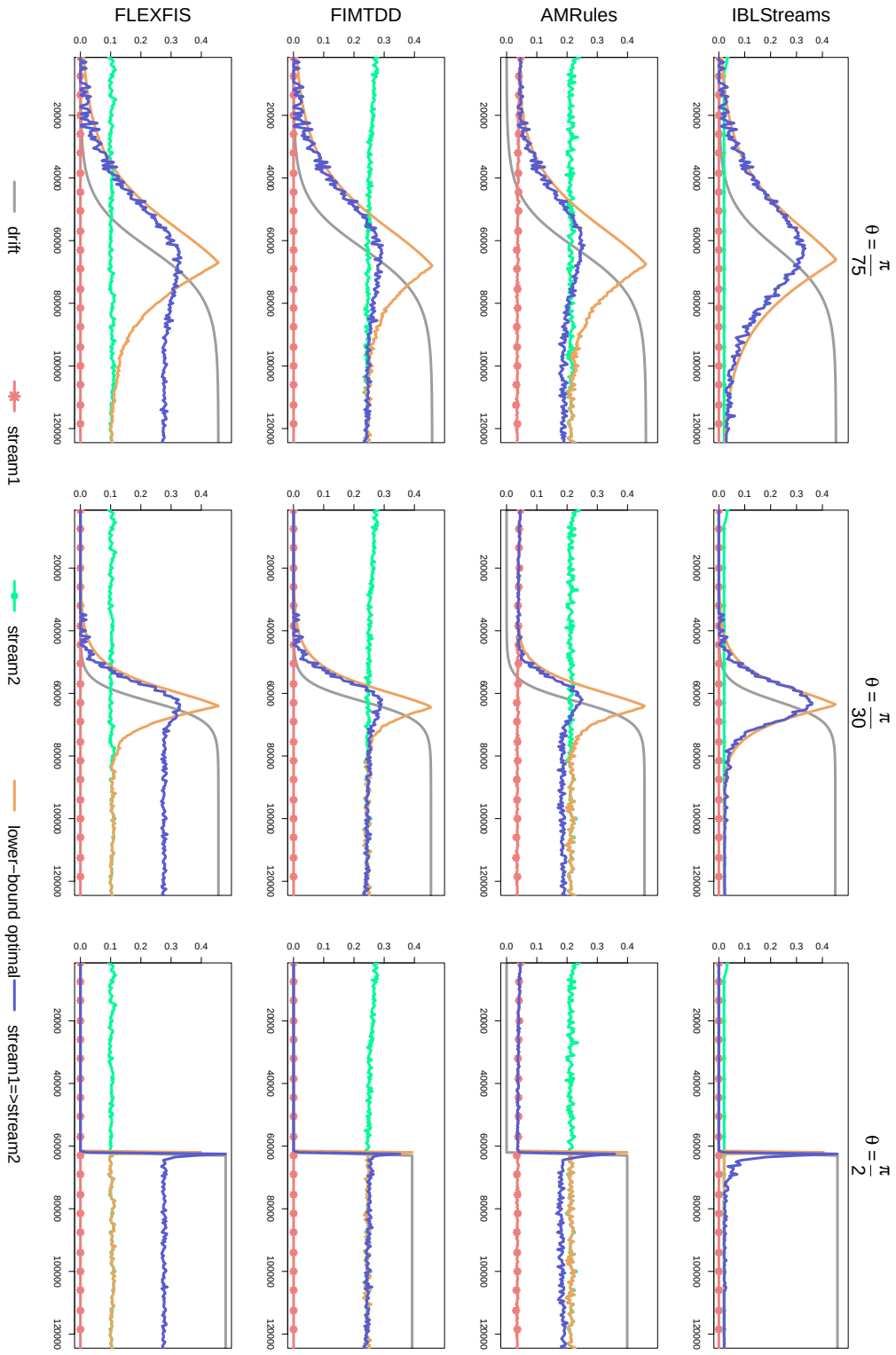


Figure 6.9: Performance curves (RMSE) on the distance to hyperplane data, with a drift from f_1 to f_3. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

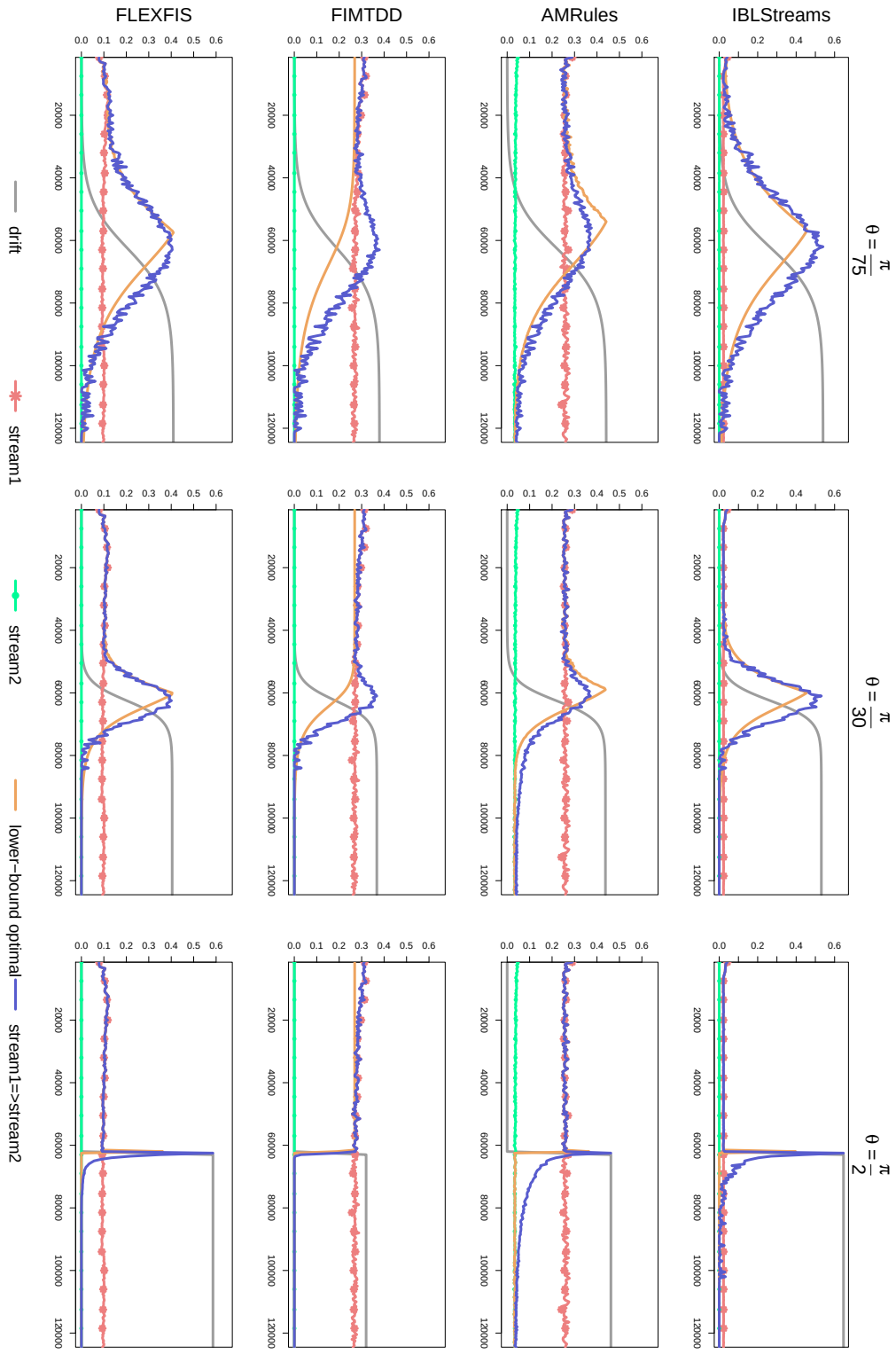


Figure 6.10: Performance curves (RMSE) on the distance to hyperplane data, with a drift from f_3 to f_1. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

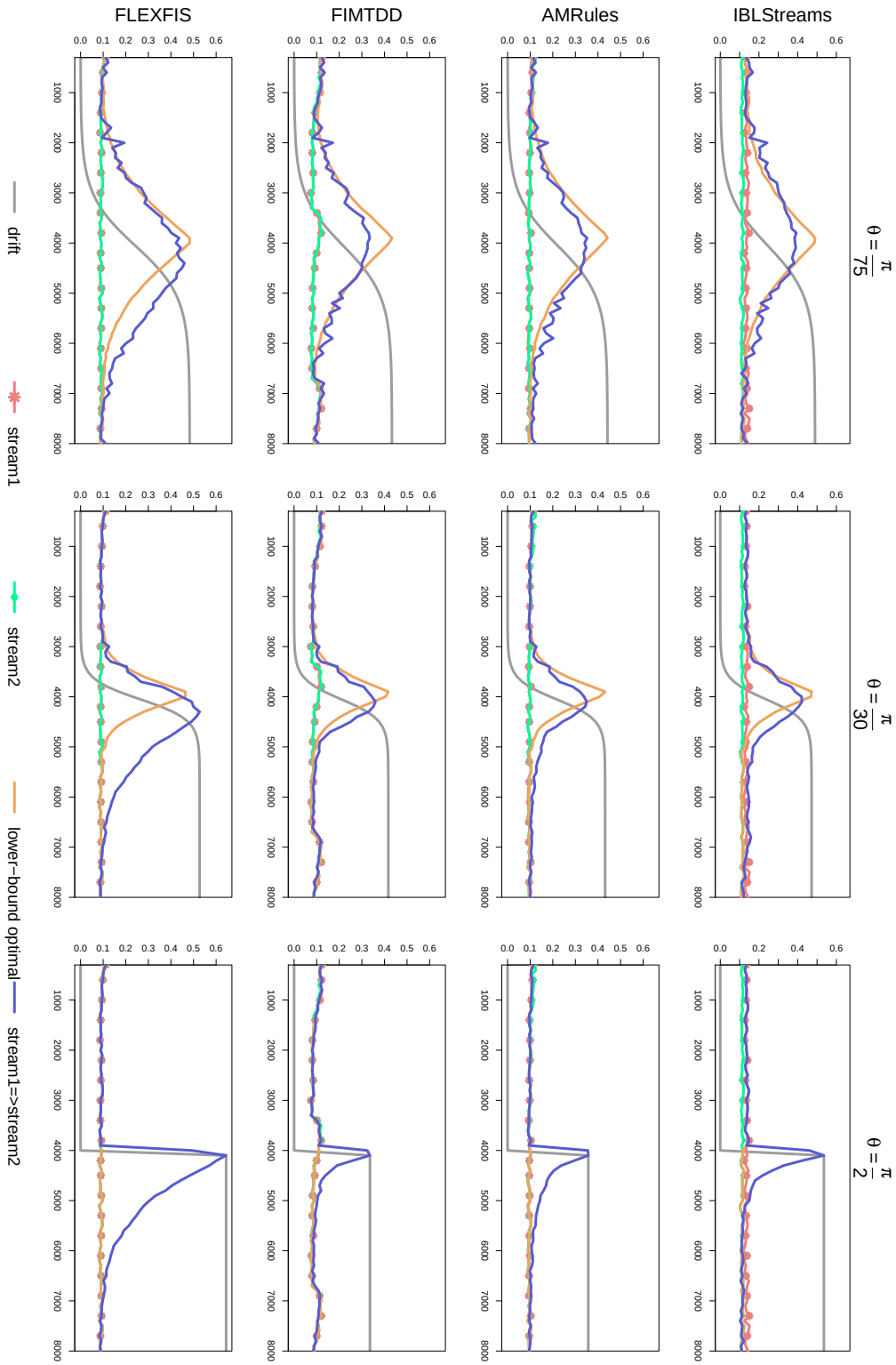


Figure 6.11: Performance curves (RMSE) on the bank32h data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

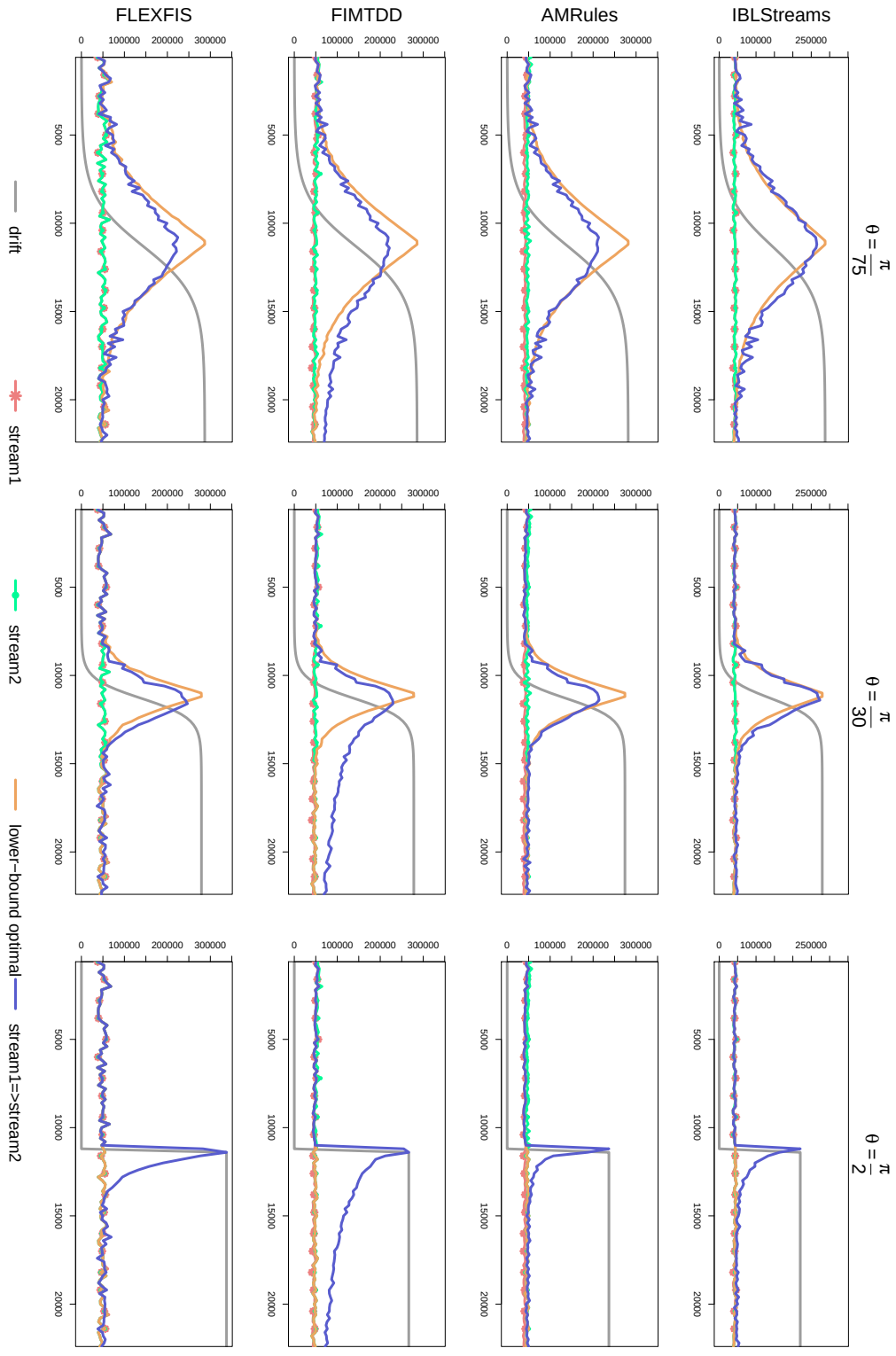


Figure 6.12: Performance curves (RMSE) on the house8L data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

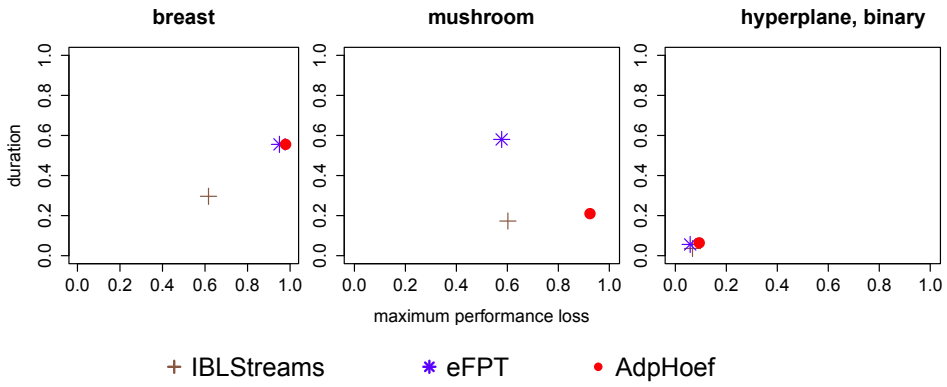


Figure 6.13: Duration versus maximum performance loss of different methods on the binary classification problems.

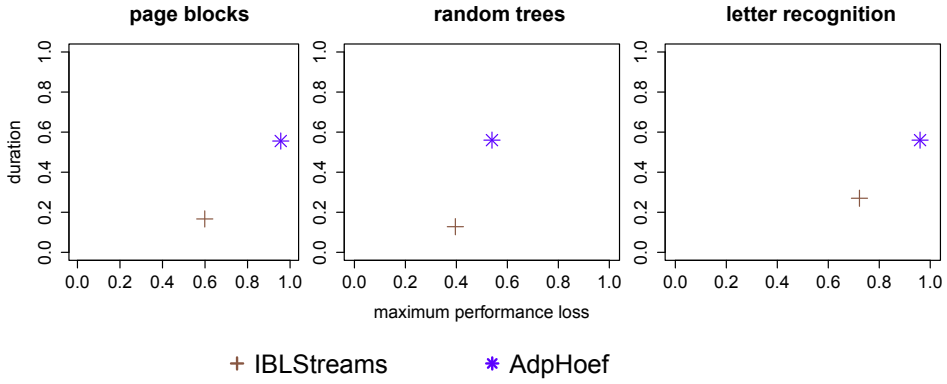


Figure 6.14: Duration versus maximum performance loss of different methods on the multiclass classification problems.

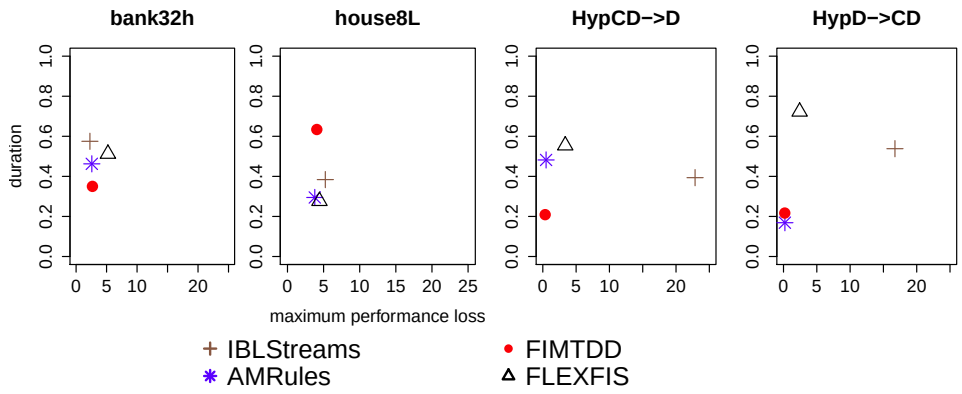


Figure 6.15: Duration versus maximum performance loss of different methods on the regression problems.

Chapter 7

Conclusion

In this thesis, we developed novel adaptive approaches for learning on non-stationary data streams. These adaptive approaches do not only generalize the observed data through the extracted abstractions, but also enforce the durable validity of the obtained knowledge with the recently observed data. With focus on supervised learning methods, two adaptive learners are introduced and evaluated: *(i)* an instance-based learning approach for regression and classification and *(ii)* a model-based approach which utilizes the fuzzy concepts in representing the data and propagating information in a tree-like structure called evolving fuzzy pattern trees. Moreover, a new type of performance comparison for adaptive learners is introduced in order to asses the learner's ability to learn in the presence of concept change and the ability to recover after the change; this comparison method is referred to as recovery analysis. We also tackled a special type of streams, namely streams of events, for which we develop an adaptive survival analysis method.

7.1 Original Contributions

The contribution of this thesis is manifold:

- Instance-based learners have the advantage of investing minor effort during the training time, with the focus on collecting observations instead of abstracting knowledge in the form of induced models. In Chapter 3, IBLStreams, an IBL approach that considers all relevance criteria as suggested by Beringer and Hüllermeier [17], is introduced and developed. This approach is competitive in terms of the generalization performance and the recovery of its performance after a concept change by repairing its case base. This repairment comes down to

the removal of a fraction of the case base. IBLStreams, compared to other supervised IBL approaches, shows superiority in many experiments. IBLStreams does not only have a competitive performance when compared with the state-of-the-art classification and regression methods on synthetic and real data streams, but also shows a relatively smooth recovery to its supposed performance on newly emerging concepts.

- Evolving fuzzy pattern trees is introduced as an extension to the fuzzy pattern tree induction methods [82, 146]. eFPT exhibits the interpretability and transparency properties enjoyed by fuzzy systems; it actually offers a tradeoff between compact interpretable models and strong generalization performance. The experiments show that eFPT is competitive in terms of accuracy and compactness of the induced models; the experimental evaluation in the recovery study also shows that eFPT recovers well, after a concept change, with a smooth recovery pattern, most of the time, compared to the Hoeffding tree.

- Recovery analysis, as a new type of analysis for adaptive models, is established in Chapter 6. The aim of this experimental protocol is to assess the ability of a learner to discover and to adapt to a concept change in the observed stream. Recovery analysis, supported with recovery measures, is introduced for two types of supervised learning problems: classification and regression. The conducted study shows that recovery analysis can help in understanding how the various learning methods detect, adapt and recover after the occurrence of a concept change.

- Event data are often produced, temporally, from many data sources; the consecutive temporal generation of events makes them naturally fit the stream model. In Chapter 5, an adaptive survival analysis for learning from streams of events is introduced. To the best of our knowledge, our approach is the first to consider survival analysis in the streaming setting; this approach adapts a variant of the Cox proportional hazard model using the sliding window technique. Under the assumption of a constant hazard rate (with time), the likelihood function, which we prove to be concave, is maximized to find the model's parameters. Moreover, the assumption of a constant hazard rate has led to an exponential hazard model which exhibits both properties: the proportional hazard and the accelerated failure time. The proposed model is evaluated, as a first proof of concept, on two real event streams. The obtained results are plausible and agree with

our expectation. The introduced approach also helped in promoting adaptive survival analysis in [102] as one of the open challenges on data streams.

7.2 Future Research

- Although IBLStreams has achieved satisfying results, in terms of performance and recovery, IBL methods still have the potential for tremendous improvements. One may think of constructing a hybrid approach in which the case base does not only contain instances but also induced rules; such an approach has the potential to improve IBL methods for two reasons: (i) rules can better summarize well occupied regions of the instance space, thus removing the need to keep numerous redundant examples from the same region and (ii) rules can be used as a temporal substitute for the case base when drifts occur and a large portion of examples is removed from the case base.

- Evolving fuzzy pattern trees, proposed in this thesis, focus only on binary classification problems; this calls for an extension that learns from streams of regression or multiclass classification problems. Motivated by the modification on the pattern tree's induction method proposed by Senge and Hüllermeier [147], we plan to induce an ensemble of trees, a forest, in parallel using techniques similar to the Hoeffding race [115], besides trying to enforce the diversity of the induced trees. In such a way, drifts can be handled through the removal and addition of trees in an adaptive way; trees in this ensemble remain interpretable when inspected individually.

- The proposed recovery analysis shows the potential toward discovering the hidden resemblance between different learning methods based on their recovery pattern; such a similarity is discovered when comparing AMRules with FIMTDD, which can only be explained by the equivalence between the tree, induced by FIMTDD, and the rules, induced by AMRules. This equivalence becomes more obvious when knowing that both methods use Hoeffding's bound and the Page-Hinkley test in the same way during the induction. We recommend a further application of the recovery analysis on more learning paradigms and methods in order to discover the points of strength and weakness in the ability of each approach to adapt and recover. Recovery analysis also shows the potential to

be extended for unsupervised learning methods in general and clustering in particular in order to evaluate the capability of a clustering method to recover after a change in the data generating distribution.

- The proposed survival analysis approach assumes a fixed set of parallel event streams, i.e., events are emitted from a fixed set of objects; this restriction causes the streams of events to contain recurrent events only.

  One may consider the case where the stream emitting objects are allowed to be removed, after an event, or to be censored, after being lost. In such a setting, the risk set becomes changing with time and events are not any more restricted to be recurrent. This requires a new formulation of the likelihood function in order to allow the risk set to be dynamically changing.

  Another extension of survival analysis on data streams may consider the characteristic properties of the events, e.g., the magnitude of an earthquake; utilizing these properties in survival analysis may have a positive effect on finding the prognostic factor.

Appendix A

Methods

In this thesis, we compare different adaptive learning algorithms for different learning tasks; each of these methods belongs to a paradigm that exhibits unique learning and recovery patterns, see Table A.1 for a brief summary of the studied methods. All used approaches, except FLEXFIS which is implemented using the fuzzy logic toolbox provided in Matlab, are implemented and offered by the MOA framework, which is described in Appendix B. IBLStreams and eFPT can be downloaded as extensions for the framework, whereas the rest of the methods can be acquired from the *2013.11* release of MOA.

A.1 Adaptive Hoeffding Tree

The Hoeffding tree [56] is an incremental decision tree approach, tailored for classification on data streams. Upon the arrival of a new training example, the algorithm examines each inner node of the tree and decides whether the current split (attribute) is still optimal, or whether an alternative split appears to be advantageous. The decision, made while choosing the optimal splitting attribute, is based on statistical hypothesis testing. More specifically, Hoeffding's inequality [81] is used to check whether the information gain of an alternative attribute is significantly higher than the gain of the currently chosen attribute.

Hoeffding's inequality states that, with probability $1 - \delta$, the difference between the observed mean and the true mean, for a random variable r of the range R, would not exceed ϵ after seeing n observations, such that

$$\epsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}} \quad . \tag{A.1}$$

The Hoeffding tree uses this bound to compare the difference between the information gains $\bar{G}(X_a)$ and $\bar{G}(X_b)$ of the two best splitting attributes X_a and X_b, respectively. Assume that the attribute X_a is better than X_b with $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b) > 0$. On the arrival of new data samples, Hoeffding's bound guarantees that the true difference (in information gain) ΔG and the empirical difference $\Delta\bar{G}$ satisfy the inequality $\Delta G > \Delta\bar{G} - \epsilon$, with probability $1 - \delta$. Observing $\Delta\bar{G} > \epsilon$ at any time means that $\Delta G > 0$, thus selecting the attribute X_a is now guaranteed to cause the largest information gain.

An adaptive version of the Hoeffding tree (AdpHoef) has been presented in [23]. This algorithm maintains a drift detection statistic in each node to judge the compatibility of the current tree or subtree with the recently received data. For each of these nodes, an alternative tree is maintained and learned on the recent data only; this alternative subtree replaces the initial subtree, rooted at that node, whenever the node's drift detector signals a change. This variant of Hoeffding trees uses the ADWIN [22] technique, a parameter-free method for detecting the rate of change in data streams. In this thesis, AdpHoef algorithm is applied for binary and multiclass classification problems.

A.2 Adaptive Model Rules

Adaptive model rules (AMRules) [4] is an induction method for rules on regression data streams. Each rule is specified by a conjunction of literals on the input attributes in the premise part and a linear function, which minimizes the root mean squared error, in the consequent part. Adaptive statistical measures are maintained in each rule in order to describe the instance subspace covered by that rule. Moreover, the performance of each rule is monitored by Page-Hinkley (PH) test [121], such that a rule is pruned as soon as its error significantly increases due to a concept change.

Each rule is initialized with a single literal and successively expanded with new literals. The best literal to be added, if any, is chosen on the basis of Hoeffding's bound, in a manner that is similar to the expansion of a Hoeffding tree. The linear function in the consequent part is learned online by applying the stochastic gradient descent method; to this end, the delta rule

$$w_i = w_i + \eta(y - \hat{y})x_i \tag{A.2}$$

updates the weight w_i associated with the ith attribute, where η is a small positive learning rate and $(y - \hat{y})$ is the committed prediction error.

A.3 Fast Incremental Model Trees with Drift Detection

The fast incremental model trees with drift detection (FIMTDD) [86] is a tree-based approach for inducing model trees for regression on data streams. It combines properties of Hoeffding trees and AMRules. Similar to Hoeffding trees, it uses Hoeffding's bound (A.1) to choose the best splitting attribute. Since FIMTDD tackles regression problems, attributes are evaluated in terms of the achieved reduction in standard deviation of the target attribute at the new subtree.

Each leaf node of the induced tree contains a linear function, which is learned in the subspace covering the instances that fall into that leaf node; this function is learned using stochastic gradient descent. Similar to AMRules, FIMTDD employs Page-Hinckley (PH) test for change detection at each internal and leaf node. A significant increase in the error indicates a concept change, which triggers the replacement of a subtree by an alternative subtree, learned from the recently observed data.

A.4 FLEXible Fuzzy Inference Systems

The flexible fuzzy inference systems (FLEXFIS) [110] learns the so-called Takagi-Sugeno-Kang (TSK) fuzzy system [165]. A fuzzy TSK system contains a set of rules, each of which has the form:

$$\text{Rule}_i \ : \text{IF} \quad (x_1 \text{ IS } \mu_{i1}) \ \text{AND} \ ... \ \text{AND} \ (x_p \text{ IS } \mu_{ip}) \tag{A.3}$$

$$\text{THEN} \quad l_i(\boldsymbol{x}) = w_{i0} + w_{i1}x_1 + w_{i2}x_2 + ... + w_{ip}x_p \ , \tag{A.4}$$

where $(x_1, \ldots, x_p)^\top$ is the vector representation of the instance $\boldsymbol{x}$ and μ_{ij} is a fuzzy set characterizing the jth component of the ith rule's premise.

The premise part of $\texttt{Rule}_i$ determines the fuzzy membership degree at which the instance $\boldsymbol{x}$ belongs to the rule $\texttt{Rule}_i$. By modeling the AND operator as a t-norm [96], each premise propagates a membership degree by applying the t-norm operator on the p-dimensional fuzzy membership degrees vector. The conclusion part is a linear function $l_i(\boldsymbol{x})$ of the input vector's components.

The final prediction of a TSK system, with C rules, is produced by the weighted average of the outputs of the single rules. To this end, each output is weighted by the normalized fuzzy memberships produced by the rules' antecedents. Consequently,

the prediction for the instance $\boldsymbol{x}$ is

$$\hat{y} = \sum_{i=1}^{C} \Psi_i(\boldsymbol{x}) \cdot l_i(\boldsymbol{x}) \ , \tag{A.5}$$

s.t. $\mu_i(\boldsymbol{x})$ is the activation degree of the ith rule and $\Psi_i(x)$ is the normalized activation degree

$$\Psi_i(\boldsymbol{x}) = \frac{\mu_i(\boldsymbol{x})}{\sum_{j=1}^{C} \mu_j(\boldsymbol{x})} \ . \tag{A.6}$$

In summary, a TSK fuzzy system models the mapping from the p-dimensional input space to the output space, through the application of both components *(i)* the fuzzy sets μ_{ij} and *(ii)* the weight vector $\boldsymbol{w_i} = (w_{i0}, w_{i1}, \ldots w_{ip})^\top$.

FLEXFIS allows the learning and the adaption of these components incrementally and makes use of online clustering techniques (that allow for the dynamic creation, merge and removal of clusters) in order to specify each rule's antecedents. FLEXFIS starts by clustering the incoming stream of examples using an incremental vector quantization (VQ) method. In this way, a distance-based clustering is performed, such that a new instance is added to the nearest cluster only if its distance is smaller than a predefined threshold; otherwise a new cluster is created for the new instance. Whenever an example is added to the nearest cluster, the cluster's center and statistics are updated. The resulting clusters are projected onto all input dimensions and the result of the projection defines the centers and the width of the Gaussian fuzzy sets; these fuzzy sets form the literals of the TKS rule's premise. The linear function in the consequent parts is learned by applying the recursive weighted least squares estimation (RWLS) [108].

FLEXFIS employs both passive and active adaptation strategies. The passive adaptation strategy occurs when the cluster statistics are continuously adapted and when the linear function's coefficients are learned in an online manner. The active adaptation is performed by checking the homogeneity of the induced clusters, such that two similar clusters can be merged and a large cluster can be split into two clusters if it potentially covers more than one concept. FLEXFIS is implemented in Matlab and offers a function for finding the optimal values of the different parameters and thresholds.

	Learning Problem			Model			Approach		Change Detection	
	Binary Classification	Multiclass Classification	Regression	Tree Structure	Rule-based	Instance-based	Fuzzy	Hoeffding Bound	Page-Hinkley	Stat. Hypot. Testing
eFPT	✓			✓			✓			✓
IBLStreams	✓	✓	✓			✓				✓
FLEXFIS			✓		✓		✓			
AdpHoef	✓	✓		✓				✓	✓	
AMRules			✓		✓			✓	✓	
FIMTDD			✓	✓				✓	✓	

Table A.1: Summary of the used learning algorithms; with their computational and structural properties.

Appendix B

MOA

A few frameworks and software systems for mining data streams have been released in recent years, including VFML[1] [84] and the MOA[2] (Massive Online Analysis) [24] framework. VFML is a toolkit for mining high-speed data streams and very large data sets. MOA is an open source software for mining and analyzing large data sets in a stream-like manner; it is implemented in Java and is closely related to WEKA[3] [180], the Waikato Environment for Knowledge Analysis, which is currently the most commonly used machine learning software. MOA contains a large collection of machine learning algorithms for classification, regression, clustering, outlier and concept drift detection. For supervised learning, it supports the development of classifiers that can learn either in a purely incremental mode, or in batch mode first (on an initial part of a data stream) and incrementally afterward. The development of an evolving classifier can be achieved by implementing a Java interface called *Classifier*. This operation simulates the case of online learning, which implies that each instance is accessed only once for learning and then discarded. A few incremental classifiers are already included in MOA, notably the Hoeffding tree [85] and the Adaptive Hoeffding Trees [23], a state-of-the-art classifier often applied as a baseline in experimental studies. Some meta learning techniques are implemented, too, such as online bagging and boosting both for static [124] and evolving streams [27].

MAO allows and supports other researches to add their approaches through what is called MOA extensions[4]. Our instance-based learner IBLStreams is also publicly available as a MOA extension.

[1] `http://www.cs.washington.edu/dm/vfml/`, accessed on July 13, 2015
[2] `http://moa.cms.waikato.ac.nz`, accessed on November 18, 2015
[3] `http://www.cs.waikato.ac.nz/ml/weka`, accessed on May 23, 2015
[4] `http://moa.cms.waikato.ac.nz/moa-extensions`, accessed on May 23, 2015

B.1 Stream Generators

MOA supports the incremental learning from data streams, which is maintained as a *Stream* object. This stream can either be attached to a real-world data set and serialized as a stream, or synthetically generated in an online manner. MOA supports the simulation of data streams by means of synthetic stream generators. An example is the hyperplane generator that was originally used in [85]. It generates data for a binary classification problem, taking a random hyperplane in d-dimensional Euclidean space as a decision boundary. Another important stream generator is the random trees generator whose underlying model is a decision tree for a desired number of attributes and classes.

Besides offering synthetic data generators, MOA offers the ability to simulate a concept drift through instantiating the class *ConceptDriftStream*. Appendix D elaborates more on the synthetic data stream generators offered by MOA; it also explains how a concept drift or a sampling drift can be simulated.

B.2 Online Evaluation

The evaluation of an evolving classifier learning from a data stream is clearly a non-trivial issue. Compared to standard batch learning, single-valued performance cannot represent the properties of the learned model in a non-stationary environment. MOA offers different solutions for this problem.

Holdout Evaluation

The holdout procedure is a generalization of the cross-validation procedure commonly used in batch learning. The training and testing phase of a classifier are interleaved as follows: the classifier is trained incrementally on a block of M instances and then evaluated (but no longer adapted) on the next N instances, then again trained on the next M and tested on the subsequent N instances, and so forth.

Test-then-train Evaluation

While the holdout procedure uses an instance either for training or for testing, each instance is used for both in the test-then-train approach: First, the model is evaluated on the instance and then a single incremental learning step is carried out. The advantage of applying this evaluation scenario is that all instances are utilized for both testing and training, without any loss of information present in the holdout

instances such as in the previous evaluation method. A holdout block does not only hide vital information, that could help in updating the trained model, but also causes a delay of any detecting of an occurring concept change during the holdout phase.

Appendix C

M-Tree

M-tree [38, 39] is an index structure that supports the storage and the retrieval of data objects based on their similarities. This is accomplished using a metric distance d in the metric space of the data objects. M-tree is a tree-like structure with internal nodes that are hyperspheres in the metric space; the hypersphere of an internal node n has the radius r_n and is centered by the data object o_n, whose distance to the parent node's center is known and maintained.

An M-tree is built in an cumulative way through the successive insertion of new data objects in a top-down manner. In order to insert a new data object o_n, the M-tree finds the node u, with the center o_u and the radius r_u, which satisfies $d(o_u, o_n) \leq r(r_u)$.

For a given query object o_q, M-tree supports two search strategies *(i)* range queries and *(ii)* k-NN queries. By applying the triangular inequality, multiple paths can be pruned during the search for objects that resemble the query o_q.

Although the previous works of IBLStreams (see Chapter 3) and IBL-DS [17] were utilizing the Query processing library XXL [52], IBLStreams is shifted in this thesis to a simpler implementation of M-tree[1] that is hosted in the web-based Git repository hosting service GitHub.

IBLStreams approach utilizes an M-Tree index structure for indexing inserted samples into the case base. The employed metric distance is the SVDM, which is introduced in the next subsection.

C.1 Distance Function

The key assumption behind the nearest neighbor principle lies in the conjecture that similar objects tend to belong to the same class; this assumption leads to results that highly depend on the employed similarity or distance metric.

[1] `https://github.com/erdavila/M-Tree`, accessed on July 13, 2015

We utilize a modified incremental version of the *simple value difference metric* (SVDM) [54, 55] to construct the M-tree index, as suggested in [17]. SVDM is a simplified version of the VDM distance measure [163]. The similarity between two vectors $\boldsymbol{x} = (x_1, \cdots, x_d, C_x)^\top, \boldsymbol{y} = (y_1, \cdots, y_d, C_y)^\top \in \mathcal{D}^d \times \mathcal{C}$ depends on the similarities between the vectors' components, where C is the output space and D_i is the input space for the ith feature which can be either numeric or nominal.

The distance δ_i for a numerical feature is normalized by the support of that feature:

$$\delta_i(x_i, y_i) = \left| \frac{x_i - y_i}{\max_{D_i} - \min_{D_i}} \right|,$$

with $\min_{D_i}$ and $\max_{D_i}$ are the minimum and maximum observed values for the ith feature. In this way, features of large support are not over-weighted when calculating the distance. For nominal features, SVDM suggests a similarity based the conditional probability of the target feature given the nominal feature:

$$\delta_i(x_i, y_i) = \left| \sum_{l=1}^{c} P(C_l|x_i) - P(C_l|y_i) \right|.$$

The distance between the two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by aggregating the featurewise distances on all features:

$$d(\boldsymbol{x}, \boldsymbol{y}) = SVDM(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{d} \sum_{i=1}^{d} \delta_i(x_i, y_i)^2.$$

Appendix D

Data Sets

This appendix presents the data sets used for all empirical evaluations throughout the thesis; data sets vary in different aspects:

- Type: (i) synthetic and (ii) real-world data.
- Learning task: (i) binary classification, (ii) multiclass classification and (iii) regression.
- Nature: (i) stream of examples versus (ii) stream of events.

Table D.3 gives an overview of the used data sets and summarizes their attributes, size and origin.

D.1 Synthetic Data Sets

The usage of synthetic data offers the flexibility in designing guarded experiments, with the aim of evaluating the model's performance in a particular environment under particular circumstances. Above all, synthetic data is useful for simulating a concept drift.

D.1.1 Hyperplane data

Learning task: Binary classification.
An example of the hyperplane data generator was originally used in [85]. It generates a binary classification data by taking a random hyperplane in a d-dimensional Euclidean space as a decision boundary.

The output for an instance $\boldsymbol{x} \in \mathcal{R}^d$ is determined by the sign of $\boldsymbol{w}^\top \boldsymbol{x}$, where $\boldsymbol{w}$ is the normal vector of the hyperplane. In other words, the problem is to predict in

which of the two half spaces, defined by the cutting hyperplane, the instance $\boldsymbol{x}$ resides. The coefficient vector $\boldsymbol{w}$ is produced once randomly and then fixed for further usage, thus fixing the concept to be learned; whereas the learning examples are sampled uniformly from the input space.

A hyperplane data generator is implemented in MOA and can be used by instantiating the class *HyperplaneGenerator*. This generator creates data samples with only numerical attributes; it allows a percentage of noisy examples, by simply inverting the assigned class label. The following list describes the important attributes characterizing the hyperplane data:

- The seed according to which the coefficient vector $\boldsymbol{w}$ is generated.
- The seed according to which each new instance $\boldsymbol{x}$ is generated.
- The number of dimensions d.
- The allowed percentage of noise.

D.1.2 Distance to hyperplane data

Learning task: Regression.
This is another synthetic data set that we created by modifying the *HyperplaneGenerator* in MOA as follows: The output for an instance $\boldsymbol{x}$ is not determined by the sign of $\boldsymbol{w}^\top \boldsymbol{x}$, where $\boldsymbol{w}$ is the normal vector of the hyperplane, but by the absolute value $y = f_1(\boldsymbol{x}) = |\boldsymbol{w}^\top \boldsymbol{x}|$. Hence, the problem is to predict the distance from $\boldsymbol{x}$ to the hyperplane. Similarly, one can also generate the squared distance $y = f_2(\boldsymbol{x}) = (\boldsymbol{w}^\top \boldsymbol{x})^2$ and the cubed distance $y = f_3(\boldsymbol{x}) = |\boldsymbol{w}^\top \boldsymbol{x}|^3$, which are arguably more difficult to learn than the absolute distance.

We added the implementation of the distance to hyperplane data in MOA in the class *HyperplaneGeneratorReg*; this generator allows a certain percentage of instances to be intentionally corrupted to simulate noisy samples, by adding a random ϵ to the distance. The following list describes the important attributes characterizing this data generator:

- The seed according to which the coefficient vector $\boldsymbol{w}$ is generated.
- The seed according to which each new instance $\boldsymbol{x}$ is generated.
- The number of dimensions d.
- The allowed percentage of noise.

- The type of the distance applied Distance, SquaredDistance or CubedDistance.

D.1.3 Random trees data

Learning Task: Binary and multiclass classification.
Another important stream generator is the random trees generator. Its underlying model is a decision tree for a desired number of attributes and classes. The tree is built by forming internal nodes with conditions on randomly chosen attributes; after building the random tree, class labels are assigned randomly to leaf nodes. Instances are sampled uniformly from the input space, while class labels are determined by the tree. Both types of attributes, numerical and nominal, can be employed in this synthetic data.

This data can be generated for an arbitrary number of attributes and classes using the *RandomTreeGenerator* class offered by MOA. The following list describes the important attributes characterizing the random trees data:

- The number of classes to be generated.
- The seed according to which the decision tree is randomly generated.
- The seed according to which each new instance $\boldsymbol{x}$ is generated.
- The maximum depth of the random tree.
- The number of dimensions.

D.1.4 Radial basis function data

Learning Task: Binary and multiclass classification.
This data is of a nature complexer than the previously described ones. The data generating process starts by choosing random centroids for multivariate Gaussian distributions and randomly assigning class labels to them. Moreover, each centroid is assigned a random covariance matrix and a weight. A new data sample is generated by first choosing a random centroid, with respect to its weight, and then by sampling an instance from the Gaussian distribution associated with the chosen centroid.

An RBF data generator is implemented in MOA and can be used by instantiating the class *RandomRBFGenerator*, which allows numeric attributes only. The following list describes the important attributes characterizing the RBF data:

- The number of classes to be considered.

- The seed according to which the centroids are generated.
- The seed according to which each new instance $\boldsymbol{x}$ is generated.
- The number of dimensions d.
- The number of centroids in the model.

D.1.5 SEA concept functions

Learning Task: Binary classification.
SEA concept functions are decision rules with three numeric attributes $a, b, c \in [0, \cdots, 10]$. The binary decision is made based on the first two attributes and ignoring the third one; the binary class label is assigned according to the inequality $a + b \leq \theta$. This data set was introduced in [164] with four decision functions each with a different threshold θ, such that $\theta \in \{7, 8, 9, 9.5\}$. A certain percentage of instances can be intentionally corrupted to simulate noisy samples, by simply inverting the class label.

An SEA concept functions data generator is implemented in MOA; it can be used by instantiating the class *SEAGenerator*. The following list describes the important attributes characterizing the SEA data:

- The seed according to which each new instance $\boldsymbol{x}$ is generated.
- The decision function to be considered, i.e the value of the threshold θ.
- The allowed percentage of noise.

D.1.6 STAGGER concept functions

Learning Task: Binary classification.
Initially introduced by Schilmmer in [142], the STAGGER functions are Boolean functions based on three nominal attributes: size, shape and color. MOA's data generator can be used by instantiating the class *STAGGERGenerator* which implements the three Boolean functions, as defined in the original paper. The following list describes the important attributes characterizing the STAGGER data:

- The Boolean function to be considered.
- The seed according to which each new instance $\boldsymbol{x}$ is generated.

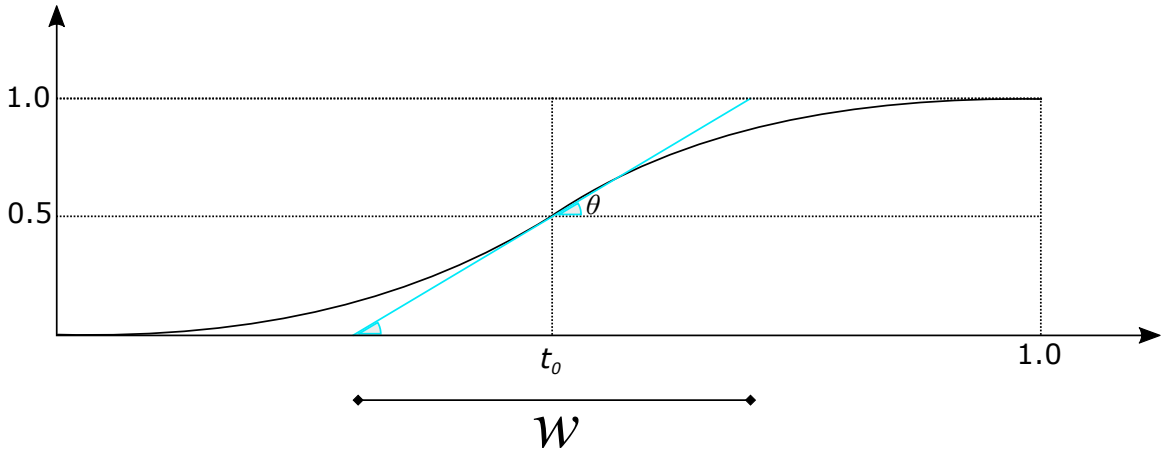


Figure D.1: The sigmoid function.

D.2 Synthetic Data Manipulation

Not only data sets can be synthetic, but also synthetic effects can be integrated into real-world and synthetic data sets. In this section, we present two such methods to simulate a change in a data generating process.

D.2.1 Concept drift simulation

As explained in Chapter 2, a concept change occurs when the so far observed concept becomes obsolete and a new concept begins to emerge. This scenario can be simulated using the functionality of the class *ConceptDriftStream* offered by MOA. The idea underlying this procedure is to mix two pure distributions in a probabilistic way, through gradually varying the corresponding probability degrees. In the beginning, examples are taken from the first pure stream with probability 1, this probability is decreased in favor of the second stream in the course of time. More specifically, the probability is controlled by means of the sigmoid function

$$f(t) = \left(1 + e^{-4(t-t_0)/w}\right)^{-1} \quad .$$

This function has two parameters: t_0 is the mid point of the change, while w is the length of this change. The length of the drift is related to the tangent of the sigmoid function at the center of the drift by $\tan\theta = \frac{1}{w}$, as explained in Figure D.1.

D.2.2 Sampling drift simulation

Instead of assuming that the data is always uniformly sampled from each dimension of the input space, other sampling distributions can be applied. When the properties

of the sampling distribution change over time, e.g., by gradually adjusting its parameters, a change in the sampling distribution occurs, leading to a virtual drift, see Section 2.3. We added this ability to MOA by implementing the class *SamplingDriftStream* which uses a Gaussian sampling distribution on each dimension $\sim \mathrm{G}(\mu, 0.1)$. A change can be realized by either continuously changing μ by adding a small fraction $\delta \in [0.0001, 0.001]$ after each data sample, or in a discrete manner by adding a larger $\delta \in [0.1, 0.3]$ after each N sampled instances.

D.3 Real Data Sets

Most of the real data sets used in this thesis are standard benchmarks taken from the UCI repository[1] [107]. Two of the real data sets are acquired from the DELVE[2] repository.

D.3.1 Cover type data

Learning Task: Multiclass classification.
The aim of this data set is to predict the forest cover type from cartographic observations. Observations, 30×30 meter cells, are labeled from the data provided by the USFS (US Forest Service). It contains 581,012 instances, each of which is described by ten numeric attributes and 44 binary attributes. The binary attributes are worthless, as they represent a binarization of two nominal attributes. Thus, the binary attributes are replaced by two nominal attributes Wilderness_Area and Soil_Type, which have now 4 and 40 nominal values respectively. The target attribute, cover type, is categorical and takes one of seven unique values.

D.3.2 Mushroom data

Learning Task: Binary classification.
The objective of this data set is to predict whether a mushroom is edible or poisonous. The data takes samples from 23 species of gilled mushrooms; each mushroom is described by 21 nominal attributes, after removing one attribute which has missing values in 30% of the cases. In total, this data set contains 8,124 data samples.

[1] `http://archive.ics.uci.edu/ml/`, accessed on October 8, 2015
[2] `http://www.cs.utoronto.ca/~delve/data/datasets.html`, accessed on October 8, 2015

D.3.3 Page blocks data

Learning Task: Multiclass classification.
The aim of this data set is to classify blocks of an image into one of five classes: text, horizontal line, picture, vertical line and graphic. The blocks are segmented and described by various attributes and pixel information. In total, the data contains 5,473 blocks, each described by ten numerical attributes.

D.3.4 Letter recognition

Learning Task: Multiclass classification.
The problem here is to recognize 26 English letters, which are written using 20 different fonts. After the post-processing of the original graphical representation, each letter is described by 16 primitive numerical features. In total, the data set comprises 20,000 examples.

D.3.5 StatLog (shuttle) data

Learning Task: Multiclass classification.
The objective of this data set is to learn the mapping between the various types of shuttles and their nine numeric attributes. One of the classes is dominant and covers about 80% of the data set. The total number of shuttles is 58,000; with seven different shuttle types to be distinguished.

D.3.6 Skin segmentation data

Learning Task: Binary classification.
The target of this data set is to distinguish skin from non-skin samples, based on their numeric color (Blue, Green and Red) attributes. Samples are randomly selected from images of people; the group of people varies in terms of age, race and gender. The total number of samples is 245,057.

D.3.7 MAGIC gamma telescope data

Learning Task: Binary classification.
This is a simulation data set for modeling the registration of gamma particles using the imaging technique, in a ground-based atmospheric Cherenkov gamma telescope. Depending on the energy, the task is to distinguish the collected photons, in the shower image, caused by primary gammas from those caused by the cosmic rays in

the upper atmosphere. This data set contains 19,020 data samples, described by ten numeric attributes.

D.3.8 Breast cancer Wisconsin

Learning Task: Binary classification.
This data comprises a set of reported clinical cases with the task of classifying them as benign or malignant. Each case is described by nine integer attributes. We remove the cases with missing values which form about 2% of the whole data; the number of remaining cases is 683.

D.3.9 Parkinson's telemonitoring data

Learning Task: Regression, multi-target prediction.
This data set consists of biomedical voice measurements from 42 Parkinson patients at an early stage of the disease. There are 18 numerical measurements for each recording, and the total number of recordings is 5,875. The target of this data set is to predict the attribute "motor UPDRS" or the attribute "total UPDRS", thus this data set can be either split into two independent regression problems or be studied as a multi-target problem.

D.3.10 Slice localization data

Learning Task: Regression.
The target here is to predict the relative location of computed tomography (CT) slices on axial axis. This data set is extracted from 53,500 images taken for 74 patients, 43 males and 31 females. Each CT image is described in terms of 384 features, after removing the patient's ID. The target attribute is the relative location of the CT slice on the axial axis of the human body; this attribute is a numeric value in the range $[0, 180]$, where 0 denotes the top of the head and 180 is the soles of the feet.

D.3.11 Bank32h

Learning Task: Regression.
This data is acquired from the DELVE repository. It is generated by simulating the way in which customers, from different rural areas, select and reject banks; this selection is based on the waiting queues in a series of banks successfully visited in order to accomplish different tasks. The data set contains 8,192 cases, each of which has 32 numeric attributes.

D.3.12 Census-house

Learning Task: Regression.
This data is also obtained from the DELVE repository; it is a collection of data sets designed on the basis of the US census data of the year 1999. The purpose of this data set is to predict the median price of houses in different regions based on the demographic properties. Each house has eight attributes and total number of 22,784 houses.

D.4 Event Streams

In this section, we present two types of event streams; the focus of these streams is not the instances but the events they emit, exhibit or suffer from. The event streams presented here are used as a proof of concept for our survival analysis approach, see Chapter 5.

D.4.1 Earthquake event stream

A stream of earthquakes can be obtained from the United States Geological Survey (USGS)[3], specifically from the catalog of the National Earthquake Information Center (NEIC)[4] whose mission is to quickly discover the most recent destructive earthquakes, in terms of location and magnitude, and to broadcast this information to international agencies and scientists.

Entries in the USGS/NEIC catalog can be added or modified at any time, as they are under continuous auditing to maintain their correctness and coherency; the online catalog retains only significant earthquakes with a magnitude[5] larger than 2.5, despite the very few micro-earthquakes with a magnitude less than one.

This data set can be observed as a stream of events, in which each event is an earthquake identified by its geographic coordinate, the exact time of occurrence, up to the second, the magnitude and depth. Table D.1, which is also presented in Chapter 5, depicts five earthquakes with their occurrence time and attributes; these earthquakes occurred on the 1st of Jan 2012.

[3] `http://www.usgs.gov`, accessed on October 8, 2015

[4] `http://earthquake.usgs.gov/contactus/golden/neic.php`, accessed on October 8, 2015

[5] We quote the USGS Earthquake Magnitude Policy: "Typical additional information can include that the magnitude was estimated using an extension of the concept originally developed by Richter, and/or that there are several different methods for estimating the size of an earthquake, all of which are consistent with the Richter scale, and a description of the measurement technique used." `http://earthquake.usgs.gov/aboutus/docs/020204mag_policy.php`, accessed on October 8, 2015

D.4.2 Twitter stream

Twitter[6] is an online microblogging web site; it is a service that allows users to send short messages of up to 140 characters known as *tweets*. Every tweet is attributed by some meta data, including the ID of the user who wrote it and the time at which the tweet was sent. Further attributes can also be extracted from the tweet with the permission of the user, such as the user's geolocation from which the tweet was posted. The geolocation is acquired from the embedded GPS functionality in the mobile device; it is represented as a tuple (`lat`, `long`) with entries for the latitude and the longitude. Table D.2 shows an example of Twitter data written in Json[7] format; this example contains two Twitter messages after obfuscating some attributes and removing unimportant ones; important attributes are written in bold. The shown messages are artificially created without any real user information.

Twitter streams can form the source of topic-based and spatial-based event streams, either by restricting the messages to contain specific keywords, or to be produced from a certain country/city/geolocation.

[6] `http://www.twitter.com`, accessed on October 9, 2015
[7] `http://json.org`, accessed on October 9, 2015

Year	Month	Day	UTC Time hhmmss.mm	Latitude	Longitude	Mag.	Depth	Catalog
2012	01	01	003008.77	12.008	143.487	5.1	35	PDE-W
2012	01	01	003725.28	63.337	-147.516	3.0	65	PDE-W
2012	01	01	004342.77	12.014	143.536	4.4	35	PDE-W
2012	01	01	005008.04	-11.366	166.218	5.3	67	PDE-W
2012	01	01	012207.66	-6.747	130.007	4.2	145	PDE-W

Table D.1: A sample earthquake data containing 5 earthquakes occurred on the first day of 2012.

favorited:false, **text**:'Stau: A8 München Richtung Stuttgart 6 km zur Ausfahrt im Schneckentempo..', truncated:false, **created_at**:Fri Feb 10 10:38:47 +0000 2012, retweeted:false, retweet_count:0, coordinates:type:Point, **coordinates**:[9.55755, 48.6333], ..., entities:user_mentions:[], urls:[], hashtags:[], geo:type:Point, coordinates:[48.6333, 9.55755], ..., place:bounding_box:type:Polygon, coordinates:[[[9.534815, 48.616779], [9.594667, 48.616779], [9.594667, 48.640891], [9.534815, 48.640891]]], place_type:city, ..., country_code:DE, attributes:, full_name:Aichelberg, Göppingen, name:Aichelberg, id:29ef9f01a553e601, country:Germany, ..., id_str:###, user:default_profile:true, notifications:null, ..., time_zone:Berlin, created_at:Fri Sep 03 14:25:38 +0000 2010, verified:false, geo_enabled:true..., favourites_count:0, lang:de, ..., followers_count:335, ..., location:Karlsruhe, ..., name:###, ..., listed_count:21, following:null, screen_name:###, id:###, ..., statuses_count:10935, utc_offset:3600, friends_count:0, ..., id:###, ...
text:'top atmosphere in Weserstadion today, a very good match...', ..., **created_at**:Tue Apr 10 21:37:28 +0000 2012, place:bounding_box:type:Polygon, coordinates:[[[8.481599, 53.011035], [8.990593, 53.011035], [8.990593, 53.228969], [8.481599, 53.228969]]], country:Germany, attributes:, full_name:Bremen, Bremen, .., country_code:DE, name:Bremen, id:9467fbdc3cdbd2ef, place_type:city, coordinates:type:Point, **coordinates**:[8.837596, 53.06693] , retweeted:false, in_reply_to_status_id:null, ..., truncated:false, contributors:null, possibly_sensitive:false, in_reply_to_screen_name:null, favorited:false, user:default_profile:false, follow_request_sent:null, lang:de, friends_count:200, ..., is_translator:false, created_at:Sat May 23 13:01:45 +0000 2009, id_str:###, ..., url:null, following:null, verified:false, ..., location:Germany, ..., statuses_count:4537, ..., time_zone:Berlin, .., utc_offset:3600, followers_count:432, ..., id:###, retweet_count:0

Table D.2: A sample Twitter data containing two Twitter messages.

	Learning Task			Properties					Source	
	Binary Cl.	Multiclass Cl.	Regression	#All Attributes	#Numeric Attributes	#Nominal Attributes	#Instances	Target Attribute	Origin	**R**eal/**A**rtif./**S**im.
cover type		✓		12	10	2	581,012	7-classes	UCI	R
mushroom	✓			21	-	21	8,124	binary	UCI	R
breast	✓			9	9	-	699	binary	UCI	R
page blocks		✓		10	10	-	5,473	5-classes	UCI	R
letter		✓		16	16	-	20,000	26-Classes	UCI	R
StatLog		✓		9	9	-	58,000	7-classes	UCI	R
skin seg.	✓			3	3	-	245,057	binary	UCI	R
MAGIC	✓			10	10	-	19,020	binary	UCI	S
Parkinson's tel.										
motor UPDRS			✓	18	18	-	5,875	[5.0377,39.511]	UCI	R
total UPDRS			✓	18	18	-	5,875	[7,54.992]	UCI	R
slice loc.			✓	384	384	-	53,500	[0,180]	UCI	R
bank32h			✓	32	32	-	8,192	[0,0.819665]	DELVE	S
house8L			✓	8	8	-	22,784	[0,500001]	DELVE	R
hyperplane	✓			✓	✓	-	∞	binary	MOA	A
			✓	✓	✓	-	∞	distance	MOA	A
			✓	✓	✓	-	∞	squared dis.	MOA	A
			✓	✓	✓	-	∞	cubed dis.	MOA	A
random trees	✓	✓		✓	✓	✓	∞	multi	MOA	A
RBF	✓	✓		✓	✓	-	∞	multi	MOA	A
SEA	✓			3	3	-	∞	binary	MOA	A
STAGGER	✓			3	-	3	∞	binary	MOA	A

Table D.3: Summary of the data sets used in this thesis.

Appendix E

Incremental Statistics

Based on the formal definitions of the sample mean

$$\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{E.1}$$

and the unbiased variance

$$s_n^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x}_n)^2}{n-1} = \frac{\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2/n}{n-1} = \frac{\sum_{i=1}^{n} x_i^2 - n\bar{x}_n^2}{n-1} \ , \tag{E.2}$$

for a sample data of size n, we derive the sample mean and variance on both an incremental sample and on a sliding window of samples.

E.1 Incremental Moments

In the case where the data sample's size n is increasing with time, as a result of accumulating data, the *incremental sample mean* becomes

$$\begin{aligned} \bar{x}_n &= \frac{1}{n}\sum_{i=1}^{n} x_i \ , \\ \bar{x}_{n+1} &= \frac{n}{n+1}\bar{x}_n + \frac{1}{n+1}x_{n+1} \ . \end{aligned} \tag{E.3}$$

The *incremental variance* is derived in [166] by defining $M_{2,n}$, which for simplification holds the nominator of (E.2)

$$M_{2,n} = \sum_{i=1}^{n}(x_i - \bar{x}_n)^2 \ . \tag{E.4}$$

The difference of $M_{2,n}$ between two consecutive samples n and $n+1$ is then

$$M_{2,n+1} - M_{2,n} = \sum_{i=1}^{n+1}(x_i - \bar{x}_{n+1})^2 - \sum_{i=1}^{n}(x_i - \bar{x}_n)^2 \tag{E.5}$$

$$= \sum_{i=1}^{n+1} x_i^2 - (n+1)\bar{x}_{n+1}^2 - \sum_{i=1}^{n} x_i^2 + (n)\bar{x}_n^2$$

$$\dots$$

$$= (x_{n+1} - \bar{x}_n)(x_{n+1} - \bar{x}_{n+1}) \ . \tag{E.6}$$

The incremental variance is then derived by substituting (E.4) and (E.6) in (E.2)

$$s_{n+1}^2 = \frac{M_{2,n+1}}{n} \tag{E.7}$$

$$= \frac{M_{2,n} + (x_{n+1} - \bar{x}_n)(x_{n+1} - \bar{x}_{n+1})}{n}$$

$$= \frac{(n-1)s_n^2 + (x_{n+1} - \bar{x}_n)(x_{n+1} - \bar{x}_{n+1})}{n} \ . \tag{E.8}$$

E.2 Shifting Moments

In the case of a sliding window, of fixed-size n, over data samples, the *shifting sample mean* $\bar{x}_{t+1}$ at the instance $t+1$ is derived from the shifting sample mean found at the previous instance t

$$\bar{x}_t = \frac{1}{n}\sum_{i=t-n+1}^{t} x_i$$

$$\bar{x}_{t+1} = \frac{1}{n}\sum_{i=t-n+2}^{t+1} x_i = \bar{x}_t + \frac{x_{t+1} - x_{t-n+1}}{n} \ . \tag{E.9}$$

Similarly, the *shifting variance* at the instance $t+1$ is

$$s_t^2 = \frac{\sum_{i=t-n+1}^{t}(x_i - \bar{x}_t)^2}{n-1} = \frac{\sum_{i=t-n+1}^{t} x_i^2 - n\bar{x}_t^2}{n-1}$$

$$s_{t+1}^2 = \frac{\sum_{i=t-n+2}^{t+1}(x_i - \bar{x}_{t+1})^2}{n-1} = \frac{\sum_{i=t-n+2}^{t+1} x_i^2 - n\bar{x}_{t+1}^2}{n-1} \tag{E.10}$$

$$s_{t+1}^2 = s_t^2 + \frac{n\bar{x}_t^2 - n\bar{x}_{t+1}^2 - x_{t-n+1}^2 + x_{t+1}^2}{n-1} \ . \tag{E.11}$$

Bibliography

[1] Charu C. Aggarwal. A survey of stream classification algorithms. In Charu C. Aggarwal, editor, *Data Classification: Algorithms and Applications*, pages 245–274. CRC Press, Boca Raton, FL, USA, 2014.

[2] David W. Aha, editor. *Lazy Learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[3] David W. Aha, Dennis F. Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[4] Ezilda Almeida, Carlos Abreu Ferreira, and João Gama. Adaptive model rules from data streams. In *ECML PKDD 2013, Machine Learning and Knowledge Discovery in Databases European Conference*, pages 480–492, Prague, Czech Republic, 2013.

[5] Amineh Amini, Teh Ying Wah, and Hadi Saboohi. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, 29(1):116–141, 2014.

[6] Plamen P. Angelov. *Evolving Rule-based Models: A Tool for Design of Flexible Adaptive Systems*. Springer-Verlag, London, UK, 2002.

[7] Plamen P. Angelov. Evolving takagi-sugeno fuzzy systems from data streams (ets+). In Plamen P. Angelov, Dimitar P. Filev, and Nik Kasabov, editors, *Evolving Intelligent Systems: Methodology and Applications*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2010.

[8] Plamen P. Angelov and Dimitar P. Filev. An approach to online identification of takagi-sugeno fuzzy models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(1):484–498, 2004.

[9] Plamen P. Angelov, Dimitar P. Filev, and Nik Kasabov, editors. *Evolving Intelligent Systems: Methodology and Applications*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2010.

[10] Plamen P. Angelov and Nikola K. Kasabov. Evolving computational intelligence systems. In *Proceedings of the 1st International Workshop on Genetic Fuzzy Systems*, pages 76–82, Granada, Spain, 2005.

[11] Plamen P. Angelov and Nikola K. Kasabov. Evolving intelligent systems, eIS. *IEEE SMC eNewsLetter*, 15:1–13, 2006.

[12] Plamen P. Angelov, Edwin Lughofer, and Xiaowei Zhou. Evolving fuzzy classifiers using different model architectures. *Fuzzy Sets and Systems*, 159(23):3160–3182, 2008.

[13] Martin Anthony and Norman Biggs. *Computational Learning Theory: An Introduction*. Cambridge University Press, Cambridge, UK, 1992.

[14] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, Madison, WI, USA, 2002.

[15] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86, 2006.

[16] T. Bailey and A. K. Jain. A note on distance-weighted k-nearest neighbor rules. *IEEE Transactions on Systems, Man and Cybernetics*, 8(4):311–313, 4 1978.

[17] Jürgen Beringer and Eyke Hüllermeier. Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6):627–650, 2007.

[18] Joseph Berkson and Robert P. Gage. Calculation of survival rates for cancer. In *Proceedings of the staff meetings. Mayo Clinic*, volume 25, pages 270–286, 1950.

[19] Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. *CoRR*, abs/1007.0085, 2010.

[20] Albert Bifet and Ricard Gavaldà. Kalman filters and adaptive windows for learning in data streams. In *The 9th International Conference on Discovery Science (DS-2006)*, pages 29–40, Barcelona, Spain, 2006.

[21] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, pages 443–448, Minneapolis, MN, USA, 2007.

[22] Albert Bifet and Ricard Gavaldà. Mining adaptively frequent closed unlabeled rooted trees in data streams. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 34–42, New York, NY, USA, 2008.

[23] Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *Proceedings of IDA 2009, the 8th International Symposium on Intelligent Data Analysis*, pages 249–260, Lyon, France, 2009.

[24] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[25] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. Mining frequent closed graphs on evolving data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 591–599, San Diego, CA, USA, 2011.

[26] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, Proceedings.*, pages 299–310, Hyderabad, India, 2010.

[27] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148, Paris, France, 2009.

[28] J. Frédéric Bonnans, Gilbert J. Charles, Claude Lemaréchal, and Claudia A. Sagastizábal. *Numerical Optimization: Theoretical and Practical Aspects.* Springer-Verlag New York, NY, USA, 2006.

[29] Léon Bottou. On-line learning in neural networks. chapter On-line Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, Cambridge, UK, 1998.

[30] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140.

[31] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and regression trees.* Wadsworth and Brooks, Monterey, CA, USA, 1984.

[32] David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical Report 4148, RSRE, 1988.

[33] Dariusz Brzezinski and Jerzy Stefanowski. Accuracy updated ensemble for data streams with concept drift. In *Hybrid Artificial Intelligent Systems - 6th International Conference, HAIS 2011, Proceedings, Part II*, pages 155–163, Wroclaw, Poland, 2011.

[34] Dariusz Brzezinski and Jerzy Stefanowski. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, 265:50–67, 2014.

[35] 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, 2014.

[36] Toon Calders, Nele Dexters, Joris J.M. Gillis, and Bart Goethals. Mining frequent itemsets in a stream. *Information Systems*, 39:233 – 255, 2014.

[37] James Cheng, Yiping Ke, and Wilfred Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems*, 16(1):1–27, 2007.

[38] Paolo Ciaccia, Marco Patella, Fausto Rabitti, and Pavel Zezula. Indexing metric spaces with m-tree. In Matteo Cristani and Letizia Tanca, editors, *Atti del Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD'97)*, pages 67–86, Residenza di Costagrande, Verona, Italy, 1997.

[39] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and

Manfred A. Jeusfeld, editors, *Proceedings of 23rd International Conference on Very Large Data Bases, VLDB'97*, pages 426–435, Athens, Greece, 1997.

[40] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 01 1967.

[41] David Roxbee Cox. Regression models and life tables. *Journal of the Royal Statistical Society B*, 34:187–220, 1972.

[42] David Roxbee Cox and David Oakes. *Analysis of Survival Data*. Chapman & Hall, London, UK, 1984.

[43] Sidney J. Cutler and Fred Ederer. Maximum utilization of the life table method in analyzing survival. *Journal of chronic diseases*, 8(6):699–712, 1958.

[44] Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, UK, 1859.

[45] Belur V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1991.

[46] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[47] D. J. Davis. An analysis of some failure data. *Journal of the American Statistical Association*, 47(258):113–150, 1952.

[48] A. P. Dawid. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278–292, 1984.

[49] Jonathan de Andrade Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André Carlos Ponce Leon Ferreira de Carvalho, and João Gama. Data stream clustering: A survey. *ACM Computing Surveys*, 46(1):13:1–13:31, 2013.

[50] Magdalena Deckert and Jerzy Stefanowski. Rill: Algorithm for learning rules from streaming data with concept drift. In Troels Andreasen, Henning Christiansen, Juan-Carlos Cubero, and Zbigniew W. Raś, editors, *Foundations of Intelligent Systems - 21st International Symposium, ISMIS 2014, Proceedings*, pages 20–29, Roskilde, Denmark, 2014.

[51] Sarah Jane Delany, Pádraig Cunningham, Alexey Tsymbal, and Lorcan Coyle. A case-based technique for tracking concept drift in spam filtering. In Ann Macintosh, Richard Ellis, and Tony Allen, editors, *Applications and Innovations in Intelligent Systems XII*, pages 3–16. Springer London, UK, 2005.

[52] Jochen Van den Bercken, Björn Blohsfeld, Jens-Peter Dittrich, Jürgen Krämer, Tobias Schäfer, Martin Schneider, and Bernhard Seeger. XXL - A library approach to supporting efficient implementations of advanced database queries. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases, VLDB 2001*, pages 39–48, Roma, Italy, 2001.

[53] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[54] Pedro Domingos. Rule induction and instance-based learning: A unified approach. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, pages 1226–1232, Montreal, QC, Canada, 1995.

[55] Pedro Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(4):141–168, 1996.

[56] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, USA, 2000.

[57] Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

[58] Didier Dubois and Henri Prade. *Fuzzy sets and systems: Theory and applications*. Academic Press, Inc., Orlando, FL, USA, 1980.

[59] Harald Dyckhoff and Witold Pedrycz. Generalized means as model of compensative connectives. *Fuzzy sets and Systems*, 14(2):143–154, 1984.

[60] Martin Ester, Hans peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining KDD-96*, pages 226–231, Portland, OR, USA, 1996.

[61] Vladimir Estivill-Castro. Why so many clustering algorithms: A position paper. *ACM SIGKDD Explorations Newsletter*, 4(1):65–75, 2002.

[62] Manning Feinleib. A method of analyzing log-normally distributed survival data with incomplete follow-up. *Journal of the American Statistical Association*, 55(291):534–545, 1960.

[63] Francisco J. Ferrer-Troyano, Jesús S. Aguilar-Ruiz, and José Cristóbal Riquelme Santos. Incremental rule learning and border examples selection from numerical data streams. *Journal of Universal Computer Science*, 11(8):1426–1439, 2005.

[64] Francisco J. Ferrer-Troyano, Jesús S. Aguilar-Ruiz, and José Cristóbal Riquelme Santos. Data streams classification by incremental rule learning with parameterized generalization. In *the 2006 ACM Symposium on Applied Computing (SAC)*, pages 657–661, Dijon, France, 2006.

[65] 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.

[66] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: A review. *ACM SIGMOD Record*, 34(2):18–26, 2005.

[67] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. A survey of classification methods in data streams. In Charu C. Aggarwal, editor, *Data Streams: Models and Algorithms*, pages 39–59. Springer US, Boston, MA, USA, 2007.

[68] João Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, London, UK, 1st edition, 2010.

[69] João Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012.

[70] João Gama and Mohamed Medhat Gaber. *Learning from Data Streams*. Springer-Verlag Berlin Heidelberg, Germany, 2007.

[71] João Gama and Petr Kosina. Learning decision rules from data streams. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, Barcelona, Catalonia, Spain, 2011.

[72] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence*, Lecture Notes in Computer Science, pages 286–295, São Luis, Maranhão, Brazil, 2004.

[73] João Gama, Pedro Medas, and Pedro Rodrigues. Learning decision trees from dynamic data streams. In *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC)*, pages 573–577, Santa Fe, NM, USA, 2005.

[74] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338, Paris, France, 2009.

[75] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

[76] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014.

[77] Jing Gao, Wei Fan, Jiawei Han, and Philip S. Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, pages 3–14, Minneapolis, MN, USA, 2007.

[78] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery*, 4(2-3):127–162, 2000.

[79] Christophe Giraud-Carrier. A note on the utility of incremental learning. *AI Communications*, 13(4):215–224, 2000.

[80] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[81] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[82] Zhiheng Huang, Tamás D. Gedeon, and Masoud Nikravesh. Pattern trees induction: A new machine learning method. *IEEE Transactions on Fuzzy Systems*, 16(4):958–970, 2008.

[83] Eyke Hüllermeier. Fuzzy methods in machine learning and data mining: Status and prospects. *Fuzzy Sets and Systems*, 156(3):387–406, 2005.

[84] Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams. 2003.

[85] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, USA, 2001.

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

[87] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer-Verlag New York, NY, USA, 2014.

[88] Cezary Z. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(1):1–14, 1998.

[89] John D. Kalbfleisch and Ross L. Prentice. *The statistical analysis of failure time data*, volume 360. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2nd edition, 2011.

[90] John H.K. Kao. Computer methods for estimating weibull parameters in reliability studies. *IRE Transactions on Reliability and Quality Control*, PGRQC-13:15–22, 1958.

[91] Edward L. Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958.

[92] Nikola K. Kasabov. Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 31(6):902–918, 2001.

[93] Nikola K. Kasabov. *Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines*. Springer-Verlag, London, UK, 1st edition, 2003.

[94] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, VLDB '04, pages 180–191, Toronto, ON, Canada, 2004.

[95] David G. Kleinbaum and Mitchel Klein. *Survival Analysis A Self-Learning Text*. Springer-Verlag New York, NY, USA, 2nd edition, 2005.

[96] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

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

[98] Janet L. Kolodner. *Case-based Reasoning*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1993.

[99] Petr Kosina and João Gama. Handling time changing data with adaptive very fast decision rules. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Bristol, UK, 2012.

[100] Bart Kosko. Fuzzy entropy and conditioning. *Information Sciences*, 40(2):165–174, 1986.

[101] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

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

[103] Brian Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2013.

[104] Mihai M. Lazarescu, Svetha Venkatesh, and Hung H. Bui. Using multiple windows to track concept drift. *Intelligent Data Analysis*, 8(1):29–59, 2004.

[105] Elisa T. Lee. *Statistical Methods for Survival Data Analysis*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2nd edition, 1992.

[106] Vincent Lemaire, Christophe Salperwyck, and Alexis Bondu. A survey on supervised classification on data streams. In *Business Intelligence - 4th European Summer School, eBISS 2014*, volume 205 of *Lecture Notes in Business Information Processing*, pages 88–125, 2014.

[107] Moshe Lichman. UCI machine learning repository, 2013.

[108] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1999.

[109] D. O. Loftsgaarden and C. P. Quesenberry. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics*, 36(3):1049–1051, 1965.

[110] Edwin Lughofer. FLEXFIS: A robust incremental learning approach for evolving takagi-sugeno fuzzy models. *IEEE Transactions on Fuzzy Systems*, 16(6):1393–1410, 2008.

[111] Edwin Lughofer. *Evolving Fuzzy Systems: Methodologies, Advanced Concepts and Applications*. Springer-Verlag Berlin Heidelberg, Germany, 2011.

[112] James B. MacQueen. Some methods for classification and analysis of multivariate observation. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, CA, USA, 1967.

[113] Robert W. Makuch. Adjusted survival curve estimation using covariates. *Journal of chronic diseases*, 35(6):437–443, 1982.

[114] Ebrahim H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. *Proceedings of the Institution of Electrical Engineers*, 121(12):1585–1588, 1974.

[115] Oded Maron and Andrew W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems 6, Proceedings of the 7th NIPS Conference*, pages 59–66, Denver, CO, USA, 1993.

[116] Colin McDiarmid. On the method of bounded differences. In J. Siemons, editor, *Surveys in combinatorics*, volume 141 of *London Mathematical Society Lecture Note Series*, pages 148–188. Cambridge University Press, Cambridge, UK, 1989.

[117] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, pages 18–32, Avignon, France, 1996.

[118] Karl Menger. Statistical metrics. *Proceedings of the National Academy of Sciences of the United States of America*, 28(12):535–537, 1942.

[119] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1st edition, 1997.

[120] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[121] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi. Test of Page-Hinkley, an approach for fault detection in an agro-alimentary production system. In *Proceedings of the Asian Control Conference*, volume 2, Melbourne, Australia, 2004.

[122] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and Information Systems*, 45(3):535–569, 2014.

[123] Nilsson Nils. *Learning Machines*. McGrwa-Hill, New Yowrk, NY, USA, 1965.

[124] Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, Key West, FL, USA, 2001.

[125] E. S. PAGE. Continuous inspection schemes. *Biometrika*, 41(1-2):100–115, 1954.

[126] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 09 1962.

[127] Witold Pedrycz and Fernando Gomide. *An introduction to fuzzy sets: analysis and design*. Mit Press, London, UK, 1998.

[128] Robi Polikar, Lalita Upda, Satish S. Upda, and Vasant Honavar. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(4):497–508, 2001.

[129] John Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[130] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[131] K. A. Rasmani and Q. Shen. Weighted linguistic modelling based on fuzzy subsethood values. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, volume 1, St. Louis, MO, USA, 2003.

[132] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[133] S. W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1):97–101, 1959.

[134] Frank Rosenblatt. The perceptron — a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[135] 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*, 33(2):191–198, 2012.

[136] Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions On Knowledge And Data Engineering*, 25(6):1272–1279, 2013.

[137] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *Proceedings of the 12th IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1393–1400, Kyoto, Japan, 2009.

[138] Marcos Salganicoff. Density-adaptive learning and forgetting. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 276–283, University of Massachusetts, Amherst, MA, USA, 1993.

[139] Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, 1997.

[140] Steven Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6(3):251–276, 1991.

[141] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[142] Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[143] George S. Sebestyen. *Decision-Making Processes in Pattern Recognition*. Macmillan, New Yowrk, NY, USA, 1962.

[144] Robin Senge. *Machine Learning Methods for Fuzzy Pattern Tree Induction*. PhD thesis, Philipp University of Marburg, 2014.

[145] Robin Senge and Eyke Hüllermeier. Pattern trees for regression and fuzzy systems modeling. In *Proceedings of FUZZ-IEEE 2010, the IEEE International Conference on Fuzzy Systems*, pages 1–7, Barcelona, Spain, 2010.

[146] Robin Senge and Eyke Hüllermeier. Top-down induction of fuzzy pattern trees. *IEEE Transactions on Fuzzy Systems*, 19(2):241–252, 2011.

[147] Robin Senge and Eyke Hüllermeier. Fast fuzzy pattern tree learning for classification. *IEEE Transactions on Fuzzy Systems*, 23(6):2024–2033, 2015.

[148] John C. Shafer, Rakesh Agrawal, and Manish Mehta. Sprint: A scalable parallel classifier for data mining. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 544–555, San Francisco, CA, USA, 1996.

[149] Ammar Shaker and Eyke Hllermeier. Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. *Neurocomputing*, 150, Part A(0):250 – 264, 2015.

[150] Ammar Shaker and Eyke Hüllermeier. Hazard analysis on data streams. In *Proceedings of the 22th Workshop Computational Intelligence*, Dortmund, Germany, 2012.

[151] Ammar Shaker and Eyke Hüllermeier. IBLStreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4):235–249, 2012.

[152] Ammar Shaker and Eyke Hüllermeier. Instance-based classification and regression on data streams. In Moamar Sayed-Mouchaweh and Edwin Lughofer, editors, *Learning in Non-Stationary Environments*, pages 185–201. Springer New York, 2012.

[153] Ammar Shaker and Eyke Hüllermeier. Event history analysis on data streams: An application to earthquake occurrence. In Georg Krempl, Indrė Žliobaitė, Yin Wang, and George Forman, editors, *Proceedings of RealStream 2013, the 1st International Workshop on Real-World Challenges for Data Stream Mining.*, pages 43–46. Otto-von-Guericke University Magdeburg, Magdeburg, Germany, Prague, Czech Republic, 2013.

[154] Ammar Shaker and Eyke Hüllermeier. Recovery analysis for adaptive learning from non-stationary data streams. In Robert Burduk, Konrad Jackowski, Marek Kurzyǹski, Michałl Wożniak, Andrzej, and Żołlnierek, editors, *Proceedings of the 8th International Conference on Computer Recognition Systems CORES*

2013, volume 226 of *Advances in Intelligent Systems and Computing*, pages 289–298, Milkow, Poland, 2013.

[155] Ammar Shaker and Eyke Hüllermeier. Survival analysis on data streams: Analyzing temporal events in dynamically changing environments. *International Journal of Applied Mathematics and Computer Science*, 24(1):199–212, 2014.

[156] Ammar Shaker, Robin Senge, and Eyke Hüllermeier. Evolving fuzzy pattern trees for binary classification on data streams. In *Proceedings of the 20th Workshop Computational Intelligence*, Dortmund, Germany, 2010.

[157] Ammar Shaker, Robin Senge, and Eyke Hüllermeier. Evolving fuzzy pattern trees for binary classification on data streams. *Information Sciences*, 220:34–45, 2013.

[158] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

[159] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[160] Dan Simon, editor. *Evolutionary Optimization Algorithms*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2013.

[161] Jasmina Smailović, Miha Grčar, Nada Lavrač, and Martin Žnidaršič. Stream-based active learning for sentiment analysis in the financial domain. *Information Sciences*, 285:181–203, 2014.

[162] Herbert Spencer. *The Principles of Biology*. Williams and Norgate, London, UK, 1864.

[163] Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29:1213–1228, 1986.

[164] William N. Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, San Francisco, CA, USA, 2001.

[165] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.

[166] Katharina Tschumitschew and Frank Klawonn. Incremental statistical measures. In Moamar Sayed-Mouchaweh and Edwin Lughofer, editors, *Learning in Non-Stationary Environments*, pages 21–55. Springer New York, 2012.

[167] Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical report, Department of Computer Science, Trinity College Dublin, Ireland, 2004.

[168] Vernon Turner, John F. Gantz, David Reinsel, and Stephen Minton. The digital universe of opportunities: Rich data and the increasing value of the internet of things. *White Paper, International Data Corporation (IDC), sponsored by EMC Corporation*, April 2014.

[169] Paul E. Utgoff. ID5: an incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, USA, 1988.

[170] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, NY, USA, 1995.

[171] Indrė Žliobaitė. Learning under concept drift: an overview. Technical report, Vilnius University, Lithuania, 2009.

[172] Indrė Žliobaitė. Controlled permutations for testing adaptive learning models. *Knowledge and Information Systems*, 39(3):565–578, 2014.

[173] Indrė Žliobaitė, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):27–39, 2014.

[174] Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 06 1945.

[175] Boyu Wang and Joelle Pineau. Online ensemble learning for imbalanced data streams. *CoRR*, abs/1310.8004, 2013.

[176] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai-Long Nguyen, and François Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[177] L. J. Wei. The accelerated failure time model: a useful alternative to the cox regression model in survival analysis. *Statistics in medicine*, 11(14-15):1871–1879, 1992.

[178] Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *Proceedings of the European Conference on Machine Learning*, Vienna, Austria, 1993.

[179] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[180] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition, 2005.

[181] Ronald R Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190, 1988.

[182] Yu Yi, Thomas Fober, and Eyke Hüllermeier. Fuzzy operator trees for modeling rating functions. *International Journal of Computational Intelligence and Applications*, 8(4):413–428, 2008.

[183] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69(2):125–139, 1995.

[184] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[185] Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 40(6):1607–1621, 2010.

[186] H-J Zimmermann and P Zysno. Latent connectives in human decision making. *Fuzzy sets and systems*, 4(1):37–51, 1980.