Algorithms for Learning Regression Trees and Ensembles on Evolving Data Streams

Elena Ikonomovska

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia, October 2012

Evaluation Board:

Asst. Prof. Dr. Bernard Ženko, Chairman, Jožef Stefan Institute, Ljubljana, Slovenia

Asst. Prof. Dr. Zoran Bosnić, Member, Faculty of Computer and Information Science, University of Ljubljana, Slovenia

Dr. Albert Bifet, Member, Yahoo Research, Barcelona, Spain

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Elena Ikonomovska

ALGORITHMS FOR LEARNING REGRESSION TREES AND ENSEMBLES ON EVOLVING DATA STREAMS

Doctoral Dissertation

ALGORITMI ZA UČENJE REGRESIJSKIH DREVES IN ANSAMBLOV IZ SPREMENLJIVIH PODATKOVNIH TOKOV

Doktorska disertacija

Supervisor: Prof. Dr. Sašo Džeroski

Co-Supervisor: Prof. Dr. João Gama

Ljubljana, Slovenia, October 2012

To my mother Slavica

Contents

Abstract

In this thesis we address the problem of learning various types of decision trees from time-changing data streams. In particular, we study online machine learning algorithms for learning regression trees, linear model trees, option trees for regression, multi-target model trees, and ensembles of model trees from data streams. These are the most representative and widely used models in the category of interpretable predictive models.

A data stream is an inherently unbounded sequence of data elements (numbers, coordinates, multi-dimensional points, tuples, or objects of an arbitrary type). It is characterized with high inbound rates and non-stationary data distributions. Real-world scenarios where processing data streams is a necessity come from various management systems deployed on top of sensor networks, that monitor the performance of smart power grids, city traffic congestion, or scientific studies of environmental changes.

Due to the fact that this type of data cannot be easily stored or transported to a central database without overwhelming the communication infrastructure, data processing and analysis has to be done in-situ and in real-time, using constant amount of memory. To enable in-situ real-time learning it is crucial to perform an incremental computation of unbiased estimates for various types of statistical measures. This requires methods that would enable us to collect an appropriate sample from the incoming data stream and compute the necessary statistics and estimates for the evaluation functions on-the-fly.

We approached the problem of obtaining unbiased estimates on-the-fly by treating the evaluation functions as random variables. This enabled the application of existing probability bounds, among which the best results were achieved when using the Hoeffding bound. The algorithms proposed in this thesis therefore use the Hoeffding probability bound for bounding the probability of error when approximating the sample mean of a sequence of random variables. This approach gives us the statistical machinery for scaling up various machine learning tasks.

With our research we address three main sub-problems as part of the problem of learning tree-based model from time-changing data streams. The first one is concerned with the non-stationarity of concepts and the need for an informed adaptation of the decision tree. We propose online change detection mechanisms integrated within the incrementally learned model. The second subproblem is related to the myopia of decision tree learning algorithms while searching the space of possible models. We address this problem trough a study and a comparative assessment of online option trees for regression and ensembles of model trees. We advise the introduction of options for improving the performance, stability and quality of standard tree-based models. The third subproblem is related to the applicability of the proposed approach to the multi-target prediction task. This thesis proposes an extension of the predictive clustering framework in the online domain by incorporating Hoeffding bound probabilistic estimates. The conducted study opened many interesting directions for further work.

The algorithms proposed in this thesis are empirically evaluated on several stationary and non-stationary datasets for single and multi-target regression problems. The incremental algorithms were shown to perform favorably to existing batch learning algorithms, while having lower variability in their predictions due to variations in the training data. Our

change detection and adaptation methods were shown to successfully track changes in real-time and enable appropriate adaptations of the model. We have further shown that option trees improve the accuracy of standard regression trees more than ensemble learning methods without harming their robustness. At last, the comparative assessment of single target and multi-target model trees has shown that multi-target regression trees offer comparable performance to a collection of single-target model trees, while having lower complexity and better interpretability.

Povzetek

V disertaciji obravnavamo problem učenja različnih vrst odločitvenih dreves iz podatkovnih tokov, ki se spreminjajo v času. Posvetimo se predsvem študiju sprotnih (online) algoritmov strojnega učenja za učenje regresijskih dreves, linearnih modelnih dreves, opcijskih dreves za regresijo, več-ciljnih modelnih dreves in ansamblov modelnih dreves iz časovnih podatkovnih tokov. Gre za najbolj reprezentativne in pogosto uporabljene razrede modelov iz skupine interpretabilnih napovednih modelov.

Podatkovni tok je neomejeno zaporedje podatkov (števil, koordinat, večdimenzionalnih točk, n-terk ali objektov poljubnega tipa). Zanj je značilna visoka frekvenca vhodnih podatkov, katerih porazdelitve niso stacionarne. Dejanske praktične primere, v katerih potrebujemo obdelavo podatkovnih tokov, predstavljajo raznovrstni sistemi za upravljanje z mrežami senzorjev, namenjeni nadzoru učinkovitosti inteligentnih elektro-omrežij, spremljanju prometnih zastojev v mestih, ali pa znanstvenemu raziskovanju podnebnih sprememb.

Ker tovrstnih podatkov ni mogoče preprosto shranjevati ali prenašati v centralno bazo podatkov, ne da bi s tem preobremenili komunikacijsko infrastrukturo, jih je potrebno obdelovati in analizirati sproti in na mestu kjer so, ob uporabi konstantne količine pomnilnika. Pri učenju iz podatkovnih tokov je najpomembnejša naloga inkrementalno računanje nepristranskih približkov raznih statističnih mer. V ta namen potrebujemo metode, ki omogočajo implicitno zbiranje ustreznih vzorcev iz vhodnega podatkovnega toka in sproten izračun potrebnih statistik.

V disertaciji smo pristopili k problemu izračunavanja nepristranskega približka cenilne funkcije tako, da jo obravnavamo kot naključno spremenljivko. To nam je omogočilo uporabo obstoječih verjetnostnih mej, med katerimi so bili najboljši rezultati doseženi s Hoeffdingovo mejo. Algoritmi, ki jih predlagamo v disertaciji, uporabljajo Hoeffdingovo mejo verjetnosti za omejitev verjetnosti napake približka srednje vrednosti vzorca iz zaporedja naključnih spremenljivk. Ta pristop nam daje statistični mehanizem za učinkovito reševanje različnih nalog strojnega učenja, ki jih obravnavamo v disertaciji.

Z našim raziskovalnim delom se posvečamo reševanju treh glavnih podproblemov, ki jih srečamo pri učenju drevesnih modelov iz časovno spremenljivih podatkovnih tokov. Prvi podproblem zadeva nestacionarnost konceptov in potrebo po informiranem in smiselnem prilagajanju odločitvenega drevesa. V disertaciji predlagamo mehanizem za sprotno zaznavanje sprememb, ki je vključen v inkrementalno naučeni model. Drugi podproblem je kratkovidnost algoritmov za učenje odločitvenih dreves pri njihovem preiskovanju prostora možnih modelov. Tega problema se lotimo s študijo in primerjalnim vrednotenjem sprotnih opcijskih dreves za regresijo in ansamblov modelnih dreves. Predlagamo uporabo opcij za izboljšanje zmogljivosti, stabilnosti in kvalitete običajnih drevesnih modelov. Tretji problem je povezan z uporabnostjo predlaganega pristopa v nalogah več-ciljnega napovedovanja. V disertaciji predlagamo razširitev napovednega razvrščanja v smeri sprotnega učenja problemih z vključitvijo verjetnostnih približkov, ki so omejeni s Hoeffdingovo mejo. Opravljene študije so odprle mnogo zanimivih smeri za nadaljnje delo.

Algoritmi, ki jih predlagamo v disertaciji so empirično ovrednoteni na več stacionarnih in nestacionarnih zbirkah podatkov za eno- in več-ciljne regresijske probleme. Inkrementalni algoritmi so se izkazali za boljše od obstoječih algoritmov za paketno obdelavo, pri čemer so

se tudi ob variabilnosti v učnih podatkih izkazali z manj nihanji v napovedih. Naše metode za zaznavanje sprememb in prilagajanje le-tem so se izkazale za uspešne pri odkrivanju sprememb v realnem času in so omogočile primerne prilagoditve modelov. Pokazali smo tudi, da opcijska drevesa bolj izboljšajo točnost običajnih regresijskih dreves kot ansambli dreves. Zmožna so izboljšanja sposobnosti modeliranja danega problema brez izgube robustnosti. Nenazadnje, primerjalno ovrednotenje eno-ciljnih in več-ciljnih modelnih dreves je pokazalo da več-ciljna regresijska drevesa ponujajo primerljivo zmogljivost kot zbirka večjega števila eno-ciljnih dreves, vendar so obenem enostavnejša in lažje razumljiva.

Abbreviations

PAC	=	Probably Approximately Correct
VC	=	Vapnik-Chervonenkis
SPRT	=	Sequential Probability Ratio Test
SPC	=	Statistical Process Control
WSS	=	Within Sum of Squares
TSS	=	Total Sum of Squares
AID	=	Automatic Interaction Detector
MAID	=	Multivariate Automatic Interaction Detector
THAID	=	THeta Automatic Interaction Detector
CHAID	=	CHi-squared Automatic Interaction Detection
QUEST	=	Quick Unbiased Efficient Statistical Tree
LDA	=	Linear Discriminant Analysis
QDA	=	Quadratic Discriminant Analysis
SSE	=	Sum of Square Errors
EM	=	Expectation-Maximization
RSS	=	Residual Sum of Squares
TDDT	=	Top-Down Decision Tree
MTRT	=	Multi-target Regression Tree
WSSD	=	Within Sum of Squared Distances
TSSD	=	Total Sum of Squared Distances
RBF	=	Radial Basis Function
RSM	=	Random Sampling Method
MSE	=	Mean Squared Error
RE	=	Relative (mean squared) Error
RRSE	=	Root Relative (mean) Squared Error
MAE	=	Mean Absolute Error
RMAE	=	Relative Mean Absolute Error
CC	=	Correlation Coefficient
PSP	=	Protein Structure Prediction
PSSM	=	Position-Specific Scoring Matrices
IMTI	=	Incremental Multi Target Induction
RSS	=	Residual Sums of Squares
RLS	=	Recursive Least Squares
ANN	=	Artificial Neural Network
PH	=	Page-Hinkley
ST	=	Single Target
MT	=	Multiple Target

1 Introduction

First will what is necessary, then love what you will.

Tim O'Reilly

Machine Learning is the study of computer algorithms that are able to learn automatically through experience. It has become one of the most active and prolific areas of computer science research, in large part because of its wide-spread applicability to problems as diverse as natural language processing, speech recognition, spam detection, document search, computer vision, gene discovery, medical diagnosis, and robotics. Machine learning algorithms are data driven, in the sense that the success of learning relies heavily on the scope and the amount of data provided to the learning algorithm. With the growing popularity of the Internet and social networking sites (e.g., Facebook), new sources of data on the preferences, behavior, and beliefs of massive populations of users have emerged. Ubiquitous measuring elements hidden in literally every device that we use provide the opportunity to automatically gather large amounts of data. Due to these factors, the field of machine learning has developed and matured substantially, providing means to analyze different types of data and intelligently ensemble this experience to produce valuable information that can be used to leverage the quality of our lives.

1.1 Context

Predictive modeling. The broader context of the research presented in this thesis is the general predictive modeling task of machine learning, that is, the induction of models for predicting nominal (classification) or numerical (regression) target values. A model can serve as an explanatory tool to distinguish between objects of different classes in which case it falls in the category of descriptive models. When a model is primarily induced to predict the class label of unknown records then it belongs to the category of predictive models.

The predictive modeling task produces a mapping from the input space, represented with a set of descriptive attributes of various types, to the space of target attributes, represented with a set of class values or the space of real numbers. The classification model can be thus treated as a black box that automatically assigns a class label when presented with the attribute set of an unknown record. With the more recent developments in the field of machine learning, the predictive modeling task has been extended to address more complex target spaces with a predefined structure of arbitrary type. The format of the output can be a vector of numerical values, a hierarchical structure of labels, or even a graph of objects.

In the focus of this thesis are tree-based models for predicting the value of one or several numerical attributes, called targets (multi-target prediction). The term that we will use in this thesis when referring to the general category of tree-based models is decision trees. The simplest types of tree-based models are classification and regression trees. While classification trees are used to model concepts represented with symbolic categories, regression

trees are typically used to model functions defined over the space of some or all of the input attributes.

Classification and regression tree learning algorithms are among the most widely used and most popular methods for predictive modeling. A decision tree is a concise data structure, that is easily interpretable and provides meaningful descriptions of the dependencies between the input attributes and the target. Various studies and reports on their applicability have shown that regression trees are able to provide accurate predictions, if applied on adequate types of problems, that is, problems which can be represented with a set of disjunctive expressions. They can handle both numeric and nominal types of attributes, and are quite robust to irrelevant attributes.

Decision trees in general can give answers to questions of the type:

1. *Which are the most discriminative (for the task of classification) combinations of attribute values with respect to the target?*

2. *What is the average value (for the task of regression) of a given target for all the examples for which a given set of conditions on the input attributes is true?*

Decision trees have several advantages over other existing models such as support vector machines, Gaussian models, and artificial neural networks. First of all, the algorithms for learning decision trees are distribution-free, that is, they have no special prior assumptions on the distribution that governs the data. Second, they do not require tuning of parameters or heavy tedious training, as in the case for support vector machines. Finally, the most important advantage of decision trees is the fact that they are easily interpretable by a human user. Every decision tree can be represented with a set of rules that describe the dependencies between the input attributes and the target attribute.

Within the category of decision trees fall structured output prediction trees (Blockeel et al., 1998; Vens et al., 2008) which are able to provide predictions of a more complex type. Among the most popular types of models that fall in this sub-category are multi-label and multi-target classification and regression trees (Blockeel et al., 1998). In this thesis, we have studied only multi-target regression trees which predict the values of multiple numerical targets. However, most of our ideas can be also extended to the case of multi-label classification trees.

A classification or regression tree is a single model induced to be consistent with the available training data. The process of tree induction is typically characterized with a limited lookahead, instability, and high sensitivity to the choice of training data. Due to the fact that the final result is a single model, classification and regression trees are unable to inform the potential users about how many alternative models are consistent with the given training data. This issue has been addressed with the development of option trees, which represent multiple models compressed in a single interpretable decision tree. Better exploration of the space of possible models can be achieved by learning ensembles of models (homogeneous or heterogeneous). In this thesis, we have considered both option trees for regression and ensembles of regression trees, which have complementary characteristics, i.e., advantages and disadvantages.

Mining data streams. Continuous streams of measurements are typically found in finance, environmental or industrial monitoring, network management and many others (Muthukrishnan, 2005). Their main characteristic is the continuous and possibly unbounded arrival of data items at high rates. The opportunity to continuously gather information from myriads of sources, however, proves to be both a blessing and a burden. The continuous arrival of data demands algorithms that are able to process new data instances in constant time in the order of their arrival. The temporal dimension, on the other hand, implies possible changes in the concept or the functional dependencies being modeled, which, in the field of machine learning is known as concept drift (Kolter and Maloof, 2005; Widmer and

Kubat, 1996). Thus, the algorithms for learning from data streams need to be able to detect the appearance of concept drift and adapt their model correspondingly.

Among the most interesting research areas of machine learning is online learning, which deals with machine learning algorithms able to induce models from continuous data feeds (data streams). Online algorithms are algorithms that process their input piece-by-piece in a serial fashion, i.e., in the order in which the input is fed to the algorithm, without having the entire input available from the start. Every piece of input is used by the online algorithm to update and improve the current model. Given their ability to return a valid solution to a problem, even if interrupted at any time before their ending, online algorithms are regarded as *any-time*. However, the algorithm is expected to find better and better solutions the longer it runs. An *adaptive* learning algorithm is an algorithm which is able to adapt its inference of models when observed evidence, conditional probabilities, and the structure of the dependencies change or *evolve* over time. Because of these features, online algorithms are the method of choice if emerging data must be processed in a real-time manner without completely storing it. In addition, these algorithms can be used for processing data stored on large external memory devices, because they are able to induce a model or a hypothesis using only a single pass over the data, orders of magnitudes faster as compared to traditional batch learning algorithms.

Learning decision trees from data streams. This thesis is concerned with algorithms for learning decision trees from data streams. Learning decision trees from data streams is a challenging problem, due to the fact that all tree learning algorithms perform a simple-to-complex, hill-climbing search of a complete hypothesis space for the *first* tree that fits the training examples. Among the most successful algorithms for learning classification trees from data streams are Hoeffding trees (Domingos and Hulten, 2000), which are able to induce a decision model in an incremental manner by incorporating new information at the time of its arrival.

Hoeffding trees provide theoretical guarantees for convergence to a hypothetical model learned by a batch algorithm by using the Hoeffding probability bound. Having a predefined range for the values of the random variables, the Hoeffding probability bound (Hoeffding, 1963) can be used to obtain tight confidence intervals for the true average of the sequence of random variables. The probability bound enables one to state, with some predetermined confidence, that the sample average for N random i.i.d. variables with values in a constrained range is within distance ε of the true mean. The value of ε monotonically decreases with the number of observations N, or in other words, by observing more and more values, the sampled mean approaches the true mean.

The problem of concept drift and change detection, one of the essential issues in learning from data streams, has also received proper attention from the research community. Several change detection methods which function either as wrappers (Gama and Castillo, 2006) or are incorporated within the machine learning algorithm (Bifet and Gavaldà, 2007) have been proposed. Hoeffding-based algorithms extended to the non-stationary online learning setup have been also studied by multiple authors (Gama et al., 2004b, 2003; Hulten et al., 2001).

Hoeffding-based option trees (Pfahringer et al., 2007) and their variants (*Adaptive Hoeffding Option Trees*) (Bifet et al., 2009b) have been studied in the context of improving the accuracy of online classification. Various types of ensembles of Hoeffding trees for online classification, including bagging and boosting (Bifet et al., 2009b), random forests (Abdulsalam et al., 2007, 2008; Bifet et al., 2010; Li et al., 2010), and stacked generalization with restricted Hoeffding trees (Bifet et al., 2012), among others, have been proposed and studied. Finally, Read et al. (2012) recently proposed an algorithm for learning multi-label Hoeffding trees which attacks the multi-label problem through online learning of multi-label (classification) trees.

The work in this thesis falls in the more specific context of algorithms for learning Hoeffding-based regression trees, algorithms for change detection, and extensions of these

concepts to more advanced and complex types of models, i.e., option trees, tree ensembles, and multi-target trees for regression.

1.2 Goals

The main goal of our research was to study the various aspects of the problem of learning decision trees from data streams that evolve over time, that is, in a learning environment in which the underlying distribution that generates the data might change over time. Decision trees for regression have not been studied yet in the context of data streams, despite the fact that many interesting real-world applications require various regression tasks to be solved in an online manner.

Within this study, we aimed to develop various tree-based methods for regression on time-changing data streams. Our goal was to follow the main developments within the line of algorithms for online learning of classification trees from data streams, that is, include change detection mechanisms inside the tree learning algorithms, introduce options in the trees, and extend the developed methods to online learning of ensemble models for regression and trees for structured output prediction.

1.3 Methodology

Our approach is to follow the well established line of algorithms for online learning of decision trees, represented with the Hoeffding tree learning algorithm (Domingos and Hulten, 2000). Hoeffding trees take the viewpoint of statistical learning by making use of probabilistic estimates within the inductive inference process. The application of the Hoeffding bound provides a statistical support for every inductive decision (e.g., selection of a test to put in an internal node of the tree), which results in a more stable and robust sequence of inductive decisions. Our goal was to apply the same ideas in the context of online learning of regression trees.

Our methodology examines the applicability of the Hoeffding bound to the split selection procedure of an online algorithm for learning regression trees. Due to the specifics of the Hoeffding bound, extending the same ideas to the regression domain is not straightforward. Namely, there exist no such evaluation function for the regression domain whose values can be bounded within a pre-specified range.

To address the issue of change detection, we studied methods for tracking changes and real-time adaptation of the current model. Our methodology is to introduce change detection mechanisms within the learning algorithm and enable local error monitoring. The advantage of localizing the concept drift is that it gives us the possibility to determine the set of conditions under which the current model remains valid, and more importantly the set of disjunctive expressions that have become incorrect due to the changes in the functional dependencies.

The online learning task carried out by Hoeffding-based algorithms, although statistically stable, is still susceptible to the typical problems of greedy search through the space of possible trees. In that context, we study the applicability of options and their effect on the any-time performance of the online learning algorithm. The goal is not only to improve the exploration of the search space, but also to enable more efficient resolution of ambiguous situations which typically slow down the convergence of the learning process.

We further study the possibility to combine multiple predictions which promises an increased accuracy, along with a reduced variability and sensitivity to the choice of the training data. A natural step in this direction is to study the relation between option trees and ensemble learning methods which offer possibilities to leverage the expertise of multiple regression trees. We approach it with a comparative assessment of online option trees for

regression and online ensembles of regression trees. We evaluate the algorithms proposed on real-world and synthetic data sets, using methodology appropriate for approaches for learning from evolving data streams.

1.4 Contributions

The research presented in this thesis addresses the general problem of automated and adaptive any-time regression analysis using different regression tree approaches and tree-based ensembles from streaming data. The main contributions of the thesis are summarized as follows:

- We have designed and implemented an online algorithm for learning model trees with change detection and adaptation mechanisms embedded within the algorithm. To the best of our knowledge, this is the first approach that studies a complete system for learning from non-stationary distributions for the task of *online regression*. We have performed an extensive empirical evaluation of the proposed change detection and adaptation methods on several simulated scenarios of concept drift, as well as on the task of predicting flight delays from a large dataset of departure and arrival records collected within a period of twenty years.
- We have designed and implemented an online option tree learning algorithm that enabled us to study the idea of introducing options within the proposed online learning algorithm and their overall effect on the learning process. To the best of our knowledge, this is the first algorithm for learning option trees in the online setup without capping options to the existing nodes. We have further performed a corresponding empirical evaluation and a comparison of the novel online option tree learning algorithm with the baseline regression and model tree learning algorithms.
- We have designed and implemented two methods for learning tree-based ensembles for regression. These two methods were developed to study the advantages of combining multiple predictions for online regression and to evaluate the merit of using options in the context of methods for learning ensembles. We have performed a corresponding empirical evaluation and a comparison with the online option tree learning algorithm on existing real-world benchmark datasets.
- We have designed and implemented a novel online algorithm for learning multiple target regression and model trees. To the best of our knowledge, this is the first algorithm designed to address the problem of online prediction of multiple numerical targets for regression analysis. We have performed a corresponding empirical evaluation and a comparison with an independent modeling approach. We have also included a batch algorithm for learning multi-target regression trees in the comparative assessment of the quality of the models induced with the online learning algorithm.

To this date and to the best of our knowledge there is no other work that implements and empirically evaluates online methods for tree-based regression, including model trees with drift detection, option trees for regression, online ensemble methods for regression, and online multi-target model trees. With the work presented in this thesis we lay the foundations for research in online tree-based regression, leaving much room for future improvements, extensions and comparisons of methods.

1.5 Organization of the Thesis

This introductory chapter presents the general perspective on the topic under study and provides the motivation for our research. It specifies the goals set at the beginning of the

thesis research and presents its main original contributions. In the following, we give a chapter level outline of this thesis, describing the organization of the chapters which present the above mentioned contributions.

Chapter 2 gives the broader context of the thesis work within the area of online learning from the viewpoint of several areas, including statistical quality control, decision theory, and computational learning theory. It gives some background on the online learning protocol and a brief overview of the research related to the problem of supervised learning from time-changing data streams.

Chapter 3 gives the necessary background on the basic methodology for learning decision trees, regression trees and their variants including model trees, option trees and multi-target decision trees. It provides a description of the tree induction task through a short presentation of the history of learning decision trees, followed by a more elaborate discussion of the main issues.

Chapter 4 provides a description of several basic ensemble learning methods with a focus on ensembles of homogeneous models, such as regression or model trees. It provides the basic intuition behind the idea of learning ensembles, which is related to one of the main contributions of this thesis that stems from the exploration of options in the tree induction.

In Chapter 5, we present the quality measures used and the specifics of the experimental evaluation designed to assess the performance of our online algorithms. This chapter defines the main criteria for evaluation along with two general evaluation models designed specially for the online learning setup. In this chapter, we also give a description of the methods used to perform a statistical comparative assessment and an online bias-variance decomposition. In addition, we describe the various real-world and simulated problems which were used in the experimental evaluation of the different learning algorithms.

Chapter 6 presents the first major contribution of the thesis. It describes an online change detection and adaptation mechanism embedded within an online algorithm for learning model trees. It starts with a discussion on the related work within the online sequential hypothesis testing framework and the existing probabilistic sampling strategies in machine learning. The main parts of the algorithm are further presented in more detail, each in a separate section. Finally, we give an extensive empirical evaluation addressing the various aspects of the online learning procedure.

Chapter 7 presents the second major contribution of the thesis, an algorithm for online learning of option trees for regression. The chapter covers the related work on learning Hoeffding option trees for classification and presents the main parts of the algorithm, each in a separate section. The last section contains an empirical evaluation of the proposed algorithm on the same benchmark regression problems that were used in the evaluation section of Chapter 5.

Chapter 8 provides an extensive overview of existing methods for learning ensembles of classifiers for the online prediction task. This gives the appropriate context for the two newly designed ensemble learning methods for online regression, which are based on extensions of the algorithm described previously in Chapter 6. This chapter also presents an extensive experimental comparison of the ensemble learning methods with the online option tree learning algorithm introduced in Chapter 7.

Chapter 9 presents our final contribution, with which we have addressed a slightly different aspect of the online prediction task, i.e., the increase in the complexity of the space of the target variables. The chapter starts with a short overview of the existing related work in the context of the online multi-target prediction task. Next, it describes an algorithm for learning multi-target model trees from data streams through a detailed elaboration of the main procedures. An experimental evaluation is provided to support our theoretical results that continues into a discussion of some interesting directions for further extensions.

Finally, Chapter 10 presents our conclusions. It presents a summary of the thesis, our original contributions, and several directions for further work.

2 Learning from Data Streams

When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong.

The First Clarke's Law by Arthur C. Clarke, in "Hazards of Prophecy: The Failure of Imagination"

Learning from abundant data has been mainly motivated by the explosive growth of information collected and stored electronically. It represents a major departure from the traditional inductive inference paradigm in which the main bottleneck is the lack of training data. With the abundance of data however, the main question which has been frequently addressed in the different research communities so far is: "*What is the minimum amount of data that can be used without compromising the results of learning?*". In this chapter, we discuss various aspects of the problem of supervised learning from data streams, and strive to provide a unified view from the perspectives of statistical quality control, decision theory, and computational learning theory.

This chapter is organized as follows. We start with a high-level overview on the requirements for online learning. Next, we discuss the supervised learning and regression tasks. We propose to study the process of learning as a search in the solution space, and show more specifically how each move in this space can be chosen by using a sub-sample of the training data. In that context we discuss various sampling strategies for determining the amount of necessary training data. This is related to the concept of Probably Approximately Correct learning (PAC learning) and sequential inductive learning. We also present some of the most representative online learning algorithms. In a final note, we address the non-stationarity of the learning process and discuss several approaches for resolving the issues raised in this context.

2.1 Overview

Learning from data streams is an instance of the online learning paradigm. It differs from the batch learning process mainly by its ability to incorporate new information into the existing model, without having to re-learn it from scratch. Batch learning is a finite process that starts with a data collection phase and ends with a model (or a set of models) typically after the data has been maximally explored. The induced model represents a stationary distribution, a concept, or a function which is not expected to change in the near future. The online learning process, on the other hand, is not finite. It starts with the arrival of some training instances and lasts as long as there is new data available for learning. As such, it is a dynamic process that has to encapsulate the collection of data, the learning and the validation phase in a single continuous cycle.

Research in online learning dates back to second half of the previous century, when the Perceptron algorithm has been introduced by Rosenblatt (1958). However, the online machine learning community has been mainly preoccupied with finding theoretical guarantees for the learning performance of online algorithms, while neglecting some more practical issues. The process of learning itself is a very difficult task. Its success depends mainly on the type of problems being considered, and on the quality of available data that will be used in the inference process. Real-world problems are typically very complex and demand a diverse set of data that covers various aspects of the problem, as well as, sophisticated mechanisms for coping with noise and contradictory information. As a result, it becomes almost impossible to derive theoretical guarantees on the performance of online learning algorithms for practical real-world problems. This has been the main reason for the decreased popularity of online machine learning.

The stream data mining community, on the other hand, has approached the online learning problem from a more practical perspective. Stream data mining algorithms are typically designed such that they fulfill a list of requirements in order to ensure efficient online learning. Learning from data streams not only has to incrementally induce a model of a good quality, but this has to be done efficiently, while taking into account the possibility that the conditional dependencies can change over time. Hulten et al. (2001) have identified several desirable properties a learning algorithm has to posses in order to efficiently induce up-to-date models from high-volume, open-ended data streams. An online streaming algorithm has to possess the following features:

- It should be able to build a decision model using a single-pass over the data;
- It should have a small (if possible constant) processing time per example;
- It should use a fixed amount of memory; irrespective of the data stream size;
- It should be able to incorporate new information in the existing model;
- It should have the ability to deal with concept drift; and
- It should have a high speed of convergence;

We tried to take into account all of these requirements when designing our online adaptive algorithms for learning regression trees and their variants, as well as, for learning ensembles of regression and model trees.

2.2 Supervised Learning and the Regression Task

Informally speaking, the inductive inference task aims to construct or evaluate propositions that are abstractions of observations of individual *instances* of members of the same class. Machine learning, in particular, studies automated methods for inducing general functions from specific examples sampled from an unknown data distribution. In its most simple form, the inductive learning task ignores prior knowledge, assumes a deterministic, observable environment, and assumes that examples are given to the learning agent (Mitchell, 1997). The learning task is in general categorized as either *supervised* or *un-supervised* learning. We will consider only the *supervised* learning task, more specifically supervised learning of various forms of tree structured models for *regression*.

2.2.1 The Task of Regression

Before stating the basic definitions, we will define some terminology that will be used throughout this thesis. Suppose we have a set of objects, each described with many attributes (features or properties). The attributes are independent observable variables, numerical or nominal. Each object can be assigned a single real-valued number, i.e., a value

of the dependent (target) variable, which is a function of the independent variables. Thus, the input data for a learning task is a collection of records. Each record, also known as an instance or an example is characterized by a tuple ($\mathbf{x}$,y), where $\mathbf{x}$ is the attribute set and y is the target attribute, designated as the class label. If the class label is a discrete attribute then the learning task is classification. If the class label is a continuous attribute then the learning task is regression. In other words, the task of regression is to determine the value of the dependent continuous variable, given the values of the independent variables (the attribute set).

When learning the target function, the learner L is presented with a set of *training examples*, each consisting of an input vector $\mathbf{x}$ from X, along with its target function value $y = f(x)$. The function to be learned represents a mapping from the attribute space X to the space of real values Y, i.e., $f : X \rightarrow \mathscr{R}$. We assume that the training examples are generated at random according to some probability distribution D. In general, D can be any distribution and is not known to the learner. Given a set of training examples of the target function f, the problem faced by the learner is to hypothesize, or estimate, f. We use the symbol H to denote the set of all possible hypotheses that the learner may consider when trying to find the true identity of the target function. In our case, H is determined by the set of all possible regression trees (or variants thereof, such as option trees) over the instance space X. After observing a set of training examples of the target function f, L must output some hypothesis h from H, which is its estimate of f. A fair evaluation of the success of L assesses the performance of h over a set of new instances drawn randomly from X,Y according to D, the same probability distribution used to generate the training data.

The basic assumption of inductive learning is that: *Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over unobserved testing examples.* The rationale behind this assumption is that the only information available about f is its value over the set of training examples. Therefore, inductive learning algorithms can at best guarantee that the output hypothesis fits the target function over the training data. However, this fundamental assumption of inductive learning needs to be re-examined under the learning setup of changing data streams, where a target function f might be valid over the set of training examples only for a fixed amount of time. In this new setup which places the afore assumption under a magnifying glass, the inductive learning task has to incorporate machinery for dealing with *non-stationary* functional dependencies and a possibly *infinite* set of training examples.

2.2.2 Learning as Search

The problem of learning a target function has been typically viewed as a problem of search through a large space of hypotheses, implicitly defined by the representation of hypotheses. According to the previous definition, the goal of this search is to find the hypothesis that best fits the training examples. By viewing the learning problem as a problem of search, it is natural to approach the problem of designing a learning algorithm through examining different strategies for searching the hypothesis space. We are, in particular, interested in algorithms that can perform efficient search of a very large (or infinite) hypothesis space through a sequence of decisions, each informed by a statistically significant amount of evidence, through the application of probability bounds or variants of statistical tests. In that context, we address algorithms that are able to make use of the general-to-specific ordering of the set of all possible hypothesis. By taking the advantage of the general-to-specific ordering, the learning algorithm can search the hypothesis space without explicitly enumerating every hypothesis. In the context of regression trees, each conjunction of a test to the previously inferred conditions represents a refinement of the current hypothesis.

The inductive bias of a learner is given with the choice of hypothesis space and the set of assumptions that the learner makes while searching the hypothesis space. As stated by

Mitchell (1997), there is a clear futility in bias-free learning: *a learner that makes no prior assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.* Thus, learning algorithms make various assumptions, ranging from "the hypothesis space H includes the target concept" to "more specific hypotheses are preferred over more general hypotheses". Without an inductive bias, a learner cannot make inductive leaps to classify unseen examples. One of the advantages of studying the inductive bias of a learner is in that it provides means of characterizing their policy for generalizing beyond the observed data. A second advantage is that it allows comparison of different learning algorithms according to the strength of their inductive bias.

Decision and regression trees are learned from a *finite* set of examples based on the available attributes. The inductive bias of decision and regression tree learning algorithms will be discussed in more detail in the following chapter. We note here that there exist a clear difference between the induction task using a finite set of training examples and the induction task using an infinite set of training examples. In the first case, the learning algorithm uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis. Several interesting questions arise when the set of instances X is not finite. Since only a portion of all instances is available when making each decision, one might expect that earlier decisions are doomed to be improperly informed, due to the lack of information that only becomes available with the arrival of new training instances. In what follows, we will try to clarify some of those questions from the perspective of statistical decision theory.

2.3 Learning under a Sampling Strategy

Let us start with a short discussion of the practical issues which arise when applying a machine learning algorithm for knowledge discovery. One of the most common issues is the fact that the data may contain noise. The existence of noise increases the difficulty of the learning task. To go even further, the concept one is trying to learn may not be drawn from a pre-specified class of known concepts. Also, the attributes may be insufficient to describe the target function or concept.

The problem of learnability and complexity of learning has been studied in the field of computational learning theory. There, a standard definition of sufficiency regarding the quality of the learned hypothesis is used. Computational learning theory is in general concerned with two types of questions: "How many training examples are sufficient to successfully learn the target function?" and "How many mistakes will the learner make before succeeding?".

2.3.1 Probably Approximately Correct Learning

The definition of "success" depends largely on the context, the particular setting, or the learning model we have in mind. There are several attributes of the learning problem that determine whether it is possible to give quantitative answers to the above questions. These include the complexity of the hypothesis space considered by the learner, the accuracy to which the target function must be approximated, the probability that the learner will output a successful hypothesis, or the manner in which the training examples are presented to the learner.

The *probably approximately correct* (PAC) learning model proposed by Valiant (1984) provides the means to analyze the sample and computational complexity of learning problems for which the hypothesis space H is finite. In particular, learnability is defined in terms of how closely the target concept can be approximated (under the assumed set of hypotheses H) from a reasonable number of randomly drawn training examples with a reasonable amount of computation. Trying to characterize learnability by demanding an error rate of $error_D(h) = 0$ when applying h on future instances drawn according to the probability

distribution D is unrealistic, for two reasons. First, since we are not able to provide to the learner all of the training examples from the instance space X, there may be multiple hypotheses which are consistent with the provided set of training examples, and the learner cannot deterministically pick the one that corresponds to the target function. Second, given that the training examples are drawn at random from the unknown distribution D, there will always be some nonzero probability that the chosen sequence of training example is misleading.

According to the PAC model, to be able to eventually learn something, we must weaken our demands on the learner in two ways. First, we must give up on the zero error requirement, and settle for an approximation defined by a constant error bound ε, that can be made arbitrarily small. Second, we will not require that the learner must succeed in achieving this approximation for *every* possible sequence of randomly drawn training examples, but we will require that its probability of failure be bounded by some constant, δ, that can be made arbitrarily small. In other words, we will require only that the learner *probably* learns a hypothesis that is *approximately* correct. The definition of a **PAC-learnable** concept class is given as follows:

> Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is **PAC-learnable** by L using H if for all $c \in C$, distributions D over X, ε such that $0 < \varepsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1-\delta)$ output a hypothesis $h \in H$ such that $error_D(h) \leq \varepsilon$, in time that is polynomial in $1/\varepsilon$, $1/\delta$, n, and $size(c)$.

The definition takes into account our demands on the output hypothesis: low error (ε) high probability ($1-\delta$), as well as the complexity of the underlying instance space n and the concept class C. Here, n is the size of instances in X (e.g., the number of independent variables).

However, the above definition of PAC learnability implicitly assumes that the learner's hypothesis space H contains a hypothesis with arbitrarily small error ε for every target concept in C. In many practical real world problems it is very difficult to determine C in advance. For that reason, the framework of *agnostic learning* (Haussler, 1992; Kearns et al., 1994) weakens the demands even further, asking for the learner to output the hypothesis from H that has the minimum error over the training examples. This type of learning is called agnostic because the learner makes no assumption that the target concept or function is representable in H; that is, it doesn't know if $C \subseteq H$. Under this less restrictive setup, the learner is assured with probability $(1-\delta)$ to output a hypothesis within error ε of the best possible hypothesis in H, after observing m randomly drawn training examples, provided

$$m \geq \frac{1}{2\varepsilon^2}(\ln(1/\delta) + \ln|H|). \tag{1}$$

As we can see, the number of examples required to reach the goal of close approximation depends on the complexity of the hypothesis space H, which in the case of decision and regression trees and other similar types of models can be *infinite*. For the case of infinite hypothesis spaces, a different measure of the complexity of H is used, called the Vapnik-Chervonenkis dimension of H (VC dimension, or $VC(H)$ for short). However, the bounds derived are applicable only to some rather simple learning problems for which it is possible to determine the $VC(H)$ dimension. For example, it can be shown that the VC dimension of linear decision surfaces in an r dimensional space (i.e., the VC dimension of a perceptron with r inputs) is $r+1$, or for some other well defined classes of more complex models, such as neural networks with predefined units and structure. Nevertheless, the above considerations have lead to several important ideas which have influenced some recent, more practical, solutions.

An example is the application of general Hoeffding bounds (Hoeffding, 1963), also known as additive Chernoff bounds, in estimating how badly a single chosen hypothesis deviates from the best one in H. The Hoeffding bound applies to experiments involving a number of distinct Bernoullli trials, such as m independent flips of a coin with some probability of turning up heads. The event of a coin turning up heads can be associated with the event of a misclassification. Thus, a sequence of m independent coin flips is analogous to a sequence of m independently drawn instances. Generally speaking, the Hoeffding bound characterizes the deviation between the true probability of some event and its observed frequency over m independent trials. In that sense, it can be used to estimate the deviation between the true probability of misclassification of a learner and its observed error over a sequence of m independently drawn instances.

The Hoeffding inequality gives a bound on the probability that an arbitrarily chosen single hypothesis h has a training error, measured over set D' containing m randomly drawn examples from the distribution D, that deviates from the true error by more than ε.

$$Pr[error_{D'}(h) > error_D(h) + \varepsilon] \leq e^{-2m\varepsilon^2}$$

To ensure that the *best* hypothesis found by L has an error bounded by ε, we must bound the probability that the error of any hypothesis in H will deviate from its true value by more than ε as follows:

$$Pr[(\forall h \in H)(error_{D'}(h) > error_D(h) + \varepsilon)] \leq |H|e^{-2m\varepsilon^2}$$

If we assign a value of δ to this probability and ask how many examples are necessary for the inequality to hold we get:

$$m \geq \frac{1}{2\varepsilon^2}(\ln|H| + \ln(1/\delta)) \tag{2}$$

The number of examples depends logarithmically on the inverse of the desired probability $1/\delta$, and grows with the square of $1/\varepsilon$. Although this is just one example of how the Hoeffding bound can be applied, it illustrates the type of approximate answer which can be obtained in the scenario of learning from data streams. It is important to note at this point that this application of the Hoeffding bound ensures that a hypothesis (from the finite space H)with the desired accuracy will be found with high probability. However, for a large number of practical problems, for which the hypothesis space H is *infinite*, similar bounds cannot be derived even if we use the $VC(H)$ dimension instead of $|H|$. Instead, several approaches have been proposed that relax the demands for these difficult cases even further, while assuring that each inductive decision will satisfy a desired level of quality.

2.3.2 Sequential Inductive Learning

Interpreting the inductive inference process as search through the hypothesis space H enables the use of several interesting ideas from the field of statistical decision theory. We are interested in algorithms that are able to make use of the general-to-specific ordering of the hypotheses in H, and thus perform a move in the search space by refining an existing more general hypothesis into a new, more specific one. The choice of the next move requires the examination of a set of refinements, from which the *best* one will be chosen.

Most learning algorithms use some statistical procedure for evaluating the merit of each refinement, which in the field of statistics has been studied as the correlated selection problem (Gratch, 1994). In selection problems, one is basically interested in comparing a finite set of hypotheses in terms of their expected performance over a distribution of instances and selecting the hypothesis with the highest expected performance. The expected performance of a hypothesis is typically defined in terms of the decision-theoretic notion of expected utility.

In machine learning, the commonly used utility functions are defined with respect to the target concept c or the target function f which the learner is trying to estimate. For classification tasks, the *true error* of a hypothesis h with respect to a target concept c and a distribution D is defined as the probability that h will misclassify an instance drawn at random according to D:

$$error_D(h) \equiv Pr_{x \in D}[c(x) \neq h(x)]$$

where the notation $P_{x \in D}$ indicates that the probability is taken over the instance distribution D and not over the actual set of training examples. This is necessary because we need to estimate the performance of the hypothesis when applied on future instances drawn independently from D.

Obviously, the error or the utility of the hypothesis depends strongly on the unknown probability distribution D. For example, if D happens to assign very low probability to instances for which h and c disagree, the error might be much smaller compared to the case of a uniform probability distribution that assigns the same probability to every instance in X. The error of h with respect to c or f is not directly observable to the learner. Thus, L can observe the performance of h over the training examples and must choose its output hypothesis on this basis only.

When data are abundant, evaluating a set of hypotheses seems trivial, unless one takes into account the computational complexity of the task. Under this constraint, one has to provide an answer to the question: *"How likely is that the estimated advantage of one hypothesis over another will remain truthful if more training examples were used?"*.

As noted in the previous section, bounding the probability of failure in finding a hypothesis which is within an ε bound of the best one in H depends on the complexity of the assumed set of hypotheses and the learning setup. However, the theoretical implications are (most of the time) not useful in practice. Classical statistical approaches typically try to assume a specific probability distribution and bound the probability of an incorrect assertion by using the initial assumptions. For example, most techniques assume that the utility of hypotheses is normally distributed, which is not an unreasonable assumption when the conditions for applying the central limit theorem hold. Other approaches relax the assumptions, e.g., assume that the selected hypothesis has the highest expected utility with some pre specified *confidence.* An even less restrictive assumption is that the selected hypothesis is close to the *best* with some *confidence.* The last assumption leads to a class of selection problems known in the field of statistics as *indifference-zone selection* (Bechhofer, 1954).

Unfortunately, given a single reasonable selection assumption, there is no single optimal method for ensuring it. Rather, there exist a variety of techniques, each with its own set of tradeoffs. In order to support efficient and practical learning, a sequential decision-theoretic approach can be used that relaxes the requirements for successful learning by moving from the goal of *"converging to a successful hypothesis"* to the goal of *"successfully converging to the closest possible hypothesis to the best one"*. In this context, there are some interesting cases of learning by computing a required sample size needed to bound the expected loss in each step of the induction process.

A notable example that has served as an inspiration is the work by Musick et al. (1993), in which a decision-theoretic subsampling has been proposed for the induction of decision trees on large databases. The main idea is to choose a smaller sample, from a very large training set, over which a tree of a desired quality would be learned. In short, the method tries to determine what sequence of subsamples from a large dataset will be the most economical way to choose the best attribute, to within a specified expected error. The sampling strategy proposed takes into account the expected quality of the learned tree, the cost of sampling, and a utility measure specifying what the user is willing to pay for different quality trees, and calculates the expected required sample size. A generalization of this method has been proposed by Gratch (1994), in which the so called *one-shot* induction is replaced with

sequential induction, where the data are sampled a little at a time throughout the decision process.

In the *sequential induction* scenario, the learning process is defined as an inductive decision process consisting of two types of inductive decisions: *stopping* decisions and *selection* decisions. The statistical machinery used to determine the sufficient amount of samples for performing the selection decisions is based on an open, unbalanced sequential strategy for solving correlated selection problems. The attribute selection problem is, in this case, addressed through a method of multiple comparisons, which consists of simultaneously performing a number of pairwise statistical comparisons between the refinements drawn from the set of possible refinements of an existing hypothesis. Let the size of this set be k. This reduces the problem to estimating the sign of the expected difference in value between the two refinements, with error no more than ε. Here, ε is an *indifference* parameter, that captures the intuition that, if the difference is sufficiently small we do not care if the technique determines its sign incorrectly. Stopping decisions are resolved using an estimate of the probability that an example would reach a particular node. The sequential algorithm should not partition a node if this probability is less then some threshold parameter γ. This decision should be, however, reached with a probability of success $(1-\delta)$.

The technique used to determine the amount of training examples necessary for achieving a successful *indifference-zone* selection takes into account the variance in the utility of each attribute. If the utility of a splitting test varies highly across the distribution of examples, more data is needed to estimate its performance to a given level of accuracy. The statistical procedure used is known as the sequential probability ratio test (SPRT); cf. Wald (1945). SPRT is based on estimating the likelihood of the data generated according to some specified distribution at two different values for the unknown mean, θ and $-\theta$. In this case, the assumption is that the observed differences are generated according to a normal distribution with mean ε, and a variance estimated with the current sample variance. A hypothesis is the overall best if there is a statistically significant positive difference in its comparison with the $k-1$ remaining hypotheses. Therefore, a refinement would be selected only when enough statistical evidence has been observed from the sequence of training examples.

With the combination of these two techniques, the induction process is designed as a **sequence** of *probably approximately correct* inductive decisions, instead of *probably approximately correct* learning. As a result, the inductive process will not guarantee that the learner will output a hypothesis which is close enough to the best one in H with a probability of success $(1-\delta)$. What will be guaranteed is that, given the learning setup, each inductive decision of the learner will have a probability of failure bounded with δ in estimating the advantage of the selected refinement over the rest with an absolute error of at most ε. A similar technique for PAC refinement selection has been proposed by Domingos and Hulten (2000) that employs the Hoeffding inequality in order to bound the probability of failure. The approach is closely related to the algorithms proposed in this thesis and will be discussed in more detail in Chapter 6.

2.4 The Online Learning Protocol

The learning scenario assumed while designing the algorithms and the experiments presented in this thesis follows the online learning protocol (Blum and Burch, 2000). Let us consider a sequence of input elements a_1, a_2, ..., a_j,... which arrive continuously and endlessly, each drawn independently from some unknown distribution D. In this setting, the following online learning protocol is repeated indefinitely:

1. The algorithm receives an unlabeled example.
2. The algorithm predicts a class (for classification) or a numerical value (for regression) of this example.

3. The algorithm is then given the correct answer (label for the unlabeled example).

An execution of steps (1) to (3) is called a trial. We will call whatever is used to perform step (2), the algorithm's "current hypothesis". New examples are classified automatically as they become available, and can be used for training as soon as their class assignments are confirmed or corrected. For example, a robot learning to complete a particular task might obtain the outcome of its action (correct or wrong) each time it attempts to perform it.

An important detail in this learning protocol is that the learner has to make a prediction after every testing and training example it receives. This is typical for the *mistake bound model* of learning, in which the learner is evaluated by the total number of mistakes it makes before it converges to the correct hypothesis. The main question considered in this model is "*What is the number of mistakes in prediction that the learner will make before it learns the target concept?*". This question asks for an estimation of the predictive accuracy of the learner at any time during the course of learning, which is significant in practical settings where learning must be done while the system is in use, rather than during an off-line training phase. If an algorithm has the property that, for any target concept $c \in C$, it makes at most $poly(p, size(c))$ mistakes on any sequence of examples, and its running time per trial is $poly(p, size(c))$ as well, then it is said that the algorithm learns class C in the mistake bound model. Here p denotes the cardinality of the problem, that is, the number of predictor variables $\{x_1, ..., x_p\}$.

2.4.1 The Perceptron and the Winnow Algorithms

Examples of simple algorithms that perform surprisingly well in practice under the *mistake bound model* are the Perceptron (Rosenblatt, 1958), and the Winnow (Littlestone, 1988), algorithms, which both perform online learning of a linear threshold function. The Perceptron algorithm is one of the oldest online machine learning algorithms for learning a linear threshold function. For a sequence S of labeled examples which is assumed to be consistent with a linear threshold function $\mathbf{w}^* \cdot x > 0$, where $\mathbf{w}^*$ is a unit-length vector, it can be proven the number of mistakes on S made by the Perceptron algorithm is at most $(1/\gamma)^2$, where

$$\gamma = \min_{x \in S} \frac{\mathbf{w}^* x}{||x||}.$$

The parameter "γ" is often called the margin of $\mathbf{w}^*$ and denotes the closest the Perceptron algorithm can get in approximating the true linear threshold function $\mathbf{w}^* \cdot x > 0$. The Perceptron algorithm is given with the following simple sequence of rules:

1. Initialize the iteration with $t = 1$.
2. Start with an all-zeros weight vector $\mathbf{w}_1 = 0$, and assume that all examples are normalized to have Euclidean length 1.
3. Given example x, predict positive iff $\mathbf{w}_t \cdot x > 0$.
4. On a mistake update the weights as follows:
   - If mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + x$.
   - If mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - x$.
5. $t \leftarrow t + 1$.
6. Go to 3.

In other words, if me make a mistake on a positive example then the weights will be updated to move closer to the positive side of the plane, and similarly if we make a mistake on a negative example then again the weights will be decreased to move closer to the value we wanted. The success of applying the Perceptron algorithm depends naturally on the data. If the data is well linearly-separated then we can expect that $\gamma \gg 1/n$, where n is the size of the sequence of examples S. In the worst case, γ can be exponentially small in n, which means that the number of mistakes made over the total sequence will be large. However, the nice property of the mistake-bound is that it is independent on the number of features in the input feature space, and depends purely on a geometric quantity. Thus, if data is linearly separable by a large margin, then the Perceptron is the right algorithm to use. If the data doesn't have a linear separator, then one can apply the kernel trick by mapping the data to a higher dimensional space, in a hope that it might be linearly separable there.

The Winnow algorithm similarly learns monotone disjunctions (e.g., $h = x_1 \vee x_2 \vee ... \vee x_p$) in the mistake bound model and makes only $O(r \log p)$ mistakes, where r is the number of variables that actually appear in the target disjunction. This algorithm can also be used to track a target concept that changes over time. This algorithm is highly efficient when the number of relevant predictive attributes r is much smaller then the total number of variables p.

The Winnow algorithm maintains a set of weights $w_1, ..., w_p$, one for each variable. The algorithm, in its most simple form, proceeds as follows:

1. Initialize the weights $w_1, ..., w_p$ to 1.
2. Given an example $\mathbf{x} = \{x_1, x_2, ..., x_p\}$, output 1 if

$$w_1 x_1 + w_2 x_2 + ... + w_p x_p \geq p$$

   and output 0 otherwise.

3. If the algorithm makes a mistake:
   (a) If it predicts negative on a positive example, then for each x_i equal to 1, double the value of w_i.
   (b) If it predicts positive on a negative example, then for each x_i equal to 1, cut the value of w_i in half.
4. Go to 2.

The Winnow algorithm does not guarantee successful convergence to the exact target concept. Namely, the target concept may not be linearly separable. However, its performance can still be bounded, even when not all examples are consistent with some target disjunction, if one only is able to count the number of attribute errors in the data with respect to c. Having to realize that a concept may not be learnable, a more practical question to ask is "*How badly the algorithm performs, in terms of predictive accuracy, with respect to the best one that can be learned on the given sequence of examples?*". The following section presents in more detail a very popular, practically relevant algorithm for online learning.

2.4.2 Predicting from Experts Advice

The algorithm presented in this section tackles the problem of "predicting from expert advice". While this problem is simpler than the problem of online learning, it has a greater practical relevance. A learning algorithm is given the task to predict one of two possible outcomes given the advice of n "experts". Each expert predicts "*yes*" or "*no*", and the learning algorithm must use this information to make its own prediction. After making the

prediction, the algorithm is told the correct outcome. Thus, given a continuous input of examples fed to the experts, a final prediction has to be produced after every example.

The very simple algorithm called the Weighted Majority Algorithm (Littlestone and Warmuth, 1994), solves this basic problem by maintaining a list of weights w_1, w_2, w_3, ... w_p, one for each expert, which are updated every time a correct outcome is received such that each mistaken expert is penalized by multiplying its weight by 1/2. The algorithm predicts with a weighted majority vote of the expert opinions. As such, it does not eliminate a hypothesis that is found to be inconsistent with some training example, but rather reduces its weight. This enables it to accommodate inconsistent training data. The Weighted Majority Algorithm algorithm has another very interesting property: *The number of mistakes made by the Weighted Majority Algorithm is never more than* $2.42(m+\log p)$ *where* m *is the number of mistakes made by the best expert so far.* There are two important observations that we can make based on the above described problem and algorithm. First, an ensemble of experts which forms its prediction as a linear combination of the experts predictions should be considered in the first place if the user has a reason to believe that there is a single best expert over the whole sequence of examples that is unknown. Since no assumptions are made on the quality of the predictions or the relation between the expert prediction and the true outcome, the natural goal is to perform nearly as well as the best expert so far. Second, the target distribution is assumed to be stationary, and hence the best expert will remain best over the whole sequence.

These assumptions may not be valid in practice. However, the Weighted Majority Algorithm has served as the basis for extensive research on relative loss bounds for online algorithms, where the additional loss of the algorithm on the whole sequence of examples over the loss of the best expert is bounded. An interesting generalization of these relative loss bounds given by Herbster and Warmuth (1998) which allows the sequence to be partitioned into segments, with the goal of bounding the additional loss of the algorithm over the sum of the losses of the best experts for each segment. This is to model situations in which the concepts change and different experts are best for different segments of the sequence of examples. The experts may be viewed as oracles external to the algorithm, and thus may represent the predictions of a neural net, a decision tree, a physical sensor or perhaps even of a human expert. Although the algorithms do not produce the best partition, their predictions are close to those of the best partition. In particular, when the number of segments is $k+1$ and the sequence is of length l, the additional loss of their algorithm over the best partition is bounded by $O(k\log p + k\log(l/k))$. This work is valid in the context of online regression since it applies to four loss functions: the square, the relative entropy, the Hellinger distance (loss), and the absolute loss.

2.5 Learning under Non-stationary Distributions

Given an infinite stream of instances, the challenge of every online learning algorithm is to maintain an accurate hypothesis at any time. In order for a learner to be able to infer a model, which would be a satisfactory approximation of the target concept c or function f, it is necessary to assume a sequence of training examples generated by an unknown *stationary* data distribution D. However, it is highly unlikely that the distribution will remain as is indefinitely. For that reason, throughout this work, we assume a setup in which the distribution underlying the data changes with time.

Our learning setup is thus represented with a stream of sequences S_1, S_2, ..., S_i, ... each of which represents a sequence of instances $a_{i_1}, a_{i_2}, ...$ drawn from the corresponding stationary distribution D_i. We expect that D_i will be replaced with the next significantly different stationary distribution D_{i+1} after an unknown amount of time or number of instances. Besides changes in the distribution underlying the instances in X, we must take into account the possibility of changes in the target function. For example, in the simple

case of learning monotone disjunctions we can imagine that from time to time, variables are added or removed from the target function $f = x_i \vee x_j \vee x_k$. In general, we have to expect any kind of changes in the shape of the target function or the target concept. Therefore, given a sequence of target functions $f_1, f_2, ..., f_i, ...$ the task of the learning algorithm is to take into account the changes in the distribution or the concept function, and adapt its current hypothesis accordingly.

2.5.1 Tracking the Best Expert

In the field of computational learning theory, a notable example of an online algorithm for learning drifting concepts is the extension of the learning algorithms proposed by Herbster and Warmuth (1998) in the context of tracking the best linear predictor (Herbster and Warmuth, 2001). The important difference between this work and previous works is that the predictor u_t at each time point t is now allowed to change with time, and the total online loss of the algorithm is compared to the sum of the losses of u_t at each time point plus the total cost for shifting to successive predictors. In other words, for a sequence S of examples of length l a schedule of predictors $\langle u_1, u_2, ..., u_l \rangle$ is defined. The total loss of the online algorithm is thus bounded by the loss of the schedule of predictors on S and the amount of shifting that occurs in the schedule. These types of bounds are called *shifting bounds*. In order to obtain a shifting bound, it is normal to constrain the hypothesis of the algorithm to a suitably chosen convex region. The new shifting bounds build on previous work by the same authors, where the loss of the algorithm was compared to the best shifting disjunction (Auer and Warmuth, 1998). The work on shifting experts has been applied to predicting disk idle times (Helmbold et al., 2000) and load balancing problems (Blum and Burch, 2000).

While linear combinations of "experts" have been shown suitable for online learning, a missing piece seems to be that the proposed algorithms assume that each expert (predictor) at the end of a time point or a trial (receiving a training example, predicting and receiving the correct output) is unrelated to the expert at the previous trial. Thus, there is some information loss as compared to the setup where the experts are online algorithms, able to update their hypothesis at the end of every trial.

2.5.2 Tracking Differences over Sliding Windows

Due to the assumption that the learned models are relevant only over a window of most recent data instances, there is a host of approaches based on some form of an adaptive window management or learning only over the most recent examples. A representative example is the work of Kifer et al. (2004). The idea here is to examine samples drawn from two probability distributions coming from two different time-windows, D_i and D_j, and decide whether these distributions are statistically different. In this work, the order statistics of the data are exploited, and generalizations of the Wilcoxon and Kolmogorov-Smirnoff test are defined in order to measure the distance between two data distributions. A more recent method for detecting changes and correlations was proposed by Sebastião and Gama (2007), based on the Kullback-Leibler Divergence measure between two probability distributions.

In this context, it is important to note that a change in the distribution underlying the data might not affect the truthfulness of the hypothesis induced by the learning algorithm. For example, imagine a change in the distribution which happens only in those regions of the instance space X over which the true concept c and the induced concept h do not agree, such that the number of those instances is reduced in the forthcoming sequence of training examples. This would actually result in increased accuracy, making the current hypothesis even more preferable over an alternative one. For this reason, a more intuitive approach is to monitor the learning process and act and adapt the induced models only upon significant degradation of their performance.

2.5.3 Monitoring the Learning Process

To enable proper adaptation of the induced models to the current distribution and target concept, appropriate monitoring of the learning process is necessary. The origin of the changes in the distribution or the target function is usually unknown. However, in most of the cases, these changes will make the induced model inconsistent with the newly arriving training instances. Consequently, the number (or magnitude) of mistakes made by the learner will increase, resulting in a steady or an abrupt decrease in the value of the utility measure employed. As previously, we assume that the true labels of the target variable will be supplied by an external oracle. Depending on how often these values are supplied to the learning algorithm, an appropriate monitoring framework can be designed to track the performance of the induced models.

Given a continuous signal of utility values, such as a sequence of absolute or squared error values, the performance of an online learner can be tracked and monitored using online change detection algorithms such as the CUSUM test, its successor the Page-Hinkley test by Page (1954), the ADWIN method introduced in (Bifet and Gavaldà, 2007), and the recently introduced exponentially weighted moving average (EWMA) charts by Gordon et al. (2012). These algorithms are among the most successful and powerful statistical tests for performing quality control of the inductive learning process. The main application of the CUSUM and Page-Hinkley tests is for monitoring the average of a noisy signal and detection of abrupt changes. The Page-Hinkley test considers a cumulative variable U_T defined as the cumulated difference between the observed values and their mean at any point in time:

$$U_T = \sum_{t=1}^{T} (x_t - \bar{x}_T - \delta)$$

where $\bar{x}_T = 1/T \sum_{t=1}^{T} x_t$ and δ corresponds to the magnitude of changes allowed and is used to amortize the effect of noise or fluctuations of the single on the cumulative sum. To detect an increase in the average of the signal, it maintains the minimum value of U_t $m_T = \min(U_t, t = 1, ..., T)$ and monitors the difference between m_T and U_T, i.e., $\Delta = U_T - mT$. When $\Delta > \lambda$ the test would raise an alarm for a change in the distribution. The threshold λ depends on the admissible false alarm rate.

The advantage of the CUSUM and Page-Hinkley tests is that they are memoryless, which means that there is no need of storing any subsequence of the monitored signal. The downside when using this type of methods is that they typically rely on a proper choice of values for the λ threshold and the *admissible noise* parameter δ. The book of Basseville and Nikiforov (1993) introduces a number of offline and online methods for detection of abrupt changes. In the context of regression, change detection methods are discussed for additive changes in a linear regression model, as well as for additive and non-additive changes in a linear ARMA model.

The ADWIN method keeps a sliding window W of length n with the most recently received errors or loss values and compares the distribution on two sub-windows of W, W_0 and W_1. Whenever the sub-windows exhibit distinct enough averages, the older sub-window is dropped and a change in the distribution of errors is assigned. This indicates that the performance of the learning algorithm has dropped significantly. The threshold that has to be passed in order to declare that the averages are distinct enough is computed with:

$$\varepsilon = \sqrt{\frac{1}{2m} \ln \frac{4}{D}}$$

with $m = 1/(1/n_0 + 1/n_1)$ where n_0 and n_1 denote the lengths of windows W_0 and W_1. The parameter D denotes the confidence in applying the cut, which establishes a bound on the false positive rate.

The EWMA chart method (Gordon et al., 2012) is proposed as a modular concept drift detection method used to monitor the misclassification rate of any streaming classifier. It is an online computationally efficient method with overhead $O(1)$ with no need to store data points in memory. The nice property of the EWMA charts method is that it allows the rate of false positive detections to be controlled and kept constant over time. The EWMA estimator detects an increase in the mean of a sequence of random variables. Suppose we observe the independent random variables X_1, ..., X_n which have a mean value μ_0 before the change point and μ_1 after. Assuming that both μ_0 and the standard deviation σ_X are known, the estimator is initialized with $Z_0 = \mu_0$. Its value at time t is computed with:

$$Z_t = (1-\lambda)Z_{t-1} + \lambda X_t,\ t > 0,$$

where the parameter λ is used to control how much weight is given to the more recent data compared to older data. In the original work by Roberts (1959) it is shown that, independent of the distribution of the X_t variables, the mean and the standard deviation of the estimator Z_t are:

$$\mu_{Z_t} = \mu_t,\ \sigma_{Z_t} = \sqrt{\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})}\sigma_X.$$

As long as the distribution of the X_t variables is stationary the value of Z_t will fluctuate around the mean μ_0. If the mean of the sequence of variables at time t changes to μ_1 then the value of Z_t will also change, diverging away from μ_0 towards μ_1. Thus, if at time t the difference $\Delta = Z_t - \mu_0$ is greater than $L\sigma_{Z_t}$, the method will flag a change. The parameter L is called the control limit and determines how far Z_t has to diverge from μ_0 before a change is alarmed.

In the context of monitoring the performance of an online classifier, the authors propose to apply EWMA charts on the error stream $\{X_t\}$ which is viewed as a sequence of Bernoulli random variables with a Bernoulli parameter p_t that represents the probability of misclassifying an example at time t. Detecting a concept drift in the performance of the classifier then reduces to detecting an increase in the parameter p_t of a Bernoulli distribution. Since the mean and the standard deviation, as well as the initial value of p_0 of a data stream are unknown, the authors propose a second estimator used to estimate the value of p_0 denoted with $\hat{p}_{0,t}$ and computed as follows:

$$\hat{p}_{0,t} = \frac{1}{t}\sum_{i=1} tX_i = \frac{t-1}{t}\hat{p}_{0,t} + \frac{1}{t}X_t.$$

The pre-change standard deviation is then computed with $\hat{\sigma}_{0,t} = \hat{p}_{0,t}(1-\hat{p}_{0,t})$. These values are then simply substituted in place of μ_0 and σ_X. Whenever a change is detected the classifier needs to be completely reset and retrained with post-change labelled examples. An interesting proposal is to store a small number of the most recent observations which came before the point at which a change was detected which would generally belong to the post-change distribution rather than the pre-change distribution, due to the detection delay. The stored observations can be used to train the classifier and thus, avoid experiencing drastic drop in performance after the change point.

Using ideas from the statistical process control (SPC) methodology, a very usefully change detection method has been proposed by Gama and Castillo (2006), based on performance indicators adequate for the online classification task. As in the mistake bound model, the learner is expected to return a prediction for each example from the sequence. The error rate of the learner is estimated through the probability for an incorrect prediction, which is approximately the ratio of the number of errors over the whole sequence. The SPC algorithm monitors the minimal error rate of the system p_{min} and its minimal standard deviation s_{min} over time. At any given point in time i, the algorithm compares the sum of the current

error rate p_i and its standard deviation $s_i = \sqrt{p_i(1-p_i)/i}$, with the quantity $p_{min} + \alpha * s_{min}$ or the quantity $p_{min} + \beta * s_{min}$. The values of the parameters α and β depend on the desired level of confidence in detecting a change. Using these parameters, SPC defines three possible states of the system: *in-control*, *out-of-control* and *warning*. The system can go from a state of *in-control*, through the state of *warning*, into a state of *out-of-control*, when an alarm is raised. In the case of false alarms, the system can return from the state of *warning* to the state of *in-control* without raising an alarm.

This SPC methodology allows for flexible management of false alarms and tunable confidence levels for the true positives. An interesting property of the SPC algorithms is that the error boundaries are based on the variance, whose value is expected to decrease as the confidence of the learner increases. The method can be implemented inside an online learning algorithm or as a wrapper to batch learners. Other approaches use some properties of the learning algorithm. Klinkenberg and Joachims (2000) present a theoretically well-founded method to recognize and handle concept changes using the properties of Support Vector Machines. The method adjusts the window size over which the learner infers its model, so that the estimate of the generalization error is minimized. The basic idea of adaptive window management is to adjust the window size to the current extent of concept change.

2.6 Methods for Adaptation

Change detection and adaptation are indivisible parts of the learning process and play an important role in online learning. Adaptation is basically a method of *forgetting* or *degrading* the induced models. Research in machine learning has been mostly concerned with the problem of how a system may acquire (useful) knowledge that it does not possess, and relatively little attention has been paid to the converse problem: How may a system dispose of knowledge it already possess but is no longer useful?

While learning is a process in which an organized representation of experience is constructed, forgetting is a process in which parts of that organized representation are rearranged or disposed. In machine learning, there are two main approaches for adapting to changes in the target concept or the target function (i.e., forgetting):

- *Methods for blind adaptation*: These methods are based on blind data management, consisting of instance weighting and selection. In this category, we have methods for learning over a sliding window and methods for weighting the instances or parts of the hypothesis according to their age or their utility for the learning task (Klinkenberg, 2004; Klinkenberg and Joachims, 2000; Klinkenberg and Renz, 1998). Typically, the oldest information has less importance. Weighting examples corresponds to *gradual forgetting*, while the use of time windows corresponds to *abrupt forgetting*. The disadvantage of this approach is its ignorance with respect to the existence of an actual change. Namely, the model is updated without considering whether changes have really occurred or not.

- *Methods for informed adaptation*: This category includes methods that use some form of change detection, in order to determine the point of change or the time-window in which change has occurred. The adaptation is initiated by the event of change detection and is therefore referred to as informed (Gama et al., 2003). A very useful feature of these methods is their ability for a granular adaptation of global models (such as decision rules and decision trees), where only parts of the model are affected by the change (Hulten et al., 2001).

Naturally, there is always a trade-off between the cost of an update and the anticipated gain in performance. This motivates the development of incremental learning algorithms with fast execution and response, which require only a sub-linear amount of memory for the

learning task, and are able to detect changes and perform an informed adaptation of the current hypothesis with a minimal loss of accuracy.

3 Decision Trees, Regression Trees and Variants

The only way of discovering the limits of the possible is to venture a little way past them into the impossible.

The Second Clarke's Law by Arthur C. Clarke, in "Hazards of Prophecy: The Failure of Imagination"

At the core of this research work is the problem of *any-time adaptive* learning of regression trees and different variants of regression trees, such as model trees, option regression trees, and multivariate-response trees. We will elaborate on the tree induction task through a short presentation on the history of learning decision trees, and address each of the main issues in detail. Our presentation will revolve around a single important question: *How to determine the amount of statistical evidence necessary to support each inductive decision in the learning process?*. We will review ideas from the field of statistical learning theory and unify them with existing machine learning approaches.

This chapter will explore the different dimensions of the tree induction process. We will consider both standard greedy search and the more explorative approach of using options, both a discrete-valued target function and a smooth regression surface, and both the single-target prediction case and the problem of predicting the values of multiple target variables. We will provide background for the reader to understand the different types of tree models and their learning from streaming data and thus our contributions.

This chapter is organized as follows. We start with introducing the tree induction task and discuss the main properties of Top-Down Induction Decision Tree (TDIDT) algorithms. We proceed with following the history of decision trees which helps us introduce the two representative categories of decision tree learning algorithms, and the use of statistical tests in the tree induction task. We next discuss the main issues when learning decision and regression trees, in particular selection and stopping decisions. The last tree sections introduce model trees, decision and regression trees with options, and multi-target decision and regression trees.

3.1 The Tree Induction Task

Decision and regression trees are distribution-free models. The tree induction procedure, as already described, is very simple and efficient. In addition, decision and regression trees do not require tuning of parameters or heavy tedious training. Due to the informed selection of attributes in the split tests they are quite robust to irrelevant ones. For these reasons, they are easy to implement and use.

Tree growing, also known as "hierarchical splitting", "recursive-partitioning", "group dividing" or "segmentation" , has its origin in the analysis of survey data. A classification tree or a regression tree is a set of rules for predicting the class or the numerical value of an object from the values of its predictor variables. The tree is constructed by recursively partitioning

a learning sample of data in which the class label or the target variable value and the values of the predictor variables for each case are known. Each partition is represented by a node in the tree. Figure 1 gives a decision tree for the well known toy problem of predicting whether a tennis game would be played or not, depending on the weather conditions.

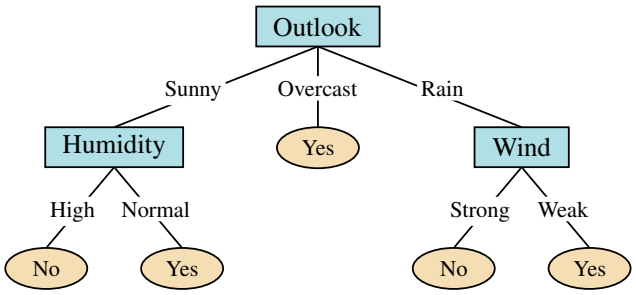


Figure 1: An illustrative decision tree for the concept of playing a tennis game. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the prediction associated with this leaf.

Decision and regression trees are known to provide efficient solutions to many complex non-linear decision problems by employing a divide-and-conquer strategy. Each division step consists of choosing a splitting test which would basically split the *feature space*, and as a result reduce the number of possible hypotheses. The divide step is then repeated on the resulting sub-spaces until a termination condition is met (i.e., all leaves of the decision tree are "pure") or until another user-defined stopping criteria has been fulfilled (e.g., maximal tree depth, node significance threshold, etc.).

In the regression setting, the problem is typically described with p predictor (explanatory) variables or attributes (x_1, x_2, ..., x_p) and a continuous response variable or a target attribute y. Given a sample of the distribution D underlying the data, a regression tree is grown in such a way that, for each node, the following steps are continuously reapplied until a termination condition is met:

1. Examine every allowable split.
2. Select and execute (create left and right children nodes) the *best* of these splits.

The *root* node comprises the entire sample which corresponds to the entire instance space X, while its children nodes correspond to the resulting subspaces X_L and X_R obtained with the execution of the best split ($X = X_L \cup X_R$). Basically, a split corresponds to a hyperplane which is perpendicular to an axis that represents one of the predictor variables. The splitting of the instance-space is performed recursively by choosing a split that improves the impurity of the node with respect to the examples that are assigned to the current region (node) which strengthens the association between the target variable and the predictor variables. In this context, two main approaches have appeared in the historical development of decision and regression trees and will be discussed in more detail in the following section.

Another point of divergence among existing tree learning algorithms is the stopping criterion. One line of algorithms is using a direct stopping rule, i.e., pre-pruning. The stopping rule refers to the criteria that are used for determining the "right-sized" decision or regression tree, that is, a tree with an appropriate number of splits. The other line of algorithms builds a tree until no more splitting is possible, and proceeds with a pruning phase in which the size of the tree is reduced (post-pruning). We will elaborate further on the meaning of "right-sized" tree and optimal predictive accuracy in a separate discussion on the important issues of learning a tree-based predictor.

From the viewpoint of learning as search, tree learning algorithms perform a simple-to-complex, hill-climbing search of a complete hypothesis space. The hill-climbing search begins

with an empty tree, progressively considering more elaborate hypotheses. The algorithm maintains only a single current hypothesis as it explores the space of all possible trees. Once it selects a splitting test and refines the current hypothesis, it never backtracks to reconsider this choice. As a consequence, this search method has no ability to determine how many alternative regression trees are consistent with the available training data, or pose new queries that optimally resolve among these competing hypotheses. Therefore, although efficient, this procedure will result in the first tree that fits the training data and is susceptible to the usual problem of converging to a locally optimal solutions.

Decision and regression trees can represent fairly complex concepts, therefore their bias component of the error is typically much lower. However, batch tree learning algorithms are characterized with a limited lookahead, instability and high sensitivity to the choice of training data. Because splitting is performed until the stopping rule can be applied decision and regression trees tend to overfit the data.

Post-pruning is very important when building decision and regression trees. The result is a higher variance component of the error, which is related to the variability resulting from the randomness of the learning sample. There exist different methods to reduce the variance of a machine learning method. The most common method for decision and regression trees is complexity control, i.e., pruning. A somewhat less explicit variance control method is model averaging. Model averaging consists in aggregating the predictions made by several models. Due to the aggregation, the variance is reduced and hence the accuracy of the aggregated predictions is typically higher. A deeper discussion and details of various model averaging method is given in Chapter 4.

3.2 The History of Decision Tree Learning

The first known algorithm for binary segmentation problems that learns binary regression trees appeared in the work of Morgan and Sonquist (1963) under the name Automatic Interaction Detector (AID). Messenger and Mandell (1972) and Morgan and Messenger (1973) extended AID for categorical outcomes (concept learning) using a so called θ criterion, which resulted in THAID (THeta AID). Gillo (1972) and Gillo and Shelly (1974) extended AID for multivariate quantitative outcome variables (MAID). The main motivation behind these early approaches was in discovering important interactions between the target variable and the explanatory predictor variables, rather than improving prediction. As statisticians, the authors were interested in finding alternative to the otherwise restricted linear model, in which the effect of the explanatory variables is mainly additive. This is slightly different from the error reduction measures which were used in the algorithms proposed later on.

In AID, for example, the splitting criterion is the largest reduction in unexplained sum of squares, which is commonly known as *residuals* or *within sum of squares* (WSS). WSS is given with:

$$WSS = \sum_{j=1}^{g} \sum_{i=1}^{n_j} (y_{ij} - \bar{y}_j)^2,$$

where $\bar{y}_j$ is the mean of all y_{ij}'s that correspond to node/group j and g denotes the number of nodes/groups that would be produced by the split. Since AID considers only binary splits, here $g = 2$. The total sum of squares TSS before the split is given with:

$$TSS = \sum_{j=1}^{g} \sum_{i=1}^{n_j} (y_{ij} - \bar{y})^2,$$

where $\bar{y}$ is the mean of all y_j's. The algorithm tries to *maximize* the η^2 coefficient that represents the percent of variance in the dependent variable explained linearly or nonlinearly by the independent variable. Given $BSS = TSS - WSS$, this is equivalent to maximizing the

ratio BSS/TSS. The criterion BSS is taken as a measure of the explanatory power of the split. MAID uses a version of this criterion generalized to the multivariate case.

3.2.1 Using Statistical Tests

The idea of introducing statistical significance tests to the splitting criteria dates back to these early methods. Kass (1975) studied significance testing for AID, namely the p-value of the BSS/TSS ratio, that he evaluates through a distribution-free permutation test. A Chi-square approximation of this test was proposed by Scott and Knott (1976). At last, CHAID (Chi-square Automatic Interaction Detector) was introduced by Kass (1980) as an evolution of AID and THAID, and is perhaps the most popular statistical supervised tree growing technique.

As indicated by its name, CHAID uses a Chi-square splitting criterion. More specifically, it uses the p-value of the Chi-square. In the case of regression tasks, the different splits are evaluated under the F statistic for the difference in mean values between the g nodes generated by the splits:

$$F = \frac{BSS/(g-1)}{WSS/(n-g)} \sim F_{(g-1),(n-g)}. \tag{3}$$

The main characteristics of CHAID that contributed to its popularity are:

1. At each node, CHAID determines for each potential predictor the optimal n-ary split it would produce, and selects the predictor on the basis of these optimal splits.

2. CHAID uses p-values with a Bonferroni correction in split selection.

The use of p-values provides stopping rules that automatically account for statistical significance. Thresholds are naturally set to the usual critical values considered for statistical significance, namely 1%, 5% or 10%. Such p-value criteria are sensitive to the number of cases involved in the split and tend to avoid splitting into too small groups.

The focus being on effects and interaction effects, the aim of these earlier methods was primarily to segment the data into groups with (as much as possible) different distributions of the outcome variable. The splitting criteria considered are thus naturally measures of the association strength between the outcome and split variables. This contrasts with more recent methods such as CART by Breiman et al. (1984), ID3 by Quinlan (1986) and C4.5 by Quinlan (1992), which attempt to maximize the homogeneity of each group by means of purity measures. Nevertheless, the splitting criteria of some of these earlier methods do incorporate purity measures such as the classification error rate (which is behind the θ criterion in THAID) or the Shannon's entropy (which was among the other alternatives proposed by Messenger and Mandell (1972)).

From the standpoint of reliable tree interpretation, a concern that has been raised on the appropriateness of methods that tend to select variables that have more splits (the variable selection bias). To address such concerns, a different approach has been taken by multiple authors, such as Loh and Vanichsetakul (1988) and Loh and Shih (1997), that separates the problem of variable selection from that of split point selection. The FACT algorithm by Loh and Vanichsetakul (1988) employs a computationally simple approach, which calculates the variance (ANOVA) F-statistic for each ordered variable. The variable with the largest F-statistic is selected and *linear discriminant analysis* (LDA) is used to find the best splitting point. Categorical variables are handled by transforming them into ordered variables. If there are m classes among the data in a node, this method splits the node into m sub-nodes.

Loh and Shih (1997) argue that FACT is free of variable selection bias only when all the predictors are ordered variables. If some are categorical variables, then the variable selection is not unbiased. Furthermore, because it uses a direct stopping rule, it is less effective than

a method that employs bottom-up pruning (such as CART) in some applications. The algorithm QUEST by Loh and Shih (1997), was designed to overcome some of the limitations of FACT, while retaining its simplicity. It employs a modification of recursive quadratic discriminant analysis (QDA) and includes a number of innovative features for improving the reliability and efficiency of the classification trees that it computes. In particular, the choice of the most appropriate attribute in QUEST is made by performing ANOVA F-tests for numerical variables and χ^2-tests for categorical variables.

Further methodological developments within this line of research include the algorithms CRUISE by Kim and Loh (2001) and GUIDE by Loh (2002). GUIDE is a regression tree learner that uses well all of the previously discussed ideas for building classification trees. The attribute selection step uses a χ^2 testing framework similar to CRUISE. However, the split selection step for numerical attributes is different in that it combines two evaluation measures, i.e., a greedy one that tries to find the smallest sum of square errors (SSE) on all possible numerical splits, and a median one that takes the best split as the median value. CTREE by Hothorn et al. (2006), unifies the statistical approaches designed to reduce variable selection bias under a framework for unbiased recursive partitioning based on conditional hypothesis testing. In addition to models for continuous and categorical data, it includes procedures applicable to ordinal or multivariate responses.

3.2.2 Improving Computational Complexity: Incremental Learning

The computational complexity of batch tree learning algorithms depends mainly on the size of the training dataset. Although the learning procedure is fast, computing the required statistics is computationally expensive. For that reason, many incremental algorithms have been proposed which examine only a subset of the training data for performing the selection step.

Incremental versions of ID3 Quinlan (1986) include ID4 by Schlimmer and Fisher (1986) and ID5 by Utgoff (1988), the latter of which is extended by Utgoff et al. (1997) through an efficient tree restructuring. An incremental version of CART is described by Crawford (1989). BOAT is also an incremental decision tree algorithm that addresses the scalability issue in data mining (Gehrke et al., 1999).

In the group of regression tree learning algorithms, the algorithms SECRET by Dobra and Gehrke (2002) and LLRT algorithm by Vogel et al. (2007) deserve a mention for their scalability and efficiency. Their main weakness is that, as the rest of the batch algorithms, they require storing of all the examples in main memory. This becomes a major problem when the datasets are larger than the main memory of the system. In such situations, subsampling, parallelization of the induction method, or other data reduction techniques need to be used.

Decision tree algorithms that address scalability issues in this manner are SLIQ by Mehta et al. (1996), SPRINT by Shafer et al. (1996), and RainForest by Gehrke et al. (1998). Chan and Stolfo (1993) considered partitioning the data into subsets that fit in memory and then developing a classifier on each subset in parallel. Their studies showed that although this approach reduces running time significantly, the multiple classifiers do not achieve the accuracy of a single classifier built by using all of the data. Some early work by Wirth and Catlett (1988) studied sampling at each node of the decision tree. However the main problem with all of these approaches, which still remains unresolved, is the failure to avoid the loss of information due to distorting the temporal locality of the data, present in data streams.

3.3 Issues in Learning Decision and Regression Trees

Practical issues in learning decision and regression trees include determining how deeply to grow the tree, choosing the appropriate splitting criterion. The former is strongly connected to handling noisy data and improving computational efficiency. Below we discuss these issues from the perspective of learning from data streams.

3.3.1 Stopping Decisions

One of the biggest issues in learning decision and regression trees is the problem of *overfitting* the training examples. The definition of *overfitting* given in the book of Tom Mitchell is as follows:

> Given a hypothesis space H, a hypothesis $h \in H$ is said to **overfit** the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

There are two main reasons why overfitting my occur. The first reason is noise in the data. Growing each branch of the tree in order to fit the training examples perfectly, i.e., with zero error, would actually mean trying to perfectly fit the noise in the training examples. Overfitting is possible even when the training examples are noise-free and might happen when a small number of examples are associated with a leaf node. In this case, it is very possible for coincidental regularities to occur, under which a particular attribute happens to partition the examples very well although there is no real connection between that attribute and the target function.

There are several approaches to avoid overfitting, grouped in two categories: approaches that stop growing the tree before all the data is perfectly partitioned, and approaches that allow the tree to overfit the data, and then post-prune the tree. Researches have debated extensively on the question which of this approaches is more successful, but it seems that there is no strong evidence in favor of either of them. Breiman et al. (1984) and Quinlan (1993) suggest that it is not appropriate to use stopping, i.e., pre-pruning criteria during tree growth because this influences the quality of the predictor. Instead, they recommend first growing the tree, and afterwards pruning the tree. C4.5 includes three post-pruning algorithms, namely reduced error pruning, pessimistic error pruning and error based pruning. CART also includes a post-pruning algorithm known as cost complexity pruning.

On the other hand, Hothorn et al. (2006) suggest that using significance tests as stopping criteria (i.e., pre-pruning) could generate trees that are similar to optimally pruned trees. In his algorithm CTREES, the stopping decision is combined with the split selection decision, and is formulated as hypothesis testing. In particular, the most significant attribute for splitting is found by first rejecting the H_0 hypothesis that there is no relationship between the input attributes and the output attribute. A permutation test linear statistic is first calculated for all predictor variables. The null hypothesis is rejected when the minimum of the adjusted (by a Bonferroni multiplier) P-values is lower than a pre-specified threshold. If the null hypothesis is not rejected, the tree stops to grow; otherwise the best splitting attribute is suggested.

Regardless of which approach is used, a key question is which criterion should be used to determine the correct *final* tree size. From the perspective of learning from data streams it is difficult to think of a *final* size of the induced decision or regression tree, since the process is rather dynamic, and the evaluation phase is interleaved with the learning phase. The fact that there is an abundance of training data makes it highly improbable that overfitting can occur due to modeling coincidental regularities. Thus, the only reason for overfitting is the possible existence of noise in the data. Under the assumption of a stationary data distribution D, the standard criteria typically used in practice are as follows:

- *Training and validation set approach*: The validation set is assumed unlikely to exhibit the same random fluctuations as the training set and is thus expected to provide a reliable estimate of the predictive accuracy. It is, however, important that the validation set is large enough itself, in order to represent a statistically significant sample of the instances.

- *Statistical test validation approach*: A statistical test is applied to estimate how likely is that further expanding would improve the accuracy over the entire instance distribution (a chi-square test, as in Quinlan (1986)). Naturally, these estimates can only be validated in hindsight.

- *Minimum Description Length approach*: Based on a measure of the complexity for encoding the training examples and the decision (regression) tree, the tree growth is stopped when this encoding size is minimized. The main assumption is that the *maximum a posteriori* (MAP) hypothesis can be produced by minimizing the description length of the hypothesis plus the description length of the data given the hypothesis, if an optimal coding for the data and the hypothesis can be chosen.

From the existing approaches and criteria for constraining tree growth, considering the fact that it is difficult to obtain a separate testing set for validation, it seems that the most appropriate method would be to use pre-pruning based on either *statistical test validation* or on some user defined criteria (such as maximum tree depth or minimal improvement of accuracy). Another promising approach is to reuse the idea for combining multiple predictions in learning ensembles of predictors and rely on *out-of-bag* estimates which are basically a side product of the bootstrapping approach proposed by Breiman (1996a, 2001). A further discussion of this method will be given in Chapter 4.

3.3.2 Selection Decisions

A second, very important, issue which is strongly present in split selection decisions, is the variable section bias, described as the tendency to choose the variables with many possible splits. In particular, Quinlan and Cameron-Jones (1995) observed that "*...for any collection of training data, there are 'fluke' theories that fit the data well but have low predictive accuracy. When a very large number of hypotheses is explored, the probability of encountering such a fluke increases.*" However when training data is abundant fitting a hypothesis by chance becomes less probable.

Hothorn et al. (2006) connect the negative sides of exhaustive search procedures (usually applied to fit regression trees) with a selection bias towards covariates with many possible splits or missing values. He argues for a statistical approach to recursive partitioning which takes into account the distributional properties of the measures. While this approach is not guaranteed to improve accuracy, it is expected to reduce the selection bias and as a result improve the veracity of the hypothesis and its interpretation. Splitting measures that try to balance the size of the child nodes, on the other hand, have shown to improve accuracy. They can be employed in the online learning scenario, by estimating the probability of an example reaching the particular leaf node.

The different methods and approaches proposed to improve the selection decision in decision and regression tree learning algorithms relate closely to strategies for choosing the next move in the search of the space of possible hypotheses $\boldsymbol{H}$. In this context, several other ideas (which have led to new types of tree-structured predictors) explore multiple paths, which correspond to different refinements of the current hypothesis or to the introduction of some randomness in the selection process. The later can be particularly useful when noise is present in the training data.

3.4 Model Trees

Tree induction methods and linear regression are both popular techniques for supervised learning tasks. The two schemes have somewhat complementary properties: the simple linear models fit by regression exhibit high bias and low variance, while tree induction fits more complex models which results in lower bias but higher variance. Therefore, a natural step to improve the performance of decision and regression trees is to combine these two techniques into one.

In the existing literature the term *model trees* has been used to refer to a more general type of decision and regression trees that are able to explore multiple representation languages in the internal (decision) nodes and in the leaves (Gama, 2004). The first work that explores the idea of using linear combination of attributes in internal nodes is CART by Breiman et al. (1984), followed by FACT (Loh and Vanichsetakul, 1988), Ltree (Gama, 1997), QUEST (Loh and Shih, 1997) and Cruise (Kim and Loh, 2001). In the regression domain, model trees mainly represent algorithms that use a linear combination of attributes or kernel regression models only in the leaves of the tree. One of the earliest works that explores the idea of using *functional leaves* is the Perceptron Tree by Utgoff (1988), where leaves can contain a general linear discriminant function. Another work that proposes a similar approach is by Quinlan (1992).

Adding a general complex function in the leaves enables the exploration of a more sophisticated representation language. Functional leaves may also implement a naive Bayes classifier, which was shown to significantly improve the performance in a study by Kohavi (1996). The most complex type of model trees employs kernel regression models in the leaves (Torgo, 1997). In a study by Gama (2004) has been further shown that using functional leaves is a variance reduction method, while using functional inner nodes is a bias reduction process.

A linear model tree is a regression tree whose leaves implement a general linear function. Unlike ordinary regression trees, model trees construct a piecewise linear (instead of a piecewise constant) approximation to the target function. The final model tree consists of a tree with linear regression functions at the leaves, and the prediction for an instance is obtained by sorting it down to a leaf and using the prediction of the linear model associated with that leaf. The linear models at a leaf typically do not incorporate all attributes present in the data, in order to avoid building overly complex models. This means that ordinary regression trees are a special case of model trees: the 'linear regression models' here do not incorporate any attribute and are just the average class value of the training instances at that node.

The existing work on learning model trees includes algorithms which build the tree as a standard regression tree, using variants of the least squares evaluation function, such as the maximization of the BSS measure previously defined. The use of the variance as the impurity measure is justified by the fact that the best constant predictor in a node is the expected value of the predicted variable for the instances that belong to the node. After the tree is fully grown, the post-pruning phase is combined with constructing linear regression models in the leaves.

This category of algorithms include M5 by Quinlan (1992) and M5' (Wang and Witten, 1997), both based on variance reduction. The same approach is also used in HTL by Torgo (1997), which replaces the linear models with more complicated models, such as kernels and local polynomials. For such regression trees, both construction and deployment of the model is expensive: However, they are potentially superior to the linear regression trees in terms of accuracy.

As noted by Karalič (1992) the variance of the response variable can be a poor estimator of the merit of a split when linear regressors are used in the leaves, especially if the points are arranged along a line which is not perpendicular to the axis of the response. To correct

this, Karalič (1992) proposed an impurity function that uses the mean square error of the linear model fitted in the leaf. The appropriate impurity function would be:

$$RSS = \sum_{i \in I_L} (y_i - \hat{f}_L(x_i))^2 + \sum_{i \in I_R} (y_i - \hat{f}_R(x_i))^2,$$

where $\hat{f}_L$ and $\hat{f}_R$ represent the optimal models for the left and the right partitions, respectively. For every possible split attribute and split point, a different pair of optimal models $\hat{f}_L$ and $\hat{f}_R$ will be obtained.

By using this split criterion, RETIS solves the aforementioned problem and builds trees of higher quality. However, if exhaustive search is used to determine the split point, the computational cost of the algorithm becomes excessively high. Namely, for real-valued predictor attributes, a linear system has to be formed and solved for all possible values in their domain. The situation is even worse for categorical attributes, since the number of linear systems that need to be solved depends exponentially on the cardinality of the domain of each attribute.

A simple modification alleviates the problem of high computational complexity by considering only a sample of all the possible split points. However, it is unclear how this would influence the accuracy of the generated trees. SMOTI by Malerba et al. (2002) allows regression models to exist only in the internal nodes of the tree, instead of only in the leaves, which accounts for both global and local effects. However, its split evaluation criterion is even more computationally intensive, because it takes into account all the linear regression models reported along the path from the root to the leaf, in addition to the mean square error of the locally fitted linear model.

The LLRT algorithm by Vogel et al. (2007) provides an improvement of the computational complexity of RETIS-style algorithms, by replacing the Singular Value Decomposition, typically used for determining the regression coefficient vector in the linear regression equation, with quicker alternatives such as Gaussian Elimination. The required matrices for all the features in the model are pre-calculated through a single scan of the data, which turns out to be equivalent to running a linear regression over the whole training dataset. The computational complexity of each split evaluation operation is $O(p^3)$, where p is the number of predictor variables used in the linear model, resulting $O(p^4)$ operations for determining each potential split. The negative side of this approach is the requirement to store the complete set of matrices in memory at any time.

As in previous attempts to learn decision and regression trees by replacing exhaustive search with some form of statistical tests for split variable section, Chaudhuri et al. (1994) incorporate the Student's t-test in the algorithm SUPPORT. The main idea is to fit a functional model or just a simple linear regressor for every node in the tree and then partition the instances into two groups: instances with positive residuals and instances with negative residuals. Unfortunately, it is not clear why the differences in the distributions of the signs of the residuals are good criteria to evaluate selection decisions. Further improvements of the same approach have been proposed later by Loh (2002), which consist mainly of a more sophisticated mechanism for handling the variable selection bias and replacing the Student's t-test with the χ^2-test.

A conceptually different approach has been proposed by Dobra and Gehrke (2002), based on transforming the regression problem into a classification problem. The main idea employed in the SECRET algorithm is to use the *expectation-maximization* (EM) algorithm on the instances associated with each node ,in order to determine two Gaussian clusters with shapes close to flat disks. After the clusters are determined, an instance is labeled with class label 1 if the probability to belong to the first cluster exceeds the probability to belong to the second cluster, or class label 2 in the opposite case. Having instances labeled in this manner, the *gini gain* or a similar split evaluation can be used to determine the split attribute and the corresponding split point. The linear regressors in the leaves are determined using least

squares linear regression. This approach avoids forming and solving a large number of linear systems of equations, which is required in an exhaustive search procedure as used by RETIS. However, its effectiveness depends on the validity of the assumption that the instances can be separated in two groups with the shapes of flat disks in the space of the response variable.

3.5 Decision and Regression Trees with Options

Option trees consist of a single structure that efficiently represents multiple decision trees. Introduced by Buntine (1992), option trees are known in the machine learning literature as an efficient variance reduction method (in the context of a bias-variance decomposition of the error). As a middle ground between trees and ensembles, they offer the double advantage of having multiple predictions per example, while having a single concise and interpretable model.

Option trees represent a variant of decision trees that contains *option nodes* (or *options*) in addition to the standard splitting nodes and leaves. The option node works as an *OR* operator, giving multiple optional splits. As such, it provides means for an interactive exploration of multiple models. An option node enables the user to choose which branch should be explored further, possibly expanding only the structure extending underneath.

When used for prediction, an example sorted to an option node will be cloned and sorted down through all of its branches, eventually ending at multiple leaves. As a result, the option node can combine multiple predictions and leverage the "knowledge" of multiple "experts". In the case of option decision trees, the commonly used rule for combining multiple predictions is majority voting. For regression tasks, the multiple predictions are typically averaged to form the final prediction.

Figure 2 illustrates a partially built option tree on the *Wine quality* dataset. If one follows the left branch of the root node, corresponding to (**Alcohol** $\leq$ **11.6**), one encounters the first option node, which further leads to a disjunction of three inequalities {(Volatile acidity ≤ 0.25) **OR** (Citric acid ≤ 0.25) **OR** (Alcohol ≤ 10.6)}. Putting it all together, one gets the following disjunction:

> {(Alcohol $\leq$ 11.6 AND Volatile acidity ≤ 0.25) **OR** (Alcohol $\leq$ 11.6 AND Citric acid ≤ 0.25) **OR** (Alcohol ≤ 10.6)}.

By deciding to follow one of the three possible branches of the option tree, only the corresponding conjunction from this set will be explored further.

In the previous sections, we have discussed extensively various batch algorithms for learning decision and regression trees. Regardless of the selection criterion employed, all of those algorithms basically pursue a hill-climbing greedy search through the space of possible hypotheses. Each selection decision thus deals with the question "*Which path is most likely to lead to the globally optimal model?*" Unfortunately, it is not possible to evaluate every path or every hypothesis in H. To provide an answer to this question, each predictive attribute (eventually accompanied with a split point) is evaluated using a chosen measure of merit which determines how well it separates the training examples *alone*. Having in mind that the selection decisions are performed at a particular point in the instance space which was reached through a sequence of previously chosen selection decisions, the choice of the best split can only be *locally optimal*.

As discussed by Kohavi and Kunz (1997), this approach will prefer attributes that score high in *isolation*, possibly overlooking combinations of attributes which may lead to better solutions globally. Breiman et al. (1984) pointed out that the greedy search is also one of the main reasons for the instability of decision trees, since small changes in the training data may result in very different trees. After selecting the split attribute and the split point, the algorithm never backtracks to reconsider earlier choices. Allowing for multiple optional tests

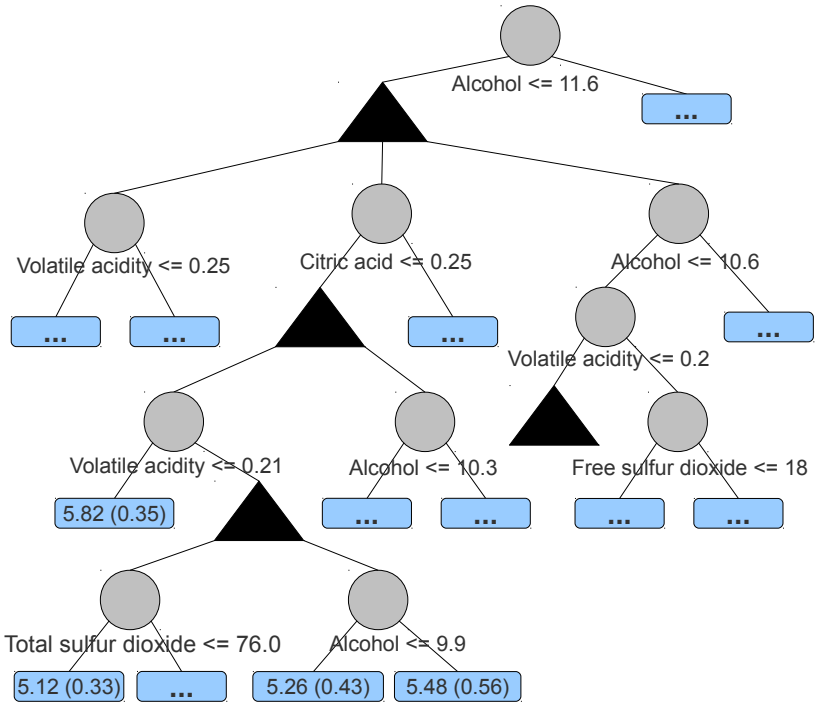


Figure 2: An illustration of an option tree. An example is classified by sorting it down every branch of an option node to the appropriate leaf nodes, then returning the combination of the predictions associated with those leaves.

enables poor early selection decision to be reconsidered later on, and duly revised. From a Bayesian perspective, the option node reduces the risk by averaging several choices.

The tree induction algorithm proposed by Kohavi and Kunz (1997) is a TDIDT induction algorithm. The evaluation criterion is based on the normalized information gain. The principal modification to the basic TDIDT algorithm is that instead of always selecting a single split, when several splits are close to the best one, the proposed algorithm creates an option node. Due to the majority voting technique employed, the minimum number of children of an option node is three (four or more choices are reasonable for multi-class problems). The pruning method is also modified to take into account the averaged error of all the children of an option node.

Kohavi and Kunz (1997) studied multiple criteria to choose the number of alternatives at an option node. The simplest one creates an option node whenever there are three or more attributes that rate a multiplicative factor (the option factor) of the best attribute with respect to the evaluation criterion. However, as using an option factor was shown to produce huge trees for some problem sets and no changes for others, Kohavi and Kunz (1997) investigated different strategies to constrain the creation of option nodes without harming the accuracy of the tree.

Their study revealed that option nodes are most useful near the root of the tree. Several explanations were offered to clarify this. The first one is from the perspective of a hill-climbing greedy search. Namely, the selection decisions that govern the splits near the root of the tree have access to little or no information on the goodness (or the optimality) of the final model that would be obtained by choosing one of several splits with similar estimated merits. On the other hand, multiple studies have confirmed that combining multiple models works best if their structures are not correlated or when they are anti-correlated. By introducing option nodes closer to the root, the sub-trees that might follow have a higher chance to be different from each other. The best practical compromise proposed by Kohavi and Kunz (1997) was to use an exponentially decaying option factor. Starting with an option factor

of x, each level is assigned a factor of $x \cdot y^{level}$, where y is the decay factor and the root node is at level 0. Option nodes were further restricted to the top five levels.

Although, in a way, option trees resemble ensembles of models, the additional structure that they provide gives two important advantages: it avoids replication, and pinpoints the uncertainty in selection decisions. Option trees are deterministic and do not rely on random bootstrap samples. Coupled with an interactive tree visualizer option trees can provide an excellent exploratory tool. The only negative aspect of learning option trees is the increased resource consumption and processing time per example.

3.6 Multi Target Decision and Regression Trees

Multi target decision and regression trees represent an interesting extension of the tree-based prediction paradigm to a multiple response setting, as they are able to provide predictions for multiple target variables simultaneously by means of a single descriptive model. The new setup, for which multi target trees have been designed, defines each instance as a pair of two vectors $(\mathbf{x},\mathbf{y})$, where $\mathbf{x} = \{x_1, x_2, ..., x_p\}$ is the vector of *predictor* or *descriptive* variables or attributes and $\mathbf{y} = \{y_1, y_2, ..., y_t\}$ is the vector of *target* or *response* variables or attributes. The principal difference between a multi target and a single-target decision or regression tree is the number of predictions contained in the leaves. While single-target trees provide only a single prediction for the response variable, multi target trees provide a vector of predictions, each corresponding to one of the multiple response variables.

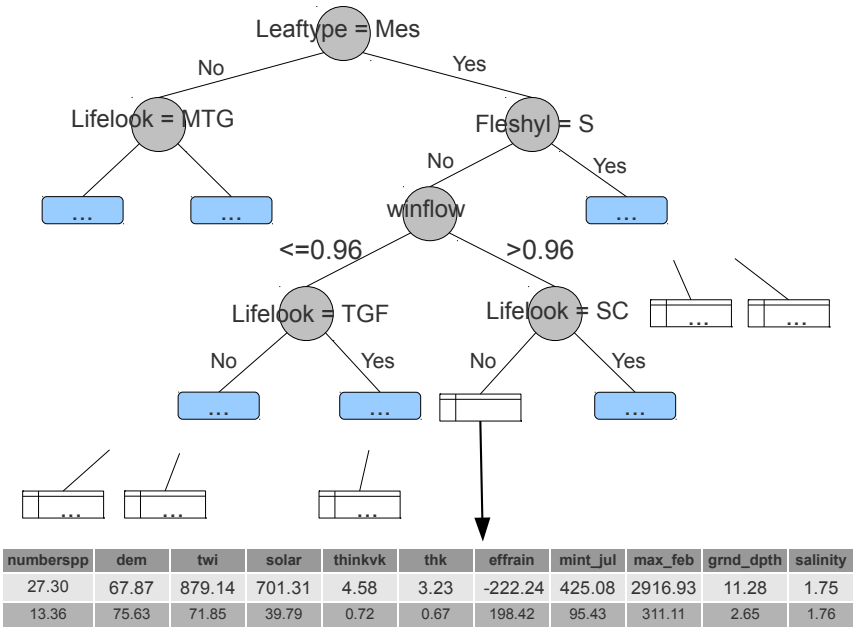
In Figure 3, we give an example of a partially constructed multi target regression tree (MTRT) for the Vegetation Clustering problem studied by Gjorgjioski et al. (2003). The tree predicts the environmental characteristics of a site (climatic, radiometric and topographic) given information on the species (physiognomy, pheology and phylogeny) present at that site. Each leaf of the multi target regression tree stores a vector of real values which represent the constant regressors for the different target attributes. Each prediction (second row) is accompanied with a standard deviation (third row), as a measure of the confidence in the prediction. Instead of a constant predictor, the leaf can contain a more complex functional model per target, like a linear regressor in the case of multi target model trees (Appice and Džeroski, 2007). As illustrated, each leaf stores a constant prediction (the instance average) and its confidence (standard deviation), for each of the responses.

Existing approaches to building multi target decision and regression trees can be, in general, divided in two categories. The first category are the covariance-aware methods, which try to make the best use of the covariance matrix and the dependencies among the target variables explicitly in the split evaluation function. The second category are the covariance-agnostic methods, which transform the problem into a type of a clustering problem in which the goal is to obtain homogeneous clusters of instances in the space of the response variables (i.e., $\mathscr{R}^t$) .

3.6.1 Covariance-Aware Methods

In the first category, we have the pioneering work of Segal (1992), which proposed to extend the regression tree methodology by modifying the split evaluation function to accommodate multiple responses. He considered two types of split evaluation functions. The first type is based on a modified node impurity measure, which aims at improving within-node homogeneity. The second type is based on two-sample statistical tests, such as the Hotelling's T^2 and promotes between-node separation.

Let $V(\theta)$ denote the model covariance matrix dependent on some unknown parameters θ. By taking θ to be equal to the sample covariances s_{ij}, we get $V = S$, where S is the pooled sample covariance matrix. This allows us to proceed without making any assumptions on the covariance structure. However, by restricting the dimension of θ significant gains in



numberspp	dem	twi	solar	thinkvk	thk	effrain	mint_jul	max_feb	grnd_dpth	salinity
27.30	67.87	879.14	701.31	4.58	3.23	-222.24	425.08	2916.93	11.28	1.75
13.36	75.63	71.85	39.79	0.72	0.67	198.42	95.43	311.11	2.65	1.76

Figure 3: An example of a multi target regression tree for the Vegetation Clustering problem.

both efficiency and interpretation can be obtained. An immediate generalization of the least squares measure used for regression trees is given with the following equations:

$$WSS = \sum_{j=1}^{g} \sum_{i=1}^{N_j} (y_{ij} - \bar{y}_j)' V(\theta)(y_{ij} - \bar{y}_j),$$

$$TSS = \sum_{j=1}^{g} \sum_{i=1}^{N_j} (y_{ij} - \bar{y})' V(\theta)(y_{ij} - \bar{y}), \tag{4}$$

where g denotes the number of nodes, and n_j is the number of instances associated with the corresponding node. The split evaluation function would be (as before) the maximization of the difference $BSS = TSS - WSS$. This basically corresponds to replacing the variance with an average Mahalanobis distance to the "center of mass" in the leaf.

Analogous to this within-node measure of loss, Segal (1992) studied functions that assess how closely the sample covariance matrix conforms to the hypothesized covariance matrix. Along this line, they proposed a likelihood-ratio test for equality of the two covariance matrices that correspond to the partitions obtained with a split. The split evaluation criterion is thus to maximize the value of the likelihood-ratio split function over all candidate splits.

3.6.2 Covariance-Agnostic Methods

The second category of algorithms approaches the problem from the perspective of obtaining homogeneous clusters in the space of target variables. As noted by Segal (1992), the problem of modeling multiple response variables falls somewhere between finding a proper classification model and clustering. Early works proposed to use some clustering algorithm on the multiple responses and then examine how covariates distribute within and between these clusters.

A more recent study by Geurts et al. (2006) proposed an algorithm called Output Kernel Trees (OK3) that employs a mapping of the loss function from the output space into some Hilbert space $\mathscr{H}$ through a definition of a kernel over the output space. As a consequence, OK3 builds a tree that maps subregions of the input space (defined by hyperplanes parallel

to the axes) into hyper-spheres defined in the feature space, corresponding to the output kernel. For example, given a loss function $l(f(x),y) = \|f(x)-y\|^2$, where $\|\cdot\|$ denotes the Euclidean norm in the output space $\mathscr{R}^p$, the empirical variance of the response vector $\mathbf{y}$ for all the instances in the subset S is computed as:

$$var\{\mathbf{y}|S\} = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{y_i}-\bar{\mathbf{y}}\|^2. \tag{5}$$

Geurts et al. (2006) have leveraged the fact that the computation of (5) requires only scalar products between the response variables, which can be used to derive a direct way to handle complex output spaces. Defining a kernel $k : \mathscr{Y} \times \mathscr{Y} \to \mathscr{R}$, which includes a mapping $\phi : \mathscr{Y} \to \mathscr{H}$ into some Hilbert space $\mathscr{H}$ such that

$$k(y_i,y_j) = \langle \phi(y_i), \phi(y_j) \rangle,$$

the variance on the output response vector in $\mathscr{H}$ becomes:

$$var\{\phi(\mathbf{y})|S\} = \frac{1}{N}\sum_{i=1}^{N}k(y_i,y_i) - \frac{1}{N^2}\sum_{i,j=1}^{N}k(y_i,y_j). \tag{6}$$

This provides the means to extend the inductive inference task to a large class of complex output spaces, over which various kernels have already been defined (Geurts et al., 2006). The type of kernel used in the work of Geurts et al. (2006) is a radial basis function (RBF) kernel: $k(y_i,y_j) = exp(-\|y_i-y_j\|^2/2\sigma^2)$ with $\sigma = 7.0711$.

The downside of this approach is that it becomes computationally expensive to make predictions, since the method provides a direct computation only in the mapped Hilbert space. Naturally, we are (most of the time) interested in making predictions in $\mathscr{Y}$, which requires the computation of a pre-image $\hat{y} = \phi^{-1}(\hat{\phi})$. Since we actually don't want to have access to the feature space and cannot assume that there is a point $\hat{y} \in \mathscr{Y}$ such that $\phi(\hat{y}) = \hat{\phi}$, only an approximation can be obtained. This requires us to find the response vector that mapped in $\mathscr{H}$ is the *closest* to the center of mass in S. The computation of the split evaluation function gets also a quadratic complexity of $O(N^2)$ for a subset of size N.

Multi target regression trees have also been studied as a special instance of predictive clustering trees (PCTs), proposed by Blockeel et al. (1998). Within this framework, Struyf and Džeroski (2006) have employed the main principle of minimizing the intra-cluster variance towards obtaining more homogeneous clusters in the output space for learning multi target regression trees. In addition, each cluster is associated with a predictive model. The model describes the cluster, and based on the values of the descriptive attributes (the regressors), predicts the values of the target attributes.

The ideas behind the predictive-clustering methodology are best understood from a viewpoint, taken by Langley (1994) and Fisher and Hapanyengwi (1993), in which both clustering and concept-learning are instantiations of the same general technique, the induction of concept hierarchies. To apply the ideas used for top-down induction of clustering trees, one needs to be able to define a distance function d that computes the distance between two examples or prototypes, and a prototype function p that computes the prototype of a set of examples. Given a cluster C and a splitting test h that will result in two disjoint sub clusters C_1 and C_2 of C, a distance $d(p(C_1),p(C_2))$ is computed. The best splitting test h is the one that maximizes this distance. For regression tasks, the prototype function is simply the mean. Thus, the impurity of a node that represents a cluster C_j is redefined with the within nodes sum of squared distances (WSSD) between all instances belonging to a node (the cluster) and the prototype of that cluster p_j:

$$WSSD = \sum_{j=1}^{g}\frac{1}{|C_j|}\sum_{y_i \in C_j} d(y_i,p_j)^2, \tag{7}$$

where g is the number of groups (nodes). For the total sum of squared distances defined appropriately as:

$$TSSD = \frac{1}{|C|} \sum_{y \in C} d(y,p)^2, \tag{8}$$

this gives a split evaluation measure of a similar form as before, having to maximize the difference: $BSSD = TSSD - WSSD$.

Maximizing inter-cluster distances corresponds to minimizing intra-cluster distances. Therefore, a typical instance of this principle is minimizing the intra-cluster variance as proposed by Kramer (1996). Geurts et al. (2006) noted that when the distance metric in the output feature space is the Euclidean distance, the PCT-based approach will produce the same trees as the OK3 algorithm.

Although there are differences in the treatment of the impurity of a node, or the computation of the predictions and the split evaluation function, it is important to notice that the same ideas for retaining a statistical support and minimizing the variance selection bias which were discussed in the previous sections can be applied to the problem of multi target prediction as well. This is particularly true for the category of algorithms which treat the problem of learning multi target regression trees as a clustering problem in the space of response variables. Transferring the online learning framework to the task multi target prediction will be discussed in more detail in Chapter 9.

4 Ensembles of Decision and Regression Trees

Any sufficiently advanced technology is indistinguishable from magic.

The Third Clarke's Law by Arthur C. Clarke, in "Hazards of Prophecy: The Failure of Imagination"

While our main interest is in the inductive inference of *descriptive interpretable* models, we are also interested in models of high predictive performance. In this context, we embarked on a theoretical and experimental comparison that spans over the vast taxonomy of methods for constructing ensembles of predictive models. There are several reasons why such a comparison is interesting and necessary, among which we favor the statistical one. Aggregating multiple predictors can bring the resulting model closer to the optimal or *best* hypothesis in comparison to a greedy search strategy in a constrained hypothesis space. In this chapter, we present several methods for learning ensembles of homogeneous models, in particular ensembles of decision and regression trees.

The chapter is organized as follows. We start with presenting an intuitive explanation on the question why ensembles of estimators perform better than a single estimator, and continue with a more precise theoretical approach that has been used mainly in the regression domain. In that context, we present the bias-variance and the bias-variance-covariance decomposition of the error, which gives an explanation on the source of error when learning ensembles of models. The succeeding section describes methods for learning ensembles, which are in general divided in two categories: those that obtain diverse trajectories in the hypothesis space by diversifying the set of accessible hypotheses and those that achieve this by diversifying the strategy of traversal. The last section presents algorithms for learning ensembles of classifiers for concept drift detection, due to their widespread use for learning under non-stationary distributions.

4.1 The Intuition Behind Learning Ensembles

Ensemble learning algorithms construct a set of predictive models known as *base models* or *experts*, and combine their predictions into a single final prediction. The learned ensemble represents a single hypothesis, which is not necessarily contained within the hypothesis space of the models from which it is built. Ensemble models have been shown, both theoretically and empirically, to outperform single predictors on a wide range of tasks. The main belief as stated in the literature on learning ensembles is that, by combining *different* base models, the ensemble can benefit from exploring different representation languages, feature spaces or search strategies.

An important component of every ensemble learning algorithm seems to be its strategy for achieving error "diversity" of the base models (Kuncheva and Whitaker, 2003). An intuitive explanation is that, if the base models make mistakes on different examples, the incorrect predictions from some of the base learners can be corrected with a majority of

correct predictions from the other members of the ensemble. Although ensembles are (most of the time) undoubtedly better predictors than a randomly chosen single predictive model, our understanding of why and how this happens is still incomplete. Luckily, in the regression context, there is a theoretical approach that explains why a convex combination of regression estimators performs better than a single estimator.

Assume the task is to learn a function $f : \mathscr{R}^p \rightarrow \mathscr{R}$ for which we have a sample of instances $(\mathbf{x}, y)$, where $\mathbf{x} =< x_1, x_2, ..., x_p >$ is an input vector and $y = f(\mathbf{x}) + \varepsilon$, drawn independently at random from some unknown stationary distribution D. ε is the additive noise with zero mean and finite variance. However, for brevity and without loss of generality, we can assume a noise level of zero in the data. The ensemble consists of M models, and the output of model i on an input $\mathbf{x}$ is $z_i = \hat{f}_i(\mathbf{x})$. Let us further assume that the combined prediction is a weighted ensemble average given with:

$$z_{ens} = f_{ens}(\mathbf{x}) = \sum_{i=1}^{M} w_i \hat{f}_i(\mathbf{x}),$$

where $w_i > 0$ and $\sum_i w_i = 1$. Consequently, the ensemble z_{ens} is a convex combination of the component models/estimators. This is a standard combination rule for regression ensembles, with its special case of non-weighted averaging where $w_i = 1/M$. The weights can be seen as our relative confidence in the correctness of the models' predictions, and are thus constrained to sum to 1.

Krogh and Vedelsby (1995) proved that for any single input $\mathbf{x}$ the squared error of the ensemble is guaranteed to be less than or equal to the average squared error of the component models:

$$(f_{ens}(\mathbf{x}) - y)^2 = \sum_{i=1}^{M} w_i (\hat{f}_i(\mathbf{x}) - y)^2 - \sum_{i=1}^{M} w_i (\hat{f}_i(\mathbf{x}) - f_{ens}(\mathbf{x}))^2. \tag{9}$$

This decomposition is best known as the Ambiguity decomposition. The detailed proof is given in Krogh and Vedelsby (1995). As the squared error is always positive, the difference of the two terms has to be positive as well. The *ensemble ambiguity* on input $\mathbf{x}$ is:

$$\overline{a}(\mathbf{x}) = \sum_{i=1}^{M} w_i (\hat{f}_i(\mathbf{x}) - f_{ens}(\mathbf{x}))^2. \tag{10}$$

It is simply the variance of the weighted ensemble around the weighted mean and measures the disagreement of the models on the input vector $\mathbf{x}$. The larger the ambiguity term, the larger the ensemble error reduction would be. However, by increasing the variability of the individual estimators, the weighted average error of the individuals might increase as well. This shows that diversity itself is not enough. The best performance can be obtained only with the right balance between diversity and individual accuracy.

The importance of the Ambiguity decomposition is that it explains mathematically the minimization of the risk of randomly choosing a single estimator by averaging multiple estimators. Of course, the error of an individual on a particular input might be lower than the average, or lower than the error of the ensemble, but for a given input vector $\mathbf{x}$ we have no way of choosing the most adequate estimator. Therefore, the risk of picking one at random can be reduced by averaging their predictions.

4.2 Bias, Variance and Covariance

The Ambiguity decomposition holds for convex combination rules and is a property of an ensemble trained on a single dataset. Therefore, the diversification of the errors is achieved only through the diversification of the inference process for each individual estimator (model).

Unlike the *bias-variance* decomposition, proposed for the first time by Kong and Dietterich (1995), it does not take into account the variability of distributions over possible training sets. For this reason, it is particularly useful to analyze the bias-variance decomposition of the squared loss for an ensemble.

In the case of a least squared regression, we have some machine learning algorithm that constructs a non-parametarized estimator $\hat{f}$ (e.g., a regression tree) that minimizes the expected mean squared error:

$$e(\hat{f}) = \int (\hat{f}(\mathbf{x}) - y)^2 p(\mathbf{x}) d(\mathbf{x}). \tag{11}$$

Since we do not have access to the true distribution $p(\mathbf{x})$, we approximate the integral with a summation over the testing dataset. Given a test example $\mathbf{x}$, the estimator produces a prediction $z = \hat{f}(\mathbf{x})$. For the sake of simplicity, the fact that y is a function of $\mathbf{x}$ will remain implicit throughout this section.

The *bias-variance* decomposition of the squared error is then:

$$\begin{aligned} E\{(\hat{f} - y)^2\} &= (E\{\hat{f}\} - y)^2 + E\{(\hat{f} - E\{\hat{f}\})^2\} \\ &= bias(\hat{f})^2 + variance(\hat{f}). \end{aligned} \tag{12}$$

The *variance* quantifies our expectation on the amount of variability of the predictions around their expected value. The *bias* quantifies the variability of the expected predictions from the true values. For best performance, these two components of the error have to be balanced against each other. The *bias* of an estimator reflects its intrinsic ability to model the given unknown function $f(\mathbf{x})$ and does not depend on the choice of the training set. On the other hand, the *variance* component measures the variability of the models produced given different training sets, and does not depend on the true target attribute values.

If the ensemble is treated as a single learning unit denoted with f_{ens}, its bias-variance decomposition can be formulated as:

$$\begin{aligned} E\{(f_{ens} - y)^2\} &= (E\{f_{ens}\} - y)^2 + E\{(f_{ens} - E\{f_{ens}\})^2\} \\ &= bias(f_{ens})^2 + variance(f_{ens}). \end{aligned} \tag{13}$$

As shown by Ueda and Nakano (1996) and Brown et al. (2005a), this decomposition can be further extended for the case of ensembles to express the correlation among ensemble members. Using the notation $\hat{f}_i$ instead of $\hat{f}_i(\mathbf{x})$ for brevity, the generalization error of the ensemble estimator f_{ens} can be decomposed into three terms:

$$E\{(f_{ens} - y)^2\} = \overline{bias}^2 + \frac{1}{M}\overline{var} + (1 - \frac{1}{M})\overline{cov}, \tag{14}$$

where

$$\overline{bias} = \frac{1}{M}\sum_{i=1}^{M}(E\{\hat{f}_i\} - y) = \frac{1}{M}\sum_{i=1}^{M} bias(\hat{f}_i) \tag{15}$$

is the averaged bias of the ensembles members,

$$\overline{var} = \frac{1}{M}\sum_{i=1}^{M} E\{(\hat{f}_i - E\{\hat{f}_i\})^2\} = \frac{1}{M}\sum_{i=1}^{M} variance(\hat{f}_i) \tag{16}$$

is the averaged variance of the ensemble members, and

$$\overline{cov} = \frac{1}{M(M-1)} \sum_{i=1}^{M} \sum_{i \neq j} E\{(\hat{f}_i - E\{\hat{f}_i\})(\hat{f}_j - E\{\hat{f}_j\})\}. \tag{17}$$

is the averaged covariance of the ensemble members. The *bias-variance-covariance* decomposition illustrates that, in addition to the bias and the variance of individual estimators, the generalization error of an ensemble also depends on the covariance between the individual members. A negative correlation among two ensemble members contributes to a decrease in the generalization error. Thus, we would ideally like to decrease the covariance, without causing any increase in the bias or variance terms.

By plugging in the formula for ensemble ambiguity (10) to (14), in the special case when $w_i = 1/M$, it can be shown that the *expected ensemble ambiguity* sums up the variance of the ensemble plus an undefined term Ω, as given with the following decomposition:

$$\begin{aligned} E\{\overline{a}\} &= \frac{1}{M} \sum_{i=1}^{M} E\{(\hat{f}_i - f_{ens})^2\} \\ &= \frac{1}{M} \sum_{i=1}^{M} E\{(\hat{f}_i - E\{f_{ens}\})^2\} - E\{(f_{ens} - E\{f_{ens}\})^2\} \\ &= \Omega + var(f_{ens}) \end{aligned} \tag{18}$$

With a further analysis, it has been shown that the term Ω is a sum of the averaged variance of the ensemble members plus the average squared deviation of the expectations of the individuals from the expectation of the ensemble. This exemplifies the diversity trade-off, which tells us that the diversity of an ensemble cannot be improved without affecting the other parts of the total loss.

4.3 Methods for Generating Ensembles

Our presentation of the taxonomy of ensemble learning methods is structured according to the various methods for introducing *diversity* among the ensemble members. The standard way of categorizing ensemble learning methods is by dividing them into two general groups: methods that modify the training data and methods that modify the learning process. We will use the same taxonomy with a different perspective, which we believe gives a more intuitive view with respect to the inductive inference process.

Through the previous chapters, we have viewed the learning process as search through the hypothesis space. Consequently, when building an ensemble we would like the individual estimators to occupy different points in the hypothesis space, possibly all near to the optimal one. We will therefore look at two categories of methods for achieving diversity in the ensemble's errors: those that obtain diverse trajectories in the hypothesis space for the base level learning algorithms by diversifying the set of accessible hypotheses and those that achieve this by diversifying the strategy of traversal:

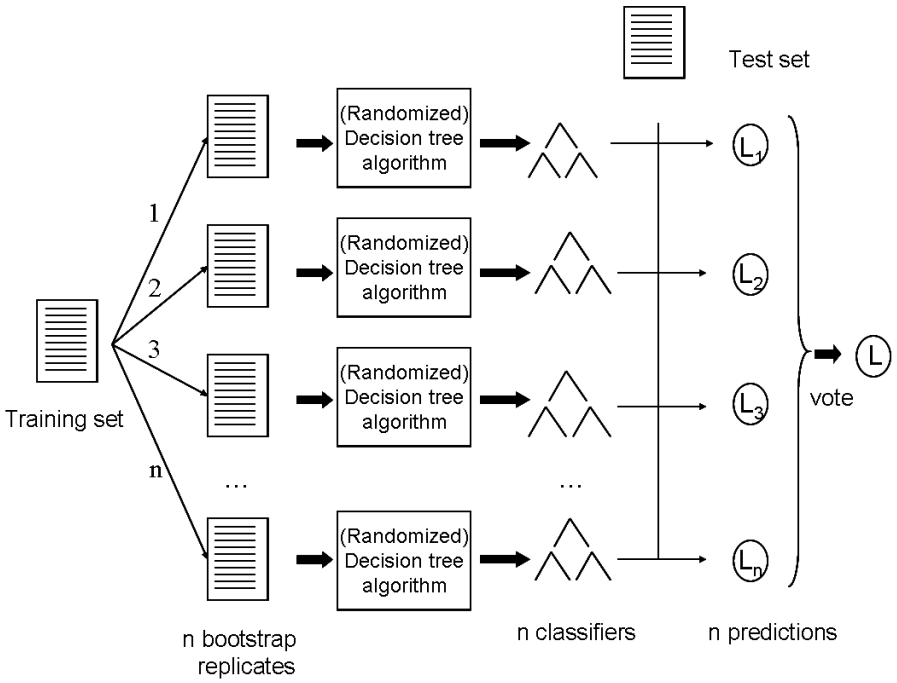
- *Diversifying the Set of Accessible Hypotheses* - These methods vary the set of hypotheses which are available to the learning algorithm in two ways, by diversifying the input space or by diversifying the output space for the different ensemble members. It should be noted here that the space of all possible hypotheses H remains the same, although a different subset $H_i \subseteq H$ would be accessible for each of the individual estimators.

- *Diversifying the Traversal Strategy* - These methods basically alter the mechanism of modeling the inductive decisions, thus explicitly varying the way the algorithm traverses the hypothesis space, thereby leading to different hypotheses. The simplest

method for diversification of the traversal strategy is by introducing a random component. Each traversal of the hypothesis space seeded with a different random seed would likely yield a different trajectory, which in the case of tree-structured predictors would correspond to a different decision or a regression tree.

Naturally, there are methods that utilize a combination of these two categories, but we are mainly interested in the strategies for diversification and their effect on the inference process.

Figure 4: An illustration of the process of learning ensembles through diversification of the training data and diversification of the traversal strategy.



A unified view is given by the illustration in Figure 4. The process starts by creating a collection of n training datasets obtained through some method for diversifying the input and/or the output space. A predictive model is further learned on each of the training datasets by using a machine learning algorithm of choice. In this case, since we are interested mainly in ensembles of decision and regression trees, the learning algorithm will construct a decision tree from each training dataset.

Optionally, it is possible to introduce some randomness in the decision tree learning algorithm which corresponds to diversifying the traversal strategy of the algorithm. In the well known Random Forest algorithm, Breiman (2001) modified the standard approach for learning a decision tree by reducing the number of combinations examined at each splitting node to a smaller subset of randomly chosen attributes. Each node is therefore split using the best among a random subset of predictors. This somewhat counterintuitive strategy turns out to perform very well as compared to many other classifiers, including discriminant analysis, support vector machines, and neural networks (Breiman, 2001).

Having an ensemble of decision trees trained on a *different sample* of the original training data, each of the learned trees is further applied on a separate test set of instances. Their predictions are at last combined using some form of a combination rule. In the classification case, a popular method to combine predictions from individual models is majority voting, i.e., the predictions are combined by using the plurality of votes in favor of a particular class. In the regression case, a simple weighted or non-weighted averaging would in general suffice.

In a final note, we would like to mention a third category of ensemble learning methods that basically represents a regularization of the variance of the ensemble by using a meta-

learning algorithm. Here, regularization of variance denotes a method for synthesizing and combining information from other models. Stacking or *stacked generalization* (Wolpert, 1992), is mainly a method for combining heterogeneous base models, such as the k-nearest neighbors method, decision trees, Naive Bayes, etc. The central idea is to learn the biases of the base predictors with respect to a particular training set, and then to filter them out. The procedure has two steps. First, a meta-learning dataset using the predictions of the base models is generated. Second, using the meta-learning set a meta model is learned, which can combine the predictions of base models into a final prediction. This method is used as some form of a combination rule. As such, it goes out of the scope of this chapter, for it does not belong to the methods that influence the inference process of the individual estimators, but rather the combining of predictions with the final ensemble.

4.3.1 Diversifying the Set of Accessible Hypotheses

We have identified three ways of diversifying the set of accessible hypotheses, each discussed in more detail in the following sections. We will discuss methods for manipulation or diversification of the training data, including the input space X and the output space Y.

4.3.1.1 Diversification of the Training Data

The most well-known methods for learning ensembles by manipulation of training data are *Bagging*, introduced by Breiman (1996a), and *Boosting*, proposed by Freund and Schapire (1997). These ensemble learning algorithms have been empirically shown to be very effective in improving the generalization performance of the individual estimators in a number of studies. In order to promote variance among the predictors, these algorithms rely on altering the training set used when learning different ensemble members.

Bagging is an acronym for *Bootstrap Aggregating*. It is the simplest ensemble learning algorithm which consists mainly of creating a number of bootstrap replicates of the training dataset. A bootstrap replicate is created by sampling randomly with replacement, i.e., an instance $a_i =< \mathbf{x_i}, y_i >$ may appear multiple times in the sample. Assume that the algorithm creates M such bootstrapped samples. The obtained bootstrap replicates are used to fit M predictive models T_m, each representing hopefully a different hypothesis from H. After the base learners have been fit, the aggregated response is the average over the T_m's when predicting a numerical outcome (regression), and a plurality vote when predicting a categorical outcome (classification). An advantage of *Bagging* is that it is a parallel algorithm, both in its training and operational phases, which enables a parallel training and execution of the M ensemble members.

The commonly accepted answer to the question "*Why does Bagging work?*" is that, bagging smooths out the estimates of the individual models and reduces the variance of the ensemble. This was empirically supported in our own experimental study as we shall see in the following chapters. In the case of classification, however, it is less clear why voting works. One generally accepted belief is that bagging transforms the prior distribution of the classifier models into a new distribution of models of higher complexity (Domingos, 1997).

Boosting has its origins in the online learning algorithm called *Hedge*(β) by Freund and Schapire (1997), developed as part of a decision-theoretic framework for online learning. Within this framework, the authors propose weighting of a set of experts in order to predict the outcome of a certain event. Each expert s_i is assigned a weight which can be interpreted as the probability that s_i is the best expert in the group. These probabilities are updated online, with the arrival of each new "training" event. The experts that predicted correctly are rewarded by increasing their weight, thus our increasing belief in their expertise, while the experts that predicted incorrectly are penalized by decreasing their weight. The *Hedge*(β) algorithm evolves the distribution of our beliefs in the experts in order to minimize the

cumulative loss of the prediction. Freund and Schapire (1997) proved an upper bound on the loss which is not much worse than the loss of the best expert in the ensemble in hindsight.

The well known *AdaBoost* (Adaptive Boosting) algorithm, which has resulted from this study, works in a similar manner. The general boosting idea is to develop a classifier team incrementally, by adding one classifier at a time. In each iteration of the process, a new training set is generated which takes into account the predictive accuracy of the classifier resulting from the previous iteration. The sampling distribution starts from uniform, and progresses towards increasing the likelihood of the "difficult" instances. Basically, the classifier that joins the team at step i is trained on a dataset which selectively sampled from D, such that the likelihood of instances which were misclassified at step $i-1$ is increased. Therefore, each succeeding classifier gets access to a different set of hypotheses, which the previous classifier was not able to explore well enough.

This is only one of the two possible implementations of *AdaBoost* which in particular deals with *resampling*. The other implementation performs reweighing instead of resampling and is thus deterministic. In this case, *AdaBoost* generates a sequence of base models $T_1, ..., T_M$ using weighted training sets such that the training examples misclassified by model T_{m-1} are given half the total weight when training the model T_m, and correctly classified examples are given the remaining half of the weight.

4.3.1.2 Diversification of the Input Space

An early work by Ho (1998) proposes a randomization in the input space in the context of ensemble learning, such that every base model is trained using a randomly chosen subset of attributes. The random subspace method (RSM) is based on random sampling of features instead of data points. Let each learning example a_i in the learning set S be a $p+1$-dimensional vector $a_i =< x_{i_1}, x_{i_2}, ..., x_{i_p}, y_i >$. RSM randomly selects d attributes from the first p, where $d < p$. By this, we obtain a d-dimensional random subspace of the original p-dimensional attribute space over which a new predictor is learned. Typically, d is kept the same for all members of the ensemble. Ho (1998) has reported good results for tree classifiers built from $d \approx p/2$ features. The common knowledge is that, when the dataset has many attributes each containing little information on the target, one may obtain better models in random subspaces than in the original attribute space (Breiman, 2001; Ho, 1998). The same idea has been used in the Random Forest algorithm of Breiman (2001), however its final effect is different since repeated random selection of features is performed within the inference process.

4.3.1.3 Diversification in the Output Space

In a study on feature selection for linear regression, Breiman (2000) has found a somewhat surprising result that adding noise to the target attribute values, while leaving the input vectors intact, worked just as well as bagging. He extended those early results to non-linear contexts in both regression and classification tasks, and showed that given a single training set by introducing extra random variation into the outputs, the predictors built on these training datasets, averaged or voted, may perform comparable in accuracy to methods such as bagging or boosting. Practically, adding random noise to the output enables us to produce a sequence of perturbed training sets with similar quality as compared to those obtained with bagging.

For the case of regression, adding a random component to the outputs can be done in a straightforward way. Breiman (2000) proposed the *Output Smearing* procedure which describes a method of adding Gaussian noise to the outputs. The procedure requires a robust estimate of the standard deviation of the outputs in the dataset. This is achieved by using a second pass over the data in which the values which deviate more than 2.5 standard

deviations from the original mean are removed and the standard deviation is re-computed. Then, the new outputs are generated as:

$$y' = y + z \cdot sd(y), \tag{19}$$

where $sd(y)$ is the standard deviation estimate and z is an independent unit normal. A maximal tree is then grown using the new set of outputs, and the whole procedure is repeated M times. The final prediction is given as the average over the predictions of the M previously grown trees. The procedure is inherently parallel, allowing us to grow M trees simultaneously. Breiman (2000) has shown that adding output noise (mean-zero Gaussian noise added to each of the outputs) with random feature selection works better than bagging, comparing favorably in terms of accuracy.

In a similar study (Geurts, 2001) has shown that significant improvements of the accuracy of a single model can be obtained by a simple perturbation of the testing vector at prediction time. The Dual Perturb and Combine algorithm (dual P&C) proposed by Geurts (2001) was shown to produce improvements comparable to those obtained with bagging. Dual P&C produces a number of perturbed versions of the attribute vector of a testing instance by adding Gaussian noise. In the context of decision trees adding Gaussian noise to the attribute vector is more or less equivalent to adding Gaussian noise to the discretization thresholds applied in the splitting tests, which in some ways equals to randomizing the discretization thresholds.

4.3.2 Diversification of the Traversal Strategy

In contrast to deterministic information-guided learning, among the most accurate methods for learning predictors we also have methods that incorporate randomness in the learning process. For example, neural networks use randomization to assign initial weights, bagging generates ensembles from randomly sampled bootstrap replicates of the training set, and Boltzmann machines are stochastic in nature. Randomizing the learning process has been often seen by many authors as an efficient method to promote diversity among ensemble models, while reducing the computational complexity and improving the accuracy. The main argument in classification is that the classification *margin* of an example allows for less precise models which can be learned more efficiently by replacing the brute force examination of all attribute-value combinations with a more efficient one (Breiman, 2001). From our point of view, this method enables diversification of the search strategy, resulting in a different traversal trajectory in the space of hypotheses for every random seed.

All of these methods have been shown to achieve remarkable accuracy, despite the counterintuitive action of injecting random bits of information in the learning process. The common belief is that randomization in the tree building process improves accuracy by avoiding overfitting, while bagging achieves that by improving the instability of decision tree learning algorithms. The most interesting representative of this group of ensemble learning algorithms is the RandomForest algorithm by Breiman (2001). The RandomForest algorithm, as described briefly in the first section, can be thought of as a variant of Bagging. The main point of departure from Bagging is that it allows for a random selection of a subset of attributes considered in the split selection phase at every node of the tree.

The accuracy of a RandomForest depends on the strength of the individual tree classifiers and the correlation among them. Breiman (2001) proved a bound on the generalization error of RandomForest, showing that RandomForest does not overfit as more trees are added. In particular, given a RandomForests of decision trees the following inequality is proven:

$$PE^* \leq \overline{\rho}(1-s^2)/s^2 \tag{20}$$

where PE^* is the generalization error of the forest, $\overline{\rho}$ is the weighted correlation between

the residuals (in case of classification the residuals are computed using a margin function) and s is the strength (expected error) of the set of classifiers. Although the bound is likely to be loose, it shows that in order to improve the generalization error of the forest, one has to minimize the correlation while maintaining strength.

A similar bound for the mean squared generalization error is derived for the case of regression, which shows that the decrease in error from the individual trees in the forest depends on the correlation between the residuals and the mean squared error of the individual trees. Let $f(\boldsymbol{\theta})$ denote a classifier for a given randomly chosen set of feature vectors $\boldsymbol{\theta}$ employed at each node. Let $PE^*(forest)$ and $PE^*(tree)$ correspond to the predictive error of the forest and the average predictive error of a single tree. The requirements for accurate regression forests are low correlation between the residuals and low error trees, as stated by the following theorem:

If for all random choices of feature vectors $\boldsymbol{\theta}$, $E[Y] = E_X[f(\boldsymbol{\theta})]$, *then* $PE^*(forest) \leq \overline{\rho} PE^*(tree)$, *where* $\overline{\rho}$ *is the weighted correlation between the residuals* $Y - f(\boldsymbol{\theta})$ *and* $Y - f(\boldsymbol{\theta}')$, *and* $\boldsymbol{\theta}$, $\boldsymbol{\theta}'$ *are independent.*

An interesting difference between RandomForests for regression and classification is that in the case of regression the correlation increases slowly with the increase of the number of features used. Basically, a relatively large number of features is required to reduce the generalization error of a tree.

4.4 Ensembles of Classifiers for Concept Drift Detection

In the first chapter, we have discussed the inherent temporal component in the process of learning from data streams. Due to its importance, a large number of algorithms and models for dealing with it have been designed and proposed thus far. Among the vast variety of methods, ensembles of classifiers have been shown to possess some advantages over single classifier methods, mainly because they are easy to scale and parallelize, and can quickly adapt to changes in the distribution or the target concept by pruning the members whose performance decreased over the most recent testing data. Consequently, there is a host of ensemble learning algorithms designed in particular for the task of concept drift management.

An early work by Street and Kim (2001) proposed an ensemble-based algorithm for detecting abrupt concept drift. The ensemble members are learned on different blocks (chunks) of data. The algorithm splits data into blocks, builds one classifier per data block independently, and retires old classifiers one at a time.

Wang et al. (2003) have developed a general framework for mining concept drifting data streams using weighted ensemble classifiers. The proposed algorithm tries to adapt to changes by assigning weights to classifiers proportional to their accuracy over the most recent data block. Both Street's and Wang's algorithms are in the spirit of traditional bagging algorithms, but they do not perform in a real streaming fashion, i.e., one example at a time.

The Dynamic Weighted Majority (DWM) algorithm by Kolter and Maloof (2007) is a well known approach which creates and removes weighted experts dynamically in response to changes in performance. Based on the Weighted Majority online learning algorithm by Littlestone (1988), this method can be used in combination with any base learning algorithm. In Kolter and Maloof (2005), an extended variant of DWM called AddExp has been proposed for classification and regression tasks. The authors were able to find bounds for the performance of AddExp relative to the actual performance of any online learner on a single subsequence.

A similar approach based on boosting and explicit change detection is proposed by Chu and Zaniolo (2004). Zhang and Jin (2006) proposed an ensemble learning algorithm with dynamic construction and organization sensitive to concept drifts. Another boosting method by Scholz and Klinkenberg (2007) was developed for managing various types of concept drift.

In short, the main principle is to split the data stream into chunks of data over which the base models are fitted. Uninformed methods assign weights to the more confident estimators, which are updated continuously with every new training example available. Informed methods basically track the performance of the ensemble members by a continuous evaluation or by using some change detection method. Some of the approaches also monitor the distribution of class labels comparing models of every two chunks. Finally, low-performing classifiers are either excluded from the ensemble or retired on disk in case they become valuable later. Some methods include a dynamic adjustment of the size of the data chunks and partial reconstruction of the models (Li et al., 2010).

Most of these methods differ in the change detection method or the adaptation strategy employed. While ensembles have been shown to be very effective for learning under concept drift, users are very often interested in the meaning of the concept drift and how it has affected the inferred models or patterns. Thus, a very challenging problem is embedding change detection methods within the learning process of decision tree induction. Such an approach can explicitly identify the stale or non-valid patterns and rules, as well as partially reconstruct the tree. A discussion on issues and strategies for change detection and adaptation in the context of decision and regression trees learned on data streams is also given in Chapter 6.

5 Experimental Evaluation of Online Learning Algorithms

Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted.

Albert Einstein

This chapter discusses issues related to the design of evaluation experiments and the choice of appropriate models and metrics for evaluating the performance of online algorithms. Traditional methods for evaluating machine learning algorithms cannot give insights into the temporal dimension of the learning process because of the assumption that the data instances are drawn at random from a stationary data distribution, representing a concept or a target function which does not change with time. We will first define the main criteria for evaluation of online algorithms and then present the basic framework for online evaluation of machine learning algorithms. The methods, metrics and datasets described in this chapter are used in the experimental evaluation performed for every algorithm proposed in this thesis.

5.1 Criteria for Online Evaluation

Unlike the batch process of learning, the process of learning from data streams has an inherent temporal component. The increasing volume of data usually demands algorithms which are able to perform learning in a single-pass. The non-stationary data distributions on the other hand demand the ability to adapt through the data to concept drifts and shifts. As a consequence, streaming and online algorithms are typically designed to track the evolution of the underlying data, and maintain or induce models whose accuracy remains high in the presence of changes.

The question of what is the best evaluation methodology for a machine learning algorithm is very difficult to answer, even in the context of batch machine learning. However, one thing is clear: The specific characteristics of the online learning scenario demand specific methods, metrics and analysis tools, which will be able to address the temporal component of the data and the learning process.

A learning process which never ends demands continuous evaluation, which has to share some of the properties of the learning process itself. An adequate design of the experimental evaluation has to provide answers on how successfully the algorithm meets the above requirements over the course of learning. A number of different evaluation methodologies have been proposed in the stream mining literature, which address various aspects of the online learning process (Castillo and Gama, 2005; Gama et al., 2004b, 2003; Holmes et al., 2005; Hulten et al., 2001). A good overview thereof is given by Gama (2010).

We will describe the online evaluation methodology which we applied in our empirical evaluation. We are interested in an evaluation methodology which will enable us to continuously monitor the process of learning and all of its aspects, in order to address issues related

to the algorithm's consumption of memory, processing time per example, evolution of the model over the course of learning, speed of convergence, as well as its ability of concept drift detection and adaptation to it.

5.2 Evaluation Metrics

In this section we discuss the error metrics that we used to measure and estimate the generalization power of the algorithm, metrics for measuring the complexity of the models induces and metrics for measuring the effectiveness of change detection methods when learning from non-stationary distributions.

5.2.1 Error Metrics

The most relevant metric for a learning algorithm is the generalization error, which is an estimate of the goodness of the fit of the induced model with respect to the target function. While the model is being fitted on a training set, our estimates of the generalization error are always obtained by using a separate independent set of instances (testing set). Standard error metrics used in the machine learning literature are the *mean squared* error (*MSE*) (Armstrong and Collopy, 1992), and the *mean absolute* error (*MAE*) (Abramowitz and Stegun, 1972).

Given a training set $\{(\mathbf{x_1},y_1), (\mathbf{x_2},y_2), ...\}$ the regression task is to construct an estimator $\hat{f}(\mathbf{x})$ that approximates an unknown target function $f(\mathbf{x})$. The quality of the approximation is evaluated with the expected mean squared error over the whole distribution of examples:

$$e(\hat{f}) = \int (\hat{f}(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}) d(\mathbf{x}). \tag{21}$$

Since we do not have access to the true distribution $p(\mathbf{x})$, we approximate the integral with a summation over a separate set of instances of size N, ideally drawn independently at random from the same distribution and held for testing purposes. Given a test example $\mathbf{x}$, the estimator produces a prediction $\hat{f}(\mathbf{x})$. The mean squared error is defined as the averaged squared difference between the predicted value $\hat{f}_i = \hat{f}(\mathbf{x_i})$ and the desired correct value $y_i = f_i = f(\mathbf{x_i})$:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (f_i - \hat{f}_i)^2. \tag{22}$$

Because the error magnitude depends on the magnitudes of possible function values, the relative mean squared error (RE) can be used instead of MSE:

$$RE = \frac{N \cdot MSE}{\sum_i (f_i - \overline{f})^2} \tag{23}$$

where the MSE is normalized with respect to the error of the baseline predictor that always predicts the average value:

$$\overline{f} = \frac{1}{N} \sum_{i=1}^{N} f_i. \tag{24}$$

The square root of *RE* is known under the term *root relative mean squared error RRSE*:

$$RRSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \frac{(f_i - \hat{f}(\mathbf{x_i}))^2}{(f_i - \overline{f})^2}} \tag{25}$$

Other error metrics that we have used are the *mean absolute error* (MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - \hat{f}(\mathbf{x_i})| \tag{26}$$

and its normalized variant the *relative mean absolute error* (RMAE):

$$RMAE = \frac{N \cdot MAE}{\sum_i |f_i - \overline{f}|} \tag{27}$$

The RE, RRSE and RMAE errors are nonnegative. For acceptable models, they should have values smaller than 1: $0 \leq RE \leq 1$, $0 \leq RRSE \leq 1$, $0 \leq RMAE \leq 1$. If for some function the relative error, the root relative mean square error or the relative mean absolute error is greater than 1, the model is completely useless. Namely, $RE = 1$, $RRSE = 1$ and $RMAE = 1$ are trivially achieved with the baseline predictor $\hat{f} = \overline{f}$.

A different type of measure that quantifies the statistical correlation between actual function values f_i and the predicted function values $\hat{f}_i$ for a dependent regression variable is the Pearson's correlation coefficient (CC):

$$CC = \frac{S_{f\hat{f}}}{S_f \cdot S_{\hat{f}}} \tag{28}$$

where

$$S_{f\hat{f}} = \frac{\sum_{i=1}^{N}[(f_i - \overline{f})(\hat{f}_i - \overline{\hat{f}})]}{N-1}$$

$$S_f = \frac{\sum_{i=1}^{N}(f_i - \overline{f})^2}{N-1}$$

$$S_{\hat{f}} = \frac{\sum_{i=1}^{N}(\hat{f}_i - \overline{f})^2}{N-1}$$

$$\overline{f} = \frac{1}{N}\sum_{i=1}^{N} f_i \text{ and } \overline{\hat{f}} = \frac{1}{N}\sum_{i=1}^{N}\hat{f}_i.$$

The correlation coefficient has values in the range [-1,1], where 1 stands for perfect positive correlation, -1 stands for perfect negative correlation, and 0 stands for no correlation at all. Thus, we are interested in positive values of correlation. As opposed to the mean squared error and the mean absolute error that need to be minimized, the learning algorithm aims to maximize the correlation coefficient. It should be also noted that the Pearson's correlation coefficient measures the linear correlation between the two variables.

The *mean square error* and the *mean absolute error* can be calculated incrementally by maintaining the sums of squared and absolute differences between the true function values f_i and the predicted function values $\hat{f}_i$. For their normalized versions, we need to incrementally calculate the sum of true function values $\sum_i^N f_i$, of predicted function values $\sum_i^N \hat{f}_i$, as well as the sums of squared predicted and true function values: $\sum_i^N \hat{f}_i^2$ and $\sum_i^N f_i^2$, and the number of observed training instances N. The later sums are necessary for the computation of the term $\sum_i^N (f_i - \overline{f})^2$, which can be decomposed into:

$$\sum_{i=1}^{N}(f_i - \overline{f})^2 = \sum_{i=1}^{N}(f_i - \frac{1}{N}\sum_{i=1}^{N} f_i)^2 = \sum_{i=1}^{N} f_i^2 + \frac{1}{N}(\sum_{i=1}^{N} f_i)^2. \tag{29}$$

The sum of their products $\sum_i f_i \hat{f}_i$ on the other hand is necessary for an online computation of the correlation coefficient. All of the above evaluation metrics can be applied in a straightforward way to evaluate multi-target predictive models: The necessary statistics (sums and counts) should be computed per target variable.

5.2.2 Metrics for Model's Complexity

For the case of interpretable models, it is also very important to have a model of low complexity, which can be interpreted easily. Having to deal with only regression trees and their variants, a straightforward way of measuring model complexity is to count the total number of nodes. If we are interested in measuring the number of rules we can extract from the tree, then a more suitable measure is the number of leaves in the tree, since each leaf corresponds to a different predictive rule (which is a conjunction of logical or comparison tests).

In the case of option trees, we use two different measures of complexity: the number of different trees represented with a single option tree, which gives us a measure which is comparable to the size of an ensemble, as well as the total number of leaves in the option tree, i.e., the number of different rules that can be extracted from the tree.

The algorithms are further expected to work under constrained resources in terms of memory, running time, and processing time per example. Therefore, we further measure the memory allocation (in MB) at any point in time during the learning process, as well as the total elapsed running time (in seconds).

5.2.3 Metrics for Change Detection

Another important dimension along which we evaluate an online learning algorithm, is its ability to adapt to concept drift. Among other measures of evaluation, it is also very important to evaluate the performance of the change detection method and the adaptation method of the algorithm. This would tell us how fast and how well the algorithm will recover and repair the inferred model after a change in the target representation.

Let us first illustrate the corresponding problem statement. Assume that one observes a piecewise constant signal disturbed by white noise. Let us assume an unknown number of jumps in the mean which occur at unknown points in time. The problem of online detection relevant for our work is the online detection of such jumps, as quickly as possible, in order to allow the detection of near successive jumps.

In this context, the adequate evaluation metrics for online change detection are: the number of false alarms, the number of true positives, and the detection delay. In a probabilistic manner of thinking, we are thus interested in the following metrics:

- *Probability of false alarms*: Measures the robustness and the sensitivity to noise. In other words, we are typically interested in a high mean time between false alarms.
- *Probability of true positives*: Quantifies the capacity of the method to detect all changes (all of the jumps in the signal).
- *Mean detection delay*: The mean time between the change (jump) and the alarm time, conditioned by a nonexistence of false alarms before the change (jump). Naturally, it is highly desirable that the alarm is given without any delay, with as few lost observations as possible.

There exists a close connection between the ability of quick change detection and the sensitivity of the method. The ability of quick change detection causes the detector to be sensitive and thus increases the risk of false alarms. In conclusion, a detector would perform optimally if, for a fixed mean time between false alarms, the delay for detection is minimized (Basseville and Nikiforov, 1993).

The above probabilities can be easily estimated when the evaluation of the change detection method is performed in a controlled environment, through counting the false alarms and the true positives over the total number of changes artificially introduced in the target function. However, it is much more difficult to assess the quality of change detection and

adaptation methods for real-world datasets, in which the number of changes in the target function cannot be known up-front, considering that the target function is unknown itself. The best evaluation method in such a scenario is to track the performance of the learning algorithm over a testing set of most recent examples, and consider an alarm for a change when its performance starts to degrade drastically.

5.3 Evaluation Approaches

There are two general approaches for evaluating online learning algorithms, which correspond to two basic methods for evaluation:

1. *Holdout Evaluation* - Periodical evaluation of the learned model is performed at some pre-defined time intervals, using a separate holdout set of testing examples. This method is a natural extension of the batch holdout method in which a (large enough) set of examples is separated from the training set and used for testing purposes Dietterich (1998).

2. *Prequential Evaluation* - The *predictive sequential* or *prequential* analysis is a sequential hypothesis testing method, where the sample size is not fixed in advance. Sequential analysis methods have been initially introduced by Wald (1947), and have been recently used by several authors (Castillo and Gama, 2005; Gama et al., 2009; Hulten et al., 2001).

5.3.1 Holdout Evaluation

In the *holdout* evaluation approach the error metrics are estimated using a separate holdout set of unseen examples, i.e., examples which have not been used for training. The error estimates are for that reason unbiased. This method should provide the most reliable estimate of the expected performance of the algorithm over any non-observed data in the near future.

The holdout evaluation can be implemented by applying a buffer over the data stream, which will continuously hold out a set of training examples. At the end of every testing phase the training examples will be given to the algorithm for updating the model.

A useful feature of this approach is that it facilitates comparison with streaming algorithms designed to perform over batches of examples. In this learning setup, every batch of examples can be used as a holdout set. A downside of this approach is that it requires additional memory to store the testing examples.

5.3.2 Prequential Evaluation

The *prequential* evaluation methodology consists of a test-and-train procedure. The model is first tested on a testing example, whenever one is available, immediately thereafter the same example is used for training.

The error statistics are usually maintained from the beginning of the learning process. Let $L(f,\hat{f})$ denote the loss function chosen to evaluate the goodness of our induced function $\hat{f}$ with respect to the true function f. The predictive sequential loss is computed by accumulating the sum of the computed losses after every test-then-train procedure for N examples observed from the data stream:

$$S_i = \sum_{i=1}^{N} L(f_i, \hat{f}_i). \tag{30}$$

Taking into account that a learning algorithm that continuously updates its theory as more information becomes available performs poorly at the beginning of learning, the measured quantities give a rather pessimistic picture of the overall performance of the algorithm.

For example, *prequential* evaluation enables us to obtain learning curves which give insight into the evolution of the learning process. Very often, however, current improvements of the model cannot be easily seen due to early mistakes.

For this reason various techniques, such as sliding windows of the most recent observed errors or fading factors can be applied. These techniques discard past mistakes or reduce their influence on the performance measures. Gama et al. (2009) in particular defend the use of predictive sequential error estimates using fading factors to assess the performance of online learning algorithms in the presence of concept drifts.

The prequential error estimated using fading factors is computed as:

$$E_i = \frac{S_i}{B_i} = \frac{L_i + \alpha \times S_{i-1}}{1 + \alpha \times B_{i-1}} \tag{31}$$

where $L_i = L(f_i, \hat{f}_i)$, $S_1 = L_1$, $B_1 = 1$, and i is the number of examples observed thus far. As we can see the fading factor is also used in the denominator of the equation to correct for the fact that $E_i \to 0$ when $i \to \infty$. They show that the prequential error with a fading factor of $\alpha = 0.975$ converges as fast to the holdout estimate as the prequential error without a fading factor estimated using a sliding window of size 50k examples.

In our experimental evaluation, we have used both evaluation methods. We have used the prequential error estimated using a sliding window, which we refer to as *prequential-window*, as well as the prequential error estimated using a fading factor, referred to as *prequential-fading*. The size of the sliding window or the value of the fading factor determine the level of aggregation and can be adjusted by the user. The sliding step in the *holdout* and the *prequential-window* evaluation determines the level of detail or the smoothness of the learning curve.

5.4 Online Bias-Variance Analysis

The *bias-variance decomposition* of the squared loss is a very useful analytical tool for evaluating the quality of models. Beside the standard bias-variance decomposition of the squared loss by Domingos (2000), defined with Equation (12), we will also use the *bias-variance-covariance* decomposition of the squared loss (defined by Equation (14)) for evaluating the quality of an ensemble. The procedure for evaluation is the same for both types of decomposition.

The batch method for *bias-variance* analysis typically consists of a 10-fold cross-validation sampling procedure for estimating the components of the mean squared error, which has been reported to provide the most reliable estimates (Bouckaert, 2008). The models are derived from 10 independent training sets and their predictions are logged over 10 independent test sets. For each test set an averaged bias and variance component is obtained. The final estimates are the averages of the 10 independent estimates of the bias and the variance. In the streaming scenario we cannot perform 10-fold cross validation. However, we can obtain 10 different bootstrap training datasets in an online fashion, and we can use a single holdout test set over which the predictions of the different estimators can be compared.

The previously described *holdout* evaluation method is very useful for online bias-variance profiling. The evaluation procedure consists of ten rounds of sampling with replacement using the online bootstrap sampling method of Oza and Russell (2001). Each round of sampling provides a different training dataset. On the other hand, the holdout set at a given moment is equal in all of the 10 rounds. In every holdout evaluation, which is performed periodically at predefined time intervals, the predictions of each estimator are logged. These are sufficient to compute the *bias-variance* and the *bias-variance-covariance* decompositions, which will be logged and recomputed in the next holdout evaluation round. This enables us to monitor the evolution of the components of the squared error over time. This evolution can give us insight into what affects the learning process and which component of the error

needs to be managed better. All of the examples used for testing are buffered in the order of their arrival and sequentially dispatched to the learning algorithm for further training.

5.5 Comparative Assessment

One of the required tasks in the evaluation of newly developed machine learning algorithms is a comparative assessment with respect to existing learning methods. Here the implicit hypothesis is that the enhancements proposed in the new learning algorithm yield an improved performance over the existing algorithm(s). Various statistical tests are used to verify the hypothesis of improved performance, among which the Friedman test followed by the Nemenyi post-hoc test has been argued as one of the simplest and most robust non-parametric tests for statistical comparison of multiple classifiers or predictors over multiple datasets (Demšar, 2006).

The test proposed by Friedman (1937, 1940) is commonly used to detect differences in treatments across multiple test attempts. It deals with the problem of multiple hypothesis testing, where the usual goal is to control the *family-wise error*, i.e., the probability of making at least one Type I error in any of the comparisons. It is a non-parametric equivalent of the repeated-measures ANOVA.

The Friedman test consists of an initial ranking of the algorithms for each dataset separately, the best performing algorithm getting the rank of 1, the second best the rank of 2, etc. Let r_i^j be the rank of the j-th of k algorithms on the i-th of N datasets. Let $R_j = \frac{1}{N}\sum_i r_i^j$ denote the average ranks of the algorithms, for $j = 1,...k$. The null-hypothesis is that all the algorithms are equivalent and that ranks are equal.

The Friedman statistic

$$\chi_F^2 = \frac{12N}{k(k+1)}[\sum_j R_j^2 - \frac{k(k+1)^2}{4}] \tag{32}$$

is distributed according to χ_F^2 with $k-1$ degrees of freedom, when N and k are large enough. A better statistic, as argued by Iman and Davenport (1980), is the one used in our comparative assessment of algorithms: This is F_F statistic, defined as:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}. \tag{33}$$

The F_F statistic is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

If the null-hypothesis is rejected, the verification proceeds with a post-hoc test. For comparing the average ranks of two algorithms in a set of two-by-two comparisons, we have used the Nemenyi test (Nemenyi, 1963). The performance of two classifiers is *significantly different* if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{34}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. Details on the tables with critical values are given by Demšar (2006).

When all algorithms are compared with a control algorithm, it is advised to use a Bonferoni correction or a similar procedure instead of the Nemenyi test . The test statistic which we used, as proposed by Demšar (2006), for comparing the i-th and the j-th algorithm is:

$$z = (R_i - R_j)/\sqrt{\frac{k(k+1)}{6N}}.$$

The z value is further used to find the corresponding probability from the table of the normal distribution, which is then compared with an appropriate α.

The Wilcoxon signed-ranks test is another non-parametric alternative to the paired t-test. It ranks the differences in performance of two classifiers for each dataset, ignoring the signs, and compares the ranks for the positive and the negative differences. This test does not assume normal distributions, and is less sensitive to outliers (exceptionally good/bad performances on a few datasets). Demšar (2006) recommends the Wilcoxon signed-ranks test as more sensible than the t-test. We used this test in the statistical comparative assessment when we compared only two learning algorithms.

Let d_i be the difference between the performance scores of the two predictive models on the i-th out of N datasets. The differences are ranked according to their absolute values and average ranks are assigned in case of ties. Let R^+ be the sum of ranks for the datasets on which the second algorithm outperformed the first, and R^- the sum of ranks for the opposite. Ranks for $d_i = 0$ are split evenly among the sums; if there is an odd number of them, one is ignored:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$

Let T be the smaller of the sums, $T = \min(R^+, R^-)$. The difference is significant if the value of T is greater than the corresponding critical value. The critical values for T for a number of datasets up to 25 (or sometimes more) can be found in most text books on statistics.

In a final note, we would like to discuss an alternative approach which has not been used often in the literature of machine learning, and has been inspired by some recent advancemens in the field of stream data mining. The (additive) Hoeffding bounds (Hoeffding, 1963) can also be used to perform a comparative assessment, by treating the difference in the values of the loss functions of two learning algorithms as a random variable. The Hoeffding's inequality provides an upper bound on the probability that the sum of random variables deviates from its expected value.

Let A and B denote two learning algorithms whose performance we would like to compare using an appropriate loss function L. Let $L(A, \mathbf{x_i})$ and $L(B, \mathbf{x_i})$ denote the losses calculated for A and B given an input instance $\mathbf{x_i}$. Let $\Delta_i = \Delta(\mathbf{x_i}) = L(A, \mathbf{x_i}) - L(B, \mathbf{x_i})$ denote the corresponding difference in loss.

The general case, under which the Hoeffding's inequality applies, is exemplified with a sequence of random variables $X_1, X_2, ..., X_n$ for which we assume that they are almost surely bounded; that is, for $1 \leq i \leq n$ we can assume that $P(X_i \in [a_i, b_i]) = 1$, where $a_i, b_i \in \mathscr{R}$. Let $\overline{X} = 1/n \sum_i X_i$ be the empirical mean of these variables. Theorem 2 of Hoeffding (1963) proves the inequality

$$Pr(\overline{X} - E[\overline{X}] \geq \varepsilon) \leq e^{-\frac{2\varepsilon^2 n^2}{\sum_i^n (b_i - a_i)^2}} \tag{35}$$

which is valid for positive values of ε.

The applicability of this result is interesting from the aspect of determining the moment or the amount of training instances after which one can claim that a given method has a statistically supported (significant) advantage in terms of the used loss evaluation function. All we need is to bound the random variables observed and maintain their empirical mean at all times. This can be easily achieved by defining our random variables as $X_i = sgn(\Delta_i)$ taking on values in {-1, 1}. Let $X = \sum_i X_i$. Without any assumptions on the advantage of one method over the other the probability of the random variables X_i taking on any of the

values, -1 or 1, is equal; that is, $Pr(X_i = -1) = Pr(X_i = 1) = 1/2$. Then,

$$Pr(X \geq \varepsilon) \leq e^{\frac{-\varepsilon^2}{2n}} \tag{36}$$

for any $\varepsilon > 0$. By assigning a constant value δ to this probability, one can calculate the value of ε with:

$$\varepsilon \leq \sqrt{\frac{\ln(1/\delta)}{2n}}.$$

In this case, ε quantifies the advantage of algorithm A over algorithm B, due to its positive value. The procedure can be run like two online tests in parallel. When one of them satisfies the desired (ε, δ) approximation in the comparison, we can assert the advantage of one method over the other. However, it is possible that this difference would fluctuate around a zero mean, which can be interpreted as a situation where none of the algorithms has a statistically significant advantage over the other. This kind of procedure is very useful for any type of comparative assessment in the online learning scenario.

5.6 Datasets

Two different prediction tasks are addressed in this thesis, i.e., predicting the value of a single target attribute and predicting the values of multiple target attributes. For each of the predictive tasks, we have generated multiple artificial datasets and used multiple datasets coming from real-world problems. In the following sections, we present the methods used for generating our artificial data and describe the characteristics of every individual real-world dataset used in our experimental evaluation.

5.6.1 Artificial Datasets

Artificial datasets are extremely useful for simulating various specific problems that our algorithms are expected to handle effectively. This approach is common in the machine learning literature. As a result, various methods have been proposed by different authors. In this dissertation, we use three well known functions for generating artificial data:

- *Fried* - This dataset is generated by using a function proposed by Friedman (1991). It contains 10 continuous attributes, whose values are independently distributed according to a uniform distribution in the interval [0, 1]. From those, only 5 attributes are relevant for predicting the target value, while the rest are redundant. We generate a random vector of 10 real values, each in the interval [0,1]. Each random vector represents a training example, whose true target value is computed by using the following function:

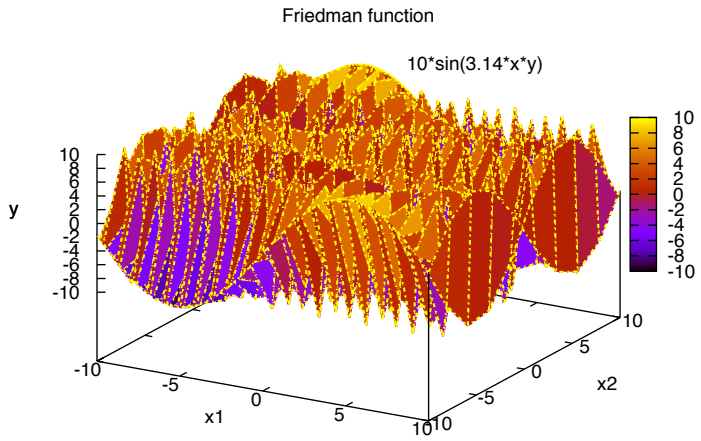
$$y = 10 \cdot \sin(\pi \cdot x_1 \cdot x_2) + 20 \cdot (x_3 - 0.5)^2 + 10 \cdot x_4 + 5 \cdot x_5 + \sigma(0,1), \tag{37}$$

  where $\sigma(0,1)$ is a random number generated from a normal distribution with mean 0 and variance 1. An illustration of Friedman's function, shown on a limited part of the space defined by its first two dimensions, is given in Figure 5.

- *Losc and Lexp* - These datasets have been both proposed by Karalič (1992). The Losc and Lexp datasets represent functional dependencies between 5 input attributes, of which 1 discrete valued and 4 real valued attributes, and the target attribute. These functions are similar to the Friedman's function, with the difference that in this case all of the 5 attributes are relevant for the prediction task. These datasets were used in order to compare our results with published results of other authors.

These artificial datasets are used in the first part of the evaluation, performed on stationary streams.

Figure 5: An illustration of Friedman's function viewed on a rectangle over the two dimensions $x = x_1$ and $y = x_2$.



5.6.1.1 Concept Drift

To simulate different types of concept drift, we have created modified versions of the above Friedman function and merged them correspondingly. Simulating concept drift enables us to control the relevant parameters when evaluating the quality of the drift detection and adaptation mechanisms. Three different scenarios of concept drift were used in our experimental evaluation:

1. **Local expanding abrupt drift** - The first type of simulated concept drift is *local* and *abrupt*. The concept drift is localized in two distinct regions of the instance space. Let x_1, x_2, x_3, x_4, x_5 denote the five relevant attributes which are used in Friedman's function. The first region with drift is defined with the conjunction of inequalities $R_1 = \{x_2 < 0.3 \wedge x_3 < 0.3 \wedge x_4 > 0.7 \wedge x_5 < 0.3\}$ and the second region is defined with the conjunction of inequalities $R_2 = \{x_2 > 0.7 \wedge x_3 > 0.7 \wedge x_4 < 0.3 \wedge x_5 > 0.7$. These regions basically correspond to small sub-trees of the induced regression tree. Let N denote the size of the dataset. There are three points of abrupt change in the training dataset, the first one at $\frac{1}{4}N$ examples, the second one at $\frac{1}{2}N$ examples and the third at $\frac{3}{4}N$ examples. During the first stationary period the algorithm is trying to model the Friedman function. At the first and the second point of change we introduce the modified functional dependencies and expand the regions R_1 and R_2. The details are given below:

   - (I change point) For the examples falling in region R_1, the new functional dependency is given with: $y_{R_1} = 10 \cdot x_1 \cdot x_2 + 20 \cdot (x_3 - 0.5) + 10 \cdot x_4 + 5 \cdot x_5 + \sigma(0,1)$. All other examples are assigned target values according to the original Friedman function. An illustration of this new function, shown on a limited part of the space defined by its first two dimensions, is given in Figure 6.
   - (II change point) We introduce a new functional dependency for region R_2 given with $y_{R_2} = 10 \cdot \cos(x_1 \cdot x_2) + 20 \cdot (x_3 - 0.5) + e^{x_4} + 5 \cdot x_5^2 + \sigma(0,1)$. Both R_1 and R_2 are

expanded by removing the last inequality from their definition ($x_5 < 0.3$ and $x_5 > 0.7$, respectively). By expanding the region where the changes in the functional dependency are valid, larger sub-trees of the induced regression tree are affected. Thus, the changes of the model are expected to be more severe.

- (III change point) The introduced functional dependencies y_1 and y_2 remain the same, while both R_1 and R_2 are further expanded by removing the last inequalities from their modified definitions ($x_4 > 0.7$ and $x_4 < 0.3$, respectively). Note that the learning algorithm has already been exposed to the changes in the function and has presumably taken actions to adapt the tree. Therefore, the effect of the change is expected to be less severe.

2. **Global reoccurring abrupt drift** - The second type of simulated change is *global* and *abrupt*. The concept drift appears over the whole instance space. We have introduced two points of change, the first occurs at $\frac{1}{2}N$ examples and the second at $\frac{3}{4}N$ examples. The change is performed by simply changing the place of every input variable with another input variable. At the first point of change, the new functional dependency is given with $y_{gl} = 10 \cdot \sin(\pi \cdot x_4 \cdot x_5) + 20 \cdot (x_2 - 0.5)^2 + 10 \cdot x_1 + 5 \cdot x_3 + \sigma(0,1)$. At the second point of change, we simulate a reoccurring concept by reverting the existing function y_{gl} to the original one.

3. **Global and slow gradual drift** - The third type of simulated drift is *global* and *gradual*. A gradual concept drift is simulated by gradually introducing training examples which belong to a different functional dependency among the majority of training examples which correspond to the initial function that generates the data. A gradual concept drift is much more difficult to handle because the old model is still valid for some instances, while other instances follow a different model. An algorithm would basically need to find a way to assign the proper examples to their corresponding model tree. The gradual change is introduced once at $\frac{1}{2}N$ examples from the start of the streaming, and for the second time at $\frac{3}{4}N$ examples from the start. The particular details of the method we used are as follows:

   - (I change point) The first gradually introduced concept is defined with the following function: $y_{glr_1} = 10 \cdot \sin(\pi \cdot x_4 \cdot x_5) + 20 \cdot (x_2 - 0.5)^2 + 10 \cdot x_1 + 5 \cdot x_3 + \sigma(0,1)$. Starting from this point, examples from the new concept are being gradually introduced among the examples from the old concept by increasing their probability of being included in the training data. After 100k examples the data stream consists only of examples which represent the new concept y_{glr_1}.
   - (II change point) The second gradually introduced concept is defined as $y_{glr_2} = 10 \cdot \sin(\pi \cdot x_2 \cdot x_5) + 20 \cdot (x_4 - 0.5)^2 + 10 \cdot x_3 + 5 \cdot x_1 + \sigma(0,1)$. The examples from the new concept will gradually replace the ones from the last concept given by y_{glr_1} in the same way as before. The drift ends after 100k examples.

5.6.1.2 Multiple Targets

Due to the scarcity of streaming datasets appropriate for the task of predicting multiple numerical targets, we have again simulated several artificial datasets. We have generated 3 artificial datasets using concepts from trees in which the targets are *related* (in one case) and *unrelated* (in the other case). In particular, a multi-target regression tree is generated at random, such that all of the splitting attributes and splitting thresholds are chosen randomly, as well as the numerical predictions per target in the leaf nodes. For every numerical target, a separate standard deviation is chosen, which is used for assigning the true value of the

Figure 6: An illustration of the first variant of the modified Friedman's function used to simulate local expanding abrupt drift, viewed on a rectangle over the two dimensions $x = x_1$ and $y = x_2$.

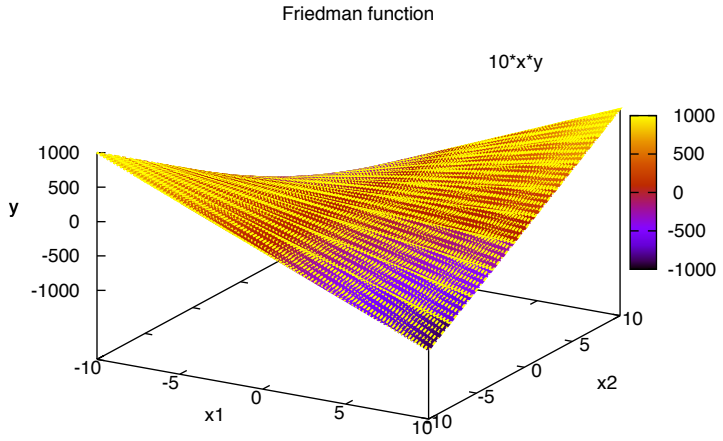


Table 1: Datasets used in the experimental evaluation and their properties: domain name, number of instances (N), number of input attributes (Attr), number of target attributes (T), number of tree levels (L), and description.

Domain	N	Attr	T	L	Description
Sim1	300000	10	5	5	Tree concepts with *related* targets
Sim2	300000	10	5	5	Tree concepts with *unrelated* targets
Sim3	300000	10	5	10	Tree concepts with *unrelated* targets
Fried*	300000	10	3	NA	Friedman's function with *unrelated* targets
Fried**	*infinite*	10	3	NA	An infinite stream of Fried* data

artificially generated training/testing examples. All of the values are chosen randomly with a standard normal distribution.

Table 1 lists all of the artificial datasets that we have generated along with their properties (domain name, number of examples, number of input attributes, number of target attributes, depth of the tree and a description). When targets are *related*, they are given constant but different values for each region of the input space. Since the trees are piece-wise constant, the targets practically behave in the same way, and are thus considered as related. When targets are said to be *unrelated* (in the case of Sim2 and Sim3) the prediction for each target is computed using a simple linear equation. By using a linear plane in the leaf nodes, the regression surface of the tree resembles a mesh of two-dimensional planes for each target. Due to the random choice of linear equations, the shape of the regression surface is not the equal for all of the targets: They behave differently in the same regions of the input space and are thus considered non-related.

The first dataset (Sim1) resembles a "prefect" regression tree of depth 5 (consisting of 5 levels), while the second dataset (Sim2) resembles a less discrete, but still non-smooth regression surface represented by a tree of the same depth. For both datasets, one of the target attributes has a higher standard deviation than the remaining targets. This would make the prediction and the modeling task much more difficult. The third artificial dataset (Sim3) is similar to Sim2 with two important differences: 1) the concepts are more complex (the synthetic random tree is much larger), 2) all of the targets have deviations which belong to the same range of values.

We have again used the Friedman's function to prepare yet another artificial dataset for a multi-target numerical prediction. We have 2 datasets, one of 300k examples and one very large (infinite), both representing a 3 target prediction task. As before, the original function was modified for two of the target attributes, but only in predefined regions of the instance space (eg. $x_3 > 0.70$, $x_4 < 0.30$ and $x_5 > 0.70$ is one of the regions). In this way, we can simulate a situation in which some of the targets behave differently for specific regions of the input space, or represent a special case of the target attribute given with the original Friedman function.

5.6.2 Real-World Datasets

The algorithms proposed in this thesis are designed for scenarios when data is abundant and delivered through a high-volume open-ended data stream. In reality, the publicly available datasets adequate for the regression task are rather small or do not contain changes in the functional dependencies. For a comparative evaluation, we have used all the available regression datasets of sufficient size. For the comparative assessment of performance on stationary data, we chose the 9 largest available "benchmark" datasets from the UCI Machine Learning Repository (Frank and Asuncion, 2010), the Delve Repository[1] and the StatLib System's[2] site. Their characteristics are given in Table 2.

Table 2: Characteristics of the benchmark datasets used for the experimental evaluation.

Dataset Name	Size	Number of attributes Nominal	Numerical	Values of the target Mean ± Std.
Abalone	4177	1	8	9.934 ± 3.224
Cal Housing	20640	0	9	206855.817 ± 115395.616
Elevators	16599	0	19	0.022 ± 0.007
House 8L	22784	0	9	50074.44 ± 52843.476
House 16H	22784	0	17	50074.44 ± 52843.476
Mv Delve	40969	3	8	-8.856 ± 10.420
Pol	15000	0	49	28.945 ± 41.726
Wind	6574	2	13	15.599 ± 6.698
Wine Quality	4898	0	12	5.878 ± 0.886

We have also used 3 other real-world datasets whose size is significantly larger compared to the datasets above. We discuss their properties in the following subsections.

5.6.2.1 Protein 3D Structure Prediction

The Infobiotics PSP [3] repository contains real-world tunable scalability benchmarks for evaluation of machine learning methods for classification and regression. PSP stands for Protein Structure Prediction, i.e., prediction of the 3D structure of a protein based on its primary sequence (a chain of amino-acids). The datasets in this repository are not concerned with predicting the exact 3D PSP problem, but rather with predicting structural features of amino acids. A good prediction of these features contributes greatly to obtaining better models for the exact 3D PSP problem. Most of these structural features are defined as continuous variables, giving rise to regression problems. However, it is also usual to discretize them and treat the resulting problems as classification problems.

The main dataset is derived from a set of 1050 protein chains and approximately 260000 amino acids. There are 140 versions of the same prediction problem: 120 in the classification domains, and 20 in the regression domain. Some versions use only discrete attributes, other

[1] http://www.cs.toronto.edu/ delve/data/datasets.html
[2] http://lib.stat.cmu.edu/index.php
[3] http://icos.cs.nott.ac.uk/datasets/psp_benchmark.html

only continuous. The number of attributes ranges from 1 to 380. The size of each dataset is 257760 instances.

Each numerical attribute represents one value of the Position-Specific Scoring Matrices (PSSM) profile of a single amino acid from the primary protein sequence. Each PSSM profile consists of 20 numerical values. The target attribute is the structural feature of one of the amino acids, positioned at a specified location in the chain sequence. The task is to predict the structural feature of the target from the local context, that is, given the window of neighbouring amino acids. For the purpose of our study, we have used the largest and most difficult dataset with 380 numerical attributes, which corresponds to a window of $\pm$ 9 amino acids, positioned left and right of the target.

5.6.2.2 City Traffic Congestion Prediction

This dataset was generated for the data mining competition that was part of the IEEE International Conference on Data Mining 2010 (ICDM)[4]. The task is to predict city traffic congestion based on simulated historical measurements. A time series of simulated congestion measurements from 10 selected road segments of Warsaw are given. There are two values recorded at each time point for a given segment, corresponding to two opposite directions of traffic flow. As a result, there are 20 possible target attributes. However, if we choose one target the rest of the remaining target attributes can be used as descriptive attributes. Congestion (the number of cars that passed a given segment) is measured in consecutive 1-minute periods of time. The training dataset which we created for the purpose of predicting the congestion at segment 10 in direction 2 is described by 619 numerical attributes. In particular, we have used the two values measured for a given time point and a given road segment plus the corresponding lagged values measured in the last half an hour, resulting in 30 values per direction. The size of the training dataset is 59952 examples.

5.6.2.3 Flight Arrival Delay Prediction

For evaluation on non-stationary real-world problems, we prepared a dataset using the data from the Data Expo competition (2009)[5]. The dataset consists of a large number of records, containing flight arrival and departure details for all the commercial flights within the U.S., from October 1987 to April 2008. The dataset was cleaned and records were sorted according to the arrival/departure date (year, month, and day) and time of flight. Its final size is around 116 million records and 5.76 GB of memory. The data is described by 13 attributes: *Year, Month, Day of Month, Day of Week, CRS Departure Time, CRS Arrival Time, Unique Carrier, Flight Number, Actual Elapsed Time, Origin, Destination, Distance,* and *Diverted*. Details on their meaning can be found at the DataExpo09's site. The target variable is Arrival Delay, given in seconds. Given that plotting and analyzing such a large number of evaluation results is difficult, we decided to use only a portion of the dataset that corresponds to the records in the period from January 2008 to April 2008. The size of this dataset is 5.810.462 examples.

5.6.2.4 Datasets with Multiple Targets

One of the real-world multi target prediction datasets we have used in our experimental evaluation is the Vegetation Condiiton problem (Kocev et al., 2009). This dataset is for prediction of the vegatation condition in Victoria State, Australia. Data provider is Matt White from the Arthur Rylah Institute for Environmental Research, Department of Sustainability and Environment (DSE) in Victoria, Australia. The prediction task is to determine the environmental characteristics of a site such as climatic, radiometric and topographic data from

[4]http://tunedit.org/challenge/IEEE-ICDM-2010
[5]http://stat-computing.org/dataexpo/2009

intrinsic characteristics of the species present at that site such as physiognomy, phenology and phylogeny data. The relationship between site properties and species properties is the functional dependency which we try to model by using a multi-target regression tree. The dataset consist of 16967 records, with 40 descriptive attributes and 6 target attributes.

The second real-world dataset, which we will refer to as Kras, represents a collection of satellite measurements and 3D data obtained with the Light Detection And Ranging (LiDAR) technology for measuring different forest properties. Multi-spectral satellite imagery is able to capture horizontally distributed spatial conditions, structures and changes. However, it cannot capture the 3D forest structure directly and is easily influenced by topographical covers and weather conditions. LiDAR, on the other hand, provides horizontal and vertical information (3D) at high spatial resolution and vertical accuracies. By combining satellite sensed data with LiDAR data, it is possible to improve the measurement, mapping and monitoring of forest properties and provide means of characterizing forest parameters and dynamics (Stojanova et al., 2010). The main objective is to estimate the vegetation height and canopy cover from an integration of LiDAR and satellite data in a diverse and unevenly distributed forest in the *Kras* (Karst) region of Slovenia. The dataset contains 60607 records, described with 160 numerical attributes and 2 target attributes.

6 Learning Model Trees from Time-Changing Data Streams

Panta rei ... everything is constantly changing, nothing remains the same!

Heraclitus

One of the main issues in traditional machine learning has been the problem of inferring high quality models given small, possibly noisy datasets. However, in the past two decades a different type of problems has arisen, motivated by the increasing abundance of data that needed to be analyzed. This has naturally shifted the focus towards various approaches that limit the amount of data necessary to induce models of the same quality as those induced over the whole training sample.

Although the task of learning from data streams poses some different challenges, it still requires a solution to the problem of learning under an abundance of training data. In the previous chapters we discussed several approaches to the problem that has been at the core of our research work: *What is the minimum amount of data that can be used to make an inductive selection or a stopping decision without compromising the quality of the learned regression tree?* The main conclusion is that, there is no single best method that provides a perfect solution. Rather, there are various techniques each with a different set of trade-offs. The general direction is given by the framework of sequential inductive learning, which promotes a statistically sound sampling strategy that will support each inductive decision.

In this chapter, we describe an incremental algorithm called FIMT-DD, for learning regression and model trees from possibly unbounded, high-speed and time-changing data streams. The FIMT-DD algorithm follows the approach of learning by using a probabilistically defined sampling strategy, coupled with an advanced automated approach to the more difficult problem of learning under non-stationary data distributions. The remainder of this chapter is organized as follows. We start with a discussion on the ideas behind the sequential inductive learning approach, which represents a line of algorithms for incremental induction of model trees. The next section discusses the probabilistic sampling strategy that is used to provide the statistical support for an incremental induction of decision trees. It gives the necessary background for introducing the relevant details of the algorithms we propose in the succeeding sections, for learning regression and model trees from data streams. Furher, we present our change detection and adaptation algorithms, each discussed in a separate section. The last section contains the results from the experimental evaluation on the performance of the algorithms presented in the previous sections.

6.1 Online Sequential Hypothesis Testing for Learning Model Trees

The main idea behind the sequential inductive learning approach is that each inductive decision can be reformulated as the testing of a hypothesis over a given sample of the entire

training set. By defining the level of confidence (δ), the admissible error in approximating the statistic of interest (ε), and the sampling strategy, one can determine the amount of training data required to accept or reject the null-hypothesis and reach a decision. This approach not only enables us to reach a stable statistically supported decision but also enables us to determine the moment *when* this decision should be made.

In Chapter 3 we have discussed two batch-style algorithms for learning linear model trees by a sequential hypothesis testing approach. Both CRUISE by Kim and Loh (2001) and GUIDE by Loh (2002) rely on a χ^2 testing framework for split test selection. GUIDE computes the residuals from a linear model and compares the distributions of the regressor values from the two sub-samples associated with the positive and negative residuals. The statistical test is applied in order to determine the likelihood of the examples occurring under the hypothesis that they were generated from a single linear model, and therefore a confidence level can be assigned to the split.

The same approach has been used in the incremental model tree induction algorithms BatchRA and BatchRD, as well as their online versions OnlineRA and OnlineRD, proposed by Potts and Sammut (2005). The algorithms follows the standard top-down approach of building a tree, starting with a selection decision at the root node. The null hypothesis is that the underlying target function f is a good linear approximation over the data in the complete node, i.e.,

$$H_0 : f(\mathbf{x}) = x^T \theta$$

where θ is a column vector of d parameters, given that we have d regressors (predictive attributes), and $\mathbf{x}$ is a vector of d attributes. Three linear models are further fitted to the examples observed at the node: $\hat{f}(\mathbf{x})$ using all N examples; $\hat{f}_1(\mathbf{x})$ using N_1 examples lying on one side of the split; and $\hat{f}_2(\mathbf{x})$ using N_2 examples lying on the other side of the split. The residual sums of squares are also calculated for each linear model, and denoted RSS_0, RSS_1 and RSS_2 respectively.

When the null hypothesis is not true, $RSS_1 + RSS_2$ will be significantly smaller than RSS_0, which can be tested using the Chow test (Chow, 1960), a standard statistical test for homogeneity amongst sub-samples. As noted by Potts and Sammut (2005), there is clearly no need to make any split if the examples in a node can all be explained by a single linear model. On the other hand, the node should be split if the examples suggest that two separate linear models would give significantly better predictions.

In econometrics, the Chow test is most commonly used in time series analysis to test for the presence of a structural break. The test statistic follows the F distribution with d and $N-2d$ degrees of freedom and is computed by using the following equation:

$$F = \frac{(RSS_0 - RSS_1 - RSS_2)(N-2d)}{d(RSS_1 + RSS_2)}. \tag{38}$$

The split least likely to occur under the null hypothesis corresponds to the F statistic with the smallest associated probability in the tail of the distribution. Therefore, if the null hypothesis can be rejected under the available sample of training data with the desired degree of confidence (p), the best splitting test is determined with the minimum probability value denoted with α. However, if the value of p is not small enough to reject the null hypothesis with the desired degree of confidence, no split should be made until further evidence is accumulated. For example, if $\alpha = 0.01\%$ a split is only made when there is less than a 0.01% chance that the data observed in the node truly represent a linear dependence between the target attribute and the predictor attributes (the regressors).

A major disadvantage of fitting linear models on both sides of every possible splitting point is the fact that this is intractable for the case of numerical (real-valued) attributes. The problem is partially solved by using a fixed number of k candidate splits per regressor.

Another disadvantage of the proposed method is its computational complexity, which tends to be high for a large number of regressors. The residual sums of squares are computed incrementally by using the recursive least squares (RLS) algorithm. However, each leaf stores $k(d-1)$ models which are incrementally updated with every training example. Each RLS update takes $O(d^2)$ time, hence the overall training complexity of BatchRD is $O(Nkd^3)$ where N is the total number of examples. The online version of this algorithm is termed OnlineRD.

As an alternative, a more efficient version of the same algorithm is proposed, which is based on a different splitting rule. The same splitting rule has been used in the batch algorithms SUPPORT by Chaudhuri et al. (1994) and GUIDE by Loh (2002). The algorithm computes the residuals from a linear model fitted in the leaf and compares the distributions of the regressor values from the two sub-samples associated with the positive and negative residuals. The hypothesis is that if the function being approximated is almost linear in the region of the node, then the positive and the negative residuals should be distributed evenly. The statistics used are differences in means and in variances, which are assumed to be distributed according to the Student's t distribution. These statistics cannot be applied for evaluating splits on categorical attributes, which are therefore ignored. The online version of this algorithm is termed OnlineRA.

The stopping decisions within the sequential inductive learning framework can be evaluated in an explicit or in an implicit manner. Whenever there is not enough statistical evidence to accept the null hypothesis, an implicit temporal stopping decision is made, which (in theory) may delay the split selection decision indefinitely. However, if a linear model tree is being grown to approximate a smooth convex function, then as more examples are observed, the statistical splitting rule will repeatedly split the leaf nodes of the tree. As a result, the tree will continuously grow resembling a smooth mesh structure that fractures the input space into a large number of linear models. Therefore, it is desirable to limit the growth of the tree by using some form of a pre-pruning heuristic.

Potts and Sammut (2005) have proposed to monitor the difference between the variance estimate using a single model and the pooled variance estimate using separate models on each side of a candidate split:

$$\delta = \frac{RSS_0}{N-d} - \frac{RSS_1 + RSS_2}{N_1 + N_2 - 2d}.$$

The value of the parameter δ decreases with the growth of the tree, since the approximation becomes more accurate due to the increased number of leaves. The stopping rule is thus a simple threshold monitoring task: When δ eventually falls below a pre-determined threshold value δ_0, the node will not be considered for further splitting. This is a very convenient method to limit the growth of the tree: In case the user is not satisfied with the obtained accuracy, the value of the threshold can be further decreased, and the tree will be able to grow from its leaves to form a more refined model. As long as the distribution underlying the data is stationary, no re-building or restructuring is required.

Both of the discussed algorithms use an additional pruning method, which can be also considered as a strategy for adapting to possible concept drift. The pruning takes place if the prediction accuracy of an internal node is estimated to be not worse than its corresponding sub-tree. This requires maintaining a linear model in each internal node, which increases the processing time per example to $O(hd^2)$, where h is the height of the tree.

6.2 Probabilistic Sampling Strategies in Machine Learning

Hoeffding bounds are a non-parametric method which, given a sequence of independent observations of a random variable, permit one to place a confidence interval on the underlying mean value of the random variable (Hoeffding, 1963). Namely, the Hoeffding bound can be

used to achieve an (δ, ε) approximation for the mean value of a sequence of random variables $X_1, X_2, \ldots$. enabling us to state that: With probability $1-\delta$ the true underlying mean (X_{true}) lies within distance ε of the sample mean (X_{set}):

$$Pr[|X_{true} - X_{est}| > \varepsilon] < 2e^{-2N\varepsilon^2/R^2} \tag{39}$$

R is the (known a priori) maximum range of the random variables and N is the number of samples.

We can therefore make statements like "*We are 99% confident that our estimate of the average is within* ε *of the true average*" which translates to $Pr[|X_{true} - X_{est}| > \varepsilon] < 0.01$. The confidence parameter is typically denoted by δ. Equating the right-hand side of Equation (39) with δ, we get an estimate of the precision of the approximation in terms of N, R and δ:

$$\varepsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2N}}, \tag{40}$$

The value of ε monotonically decreases as more samples are being observed, which corresponds to confidence interval shrinkage. Naturally, our estimate is improved by observing more samples from the unknown distribution, but in this case there is a fast decay in the probability of failure to bound our estimate. In particular, one of the main benefits when using the Hoeffding bound is that it gives an exponential decay in the probability density function that models the deviation of the sampled mean from its expected value.

Another advantage of the Hoeffding bound is the lack of an assumption about the distribution of the random variables. This comes at the price of a higher conservatism, which means that if we could make stronger assumptions, the same confidence intervals could be reached using fewer examples. However, Hoeffding bounds have been used extensively in a variety of tasks in machine learning. In Chapter 2, we have already shown how the Hoeffding bound can be applied in the context of *PAC* learning for determining the number of training instances which is necessary to find a hypothesis in $\mathscr{H}$ that is within an ε distance from the best one H^* with probability of $1-\delta$.

Hoeffding bounds have been also used in *racing models* for the task of automated model selection (Maron and Moore, 1994, 1997). The main idea is to *race* competing models, until a clear winner is found. The racing algorithm starts with a collection of models, each of which is associated with two pieces of information: a current estimate of its average error and the number of test points it has processed. At each iteration of the algorithm, a random instance is selected from a given test set. Then, for each model a leave-one-out error is computed using the testing instance; this is used to update the model's estimate of its own average error. The Hoeffding bound is used to determine probabilistically how close the model's estimate is to the true average error. The models whose lower bound on their average error is greater than the upper bound on the average error of the best model are eliminated from the race. As compared to beam search, the set of models is known from the beginning of the evaluation process, whereas in the former algorithm the final set of models is chosen heuristically from an unknown superset. A slightly different implementation of the same idea has been used for racing features in an algorithm for feature subset selection, in the work of Moore and Lee (1994). At each step in *forward feature selection*, all the futures available to be added are raced against each other in parallel. A feature drops out of the race if the estimated upper bound on its merit is lower than the estimated lower bound on the merit of the best feature.

The same idea has been further leveraged by Kohavi (1994) in a different algorithm for feature subset selection, where the task is formulated as a state space search with probabilistic estimates. In this problem setup, each state represents a subset of features, and each move in the space is drawn from the set of possible moves {"*add one feature*", "*remove*

one feature"}. The state evaluation function $f(s)$ is an indicator of the quality of the state s, thus the goal is to find the state with the maximal value of $f^*(s)$. The problem can be approached using a simple hill climbing method or other more sophisticated alternatives, such as simulated annealing, beam search or a genetic algorithm. However, each step in the search would have to be accompanied with an evaluation phase, which makes the exploration of the search space a very tedious task. The authors have therefore proposed a probabilistic evaluation function, which gives a trade off between a decreased accuracy of estimates and an increased state exploration. By observing the evaluation function as a random variable, the Hoeffding bound is used to provide a probabilistic estimate on the true value of $f(s)$.

These early ideas on using probabilistic estimates for speeding up different machine learning tasks have inspired the work of Domingos and Hulten (2000), now considered a main reference point for incremental algorithms for learning decision trees. Domingos and Hulten (2000) have used the *feature racing* idea in a decision tree learning algorithm as a method for approximate probabilistic evaluation of the set of possible refinements of the current hypothesis, given a leaf node as a reference point. The probability of failing to make the correct selection decision (the one that would have been chosen if an infinite set of training examples was available) is bounded with the parameter δ. The sequence of probably approximately correct splitting decision as a result gives an asymptotically approximately correct model.

More precisely, given an evaluation function f and a set of refinements $r_1, r_2, ..., r_m$, the algorithm tries to estimate the difference in performance between the best two refinements with a desired level of confidence $1-\delta$. Let r_i and r_j be the best two refinements whose estimated evaluation function values after observing N examples are $\hat{f}(r_i)$ and $\hat{f}(r_j)$ and $\hat{f}(r_i) > \hat{f}(r_j)$. Let us denote the difference of interest $\Delta\hat{f} = \hat{f}(r_i) - \hat{f}(r_j)$: We consider it as a random variable and apply to it the Hoeffding inequality to bound the probability of exceeding the true difference by more than ε with $Pr[|\Delta f - \Delta\hat{f}| > \varepsilon] < 2e^{-2N\varepsilon^2/R^2}$. If $(\Delta\hat{f} - \Delta f) < \varepsilon$ with probability $1-\delta$, then $\Delta f > \Delta\hat{f} + \varepsilon$. Given that $\varepsilon > 0$, the true difference must be positive with probability $1-\delta$, which means that $f(r_i) > f(r_j)$. This is valid as long as the estimated values $\hat{f}$ can be viewed as the average of f values for the examples observed at the leaf node[1]. Thus, when $\Delta\hat{f} < \varepsilon$, there is enough statistical evidence to support the choice of r_i.

A key property of the Hoeffding tree algorithm is that it is possible to guarantee, under realistic assumptions, that the trees it produces are asymptotically arbitrarily close to the ones produced by a batch learner. Domingos and Hulten (2000) have proven that the maximal expected disagreement between the Hoeffding tree (induced by the Hoeffding tree algorithm with a desired probability δ given an *infinite* sequence of examples) and an asymptotic batch decision tree, induced by choosing at each node the attribute with the true greatest f (i.e., by using an infinite number of examples at each node), is bounded with δ/p, where p is the probability that an example reaches a leaf (assumed constant for simplicity). A useful application of the bound is that, instead of providing a desired level of confidence in each splitting decision, users can now specify as input to the Hoeffding tree algorithm the maximum expected disagreement they are willing to accept, given enough examples for the tree to settle, and an estimation of the leaf probability p.

The applicability of the Hoeffding bound is dependent on the assumption of observing a sequence of identically independently distributed random variables, which can be difficult to achieve in practice. Another caveat when using the bound is the hidden assumption on viewing the evaluation function as an average over a sequence of random variables. Finally, the Hoeffding algorithm implicitly races only two features in the refinement selection performed at the referent leaf node. This is due to the assumption that the third-best and all of the lower ranked features have sufficiently smaller merits, so that their probability of being the best refinement is very small and can be neglected. Despite these assumptions,

[1]For evaluation functions which are based on entropy reduction, the value of the range is $R = 1$.

which might be difficult to satisfy in practice, Hoeffding trees have been shown to achieve remarkable accuracy and efficiency in learning.

Hoeffding-based algorithms (Gama et al., 2004b, 2003; Hulten et al., 2001) can process millions of examples efficiently in the order of their arrival, without having to store any training data. Given enough training examples, Hoeffding trees have been show to achieve a performance comparable to that of a batch decision tree. This is empirical proof that a sequence of probably approximately correct splitting decisions, supported by using the Hoeffding probabilistic bound, can provide means for successful convergence to a hypothesis which is very near to the optimal hypothesis.

6.3 Hoeffding-based Regression and Model Trees

Following the successful line of Hoeffding-based algorithms for classification, probabilistic estimates based on the Hoeffding bound have been also used for the task of learning regression trees from an infinite stream of examples. The FIMT, FIRT-DD and FIMT-DD[2] algorithms by Ikonomovska and Gama (2008); Ikonomovska et al. (2010b, 2009) are representatives of Hoeffding-based learning algorithms in the domain of regression analysis. The FIMT algorithm is an online algorithm for learning linear model trees from stationary data streams. The statistical stability of the split selection decisions in FIMT is achieved by using the Chernoff bound. FIRT-DD is an extended version of FIMT that adds change detection abilities to FIMT when learning regression trees from time-changing data streams. FIMT-DD is an improved version of the previous algorithms that enables learning linear model trees from time-changing data streams and is based on the Hoeffding bound instead. As such will be used in the presentation of our main ideas.

The inductive inference process of FIMT-DD is very similar to the one used in the previously discussed Hoeffding-based decision tree learning algorithms, with two major differences. The first main difference is in the way the probabilistic estimate is obtained in the comparison of candidate splits. The second main difference is the incremental induction of additional linear models in the leaves of the tree.

The pseudocode of FIMT-DD is given in Algorithm 1. Like existing Hoeffding-based algorithms for classification, FIMT-DD starts with an empty leaf and reads examples in the order of their arrival. Each example is traversed to a leaf using the standard *Traverse* procedure. The pseudocode of the *Traverse* procedure is given in Algorithm 2. The procedure encapsulates the change detection and adaptation mechanisms of FIMT-DD. Regardless if change has been detected or not, it always returns a leaf node where the training example is traversed. The details of the drift detection and adaptation methods used in FIMT-DD are presented in the succeeding sections.

Every leaf has to maintain a set of sufficient statistics, updated using a corresponding procedure *UpdateStatistics*. A split selection decision is reached upon observing a sufficiently large sample of training examples, unless a pre-pruning rule is applied which corresponds to a stopping decision. The split evaluation measure is a variant of the one used by Breiman et al. (1984).

The selection decisions made by FIMT-DD are based on a probabilistic estimate of the ratio of *standard deviation reduction* (SDR) values for the two best performing candidate splits. After observing a portion of instances from the data stream in one of its leaf nodes, the algorithm finds the best split for each attribute, and then ranks the attributes according to the SDR value of the split candidate. If the splitting criterion is satisfied, FIMT-DD will introduce a *spitting test* on the best attribute, creating two new leaves, one for each branch of the split.

[2]The software is available for download at: http://kt.ijs.si/elena_ikonomovska/.

Algorithm 1 *The incremental FIMT-DD algorithm.* Pseudocode.

Input: δ - confidence level, N_{Att} - number of attributes, and n_{min} - chunk size
Output: T - current model tree
for ∞ **do**
  $e \leftarrow ReadNext()$
  $Seen \leftarrow Seen + 1$
  $Leaf \leftarrow Traverse(T, e)$ ▷ Traverses the example e to the corresponding *Leaf* node.
  $UpdateStatistics(Leaf)$ ▷ Updates the necessary statistics in the *Leaf* node.
  $UpdatePerceptron(Leaf)$
    ▷ Updates the weights of the perceptron located in the *Leaf* node.
  if $Seen \mod n_{min} = 0$ **then**
    for $i = 1 \rightarrow N_{Att}$ **do**
      $S_i = FindBestSplitPerAttribute(i)$
        ▷ Computes the best split test per attribute.
      $S_a \leftarrow Best(S_1, ..., S_{N_{Att}})$
      $S_b \leftarrow SecondBest(S_1, ..., S_{N_{Att}})$
      if $S_b/S_a < 1 - \sqrt{\frac{\ln(1/\delta)}{2 \times Seen}}$ **then**
        $MakeSplit(T, S_a)$ ▷ Creates a binary spit by applying S_a
      end if
    end for
  end if
end for

Algorithm 2 *The Traverse(T,e) procedure of the FIMT-DD algorithm.* Pseudocode.

Input: T - root node, e - training example, *PruneOnly* - boolean parameter
Output: *Leaf* - leaf node where the example e will be traversed
if $IsLeaf(T) \neq$ True **then**
  $Change \leftarrow UpdateChangeDetection(T, e)$
    ▷ Updates the statistics used in Page-Hinkley test.
  if *Change* is True **then**
    if *PruneOnly* **then**
      $Leaf \leftarrow Prune(T)$ ▷ Prunes the whole sub-tree T.
    else
      $InitiateAlternateTree(T)$ ▷ Creates an alternate tree rooted at T.
      $Leaf \leftarrow Traverse(T, e)$ ▷ Traverses the example e to the corresponding *Leaf* node.
    end if
  end if
else
  $Leaf \leftarrow T$
end if
Return (*Leaf*)

More precisely, given a leaf where a sample of the dataset S of size N has been observed, a hypothetical binary split h_A over attribute A would divide the examples in S in two disjoint subsets S_L and S_R, with sizes N_L and N_R, respectively ($S = S_L \cup S_R$; $N = N_L + N_R$). The formula for the SDR measure of the split h_A is given below:

$$SDR(h_A) = sd(S) - \frac{N_L}{N} sd(S_L) - \frac{N_R}{N} sd(S_R) \tag{41}$$

$$sd(S) = \sqrt{\frac{1}{N}(\sum_{i=1}^{N} y_i - \overline{y})^2} = \sqrt{\frac{1}{N}(\sum i = 1^N y_i^2 - \frac{1}{N}(\sum_{i=1}^{N} y_i))} \tag{42}$$

Let h_A be the best split (over attribute A) and h_B the second best split (attribute B). Let us further consider the ratio of the *SDR* values for the best two splits (h_A and h_B) as a real-valued random variable r:

$$r = SDR(h_B)/SDR(h_A). \tag{43}$$

Let us further observe the ratios between the best two splits after each consecutive example. Every observed value of the ratio can be considered as real-valued random variable r_1, r_2, ..., r_N. Let us denote the upper and lower bounds on the estimated sample average as $r^+ = \overline{r} + \sqrt{\frac{\ln(1/\delta)}{2 \times N}}$ and $r^- = \overline{r} - \sqrt{\frac{\ln(1/\delta)}{2 \times N}}$, where the true value lies in the interval $[r^-, r^+]$.

If the upper bound r^+ of the sample average is below 1, then the true average is also below 1. Therefore, after observing a sequence of N examples, the best attribute estimated from the available sample is truly the best over the whole distribution with probability $1 - \delta$. The algorithm selects the split h_A with confidence $1 - \delta$, and proceeds with evaluating the next selection decision in one of the leaves of the regression tree. The newly arriving instances are passed down along the branches corresponding to the outcome of the test.

The splitting criterion of FIMT-DD is sensitive to situations in which two or more splits have very similar or even identical *SDR* values. This type of a situation is treated as a *tie*, and is handled using the method proposed by Domingos and Hulten (2000). The main idea is that, if the confidence intervals determined by ε have shrunk substantially and still one cannot make a difference between the best splits, choosing any of them would be equally satisfying. The stopping rule can be then defined with the maximal value for ε with which the user is satisfied. Let us denote the threshold value with τ. If ε becomes smaller than τ (e.g., $\tau = 0.05$) and the splitting criterion is still not satisfied, we can argue that both of the competing candidate splits are equally good, and the one with a higher SDR value will be chosen.

6.4 Processing of Numerical Attributes

The efficiency of the split selection procedure is highly dependent on the number of possible split points. For numerical attributes with a large number of distinct values, both memory and computational costs can be very high. The common approach in the batch setting is to perform a preprocessing phase, which consists of partitioning the range of numerical attributes (discretization). This requires an initial pass of the data prior to learning, as well as sorting operations.

A separate preprocessing phase prior to learning can often be very difficult to perform in the streaming setting. Sorting is a very expensive operation. The range of possible values for numerical attributes is also unknown and can vary in case of changes in the data distribution. For classification tasks on data streams, a number of interesting solutions have been proposed: online discretization (with a pre-specified number of bins) (Domingos and Hulten, 2000), Gaussian-based methods of choosing continuous spitting tests for two-class

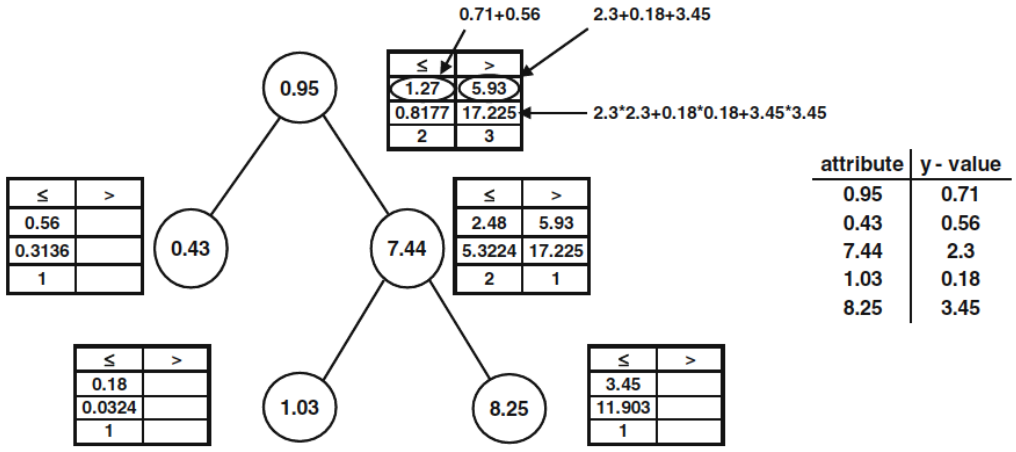


Figure 7: Illustrative example of a E-BST data structure constructed from the given table of pair of values (right)

problems (Gama et al., 2004b) and multi-class problems (Pfahringer et al., 2008), and an exhaustive method based on binary search trees (Gama et al., 2003).

The first method used in (Domingos and Hulten, 2000) is based on an online discretization in which the boundaries of every derived subrange are determined online from the first n unique values observed from the data sequence. For every new unique value a new bin is created until the pre-specified limit of n beans is reached. Once the bins are created they remain fixed till the end of the sequence, updating only their counts. Each bin contains counts per class, which and used to compute the information gain of the splitting point at one of its boundaries. Due to the fact that the boundaries of the bins are determined from the prefix of the data sequence, the method is very sensitive to skewed distributions.

The second category of methods are based on a Gaussian approximation of the distribution of values. These types of methods are very computationally efficient and require maintaining only few counts per class. In particular, the method proposed by (Gama et al., 2004b) uses a modified form of quadratic discriminant analysis. It is based on the assumption that the distribution of values of an attribute follows a normal distribution for both of the classes. By applying the equality:

$$p_1\phi\{(\mu_1,\sigma_1)\} = p_2\phi\{(\mu_2,\sigma_2)\}$$

where ϕ denotes the normal density function and p_1 (p_2) denote the estimated probabilities that an example belongs to class c_1 (c_2), the range of the numerical attribute is divided into three sub-ranges $(-\infty, d_1)$, (d_1, d_2) and $(d_2, +\infty)$, having d_1 and d_2 as the possible roots of the equation. The root that is equally distant to the means of both classes is chosen as the split that will be evaluated for the particular numerical attribute. Therefore, this method requires only the mean and variance per class of each numerical attribute. These statistics can be estimated on-the-fly from the sample of examples. The multi-class variant proposed in (Pfahringer et al., 2008) divides the range between the minimum and the maximum observed values into n parts (e.g., $n = 100$). For each of the possible splits the method estimates the parameters of the Gaussian distributions of each class, which are used to estimate the distribution of classes to the left and right of the split (based on the area under the normal distribution per class).

Although most efficient, the methods based on a Gaussian approximation are crude compared to the exhaustive method proposed in (Gama et al., 2003). This method is based on a binary tree structure created for every numerical attribute. Each node of the structure hold one possible split from the range of the attribute, as well as two arrays consisted of

counts of examples per class computed from the start of the sequence. The first array corresponds to the instances that have passed through this node with values less then or equal to the split value. The second array corresponds to the instances with vales greater than the split value. A previously non-observed value k is being inserted in the tree right below the node with the smallest split value which is greater than k. The size of the tree therefore depends on the number of unique values n. Insertion of new values in the binary tree structure has time complexity on average $O(\log n)$ and $O(n)$ in the worst case.

Based on this method we have developed an extended binary search tree structure (E-BST), which is an adaptation, tailored for regression trees. The basis of our approach is to maintain an E-BST for every numerical attribute in the leaves of the regression tree. The E-BST holds the necessary statistics and enables us to sort values on the fly. Each node in the E-BST has a test on a value from the attribute range (in the form: $\leq key$). The keys are the unique values from the domain of the corresponding numerical attribute. Further, each node of the E-BST structure stores two arrays of three elements. The first array maintains statistics for the instances reaching the node with attribute values less than or equal to the key: (1) a counter of the number of instances that have reached the node, (2) the sum of their target attribute values $\sum y$ and (3) the sum of the squared target attribute values $\sum y^2$. The second array maintains these statistics for the instances reaching the node with attribute values greater than the key.

The E-BST structures are incrementally updated with every new example that is reaching the leaf of the regression tree where they are stored. The process starts with an empty E-BST. Unseen values from the domain of the attribute are inserted in the structure as in a binary search tree. The main difference is that while traversing the E-BST, the counts in all the nodes on the way are additionally updated.

As an example, consider the E-BST structure in Figure 7. It is constructed from the following sequence of (numerical attribute, target attribute) pairs of values: {(0.95, 0.71), (0.43, 0.56), (7.44, 2.3), (1.03, 0.18), (8.25, 3.45)}. The root holds the key 0.95 since it is the first from the sequence. The first column which corresponds to the case $value \leq key$ is initialized with the values (0.71, 0.712, 1). The next value 0.43 is a new key inserted to the left of the root. The first column of the new node is initialized with (0.56, 0.562, 1) and the first column of the root is also updated with the values (0.71 + 0.56, 0.712 + 0.562, 2). This procedure continues for the remaining pairs of values. The last value inserted in the tree will update the second column of the root node (0.95) and the node (7.44).

To evaluate the possible splits, the whole tree must be traversed in an in-order fashion (in order to obtain a sorted list of distinct values). The computation of the standard deviation reduction for each possible splitting point is performed incrementally. This is possible because all the nodes in the structure have the information about how many examples have passed that node with attribute values less or equal and greater than the key, as well as the sums of their target attribute values.

The algorithm for finding the best splitting point *FindBestSplit* is a recursive algorithm which takes as input a pointer to the E-BST. Its pseudo code is given in Algorithm 3. As it moves through the tree in in-order it maintains the sum of all target attribute values for instances falling to the left or on the split point (*sumLeft*), the sum of all target attribute values for instances falling to the right of the split point (*sumRight*), the corresponding sums of squared target attribute values (*sumSqLeft* and *sumSqRight*), the number of instances falling to the right of the split point (*countRightTotal*), and the total number of instances observed in the E-BST structure (*total*). From these, the SDR statistics for each split point is calculated by using a corresponding procedure *ComputeSDR*(). When returning from the left or the right child node, these counts must be returned as they were at the parent node.

Insertion of a new value in the E-BST structure has time complexity $O(\log(n))$ on average and $O(n)$ in the worst case, where n is the number of unique values seen thus far in the leaf. In the case of a balanced E-BST (AVLTree) (Sedgewick, 1983) the worst insertion time is

Algorithm 3 *FindBestSplit: An extended FindBestSplit procedure for numerical attributes.* Pseudocode.

Input: *Node* - the root of the extended binary search tree structure
Output: h - the split that gives the maximal standard deviation reduction (SDR)

```
if Node → leftChild ≠ ∅ then
    FindBestSplit(Node → leftChild)
end if
                                ▷ Updates the sums and counts for computing the SDR value
sumTotalLeft = sumTotalLeft + Node → sumLeft
sumTotalRight = sumTotalRight − Node → sumLeft
sumSqTotalLeft = sumSqTotalLeft + Node → sumSqLeft
sumSqTotalRight = sumSqTotalRight − Node → sumSqLeft
countRightTotal = countRightTotal − Node → countLeft
if maxSDR < ComputeSDR(T) then
    maxSDR = ComputeSDR(T)
    h = Node → key
end if
if Node → rightChild ≠ ∅ then
    FindBestSplit(Node → rightChild)
end if
                                ▷ Updates the sums and counts for returning to the parent node
sumTotalLeft = sumTotalLeft − Node → sumLeft
sumTotalRight = sumTotalRight + Node → sumLeft
sumSqTotalLeft = sumSqTotalLeft − Node → sumSqLeft
sumSqTotalRight = sumSqTotalRight + Node → sumSqLeft
countRightTotal = countRightTotal + Node → countLeft return h
```

$O(\log(n))$. Determining the best split point has a time complexity of $O(n)$.

Although the tree is very good for efficient sorting of the numerical values, its memory consumption in some situations can be a concern. For reducing the memory requirements, several strategies can be employed. For example, if we do not maintain a balanced tree, the second array for the instances falling to the right of the split point can be removed, and this information can be kept only at the root node. Another option is reducing the precision of the real values stored as unique keys. Rounding to three decimal places can substantially reduce the amount of memory allocated.

To further reduce the memory requirements, we propose a method for disabling bad split points that will result in dropping parts of the E-BST structure. The procedure is initiated whenever the splitting criterion is not satisfied. Let $SDR(h_1)$ be the greatest reduction in the standard deviation measured at the last evaluation and $SDR(h_2)$ the second highest reduction over a different attribute. Let $r = SDR(h_2)/SDR(h_1)$ be the corresponding ratio. Every split point with $SDR(h_i)/SDR(h_1) < r - 2\varepsilon$ is considered bad.

The rationale of this method is as follows: If the upper bound of the ratio (for any split point) is smaller than the lower bound of r, then the true reduction of that split relative to the best one is definitely smaller. Therefore, every other split point with a ratio below this value is not a competing point and can be safely removed. The algorithm starts from the leftmost node of the tree structure that has been updated with at least 5% of the examples observed in the whole structure. The SDR values of the split points are computed the same way as previously. If the children of a node that has to be removed are good split points, or only the right child remained, its key will be replaced with the key of the in-order successor.With this action the arrays of the successor and all the predecessor nodes have to be updated correspondingly. If the node that has to be removed does not have an in-order

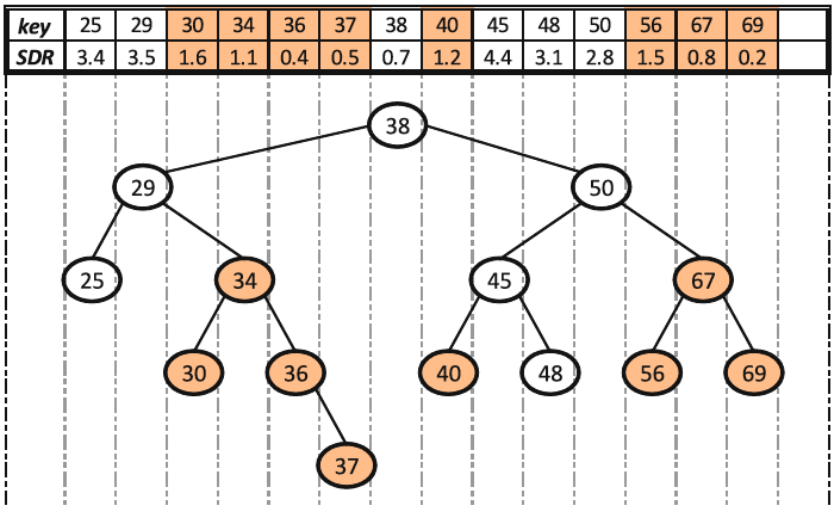


Figure 8: Illustrative example of a E-BST and disabled regions with bad split points.

successor, then we use the key from the in-order predecessor: In this case, the updating strategy is more complicated, because all the predecessor nodes on the path to the node (being removed) have to be updated.

When a node is removed from the tree no information is lost because its sums and counts are held by upper nodes. This method enables us to save a lot of memory without losing much of the relevant information. A simpler and less costly procedure is to prune only the nodes whose children are all bad split points. In this case no updating is necessary. Figure 8 gives an illustrative example of an E-BST with split points disabled by using the simpler procedure. The threshold is 1.6 and the colored (grey) keys (30, 34, 36, 37, 56, 67, and 69) are removed from the tree, leaving as possible split points the uncolored keys (25, 29, 38, 40, 45, 48, and 50).

6.5 Incremental Linear Model Trees

In Chapter 3, we have discussed several batch methods for learning linear model trees, which can be mainly categorized based on the approach taken for solving the selection decision. One category of algorithms ignores the fact that the predictions will be computed using linear models instead of a constant regressor, and uses a variant of the least squares split evaluation function. That said, there is no guarantee that the split chosen will minimize the mean squared error of the model tree.

The other category of algorithms tries to minimize the least squares error taking into account the hypothetical linear models on both sides of each candidate split. While the second category of algorithms gives theoretical guarantees that the mean squared error of the model tree will be minimized, its computational complexity is much higher compared to the first category of algorithms. Another problematic aspect of fitting linear models on each side of the candidate splits is that the problem is intractable for real-valued (numerical) attributes.

For these reasons, we have chosen an approach which is agnostic to the existence of linear models in the leaves. The model tree is thus built just like a regression tree, using the standard deviation reduction as an evaluation function. The linear models are computed after the execution of a split by using a computationally lightweight method based on online learning of un-thresholded *perceptrons*.

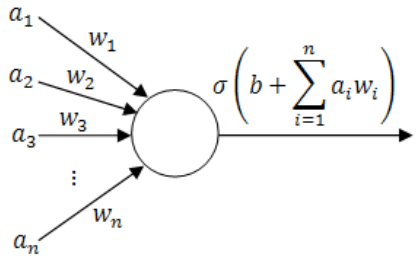


Figure 9: An illustration of a single-layer perceptron

The perceptron is the basic building block of an artificial neural network (ANN) system, invented by Frank Rosenblat. The simplest perceptron model is depicted in Figure 9. Here, rach w_i is a real-valued weight that determines the contribution of input x_i to the perceptron output o.

All the numerical attributes are mapped to a corresponding input, while the target attribute is mapped to the perceptron's output. The symbolical or categorical attributes have to be previously transformed into a set of binary attributes. More precisely, given inputs x_1 through x_n, the output o computed by the perceptron is:

$$o = f(\sum_{i=1}^{n} w_i x_i - b) = f(\mathbf{w}\mathbf{x} - b) \tag{44}$$

where b is a threshold and f is an activation function. The activation function can be, e.g., a step function, a linear function, a hyperbolic tangent function or a sigmoid function.

Learning a perceptron involves choosing values for the weights, including the threshold b, and is conditioned upon the function used to measure its error with respect to the desired output. Since we are dealing with a regression task, we want to obtain a *linear unit* which will give a best-fit approximation to the target concept. In that sense, the least squared error is a natural choice of an error function, E(w):

$$E(\mathbf{w}) = \frac{1}{2}(y-o)^2$$

where y and o are the target value and the perceptron output value for a given training example x.

The perceptron learning procedure is given as follows:

1. Initialize the weights $w_1, ..., w_n$, and the threshold b to random values.
2. Present a vector $\mathbf{x}$ to the perceptron and calculate the output o.
3. Update the weights using the following training rule: $\mathbf{w} \leftarrow \mathbf{w} + \eta(y-o)\mathbf{x}$, where η is the learning rate.
4. Goto step 2 until a user specified error is reached.

The training rule used in this learning procedure is known as the *delta* or Widrow-Hoff additive rule, for which it is known that the learning process converges even in the case when the examples do not follow a linear model. The learning rule adjusts the weights of the inputs until the perceptron starts to produce satisfying outputs. However, the threshold parameter b remains fixed throughout learning, once the perceptron is initialized. Depending on the value of b, this would potentially slow down the convergence. As a solution, an additional input is presented to the perceptron $x_0 = 1$ with a fixed value, and a weight $w_0 = -b$ initialized to a random value. With this modification, the convergence of the training algorithm is improved and the model is simplified.

The key idea behind the delta rule is to use *gradient descent* to search the hypothesis space H, which in this case is the set of all possible real-valued weight vectors; that is, $H = \{\mathbf{w}|\mathbf{w} \in \mathscr{R}^{n+1}\}$. For linear units, the error surface which is associated with the hypothesis space is always parabolic with a single global minimum. The direction of moving along this surface can be found by computing the derivative of E with respect to each component of the vector $\mathbf{w}$, known as the *gradient*. Gradient descent search determines the direction that produces the steepest increase of E. This information is used to modify the weight vector such that it will be moved in a direction that *decreases* E.

In other words, $w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$, thus the steepest descent is achieved by altering each component w_i in proportion to the corresponding partial derivative $\frac{\partial E}{\partial w_i}$:

$$w_i \leftarrow w_i + \eta (o - y) x_i \text{ , for } i \neq 0. \tag{45}$$

The weights are updated with every training example arriving in the leaf instead of using a batch update. The learning rate η can be kept constant or it can decrease with the number of examples seen during the process of learning. The second option gives a smooth decay of the learning rate by using a formula where the value of the learning rate is inversely proportional to the number of instances:

$$\eta = \frac{\eta_0}{1 + n\eta_d}. \tag{46}$$

where n is the number of examples used for training. As proposed in our previous work (**?**), the initial learning rate η_0 and the learning rate decay parameter η_d should be set to appropriately low values (e.g., $\eta_0 = 0.1$ and $\eta_d = 0.005$). If a constant learning rate is preferred, it is recommended to be set to some small value (e.g., 0.01). In the experimental evaluation presented in the following sections we have named the two versions of the FIMT algorithm as FIMT_Decay (with a decaying learning rate) and FIMT_Const (with a constant learning rate).

The perceptron represents a linear approximation of the training examples observed at the leaf node. Consequently, the weights of the perceptron are equal to the parameters of the fitted linear model at the node. Compared to existing approaches, this process is of linear complexity, given that a stochastic gradient descent method is used.

An important part of the process is the normalization of the inputs before they are presented to the perceptron. This is necessary to ensure that each of them will have the same influence during the process of training. The normalization can be performed incrementally by maintaining the sums: $\sum_i x_j$ and $\sum_i x_j^2$, for $j = 1, ..., n$. FIMT-DD uses a variant of the studentized residual equation, where instead of dividing with one standard deviation (σ), it divides by three standard deviations. The equation is given with:

$$x'_i = \frac{x_i - \overline{x}}{3\sigma}. \tag{47}$$

The learning phase of each perceptron is performed in parallel with the process of growing a node, and ends with a split or when a pre-pruning rule is applied to the leaf. If a split was performed, the linear model in the parent node is passed down to the children nodes. This avoids training from scratch: The learning continues in each of the new leaves, independently and accordingly to the examples that will be observed in each leaf separately. Since the feature space covered by the parent node is divided among the children nodes, this can be seen as fine tuning of the linear model in the corresponding sub-region of the instance-space.

Compared to the approach of Potts and Sammut (2005) FIMT-DD has a significantly lower computational complexity which is linear in the number of regressors. An important disadvantage of FIMT-DD is that the split selection process is invariant to the existence of linear models in the leaves. The process of learning linear models in the leaves will not explicitly reduce the size of the regression tree. However, if the linear model fits well

the examples assigned to the leaf, no further splitting is necessary and pre-pruning can be applied.

6.6 Drift Detection Methods in FIMT-DD

The learning algorithm presented thus far models the functional dependencies by minimizing the mean squared error of the regression tree. In our online learning scenario, however, it is expected that the underlying functional dependencies will change over time. The change can affect the whole instance space or only some regions of it. As a result, the affected parts of the model tree or the whole model will become poor approximations of the true functional dependencies, and their predictions will most likely be incorrect. The error of the whole model will consequently increase, making it unsuitable for descriptive or predictive purposes.

While the predictive accuracy of the model tree is very important, we are primarily interested in informed detection of changes, which would enable us to alter the structure of the model tree as a response to the change in the functional dependencies. An intuitive approach is to assume that the increase in the error is due to a smaller portion of incorrect sub-trees which are being affected with the change, leaving larger portion of non-affected sub-trees that still perform accurately. This suggests a simple method to detect changes: monitor the quality of every sub-tree by monitoring the evolution of its error. If the error starts increasing, this may be a sign of change in the target function.

6.6.1 The Page-Hinkley Test

From the various methods discussed in Chapter 2, we are further interested in online change detection, due to the fact that online learning algorithms do not have access to the full history of the data stream. One of the most successful online change detection tests is the Page-Hinkley (PH) test proposed by Mouss et al. (2004). The PH test is a sequential adaptation of the detection of abrupt change in the average of a Gaussian signal. At any point in time, the test considers two variables: 1) a cumulative sum $m(T)$, and 2) its minimal value $M(T) = \min_{t=1,\ldots,T} m(t)$, where T is the number of observed examples. The cumulative sum $m(T)$ is defined as the cumulative difference between the monitored signal x_t and its current mean value $\bar{x}(T)$, corrected with an additional parameter α:

$$m(T) = \sum_{t=1}^{T} (x_i - \bar{x}(T) - \alpha) \tag{48}$$

where

$$\bar{x}(T) = \frac{1}{T} \sum_{t=1}^{T} x_t. \tag{49}$$

The parameter α corresponds to the minimal absolute amplitude of change that we wish to detect. It should be adjusted according to the expected standard deviation of the signal. Larger deviations will require larger values for α, but if α is set to a too large value, it can produce larger detection delays. Our experiments have shown that setting α to 10% of the standard deviation gives best results.

The PH test monitors the difference between the minimum $M(T)$ and $m(T)$, defined as: $PH(T) = m(T) - M(T)$. When this difference is greater than a user-specified threshold λ, i.e., $PH(T) > \lambda$, the test will trigger an alarm that informs about a change in the mean of the signal. The threshold parameter λ corresponds to the admissible false alarm rate. Increasing λ entails fewer false alarms (higher precision) at the cost of a larger number of missed events (lower recall).

The variable x_t, which is monitored over time, is in our case the absolute error at a given node or a sub-region of the instance space, i.e., $x = |y - o|$. Here y is the true value of the target attribute and o is the prediction from the constant regressor. Having multiple nodes in the tree we have multiple instantiations of the change detection test running in parallel.

The model tree divides the instance space into multiple nested regions R_i. Each node corresponds to a hyper-rectangle or a region of the instance space and is associated with a corresponding variable $x_t^{R_i}$. The root node covers all of the hyper-rectangles, while subsequent nodes cover sub-regions (hyper-rectangles) nested in the region covered by their parent node. Consequently, the structure of the regression tree gives multiple views of the instance space and the associated error, at different levels of granularity.

For computing the absolute loss, we have therefore considered two possible alternatives: 1) using the prediction from the node where the error is being monitored, or 2) using the prediction from the leaf node where the example is assigned. Consequently, we propose two different methods for change detection:

- Top-Down (TD) method: The error is computed using the prediction from the current node. The computation can be performed while the example is passing the node on its path to the leaf. Therefore, the loss will be monitored in the direction from the top towards the "bottom" of the tree.

- Bottom-Up (BU) method: The error is computed using the prediction from the leaf node. The example must therefore reach the leaf first. The computed difference at the leaf will be then back-propagated to the root node. While back-propagating the error (re-tracing the path the example took to reach the leaf), the PH tests located in the internal nodes will be updated correspondingly.

The idea for the BU method is based on the following observation: When moving towards the "bottom" of the tree, predictions in the internal nodes become more accurate (as a consequence of splitting). In case of concept drift, using more accurate predictions will emphasize the error, shorten the delays and reduce the number of false alarms. This was investigated and confirmed by empirical evaluation, which has shown that, when using the TD method, most of the false alarms were triggered at the root node and its immediate descendants.

6.6.2 An improved Page-Hinkley test

Tuning the values of the parameters α and λ is a tedious task. Our experiments have shown that uniformly good results (over the whole collection of real-world and artificial datasets) can be achieved for the combination of values: $\alpha = 0.005$ and $\lambda = 50$. If we would like to tolerate larger variations, the value of λ can be increased to 100. For more sensible change detection and under a risk of an increased rate of false alarms, the value of α can be optionally decreased to 0.001. As a general guideline, when α is decreased, the value of λ should be increased: This is to reduce the number of false alarms. On the other hand, when the value of α is increased, the value of λ should be decreased: This is to shorten the delays in change detection.

While there are many optimization techniques that can be employed to tune the values of the parameters, we have tried to automate the process. We have transformed the parameter α into a variable whose value depends on the incrementally computed standard deviation of the absolute loss x_t. In particular, we propose an improved Page-Hinkley test where the sum $m(T)$ is computed as follows:

$$m(T) = \sum_{t=1}^{T} (x_i - \bar{x}(T) - sgn(x_T - \bar{x}(T)) \times \alpha_{PH}(T)/2) \tag{50}$$

$$\alpha_{PH}(T) = sd(x_T) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x}(T))^2}. \quad (51)$$

The value of α_{PH} is initially set to 0.01 and is periodically updated after every 50 examples observed in the particular node. By adjusting the value of α_{PH} according to the standard deviation of the signal, we minimize the effect of the variance, which results in a shorter detection delay.

6.7 Strategies for Adaptation

While proper and timely detection of change in the target function is a necessity for proper adaptation, the final outcome depends on choosing an appropriate adaptation strategy. Decision and regression trees are hierarchical data structures, whose restructuring is a difficult and computationally expensive operation. This issue has been previously addressed by Hulten et al. (2001), which have proposed the CVFDT algorithm for learning decision tress from time-changing data streams with an informed and local adaptation.

The CVFDT algorithm performs regular periodic validation of its splitting decisions by maintaining the necessary statistics at each node over a window of examples. For each split found to be invalid, it starts growing a new decision tree rooted at the corresponding node. The new sub-trees grown in parallel are intended to replace the old ones, since they are generated using data which corresponds to the new concepts. To smooth the process of adaptation, CVFDT keeps the old subtrees rooted at the influenced node, until one of the newly grown trees becomes more accurate. This enables granular and local adaptations of the current hypothesis.

Another representative of this line of algorithms is VFDTc Gama et al. (2003), which also performs continuous monitoring of the differences in the class-distributions of the examples from two time-windows. The change detection method in VFDTc evaluates past splitting decisions in an incrementally built classification tree and detects splits that have become invalid due to concept drift. The tree can be adapted correspondingly by regrowing the parts which have been removed.

In our study we have considered three possible adaptation strategies: *Do nothing*, *Prune*, and *Grow an alternate tree*:

1. The first adaptation strategy consists of further splitting and growing new sub-trees to the old structure. The new leaves would collect up-to-date statistics, which would inevitably provide better predictions. Although the predictions might improve, the structure of the tree would likely be wrong and misleading.

2. The second adaptation strategy is to prune the parts of the tree where concept drift was detected. This is a relatively severe adaptation strategy, which might result in pruning large portions of the tree. A significant decrease in the number of leaves will lead to unavoidable and drastic short-time degradation in accuracy. In these circumstances, an outdated sub-tree may still give better predictions than a single leaf.

3. The third adaptation strategy is to delay the act of pruning to the moment when an alternate sub-tree has been re-grown, which captures the new functional dependencies. The process is initiated as soon as a change is detected at some internal node, which will have to be replaced with a new tree structure. The new alternate tree will be rooted in the same place, and will grow in parallel with the old structure. Every example that will reach a node with an alternate tree initiated will be used for growing and updating both trees. The old tree will be kept until the new alternate tree becomes more accurate. The alternate tree will not be monitored for a change detection until the moment it replaces the original one.

There are several issues that need to be addressed when using the *grow an alternate tree* adaptation strategy. Due to the possibility of false alarms or improper localization of concept drift, an alternate tree might never achieve better accuracy than the original one. In order to deal with this type of situations, we propose to monitor the difference in the accuracy of the alternate tree (*AltTree*) and the original sub-tree (*OrgTree*). This is performed by monitoring the log ratio of the accumulated prequential mean squared error computed for every example observed at the node where both of the trees are rooted.

Let $S_i^{AltTree}$ be the sequence of the prequential accumulated loss (mean squared error) for the alternate tree and $S_i^{OrgTree}$ the corresponding sequence for the original tree. The prequential accumulated loss is the sum of losses accumulated for a sequence of examples. For each example, the model first makes a prediction based on the example's attribute values, and then the example is used to update the actual model. The statistic used to compare these errors is the Q statistic and is computed as $Q_i(OrgTree, AltTree) = log(S_i^{OrgTree}/S_i^{AltTree})$. The sign of Q indicates the relative performance of both models, while its value shows the strength of the difference.

However, it is well known that the prequential error is pessimistic, because it contains all the errors from the beginning of the learning, when the model is still not fully grown. These long term influences hinder the detection of recent improvement. Gama et al. (2009) propose a formula for using fading factors with the Q_i statistic:

$$Q_i^{\alpha}(OrgTree, AltTree) = \log\left(\frac{L_i(OrgTree) + f \cdot S_{i-1}^{OrgTree}}{L_I(AltTree) + f \cdot S_{i-1}^{AltTree}}\right), \tag{52}$$

where L_i is the current loss (squared error) computed from the last example. The prequential error will be updated after computing the Q_i statistic. The fading factor f can be set to a value close to 1, for example 0.975. If the Q_i statistic has values greater than zero, the original tree has a larger error than the alternate tree.

The Q_i statistic can be evaluated at predetermined time intervals of T_{min} examples (e.g., $T_{min} = 100$ examples). If it is positive, the new alternate tree will replace the old one. The old tree will be removed, or can be stored for possible reuse in case of reoccurring concepts. If the detected change was a false alarm or it was not well localized, the Q_i statistic will increase slowly, and possibly will never reach the value of zero. For this reason, we need to monitor its average. If instead of moving towards zero it starts to decrease, the alternate tree is not promising any improvement to the current model and should be removed.

In addition, a separate parameter can be used to specify a time period during which a tree is expected to grow and model the new function. When the time period for growing an alternate tree has passed or the average has started to decrease, the alternate tree will be removed from the node and the allocated memory will be released. In order to prevent premature discarding of the alternate tree, we start to monitor the average of the Q_i statistic after $10 \cdot n_{min}$ examples have been used for growing and evaluation.

6.8 Empirical Evaluation of Online and Batch Learning of Regression and Model Trees Induction Algorithms

The empirical evaluation presented in this section was designed to answer several questions about the general performance of the proposed algorithm FIMT, which involve the quality of the models produced with probabilistic sampling strategies in terms of predictive accuracy and size, stability and variability under different training sequences, robustness and memory requirements.

With respect to predictive accuracy, the main hypothesis is that, given an *infinite stream* of instances arriving at high speed, the incrementally learned model tree would have asymptotically approximately comparable accuracy to the tree learned by an assumed ideal batch

learner that is able to process an *infinite* amount of training data. Since such an ideal batch learner does not exist, we have performed a comparison with several state-of-the-art available batch learning algorithms. We have used CART by Breiman et al. (1984) from R (with default settings); M5' by Quinlan (1992) from WEKA (Hall et al., 2009) (with default settings) with the regression tree option on (M5'RT) as well with the model tree option on (M5'MT); a linear regression method LR from WEKA (Hall et al., 2009) (with default settings), the commercial algorithm CUBIST Ltd RuleQuest (2010) (with default settings), and four other algorithms for learning model trees proposed by Potts and Sammut (2005): two versions of a batch learner (BatchRD and BatchRA) and two versions of an incremental learner (OnineRD and OnlineRA). All algorithms were run on an Intel Core2 Duo / 2.4GHz CPU with 4 GB of RAM.

We have further tried to assess the stability of incrementally learned trees, their speed of convergence and sensitivity to noise. Another interesting analysis, which is particularly valuable for situations in which resources are constrained, estimates the impact of constrained memory on the accuracy. In that context we have evaluated the algorithm's predictive accuracy when performing under memory constraints of different severity. The results of comparing the above algorithms in terms of these metrics are given in the following subsections.

6.8.1 Predictive Accuracy and Quality of Models

For an algorithm to learn a concept or approximate a target function, it is necessary to observe a sufficiently long sequence of training examples generated from a stationary data distribution. This is the learning setup which is assumed in the experimental analysis presented in this section. We use all the available stationary datasets for regression tasks presented in Chapter 5. Since the real datasets used are relatively small, the evaluation was performed using the standard method of 10-fold cross-validation, using the same folds for all the algorithms included. For all the experiments, we set the input parameters of our algorithm to: $\delta = 0.01$, $\tau = 0.05$ and $n_{min} = 200$.

Due to the fact that most of the datasets are finite and of manageable size, an advantage is unintentionally given to the batch learners. Having a training set of finite size, batch learning algorithms are able to use more training examples in their early stages of building the tree, as long as the data can be loaded in main memory. They are also able to reuse the training examples at all levels of the tree. The incrementally learned trees, on the other hand, are expected to achieve similar performance as the batch learned ones, conditioned upon having a long enough stream of training instances. Therefore, all of the results presented should be analyzed, having in mind the aforementioned requirements for a successful relative comparison.

For a fair comparison, as well as to be able to evaluate the impact of linear models in the leaves, two separate evaluations were performed. First, FIMT without linear models in the leaves and without change detection (FIRT) was compared to two regression tree learning algorithms: CART, and M5'RT. Second, FIMT-DD with linear models included and without change detection (FIMT) was compared to six model tree learning algorithms: M5'MT, LR, CUBIST, BatchRD, BatchRA, OnlineRD, OnlineRA. We used two different versions of FIMT: FIMT_Const with a constant learning rate, and FIMT_Decay with a decaying learning rate.

The performance measures for CART, M5' and FIRT averaged across the 10 datasets are given in Table 3. We have used the real-world datasets described in Chapter 5 Table 2. The presented performance results represent the average value for all the results obtained by using the 10-fold cross-validation. The detailed (per dataset) results are given in Table 29 in Appendix A section A.1.

Table 3: Results from 10-fold cross-validation averaged over the real datasets for three regression tree learning algorithms. Relative error (RE), root relative mean squared error (RRSE), number of leaves, running time and the correlation coefficient (CC) are given, averaged across 10 datasets.

Algorithm	RE %	RRSE %	Leaves	Time (sec.)	CC
M5'RT	47.30	51.72	244.69	29.28	0.82
CART	75.63	62.77	4.18	9.67	0.75
FIRT	55.38	60.11	35.54	0.33	0.75

The comparison of regression tree learners shows that FIRT has similar accuracy (RRSE, CC) as batch learner CART and lower accuracy as compared to M5'RT. FIRT produces medium sized models (on average 35 leaves), which are larger than the CART trees (on average 4 leaves), but significantly smaller than the M5'RT trees (on average 245 leaves). The Nemenyi test confirmed that there is no statistically significant difference (in terms of the correlation coefficient) between FIRT and CART: this was also confirmed with a separate statistical analysis using the Wilcoxon signed-rank test.

Table 4: Results from 10-fold cross-validation averaged over the real datasets for model tree learning algorithms

Algorithm	RE %	RRSE %	Leaves	Time (sec.)	CC
M5'MT	42.10	46.09	76.71	29.54	0.85
LR	60.48	63.01	1.00	2.59	0.74
CUBIST	48.75	/	20.67	1.02	0.75
BatchRD	41.84	45.27	55.58	6.13	0.85
BatchRA	50.22	53.39	22.85	2.24	0.81
OnlineRD	49.77	53.26	9.98	32.67	0.80
OnlineRA	48.18	52.03	12.78	3.08	0.82
FIMT_Const	60.21	104.01	35.54	0.42	0.70
FIMT_Decay	59.40	74.11	35.54	0.42	0.73

The performance results of the model-tree learners averaged across the 10 datasets are given in Table 4, while detailed (per dataset) results can be found in Table 30 for the relative error (RE%), Table 31 for the size of the models (leaves) and Table 32 for the learning time (Appendix A section A.1). The comparison of model tree learners shows that FIMT_Decay and FIMT_Const have similar accuracy to LR, while the rest of the learners are better (BatchRD being the best). It is interesting to note that the split selection method of BatchRD which takes into account fitted linear models on both sides of each hypothetical split performs better compared to the selection method which is agnostic to the existence of linear models. Another interesting result is that the incremental learner OnlineRA achieves better accuracy than the commercial batch learner CUBIST. The analysis on the statistical significance of results (Friedman test for comparison of multiple classifiers followed by the Nemenyi post-hoc test) confirmed this, additionally showing that there is no significant difference between FIMT and OnlineRA, OnlineRD and BatchRA methods. There is a significant difference in accuracy between M5', CUBIST and BatchRD over FIMT and LR, while no significant difference over BatchRA, OnlineRA and OnlineRD.

We have also noticed that FIMT_Const exhibits high root relative squared error rates, which is a sign of the existence of predictions that deviate from the true value largely. The reason most likely is that some of the perceptrons have not been trained properly, due to a premature ending of the training or due to a divergence which may happen if the learning rate is set too high. The FIMT_Decay version obviously avoids the divergence by reducing the step size as the tree improves its approximation of the target function. It was also observed that linear models in the leaves rarely improve the accuracy of the incremental regression

Table 5: Results from holdout evaluation averaged over the artificial datasets Fried, Lexp, Losc with 10-fold cross-validation on a fold of 1M examples and a test set of 300k examples.

Algorithm	RE %	RRSE %	Leaves	Time (sec.)	CC
FIRT	0.16	0.19	2452.23	24.75	0.98
CUBIST	0.13	/	37.50	104.79	0.98
FIMT_Const	0.11	0.14	2452.23	27.11	0.98
FIMT_Decay	0.11	0.14	2452.23	26.93	0.98
LR	0.46	0.51	1.00	2468.61	0.83
BatchRD	0.08	0.10	27286.30	5234.85	0.99
BatchRA	0.71	0.69	56.97	2316.03	0.51
OnlineRD	0.10	0.13	6579.5	3099.82	0.98
OnlineRA	0.70	0.68	57.77	2360.56	0.53

tree on these small datasets. This could be due to the fact that the regression surface is not smooth for these datasets and there is no need of using a model tree. The largest models on average are produced by M5'MT and BatchRD. In this comparison CUBIST is a special case, because it has a special mechanism for creating rules from the tree. Online learners have lower accuracy than their batch equivalents. FIRT has a significant advantage in speed over all of the learners. CUBIST is in the second place. Another advantage of FIRT is that it can learn with a limited amount of memory (as allowed by the user), while the other algorithms do not offer this option.

The artificial datasets (Fried, Lexp and Losc) were used to simulate a learning environment in which enough training data are provided for the incremental learner FIMT. For a proper comparison, we proceeded again with a 10-fold-cross-validation procedure, for which we generated 10 random datasets, each of size 1 million examples, and one separate test set of size 300k examples. All the algorithms were trained and tested using identical folds. It should be noted that this type of evaluation is very difficult to perform in real-time for a realistic online learning scenario. The results in Table 3 show that the incremental algorithms are able to achieve better accuracy than the batch algorithms CUBIST, LR and BatchRA. OnlilneRD achieves better accuracy than CUBIST but, on the other hand, is significantly slower. Only the two versions of FIMT-DD were able to process the data faster than CUBIST with a comparable accuracy. Detailed (per dataset) results can be found in Table 33 (Appendix A section A.1). The best algorithm is BatchRD whose models are much larger. BatchRA and OnlineRA give the worst results. The linear models in the leaves improve the accuracy of FIRT because we are now modeling smooth surfaces. The incremental model trees of FIMT_Const and FIMT_Decay have similar accuracy as OnlineRD, but are smaller in size and the learning time is at least 100 times smaller.

6.8.2 Memory and Time Requirements

One of the main advantages of FIMT-DD is that it is able to learn successfully given a constrained (small) amount of available memory. Beside the fact that FIMT-DD does not need to store its training data, and allocates memory just enough for the necessary statistics maintained in the leaf nodes and for storing the tree structure itself, it also possesses some memory management mechanisms, similar to those used in the VFDT algorithm by Hulten et al. (2001).

The way that the memory management mechanism works is by monitoring the total memory allocated at any point during the learning process. When almost all of the available memory is allocated, the memory management algorithm starts to search for leaves whose estimated error is minimal. From those least promising nodes, it deactivates just enough to lower the amount of allocated memory below the maximum allowed by the user. The reasoning behind this approach is that, by deactivating well performing leaves an advantage

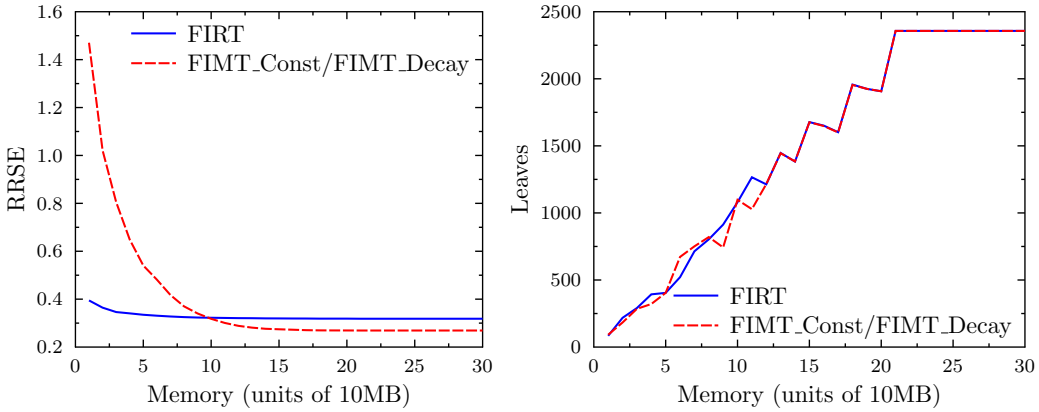


Figure 10: Ilustration of the dependency of the model's quality on the amount of available operating memory

is given to the leaves with higher error whose accuracy can improve with further refinements. Ignoring bad split points can also save some memory. It can be accompanied with ignoring the least promising attributes as well, although we have observed that this method will produce less accurate models. It is well known in the literature of learning decision and regression trees that even the least promising attributes can change the direction of the search and bring us to a better model globally Kohavi and Kunz (1997).

To evaluate the algorithm's dependency on memory constraints, we have measured the accuracy and the size of the models obtained after processing the whole sequence of training data, given different amounts of available memory. Figure 10 depicts the dependency of the model's accuracy on the available memory for the Fried dataset (the most complex one). We can observe that the quality of the regression trees does not significantly depend on the available operating memory. The models have the same quality given 20MB or 200MB of memory. For the additional option of training perceptrons in the leaves, more memory is required: FIMT achieves the same accuracy as FIRT at ∼100MB of allocated memory. This is due to the fact that for training perceptrons in each leaf node, more statistics are required to be stored, which, given the constrained memory, restricts the size of the tree. However, if the algorithm is allowed to allocate 150MB of memory FIMT can achieve better accuracy than FIRT. It is also interesting to note that, as soon as the user is satisfied with the accuracy of the model tree, the growth of the tree can be restricted and thus no more memory would be allocated as long as the data distribution and the target function remain constant.

One of the most important characteristics of the proposed algorithm, along with the anytime property and the ability of learning without storing any examples, is its processing speed. The processing speed of FIRT-DD is dependent mostly on the number of attributes, the number of distinct values of each attribute, and the frequency of change. However, as a rough indication of the processing abilities of the incremental algorithm we give the following processing times for both the simulated and the real datasets.

In Figure 11, we give a comparison of the learning times of FIRT and FIMT_Decay, with those of the algorithms OnlineRD and OnlineRA. We can see that for OnlineRD and OnlineRA, the time of learning increases almost exponentially with the number of examples. On the other hand, FIRT and FIMT_Decay manage to maintain almost constant processing time with an average speed of approximately 24000 examples per second (e/s). For the real-world datasets, the average processing speed is approximately 19000 examples per second, without the strategy for growing alternate trees enabled. This option would decrease the speed due to an increased number of statistics which need to be maintained for further growing.

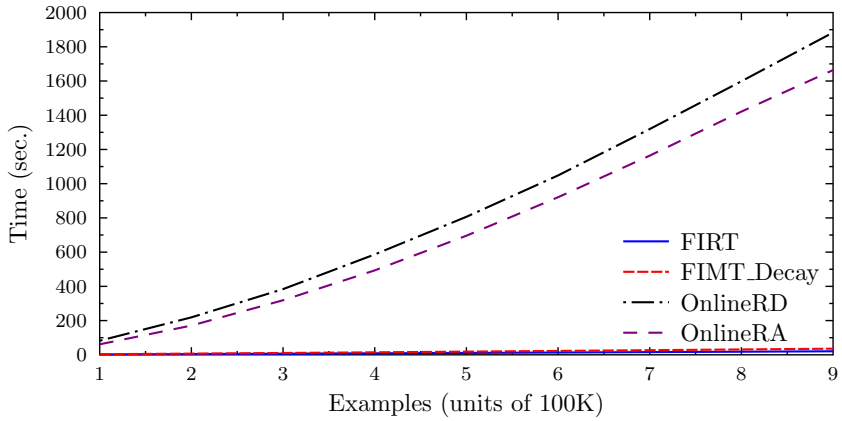


Figure 11: Comparison on the learning time by using the window-holdout method on the Fried dataset

6.8.3 Bias-Variance Analysis

A very useful analytical tool for evaluating the quality of models is the bias-variance decomposition of the mean squared error, previously discussed in Chapter 5. The bias component of the error is an indication of the intrinsic capability of the method to model the phenomenon under study. The variance measures the variability of the predictions given different training sets.

Experiments were performed for all the real datasets over the same folds used in the cross-validation and for the artificial ones over the same training sets and the same test set. The experimental methodology is the following: We train 10 models on the 10 independent training sets and log the predictions of the corresponding models over the test set. Those predictions are then used to compute the bias and the variance using Equation 12 for squared loss.

Table 6 gives the bias-variance profile of the models learned by the algorithms FIRT, FIMT_Const, OnlineRD and CUBIST. The incremental algorithms (FIRT and OnlineRD) have the smallest values for the variance component of the error, which accounts for smaller variability in predictions, given different training sets. This suggests that incrementally built trees are more stable and are less dependent on the choice of training data.

On the real datasets, CUBIST has the smallest values for the bias component of the error and FIRT has the smallest values for the variance. The bias of the incremental learners is slightly higher, but this was somewhat expected given the relatively small sized datasets. For the artificial data, incremental learners have again low variance, and for some of the problems also lower bias. For the Fried dataset, the algorithm OnlineRD is a winner. The Lexp dataset is modeled best by FIMT_Const with a significant difference in the bias component of the error. The Losc dataset is modeled best by CUBIST. On the artificial data, the OnlineRD algorithm is the most stable one.

6.8.4 Sensitivity Analysis

In this section, we present our results from the analysis of the algorithm's sensitivity to the ordering of the examples and to noise in the examples.

Stability. In the online learning scenario, the order in which examples arrive determines which sequences of training examples will be used to make the most important selection decision, i.e., the splits which are higher in the tree hierarchy. If the ordering represents a sample which is uniform across the instance space, then it is expected not to influence the choice of the best splits at any point of the tree building process. However, in practice,

Table 6: Bias-variance decomposition of the mean squared error for real and artificial datasets of the incremental algorithms FIRT, FIMT_Const, OnlineRD and the batch algorithm CUBIST.

	FIRT		FIMT_Const		OnlineRD		CUBIST	
Dataset	Bias	Variance	Bias	Variance	Bias	Variance	Bias	Variance
Abalone	1.19E+01	0.41E+01	1.22E+01	0.39E+01	1.17E+01	0.53E+01	1.16E+01	0.49E+01
Ailerons	0.00E+00	0.00E+00	1.00E-06	4.00E-06	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Mv_delve	9.80E+01	9.68E+01	9.83E+01	9.83E+01	9.79E+01	9.71E+01	9.79E+01	9.76E+01
Wind	4.26E+01	2.64E+01	4.29E+01	3.59E+01	4.20E+01	3.20E+01	4.15E+01	3.18E+01
Cal_housing	1.27E+10	7.51E+09	1.25E+10	7.75E+09	1.25E+10	7.93E+09	1.24E+10	9.37E+09
House_8L	2.58E+09	1.42E+09	2.59E+09	1.55E+09	2.59E+09	1.35E+09	2.57E+09	1.39E+09
House_16H	2.61E+09	1.09E+09	2.61E+09	1.21E+09	2.61E+09	0.93E+09	2.59E+09	1.14E+09
Elevators	0.38E-04	0.19E-04	0.59E-04	1.98E-04	0.37E-04	0.36E-04	0.37E-04	0.26E-04
Pol	1.58E+03	1.42E+03	1.58E+03	1.50E+03	1.60E+03	0.96E+03	1.56E+03	1.5E+03
Wine Quality	0.72E+00	0.21E+00	0.71E+00	0.31E+00	0.69E+00	0.24E+00	0.69E+00	0.29E+00
Fried	1.62E+00	9.20E-01	1.40E+00	4.30E-01	9.80E-01	1.30E-02	1.50E+00	9.00E-02
Lexp	2.75E-02	4.88E-02	5.57E-04	1.90E-03	1.25E-03	1.05E-03	5.91E-02	1.25E-02
Losc	1.02E-01	2.18E-02	9.61E-02	1.95E-02	1.90E-01	1.20E-05	1.33E-02	4.78E-03

it is not realistic to expect that the algorithm will observe suitable sequences for learning. Although the algorithm relies on probabilistic estimates which measure the amount of variation and the speed of convergence to the true mean values of interest, they cannot guard against orderings of examples such that the algorithm will not be able to observe the full range of possible values.

To quantify the effect of a different ordering on the performance of incremental learners, we have used both the real-world datasets given in Table 2 and the simulated datasets Fried, Losc and Lexp. The training and the testing datasets for the simulated data are of size 300K and 1000K respectively. The evaluation procedure consists of random shuffling of the training data, after which the model is evaluated on the separate testing dataset. In particular, we shuffle the training set ten times and measure the error, the learning time, the model size and the split chosen at the root node of the tree. The results show that the variance of the root relative squared error over all of the real datasets is in the range $[1.3 \times 10^{-5}, 5.6 \times 10^{-4}]$, with a mean value of 0.000215. We could observe that the split at the root node for all of the datasets and for all of the permutations of training examples remained the same most of the time. More precisely, the percentage of cases with a different split at the root node averaged over all of the different datasets is 14%. The interesting result is that, this was observed only for some of the real-world problems. These were datasets for which there exist at least two choices for a best splitting attribute in the root node.

Robustness to noise. For the analysis of the algorithm's robustness to noise we generated a separate test set of size 300K examples with a fixed amount of noise equal to 10%. The noise introduced in the artificial datasets Fried, Lexp and Losc is a perturbation factor, which shifts attributes from their true value. The offset is drawn randomly from a uniform distribution, the range of which is a specified percentage of the total value range. The size of the training sets was fixed to 1000K examples and the level of noise was systematically increased from 0 to 75%. For every noise level we measured the accuracy and the model size.

From the results, we can conclude that the incremental learners respond to noise similarly to the batch learner CUBIST. As expected, when the level of noise rises accuracy drops. The regression trees are less sensitive to noise as compared to the model trees. They have shown best results on the Fried dataset, while on the other datasets, there is no significant difference among the algorithms.

6.9 Empirical Evaluation of Learning under Concept Drift

In this section, we present the results from our empirical evaluation of the change detection and adaptation mechanisms in FIMT-DD. We have prepared a controlled environment by simulating several scenarios of different types of concept drift. Our interest is to evaluate how fast the algorithm is able to detect a change, how resilient it is to type-II errors (false negatives) and what is its ability to recover and update the model according to the new target function. For all the experiments, we have set the parameters required for the change detection test to $\alpha = 0.005$ and $\lambda = 50$ and the time interval for evaluating the alternating trees to $T_{min} = 150$.

6.9.1 Change Detection

The second part of the experimental evaluation analyzes and evaluates the effectiveness of the change detection methods proposed. The comparison is performed only on the artificial datasets (each with a size of 1 million examples), where the drift is known and controllable. This allows us to make precise measurement of delays, false alarms and undetected changes.

Tables 7 gives averaged results over 10 experiments for each of the artificial datasets. We measure the number of false alarms for each point of drift, the Page-Hinkley test delay

(number of examples monitored by the PH test before the detection) and the global delay (number of examples processed in the tree from the point of the concept drift until the moment of the detection). The delay of the Page-Hinkley test gives an indication of how fast the algorithm will be able to start the adaptation strategy at the local level, while the global delay shows how fast the change was detected globally. Table 7 contains results on change detection when the pruning strategy is used, while Table 8 contains results when the strategy of growing alternate trees is used.

The general conclusions are that both of the proposed methods are quite robust, and rarely trigger false alarms. An exception is the dataset with gradual concept drift, for which the TD method triggers one false alarm in seven of ten experiments. Both of the methods detect all the simulated changes for all the types of drift. For local abrupt concept drift, the BU method has larger global delay but smaller PH delay. This indicates that the TD method can enable faster adaptation if the localization of the drift is correct. The situation is the opposite for global abrupt drift. For global gradual drift, the situation cannot be resolved by only looking at the delays. Therefore, further discussion is provided in the following sub-sections. Experiments were performed also on datasets without drift, where the change detection methods did not trigger any false alarms.

To examine FIMT-DD's resilience to type-I (false positive) and type-II errors (false negative), we perform a set of experiments over all the datasets (stationary and non-stationary) with the change detection option activated. FIMT-DD does not generate any false positives on the stationary datasets or false negatives on the non-stationary datasets.

6.9.2 Adaptation to Change

To evaluate the change adaptation mechanism, we use the change detection methods within FIMT-DD in conjunction with two possible adaptation strategies and compare them to the "No Detection" option and to the incremental algorithms OnlineRD and OnlineRA (Potts and Sammut, 2005). Table 9 gives the resulting performance after the potential adaptation to change, estimated over the last 10000 examples of the stream, using the holdout-window evaluation method. The discussion is structured by the different scenarios of simulated drift.

The results for the Local abrupt drift dataset show that the algorithms OnlineRD and OnlineRA have poor abilities to adapt to local abrupt drift. The error of OnlineRD and OnlineRA is higher than that of FIMT-DD with the "No Detection" option. Best results are achieved with the AltTree strategy.

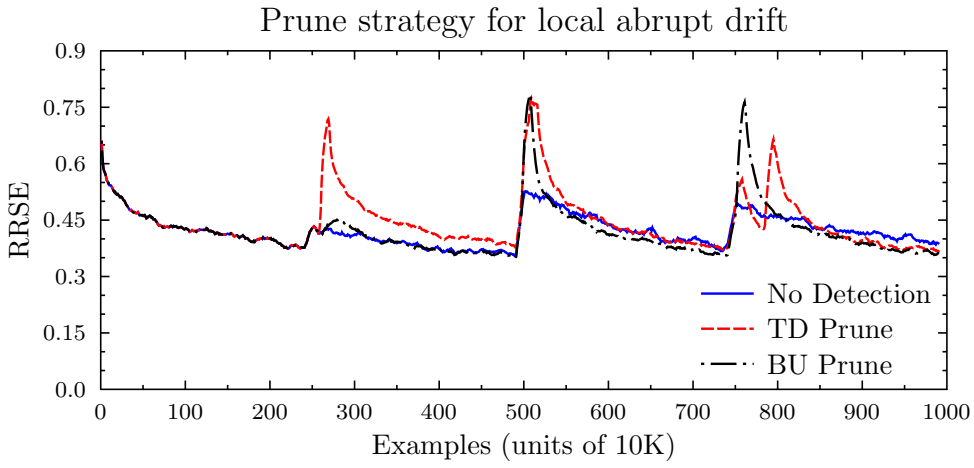


Figure 12: Learning curves for FIMT-DD with the Prune adaptation strategy to local abrupt drift in the Fried dataset. The RRSE is estimated using holdout sliding window evaluation over a window of size 10K and a sliding step 1K.

Table 7: Averaged results from the evaluation of change detection in FIMT-DD with the pruning adaptation strategy over 10 experiments. Datasets: LAD = Local Abrupt Drift, GAD = Global Abrupt Drift, GGD = Global Gradual Drift. Change detection methods: TD = Top-Down, BU = Bottom-Up. Evaluation metrics used: #FA's = number of false alarms, Global Delay = number of examples observed from the actual change point to the moment of change detection, PH Delay = number of examples observed in the node before the moment of change detection.

			Change point 250k		Change point 500k		Change point 750k	
		#FA's	Global Delay	PH Delay	Global Delay	PH Delay	Global Delay	PH Delay
LAD	TD	0.0	12071.0	4006.9	5029.3	1458.2	8631.5	894.1
	BU	0.0	30403.0	1542.1	6967.7	550.3	9519.0	601.9
GAD	TD	0.0	NA	NA	1995.6	273.3	7002.7	676.9
	BU	0.0	NA	NA	374.1	374.1	412.5	412.5
GGD	TD	0.7	NA	NA	21673.8	3473.5	17076.0	6200.6
	BU	0.0	NA	NA	18934.6	18934.6	19294.7	2850.0

Table 8: Averaged results from the evaluation of change detection in FIMT-DD with the alternate trees adaptation strategy over 10 experiments.Datasets: LAD = Local Abrupt Drift, GAD = Global Abrupt Drift, GGD = Global Gradual Drift. Change detection methods: TD = Top-Down, BU = Bottom-Up. Evaluation metrics used: #FA's = number of false alarms, #Adaptations = number of adaptation actions, Global Delay = number of examples observed from the actual change point to the moment of change detection, PH Delay = number of examples observed in the node before the moment of change detection.

				Change point 250k		Change point 500k		Change point 750k	
		#FA's	#Adaptations	Global Delay	PH Delay	Global Delay	PH Delay	Global Delay	PH Delay
LAD	TD	0.0	20.8	12071.0	4006.9	4282.7	768.4	5438.7	548.2
	BU	0.0	16.3	30403.0	1542.1	6582.6	457.3	6863.7	1399.2
GAD	TD	0.0	27.4	NA	NA	1995.6	273.3	5692.3	451.7
	BU	0.0	3.8	NA	NA	374.1	374.1	410.5	410.5
GGD	TD	0.0	43.7	NA	NA	21673.8	3473.5	12389.8	3157.9
	BU	0.0	52.8	NA	NA	18934.6	18934.6	16684.5	13186.1

Table 9: Performance results for FIMT-DD over the last 10000 examples of the data stream (never used for learning) averaged over 10 experiments for each type of simulated drift. Datasets: LAD = Local Abrupt Drift, GAD = Global Abrupt Drift, GGD = Global Gradual Drift. Change detection methods: TD = Top-Down, BU = Bottom-Up. Strategies used in the comparison: "No Detection", OnlineRD, OnlineRA, Top-Down in conjunction with the Prune and the AltTree strategy, and Bottom-Up in conjunction with the Prune and the AltTree strategy.

					Top-Down		Bottom-Up	
Data	Measures	"No Detection"	OnlineRD	OnlineRA	Prune	AltTree	Prune	AltTree
LAD	MSE±σ	4.23±0.11	5.95±1.09	9.12±0.09	4.62±0.09	3.17±0.07	3.88±0.08	3.19±0.07
	RRSE	0.38	0.45	0.56	0.396	0.33	0.37	0.33
	Leaves	2489.00	173.6	109.3	391.6	1840.69	583.20	1879.30
	Nodes	4977.00	/	/	782.2	3680.39	1165.40	3757.60
GAD	MSE±σ	3.81±0.07	1.21±0.002	2.71±0.01	4.05±0.07	3.58±0.06	3.67±0.07	3.66±0.07
	RRSE	0.39	0.22	0.33	0.398	0.38	0.38	0.38
	Leaves	2576.70	177.9	101.9	430.9	629.20	572.10	572.10
	Nodes	5152.39	/	/	860.79	1257.40	1143.19	1143.19
GGD	MSE±σ	9.41±0.19	4.35±3.53	17.88±2.0	3.25±0.06	3.56±0.06	3.38±0.06	3.95±0.07
	RRSE	0.56	0.37	0.76	0.326	0.34	0.33	0.36
	Leaves	2616.70	241.8	115.2	323.6	528.30	500.20	635.10
	Nodes	5232.39	/	/	646.20	1055.59	999.40	1269.19

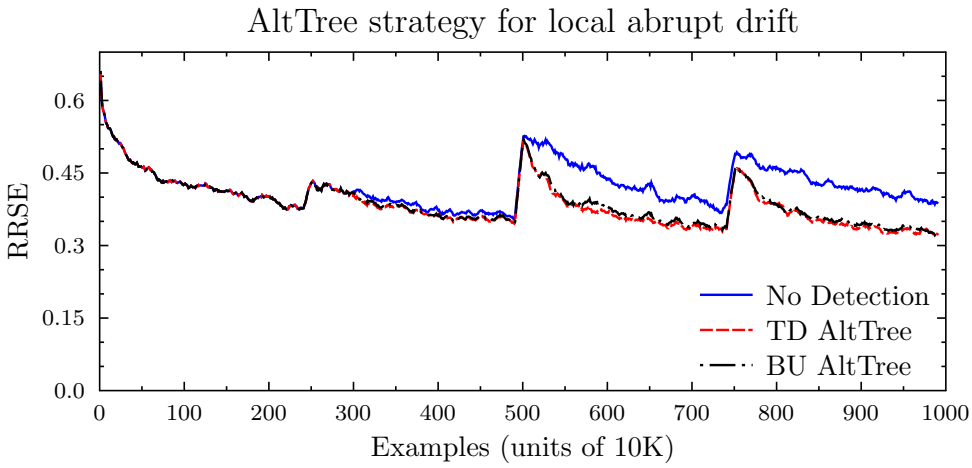


Figure 13: Learning curves for FIMT-DD with the AltTree adaptation approach to local abrupt drift in the Fried dataset. The RRSE is estimated using holdout sliding window evaluation over a window of size 10K and a sliding step 1K.

A more detailed analysis of using the "Prune" strategy has shown that the TD method mostly detects changes in the higher nodes of the tree. This can be seen in Figure 12, where for the third point of change we observe an additional peak due to the pruning of a significant number of nodes (which might be overprunning). The BU method, on the other hand, detects changes typically at the lowest node whose region is closest to the region affected by the drift. This is a sign of good localization of the concept drift. In the same figure, we observe that although at the end the results are similar, the BU change detection method enables faster adaptation to the new concept, which is due to the proper localization of change.

Figure 13 compares the learning curves for the two different change detection methods (BU and TD) and the AltTree strategy, obtained by using the sliding window evaluation on the Local Abrupt Drift dataset. In the comparison, we also include the situation when no change detection is performed (solid line). The AltTree strategy prevents reactions to false alarms. Drastic error increase (drop of accuracy) is avoided. The models have similar errors with both of the change detection methods proposed.

The analysis on the Global abrupt drift dataset yields different results. Best accuracy is achieved by the OnlineRD and OnlineRA algorithms. They are able to completely rebuild the tree at each point of change better than FIRT-DD. The reason for this, we believe is, OnlineRD and OnlineRA are in general able to induce an equally accurate model tree using fewer training examples than FIRT-DD, thus, the improvements in accuracy are greater for the same amount of data. Since the drift is global, Prune and AltTree do the same thing, re-growing the whole tree after the change, thus the results are similar. The BU method performs better due to the proper detection of drift. The TD approach does not detect the change at the root node, but somewhere at the lower nodes, which can be seen in Figure 14. This is evident also from the number of adaptations given in Table 8, which is almost seven times higher than for BU. Because of that, neither of the adaptation strategies used with TD will enable proper correction of the structure. If "No detection" is used, the size of the tree becomes around five times larger as compared to the other strategies.

For the Global gradual drift dataset, best results are achieved using the Prune strategy. The results are in favor of the TD method. The trees obtained are smaller and more accurate because of the on-time adaptation. For the same reasons, pruning is a better strategy in this case than growing alternate trees. The BU method performs worse than TD because, once a change is detected, the adaptation of the model hinders the detection of additional changes although the drift might still be present and increasing.

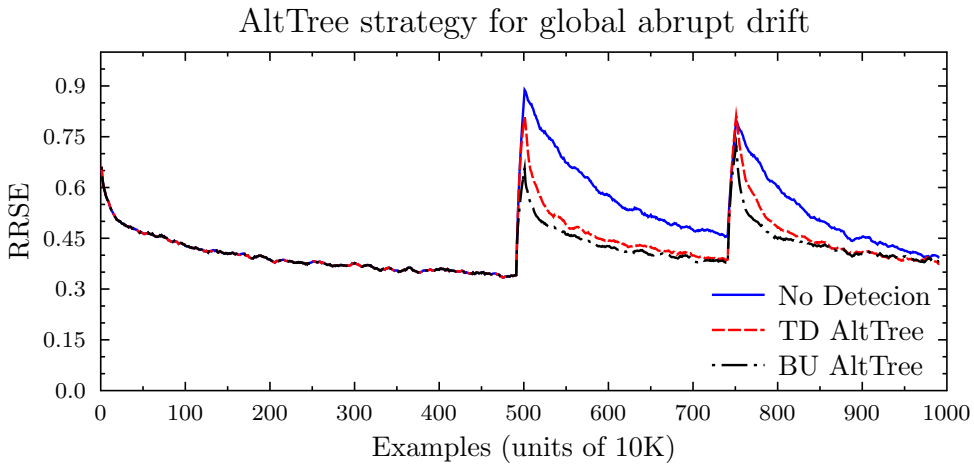


Figure 14: Learning curves for FIMT-DD with the AltTree adaptation approach to global abrupt drift in the Fried dataset. The RRSE is estimated using holdout sliding window evaluation over a window of size 10K and a sliding step 1K

As a general conclusion, when no detection is performed the tree improves its accuracy due to consecutive splitting, but becomes much larger than necessary. The structure would likely be misleading due to an improper hierarchy of splits. For different types of drift, different algorithms perform best and there is no generally applicable method.

6.9.3 Results on Real-World data

Figure 15 gives a comparison of the different adaptation strategies (both in terms of accuracy and model size) on the Airline dataset from the DataExpo09 competition. The reference case is "No detection". The AltTree strategy improves greatly the accuracy at any time, while providing significantly smaller models. When no change detection is performed the tree contains nearly 50 thousand leaves, of which only 7.5 thousand can grow further.

Table 10: The performance of FIMT-DD on the Data Expo Airline dataset, measured over the last 10K examples of the data stream (never used for learning), averaged over 10 experiments for each type of simulated drift.

		Top Down		Bottom Up	
Measures	No Det.	Prune	AltTree	Prune	AltTree
AE	17.99	14.91	14.80	14.85	14.79
MSE±σ	862.54±8.56	741.90±8.69	731.73±8.62	733.49±8.59	728.56±8.58
RE	0.99	0.82	0.81	0.81	0.80
RRSE	1.02	0.94	0.93	0.93	0.93
Nodes	86509.00	31.00	143.00	101.00	41.00
Growing nodes	7558.00	16.00	72.00	51.00	21.00

The learning curves confirm the existence of many concept drifts, which are occurring irregularly at very high rates. The changes are of both local and global type, resulting in partial or complete restructuring of the tree. The results in Table 10 obtained for the last 10K examples seen, show that the AltTree strategy achieves continuously smaller error and performs equally well with both of the change detection methods (TD or BU).

The most influential attributes were, as suspected, the *Origin*, the *Destination*, the *Flight time*, the *Day of week*, the *Day of month* and the *Departure* and *Arrival time*. Most of the changes were detected for splits on the attributes *Origin* and *Destination*, moderately on

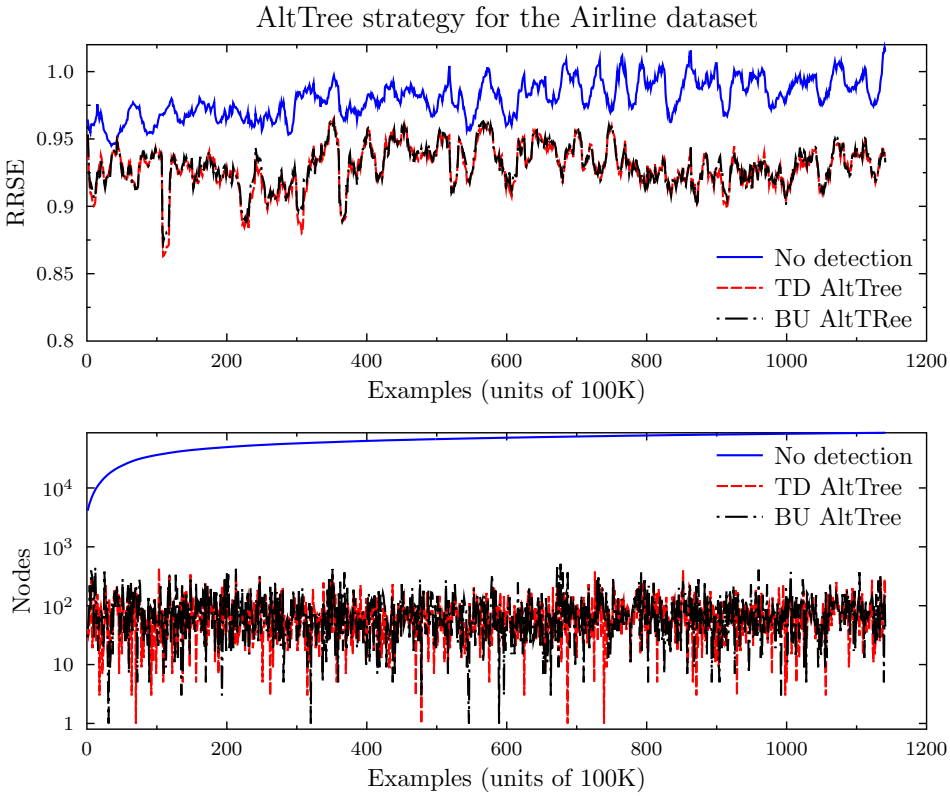


Figure 15: Learning curves for FIMT-DD on the Data Expo Airline dataset. The RRSE is estimated by using sliding prequential-window evaluation with size 1M and sliding step 100K.

the attribute *Unique Carrier* and only few on the date and the time of flying. This suggests that, the delays were most of the time related to the flying destinations and the carriers.

A change in the frequency of the flights from or to a certain destination, the closing of an airport, an airline entering a crisis or similar events can change the relations between the target attribute (the delay) and the input attributes. Due to the explicit change detection mechanism of FIMT-DD, the regression tree can be used to explain and describe such changes which occurred at irregular periods over the years.

6.10 Summary

To the best of our knowledge, FIMT-DD is the first algorithm for learning model trees from time-changing data streams with explicit drift detection. The algorithm is able to learn very fast, offering an excellent processing and prediction time per example. The only memory it requires is for storing sufficient statistics at tree leaves which is independent from the size of the input. The model tree is available for use at any time during the course of learning and will always reflect the latest changes in the functional dependencies. This is provided through local change detection and adaptation, avoiding the costs of re-growing the whole tree when only local changes are necessary.

In terms of accuracy, FIMT-DD is competitive with batch algorithms even for medium sized datasets and has smaller values for the variance component of the error. In a sense, this means that, given an infinite stream of instances arriving at high speed, the incrementally learned model tree will have asymptotically approximately comparable accuracy to the tree learned by an assumed ideal batch learner that is able to process an infinite amount of training data. In addition, the algorithm effectively maintains an up-to-date model even in the presence of different types of concept drifts. Both of the proposed methods for change

detection are quite robust, and rarely trigger false alarms. With respect to the ability to handle concept drift, the general conclusion is that the best combination of a change detection (TD and BU) and adaptation method (Prune and AltTree) depends largely on the type of concept drift.

7 Online Option Trees for Regression

You may delay, but time will not.

Benjamin Franklin

Hoeffding-based algorithms for learning decision or regression trees are considered as one of the most popular methods for online classification and regression. Due to their ability to make valid selection and stopping decisions based on probabilistic estimates, they are able to analyze large quantities of data that go beyond the processing abilities of batch learning algorithms. The probabilistic estimates enable *provably approximately correct* decisions on the estimated advantage of one attribute over another. This type of a selection decision is based on an estimation of the merit of each candidate refinement of a particular hypothesis, which corresponds to a simple hill-climbing search strategy in the space of possible hypotheses. Hill-climbing search strategies are known for their inability to avoid local minima, and tree building approaches that employ them are susceptible to myopia.

In this chapter, we discuss the idea of introducing option nodes in regression trees as a method for avoiding the problem of local minima. We start with a presentation of existing algorithms for capping options for Hoeffding (decision) trees, and continue with a presentation of our algorithm for online learning of option trees for regression. The presentation is structured in two sections, of which, the first one discusses the splitting criterion, and the second one discusses methods for aggregating multiple predictions. The last section present an experimental evaluation conducted on a number of real-world and artificial datasets. Our results show that an online option tree has better generalization power as compared to a Hoeffding-based regression tree, resulting from the better exploration of the hypothesis space. At the same time, the use of options enables us to resolve ambiguous situations and reach selection decisions sooner.

7.1 Capping Options for Hoeffding Trees

Pfahringer et al. (2007) were the first to explore the idea of using options for improving the accuracy of online classification. Their algorithm, termed *Hoeffding Option Trees*, extends the Hoeffding tree algorithm of Domingos and Hulten (2000) by introducing a mechanism for appending additional splits to the existing internal nodes. In particular, the algorithm performs periodical reevaluation of each selection decision. If a different split is probabilistically estimated to have higher merit than the existing one, it will be added as an optional split.

The question that one might ask now is: "*Why would a different split have a higher merit than the one which was previously chosen based on the collected statistical evidence?*". The answer to this question is three-fold. The first part is concerned with the probabilistic nature of the selection process. Namely, although the probability for a correct selection can be very high, there is still a non-zero chance for a failure. The second part takes into account the existence of tie situations and the assumption that the third-best split or lower

can be safely removed from the "competition". The third part is related with the possibility for changes (concept drift) in the functional dependency modeled with the tree.

In the work of Pfahringer et al. (2007), the options are not introduced during the split selection phase. Their algorithm gradually adds optional splits to the existing ones until a maximal number of allowed options is reached. When computing a prediction, the option tree uses a weighted voting strategy to combine all of the predictions produced by the alternate sub-trees. This strategy sums the the individual probabilistic predictions (per class) which were obtained from the alternate subtrees and outputs the class with the highest probability. A slightly improved version of the same algorithm termed *Adaptive Hoeffding Option Trees* has been proposed in which each leaf stores an estimation of its current error using an exponentially weighted moving average (Bifet et al., 2009b). This estimation is used in the voting scheme to adjust the weight of each node proportional to the square of the inverse of its error.

Following a similar approach, Liu et al. (2009) have proposed another variant of introducing options by using the change detection features of the CVFDT algorithm (Hulten et al., 2001). Options are appended to the existing splits based on a similar reevaluation. The main difference is in the method for combining the available predictions. The approach of Liu et al. (2009) selects the most recent and most accurate alternate sub-tree from which a prediction will be derived. An optional split would thus reflect the most recent and best performing "knowledge".

Option nodes encode valuable information on the existence of alternative splits at various parts of the tree structure. The alternative splits point out different directions for exploring the hypothesis space and emphasize the ambiguity in the model. They are an interesting direction for extending Hoeffding-based trees, as noted previously by Pfahringer et al. (2007).

However, we see some room for improvement in the way options are introduced. Our observation is that, due to the fact that the Hoeffding bound is conservative and requires a considerable number of examples to be processed before reaching a statistical stability, options will be introduced with some delay that translates into a delay in the process of learning. In addition, the proposed algorithms impose sufficient statistics to be continuously maintained in all of the internal nodes. This implies an additional increase in the memory consumption and the processing time per example. In the following section, we present our suggestions for improvements. Finally, options have so far not been used in online regression trees.

7.2 Options for Speeding-up Hoeffding-based Regression Trees

Let us look closely at the selection phase of Hoeffding-based tree learning algorithm. Given a desired level of confidence and a range for the dispersion of the random variables, the error assigned to every probabilistic estimate decreases gradually with a constant speed. While some candidate splits exhibit strong advantage over the rest, there are situations in which there is no clear difference in the estimated merits of two or more of them. The Hoeffding bound is ignorant of the value of the mean or the variance of the random variables and does not take into account situations of this type. Therefore, when the value of the estimated mean is zero, the algorithm will not be able to make an informed decision even if an infinite amount of examples are observed.

This problem has been resolved in an ad-hoc fashion with the previously discussed tie-breaking mechanism. The tie-breaking mechanism determines an amount of data which has to be processed before any decision is reached. As a result, a constant delay will be associated to every ambiguous situation.

Here, we propose a different solution, which exploits option nodes as an ambiguity resolving mechanism. The main idea as follows: If the standard selection decision is replaced with an easier selection decision, where multiple splits are chosen and added as options, there is no need to wait for more statistical evidence. A selection decision on multiple splits will be reached sooner, although under a higher approximation error. However, each of these decision can revised later when more evidence has been collected for each possible split.

There is an additional value in introducing options in the structure of the regression tree, which is not restricted to Hoeffding trees only. If we think of tree induction as search through the hypothesis space H, an option node is best understood as a branching point in the search trajectory. Each optional split represents a different direction that can be followed in the exploration of the search space. The hill-climbing search strategy is therefore replaced with a more robust one, which will enable us to revise the selection decisions in hindsight. As a result, an option tree has a different inductive bias as compared to an ordinary decision, because it will not necessarily select the *first* smallest tree that fits the training data best in the general-to-specific ordering of hypotheses. Being able to explore a larger sub-space of H, an option tree is expected to be more stable and more robust than an ordinary decision tree.

7.2.1 Ambiguity-based Splitting Criterion

We begin with a brief high-level description of the ORTO algorithm and then discuss the relevant points in more detail. The algorithm follows the standard top-down approach for learning regression trees. The pseudo-code of ORTO with a generic strategy for combining the predictions is given in Algorithm 4.

Similarly to FIMT-DD, it starts with an empty leaf and reads examples from the stream in the order of their arrival. Each example is traversed to a leaf where the necessary statistics (such as $\sum y_k$, $\sum y_k^2$) are maintained per splitting point. After a minimum of n_{min} examples have been observed in the leafs of the tree, the algorithm examines the splitting criterion. If an ambiguous situation is encountered the algorithm will proceed with creating an option node. Otherwise, a standard splitting node will be introduced and the procedure will be performed recursively for the leaf nodes in which a modulo of n_{min} examples have been observed. Differently to FIMT-DD, the ORTO algorithm makes use of a prediction rule that enables it to combine multiple predictions obtained for a single example. The prediction rule is used only in the testing phase, thus the algorithm for building an option tree is invariant on the choice of the strategy for combining the predictions.

Given the first portion of instances n, the algorithm computes the best split for each attribute and ranks these splits according to their variance reduction (VR) value. Let A_1, A_2, A_3, ..., A_d be the attribute ranking obtained after observing n examples. Let us further consider the ratio of the variance reduction values of any attribute from the subset $\{A_2, \ldots, A_d\}$ and the best one A_1, e.g., $r_n = VR(A_2)/VR(A_1)$, at the moment of observing n examples. We observe r_1, r_2, ..., r_n as independent random variables with values in the range [0,1], that is, for $1 \leq i \leq n$ it holds that $\mathrm{P}(r_i \in [0,1]) = 1$. The empirical mean of these variables is $\bar{r} = \frac{1}{n}\sum_{i=1}^{n} r_i$.

By using the Hoeffding bound in the same way as in FIMT-DD, we can apply confidence intervals on the probability for an error in approximating the true average with the estimation $\bar{r} \pm \varepsilon$, where ε is computed using the Equation 40. To allow the introduction of option nodes, we make the following modification: if, after observing n_{min} examples, the inequality $\bar{r} + \varepsilon < 1$ holds, a normal split is performed; otherwise, an option node is introduced with splits on all the attributes A_i for which the inequality:

$$VR(A_i)/VR(A_1) > 1 - \varepsilon \text{ , for } i \neq 1 \tag{53}$$

is satisfied. The assumption underlying this inequality is as follows: All the splits for which

Algorithm 4 *ORTO: An incremental option tree learning algorithm.* Pseudocode.

Input: δ - confidence level, n_{min} - chunk size, N_{Att} - number of attributes, Agg - aggregation method, y - decay factor
Output: T - current option model tree

```
for ∞ do
    e ← ReadNext()
    Seen ← Seen + 1
    Leaf ← Traverse(T, e)
                    ▷ The example e is traversed to the corresponding leaf node Leaf.
    UpdateStatistics(Leaf, e)
                ▷ Updates the split-evaluation statistics using the attribute vector of e.
    TrainThePerceptron(Leaf, e)
    if Seen mod n_min = 0 then
        for i = 1 → N_Att do
            S_i = FindBestSplit(i)                ▷ Computes the best spit per attribute.
        end for
        S_a ← Best(S_i), i = 1,...,N_Att
        S_b ← SecondBest(S_i), i = 1,...,N_Att
        if S_b/S_a < 1 − sqrt(ln(1/δ) / (2×N)) then
            MakeSplit(T, S_a)                     ▷ Creates a binary split using S_a.
        else
            O ← CreateOptionNode(T)   ▷ Transforms the leaf node T into an option node.
            MakeOption(O, S_a)                    ▷ Creates one optional split applying S_a.
            Counter ← 0
            k ← CountPossibleOptions(O)
                                ▷ Computes the number of possible optional splits.
            l ← GetCurrentLevel(T)                ▷ The root node is at level 0.
            for i = 1 → N_Att do
                if i ≠ a ∧ Counter < k × y^l ∧ S_i/S_a > 1 − sqrt(ln(1/δ) / (2×Seen)) then
                    MakeOption(O, S_i)            ▷ Creates an optional split applying S_i.
                    Counter ← Counter + 1
                end if
            end for
        end if
    end if
return Prediction(T, Agg, e)
end for
```

the inequality $\bar{r} + \varepsilon < 1$ does not hold are *approximately* equally discriminative, i.e., as discriminative or with an approximately equal variance reduction as the best one on the observed sample of the data. The fact that this inequality does not hold is an expression of the lack of the necessary confidence to discard them.

In other words, as long as the inequality $\bar{r} + \varepsilon < 1$ does not hold, the algorithm has to collect more statistical evidence in favor of the observed best split. In general, when the algorithm needs more evidence than the n_{min} initial examples to determine the best split, we either have a tie situation in which A_1 competes with one or several other candidate splits, or noisy data, which make the evaluation less reliable. Instead of waiting for more examples to be observed, which would eventually give preference to one of the splits or declare a tie situation, we accept all of the competitive splits as allowed directions for the search. With the modified splitting criterion, the tree is allowed to grow faster. At the same time, having multiple options is a strategy to overcome the one-step look-ahead "myopia" of greedy search.

7.2.2 Limiting the Number of Options

With this approach however, we might encounter situations in which multiple candidate splits are equally discriminative with respect to the target attribute. Allowing splits of this type is a reasonable decision, when the reduction of the variance is estimated to be high. However, in situations when multiple splits are equally discriminative but do not contribute a significant reduction of the variance, allowing options on all of them is unnecessary and would result in an excessive growth of the tree. For this reason, it is crucial to have some form of a restriction on the number of options or the number of trees, which will control the tree growth and the memory allocation. Having in mind that the greatest reduction of variance is achieved with splits which are positioned at the higher levels of the hierarchy; that is, near the root node, an intuitive approach is to reduce the number of maximum allowed options proportionally with the level of the node. Although there is no theoretical support to the claim that options are most useful near the root of the tree, this has been empirically shown by Kohavi and Kunz (1997) in their study on batch methods for learning option trees.

We decided to rely on the existing empirical evidence for the batch learning setup and introduce a restriction on the number of options with the depth of the tree. The number of possible options is thus given with

$$o = k \cdot \beta^{level}, \tag{54}$$

where k is the number of competitive attributes chosen using the inequality in Equation 53, and β is a decay factor. We refer to o as the decaying option factor, which will be used as an alternative to the basic algorithm (with option factor k). The levels in the tree are enumerated starting with 0 at the root.

For an intuitive comparison of option trees with ensembles of trees, it is much more interesting to use a different way to limit the size of the option tree. Instead of deciding explicitly on the number of allowed options per level or per node, one can place a constraint on the maximum number of trees represented with a single option tree by using a parameter T_{max}. When this number is reached, introducing options will no longer be allowed. This control mechanism is similar to the way the size of an ensemble is constrained; that is, by limiting the number of possible base models. Therefore, we can compare the predictive accuracy and the memory consumption of an ensemble with those of an option tree that represents an equal number of base models.

7.3 Methods for Aggregating Multiple Predictions

Decision and regression trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification or the numerical prediction of the instance. A standard node specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. Therefore, there is only one possible outcome of the test. An option node, on the other hand, acts like an aggregation function. Each branch descending from an option node corresponds to a different descendant node. Instead of a single test of some attribute, an option node contains a set of tests on multiple different attributes. Therefore, at each option node, there are multiple possible outcomes.

Having the option to choose among multiple tests provides an opportunity of exploring more possible paths in the tree, i.e., conjunctions of constraints (tests) on the attribute values. It also provides a set of predictions, which can be aggregated or combined in various ways. We have analyzed and evaluated multiple strategies for combining multiple predictions. They can be, in general, divided into two categories: 1) strategies that compute an aggregate of multiple predictions, and 2) strategies that choose an "expert" or the "best" prediction. Before dwelling on the particular details of each method proposed, we introduce the necessary notation.

Each of the strategies corresponds to a decision rule that is applied at each option node. Let m denote the number of options at a given option node. Let $p_i(t)$ denote the prediction at time t from an option $i \in \{1,...,m\}$. For clarity, we will use the shorthand p_i, unless it is necessary to use the time. We propose four different methods for combining multiple predictions and present the details of each method below. These are:

- **Averaging.** The most straightforward aggregation method for regression is by non-weighted averaging. This means that equal importance is being given to all of the options. An example which is sorted down to an option node will follow all of the branches descending form the option node. The predictions provided by the leaves where the example fell will be back-propagated to the option node and aggregated. The aggregated prediction will be further back-propagated up to the root node.

- **Selective averaging.** We have also considered the variant of a selective averaging, in which we select the predictions that will be used in the process of aggregation according to the confidence of the leaf making them. In particular, we sum the variance of all the leaves whose predictions are included in the aggregation. Let $v_i(t)$ denote the variance of the leaf i at time t. For ease of presentation, we will use the shorthand v_i. The total averaged variance is : $v = \frac{1}{m}\sum_{i=1}^{m} v_i$. Then, only the predictions from the leaves whose variance is below the average are aggregated by using simple non-weighted averaging.

  With respect to the computational complexity of the procedure for computing the predictions, we can compute an upper bound of $O(k^{\log h})$ where h is the height of the tree. However, the number of option nodes on the path from the root node to the corresponding leaf node would rarely get close to $\log h$, which makes this bound very loose. In addition it can be further reduced by applying the decaying option factor. The experimental evaluation has shown that the computational complexity can be easily controlled by allowing a small performance degradation.

- **Best path.** This method belongs to the second category, which selects a single "best" prediction from the set of multiple predictions. The best prediction in this case is obtained by determining the best path to a leaf at a given point in time. As a result, it returns a single prediction provided by the corresponding leaf.

  This method is not expected to outperform the methods which employ aggregation functions, unless there exists a single best tree that models the data particularly well

and is contained within the option tree. However, this method is useful to determine the most accurate model in the set of models compressed in the option tree. The best path will be determined as a composition of the best options. What we need is a way to evaluate the goodness of each branch descending from an option node.

As discussed previously, in the online learning scenario one may rely on a predictive sequential type of evaluation augmented with the use of fading factors. The fading factors are necessary to discard the past errors and put the emphasis on the most recent performance. We use the faded prequential mean squared error which is defined by Equation (31) from Chapter 5. The best tree at time t is composed from the subtrees below the options with the smallest faded prequential mean squared error. Basically, every training example sorted at a given option node is used for evaluating each of the options. The formula for computing the faded prequential mean squared error $L_\alpha(i,t)$ of an option i at time point t is given with:

$$M_\alpha(i,t) = \frac{S_\alpha(i,t)}{N_\alpha(i,t)} = \frac{\alpha S_\alpha(i,t-1) + SqErr(i)}{\alpha N_\alpha(i,t-1)+1}, \tag{55}$$

where α is the fading factor, $S_\alpha(i,t)$ is the faded prequential squared error and $N_\alpha(i,t)$ is the faded number of examples. Whenever a new option node is introduced, the sums are initialized to $S_\alpha(i,0) = 0$ and $N_\alpha(i,0) = 0$ for all $i \in \{1,...,m\}$. The squared error $SqErr(i)$ is simply computed as

$$SqErr(i) = (y - p_i)^2,$$

where y is the true value of the example and p_i is the prediction of option i.

- **Best Leaf.** Finally, we have also considered a method takes the prediction from the *best* or the most confident leaf. As a measure for confidence, we use the faded prequential mean squared error computed from the moment of the creation of the leaf.

From the four possible prediction rules, only two were shown to give interesting results: simple averaging and following the best path. The first strategy can be considered as a representative of the strategies that give equal importance or equal weight to each of the available predictors. An option tree with a strategy based on weighting predictions fills the gap between a single regression or model tree and an ensemble of regressors. The second strategy gives an advantage to the single best hypothesis from a set of multiple alternative hypotheses represented with the option tree. To make a distinction between these two different prediction rules of ORTO we will use the following acronyms: ORTO-A (averaging), and ORTO-BT (following the best path that represents the best tree).

7.4 Experimental Evaluation of Online Option Trees for Regression

In this section, we present the results of the experimental evaluation performed to answer the general question which inspired this work: *Do options improve the any-time performance of model trees and what is their effect in the context of bias-variance decomposition of error?*. Another interesting question which we strive to answer is whether option trees enable better exploration of the hypothesis space. We also investigate what are the costs of option trees in terms of memory and processing time.

7.4.1 Predictive Accuracy and Quality of Models

The experimental evaluation was performed on 9 of the UCI datasets used in the previous section. It consisted of a standard 10-fold cross-validation, in which we monitored the

prequential mean squared error weighted with a fading factor, as well as the evolution of the tree. The concept of predictive sequential processing of the training instances provides means for monitoring the evolution of the predictive accuracy and complexity of the model.

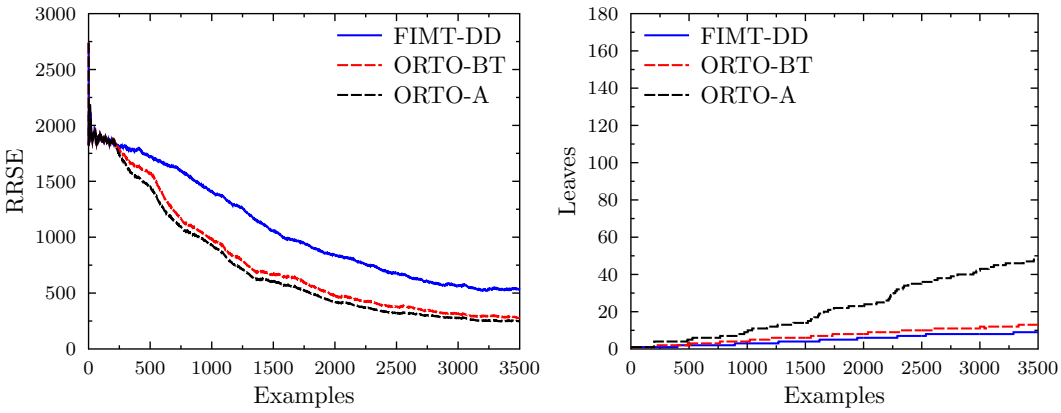


Figure 16: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Pol dataset.

Figure 16 shows the weighted prequential mean squared error (left plot) and the evolution of the tree in terms of the number of leaves (right plot), both computed for the first 3500 examples over 10 folds of the Pol dataset. Let us first look at the prequential error of FIMT-DD and ORTO-BT over the initial 3500 examples from the beginning of the tree induction process. The learning curve of ORTO-BT shows an obvious drastic drop in the root relative mean squared error. The model represents the best tree extracted from the option tree learned with ORTO. We can also notice that the regression tree learned with FIMT-DD also improves its accuracy significantly over the first training examples, but its learning rate is much lower.

The best predictive accuracy at any time is obtained by combining the predictions of all the trees represented with the option tree, that is, by using the averaging aggregation method (ORTO-A). However, this difference seems to diminish in the final stages of the learning procedure, when the models have reached the appropriate complexity. If we look at the evolution of the trees in terms of the speed with which leaves are being generated, we can see that the tree given by ORTO-BT algorithm is slightly larger than the one produced with FIMT-DD. This difference is in the range of [0,5] leaf nodes and is negligible when we compare the size of the ORTO trees to the size of the FIMT-DD trees. FIMT-DD trees are much larger and grown much faster.

The relative behavior of the three algorithms is very similar to the remaining datasets. Detailed plots are given in Appendix A section A.2. The general trend is that, for those problems for which the target function can be approximated using the available set of training examples, all three approaches give models whose accuracy is comparable at the end. An important advantage of the ORTO-A and ORTO-BT algorithms is that their learning curves reach a plato much sooner as compared to the one of the FIMT-DD algorithm. This was the exact purpose of introducing options.

Table 11 gives the results of the 10-fold cross-validation performed on the stationary UCI datasets. If we compare the performance results of the regression tree produced by FIMT-DD with the best tree that can be constructed from the non-constrained option tree ORTO-BT, we can see that (in 6 out of 9 datasets) the *best tree* from ORTO has a smaller mean squared error and more leaves. Although this advantage is not statistically significant by the Friedman statistical test accompanied with the Bonferroni-Dunn post-hoc test (at p=0.10) due to the small number of datasets used in the comparison, it is still worth to note that (in at least half of the cases) the best tree produced with options remains better

than the tree produced with FIMT-DD. This gives some evidence that the option nodes enable the tree to grow faster and provide means for a more efficient resolution of the tie situations. Larger trees tend to be more accurate for these small-sized datasets, due to the lack of data to induce a model with an appropriate complexity. While the total number of leaves in an option tree is significantly larger compared to a tree produced with FIMT-DD, its best subtree constructed by considering only the best options will be of a comparable size. A statistically significant reduction of error on the other hand can be obtained by averaging the predictions of all the subtrees in an option tree, that is, by using the ORTO-A algorithm. Our explanation is that ORTO-A smooths the crisp boundaries between the leaves, and reduces the variability in the predictions of a single regression tree.

7.4.2 Bias-Variance Analysis

To investigate the exact effect of using options and confirm the previous hypothesis, we performed a bias-variance analysis of the error for all of the algorithms and all of the datasets used in the previous comparison. The bias-variance decompositions was estimated with the same 10-fold cross-validation procedure. In Figure 17, we show a separate comparison of the bias and the variance components of the error per dataset.

The datasets are sorted by the size of the option trees in decreasing order. To be able to make a comparison across all the datasets, we normalized the error components by dividing with the corresponding error components of the regression trees produced by FIMT-DD. A result which is below a value of 100 corresponds to a smaller error component for the ORTO algorithms, while a result which is above a value of 100 tells us that the ORTO algorithm has a higher error component.

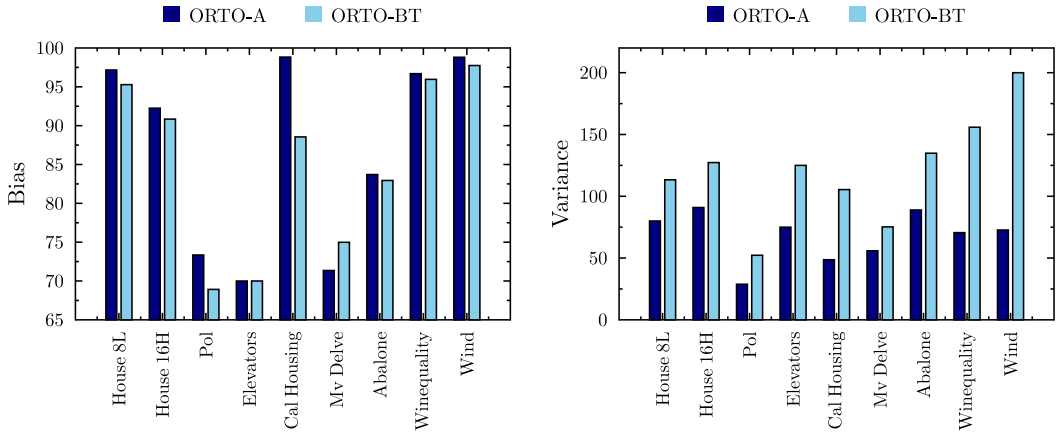


Figure 17: Bias-variance decomposition of the mean squared error of ORTO-BT and ORTO-A trees relative to their corresponding counterparts computed for the FIMT-DD trees.

The left panel in Figure 17 shows that all of the bars correspond to values which are below 100%. This is a clear indication that, in comparison to regular regression trees, option trees (with averaging or with the best tree aggregation method) have undoubtedly smaller values for the bias component of the error. Except for the Mv Delve dataset, ORTO-BT has a smaller bias error than ORTO-A. In the right panel we can observe the opposite case, where the variance component for ORTO-A is constantly smaller as compared to that of ORTO-BT trees. Relative to FIMT-DD, ORTO-A produces less variable predictions, while ORTO-BT trees tend to have an increased variance component of the error for most of the datasets (except for Pol and Mv Delve).

Our interpretation of these results is as follows. By using option nodes in the tree induction process, it is possible to obtain models which approximate the target concept better than the model produced with FIMT-DD (lower bias). However, the best single tree

Table 11: Results from 10-fold cross-validation on stationary real-world datasets. Comparison of FIMT-DD, ORTO-A, and ORTO-BT. The final MSE and size (number of leaves/rules) of models are given. Results for which ORTO-A has smaller MSE than FIMT-DD are given in italics. Results for which ORTO-BT has smaller MSE than FIMT-DD are boldfaced.

	FIMT-DD		ORTO-A		ORTO-BT	
Dataset	MSE	Size	MSE	Size	MSE	Size
Abalone	7.45E+0 ± 1.25E+0	7.80	*6.27E+0 ± 0.85E+0*	255.00	**6.41E+0 ± 0.79E+0**	14.20
Cal Housing	5.09E+9 ± 0.42E+9	50.00	*4.69E+9 ± 0.26E+9*	810.00	**4.66E+9 ± 0.23E+9**	66.19
Elevators	25.00E-6 ± 2.00E-6	23.10	*18.00E-6 ± 1.00E-6*	850.70	**20.00E-6 ± 1.00E-6**	54.20
House 8L	1.24E+9 ± 0.16E+9	51.50	*1.23E+9 ± 0.14E+9*	964.29	1.26E+9 ± 0.14E+9	429.29
House 16H	1.68E+9 ± 0.22E+9	46.70	*1.60E+9 ± 0.17E+9*	927.09	**1.64E+9 ± 0.19E+9**	76.50
Mv Delve	3.66E+0 ± 0.49E+0	111.30	*2.31E+0 ± 0.47E+0*	505.89	**2.76E+0 ± 0.27E+0**	132.80
Pol	224.17E+0 ± 105.72E+0	27.40	*123.81E+0 ± 19.24E+0*	926.29	**140.08E+0 ± 15.20E+0**	38.09
Wind	45.44E+0 ± 1.75E+0	18.70	*44.51E+0 ± 1.53E+0*	104.59	45.02E+0 ± 1.44E+0	21.10
Wine Quality	0.59E+0 ± 0.07E+0	11.70	*0.57E+0 ± 0.06E+0*	147.60	0.59E+0 ± 0.06E+0	16.40

within an option tree seems to overfit the training data and as a result a higher variability in the predictions given different training datasets can be observed (higher variance). A logical explanation for this phenomenon is the fact that our estimates of the merit of each option (when constructing the best tree or choosing the best path to a leaf) are obtained using only training data which was observed at each corresponding region of the instance space. Thus, the risk of overfitting is higher. On the other hand, by averaging the predictions coming from multiple leaves, it is possible to reduce the variability due to the different training data used and obtain better predictive accuracy in general.

7.4.3 Analysis of Memory and Time Requirements

Due to the possibility for excessive growth, option trees can be expected to have higher memory requirements. We have already discussed two different methods for constraining the size of the tree, both of which stem from the hypothesis that options are most useful near the root of the tree. If this hypothesis is true, by determining the maximal level at which option nodes can be introduced, we can significantly reduce the size of the trees (and the memory allocation) without having to sacrifice the predictive accuracy. To evaluate this hypothesis, we will analyze the effect of the decaying option factor and place a restriction on the level of the tree where an option node can be introduced. For clarity of presentation we will use the acronym ORTO-BT^3 when we allow options only at the first 3 levels, and the acronym ORTO-BT^5 when we allow options only at the first 5 levels.

We have performed the same type of 10-fold cross-validation on the same 9 stationary UCI datasets. The results are given in Table 12. In comparison to the previous results for ORTO-BT in Table 11, we can see that while the size of the best tree is substantially reduced, the error has increased only slightly. However, the restriction to maximum 3 levels of options proves to be too severe. The option trees constrained to using options only in the top 3 levels are not able to produce more than 5 models on average for some of the datasets. Here by number of models we refer to the number of different trees compressed in the option tree. This restriction was shown to harm the predictive accuracy, thus we decided to relax it and use the limit of maximum 5 levels with options counting from the root node.

A peculiar case is the Wine Quality dataset on which we observed that the best predictive accuracy was obtained with ORTO-BT^3. This leads us to think that some of the options introduced at level 4 or 5 were somewhat incorrect or led to significantly worse models than before. Another explanation is that there is a larger fluctuation of the error, due to which we've observed a misleading result.

In any case, one needs to guard against the situation when bad options are chosen and estimated as best in the testing phase. The best way is to use an unbiased estimate on the merit of each option, which was not obtained only on training data. In a final conclusion, we can confirm that by decreasing the number of options with the depth of the tree and by placing an additional constraint on the maximum level at which options can be introduced, the memory allocation can be significantly reduced without inducing a drastic increase of the error.

In addition to this evaluation, we performed a bias-variance decomposition of the error for the constrained versions of option trees. The results are shown in Figure 18. We observe an increase in the bias component of the error for 4 of the datasets, in particular for the Elevators, House 16H, Mv Delve and the Pol datasets. The relative bias error still remains below 100%, that is lower than the bias component of the FIMT-DD trees. Another difference is that, for the constrained tree, the best tree (ORTO-BT^5) has a smaller bias component as compared to the option tree with averaging (ORTO-A^5). With respect to the variance component of the error, the constrained option tree with maximum 5 levels of options, as compared to the non-constrained tree, exhibits in general smaller variance, especially for some datasets like Elevators, House 8L, House 16H. The reduction is mainly

Table 12: Results from 10-fold cross-validation on 9 stationary real-world datasets for ORTO-BT^5, and ORTO-BT^3. The final MSE, number of models and size (number of leaves/rules) of models are given.

	ORTO-BT^5			ORTO-BT^3		
Data	Models	MSE	Size	Models	MSE	Size
Abalone	63.20	6.49E+0 ± 0.89E+0	12.50	20.88	6.58E+0 ± 0.97E+0	11.70
Cal Housing	8.10	4.88E+9 ± 0.34E+9	57.29	3.20	4.89E+9 ± 0.37E+9	66.19
Elevators	94.70	19.90E-6 ± 1.29E-6	35.90	94.70	19.90E-6 ± 1.29E-6	35.90
House 8L	26.50	1.27E+9 ± 0.13E+9	59.20	14.40	1.27E+9 ± 0.14E+9	54.60
House 16H	36.50	1.64E+9 ± 0.16E+9	52.79	12.70	1.66E+9 ± 0.19E+9	47.70
Mv Delve	2.90	3.55E+0 ± 0.39E+0	112.80	2.00	3.68E+0 ± 0.54E+0	111.00
Pol	8.90	148.99E+0 ± 18.72E+0	35.50	5.20	158.38E+0 ± 28.19E+0	33.00
Wind	1.60	45.19E+0 ± 1.57E+0	19.00	1.60	45.43E+0 ± 1.91E+0	19.30
Wine Quality	15.00	0.60E+0 ± 0.06E+0	16.79	8.30	0.58E+0 ± 0.06E+0	14.20

obtained for the ORTO-BT algorithm. At the same time, constraining the number of levels in which options are introduced is not very beneficial for the ORTO-A algorithm, for which the variance component actually increased in some cases.

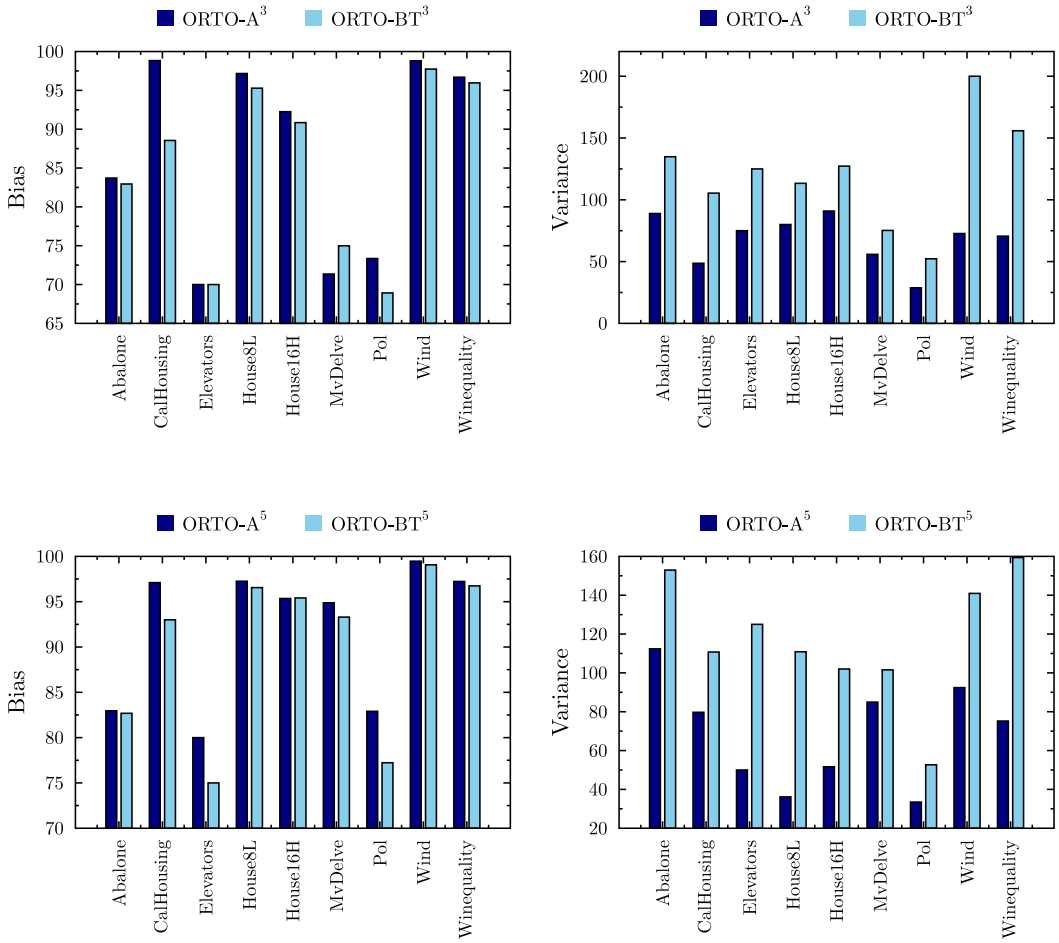


Figure 18: Bias-variance decomposition of the mean squared error of ORTO-BT^5, and ORTO-A^5 trees relative to their corresponding counterparts computed for the FIMT-DD trees.

Now, let us look at the gains in memory which were obtained by using a decaying option factor and a constraint on the number of levels containing options. In Table 15, we present the memory requirements (in MB) for all of the algorithms. The required memory is determined with the number of leaves, as the leaves store the sufficient statistics which need to be maintained in order to evaluate the candidate splits and enable growing. We can see that the savings in memory are significant when there is a constraint on the maximal level that can contain options. For most of the cases, the memory requirements are reduced by 50% and for some datasets even by an order of magnitude. The stricter limit to maximum 3 levels with options does not reduce the memory requirements significantly, but harms the predictive accuracy. For these reasons, we would not recommend a restriction that goes beyond 5 levels.

Due to the need to enable the traversal of each example through all of the branches of every option node and update all of the leaves in which an example may end up, the ORTO algorithm is naturally expected to have an increased processing time per example. In our implementation, the possible splits of an option node are examined in a sequential manner. Therefore, the relative increase in the processing time depends on the number of options nodes and their distribution in the tree structure. On average, the processing time per example of ORTO is 10 times higher than that of FIRT-DD. The results are given in

Table 13: Memory requirements (in MB): Comparison of FIMT-DD, ORTO-BT (basic version), ORTO-BT with a decaying factor and restricted to maximum 3 levels (ORTO^3), and 5 levels (ORTO^5) of options.

Data	FIMT-DD	ORTO-BT	ORTO-BT^3	ORTO-BT^5
Abalone	4.66	49.08	20.81	29.72
Cal Housing	25.92	296.64	45.02	54.77
Elevators	8.70	369.31	138.62	138.62
House 8L	25.74	451.27	197.17	249.65
House 16H	55.49	684.90	287.86	477.14
Mv Delve	59.77	219.84	99.93	106.71
Pol	7.52	375.36	44.76	50.39
Wind	13.88	60.59	22.31	22.62
Wine Quality	3.75	38.43	15.38	20.53

Table 14: Comparison of FIMT-DD (F), ORTO (O): Processing time in seconds (F:s, O:s) and in seconds per example (F:s/ex, O:s/ex). The ratio in the last column represents the relative speed of ORTO with respect to FIMT-DD in seconds per example. In addition, the number of leaves, option nods and trees for the ORTO algorithm is given.

Dataset	F:s	O:s	Leaves	Options	Trees	F:s/ex	O:s/ex	Ratio
Abalone	0.04	0.29	115	15	57	1.37E-5	9.92E-5	7.25
Cal Housing	0.65	1.82	146	13	26	4.49E-5	12.59E-5	2.80
Elevators	0.55	33.07	873	11	29	4.73E-5	2.85E-3	60.13
Pol	0.98	1.78	58	4	6	9.33E-5	16.95E-5	1.82
House 8L	0.77	4.90	267	21	38	4.83E-5	30.73E-5	6.36
House 16H	2.21	6.39	173	17	28	1.38E-4	4.00E-4	2.89
Wind	0.29	0.27	12	0	1	6.31E-5	5.87E-5	0.93
Wine Quality	0.10	0.19	35	6	11	2.92E-5	5.54E-5	1.90

Table 14. We have also noted that, as long as the number of trees is kept below or in the range [10, 20] the increase in processing time is not significant. Obviously, one simple way to reduce the processing time per example is to maintain smaller trees with options only at the higher levels of the tree.

The results presented in this section were obtained without using linear models in the leaves. We placed the focus on regression trees instead of model trees in order to obtain a clear evaluation of the advantages and disadvantages of introducing options, and without the effect of the incremental computing of linear models in the leaves of the tree. As the criterion for introducing options does not depend in any way on the presence of linear models, we performed the same evaluation procedures described above with linear models included. The results are given in Table 15.

The corresponding conclusions are very similar to the ones presented thus far, and do not provide interesting input for additional evaluation or discussion. Model trees have been shown to improve the accuracy of regression trees for most of the cases, and in general can be characterized with slightly smaller values for the variance component of the error.

Table 15: Averaged mean squared error obtained (MSE) with a 10-fold cross-validation on 9 stationary real-world datasets for the FIMT-DD, OMTO-A and OMTO-BT algorithms.

Data	FIMT-DD	OMTO-A	OMTO-BT
Abalone	13.31E+0	13.32E+0	13.47E+0
Cal Housing	20.04E+9	19.83E+9	20.12E+9
Elevators	0.00E+0	0.00E+0	0.00E+0
House 8L	0.00E+0	0.00E+0	0.00E+0
House 16H	3.77E+9	3.61E+9	3.82E+9
Mv Delve	189.18E+0	188.51E+0	189.90E+0
Pol	3.19E+3	3.22E+3	3.27E+3
Wind	75.43E+0	75.30E+0	75.69E+0
Wine quality	1.00E+0	0.98E+0	1.02E+0

7.5 Summary

The ORTO algorithm was developed to address several design issues in the learning process of Hoeffding-based regression trees. It tries to improve the myopia of split selection, the resolution of tie situations, and the possibility for changes (concept drift) in the functional dependency modeled with the tree. With the modified splitting criterion used in ORTO, the tree is allowed to grow faster, and is able to improve its accuracy using fewer training examples. The OMTO algorithm which represents an extended version of ORTO for modeling smooth surfaces was shown to deliver an improvement of the accuracy by reducing the variance component of the error.

The experimental evaluation gives evidence that, by using option nodes in the tree induction process it is possible to obtain models whose bias is lower as compared to the bias of the FIMT-DD algorithm. In addition, introducing options results in a more efficient resolution of the tie situations. From the two aggregation strategies that we considered in our study, best results were achieved when using the simple aggregation prediction rule. The second strategy, based on an estimate of the best single tree within an option tree, was observed to overfit the training data and produced a tree characterized with higher variance. We believe that this is due to the fact that the estimates of the merit of each option are obtained using only training data (observed at the corresponding region of the instance space).

Finally, option trees represent an affordable method (in terms of processing time and memory) for improving the generalization power and the robustness of tree learning algorithms. At the same time, they represent a valuable tool for studying ambiguities, which can improve our understanding of the data in hand. To the best of our knowledge, online option trees for regression have not been studied previously.

An interesting direction for further work is extending the existing change detection mechanisms for ORTO and OMTO when the data distribution is not stationary. Since the option tree extends a regression or a model tree by introducing option nodes in addition to the standard types of nodes, running change detection tests in every node is straightforward. Monitoring the signal of the averaged prediction errors can enable detecting a region affected by a concept drift where all of the sub-models fail to approximate the true regression surface. Further, the option nodes enable to monitor the performance of each sub-tree below an option node. Therefore, one can use a change detection test to monitor the ratio between the accuracy of one sub-tree over another. This gives the possibility to detect the moment when one of the trees degraded significantly as compared to the other. However, if the number of possible options is large, such an approach might suffer from an increased time complexity.

8 Ensembles of Regression Trees for Any-Time Prediction

Everything we care about lies somewhere in the middle, where pattern and randomness interlace.

James Gleick, "The Information: A History, a Theory, a Flood"

Ensemble learning methods are considered as one of the most accurate and robust machine learning approaches for solving classification and regression tasks. They have been successfully used in many real-world problems, including the well known Netflix Prize[1] competition where it was shown that best results can be achieved by combining multiple hypotheses, each of them specialized in addressing a different aspect of the data. Common knowledge on ensemble methods suggests that the hypotheses should be as accurate as possible and should have as diverse as possible distributions of errors. Although this seems as a simple requirement, it is not easily achieved. In the vast literature on learning ensemble methods, a variety of diversity measures has been proposed, but the relationship of diversity with ensemble accuracy is still ambiguous (Kuncheva and Whitaker, 2003).

In this chapter we first present a survey of the existing ensemble learning methods for online classification. The first section presents methods for online sampling from the training data stream. These methods represent the basis for implementing the online versions of bagging and boosting for learning various types of ensembles. A separate section discusses stacked generalization with restricted Hoeffding trees. We observe that most of the ideas can be easily transferred to the online regression setup. Based on these approaches we have implemented two novel ensemble learning algorithms for online regression presented in the succeeding section. We also perform an experimental comparison and evaluation of these two algorithms with single trees and option trees for regression. In the last section, we finally discuss the impact of the diversity of base models on the overall predictive accuracy of the ensemble.

8.1 Methods for Online Sampling

As discussed in Chapter 4, the simplest method to encourage diversity among the base models that constitute an ensemble is to generate different samples from the training dataset. This method modifies the training dataset each time a base hypothesis is learned. Online versions of sampling-based methods for learning ensembles have been explored for the first time in the work of Oza and Russell (2001). In the following subsections, we will discuss *online bagging* and *online boosting* as main representatives of this category.

[1] http://www.netflixprize.com/

8.1.1 Online Bagging

The batch *bagging* method generates a number of bootstrapped training sets from the original dataset. A bootstrap sample can be obtained by random sampling with replacement according to a Binomial probability distribution. In the next step, a separate model is induced on each training dataset. In the testing phase, the predictions from the base models are aggregated using majority voting. The main idea behind the online version of bagging is to transform the batch sampling procedure into an online one.

Let us first discuss the details of bootstrap sampling. The key effect of the bootstrap sampling procedure is to simulate repeated observations from an unknown population using the available sample as a basis. A training sample generated with batch bootstrap sampling contains K copies of each of the original training examples, where K is a random variable distributed according to the Binomial distribution. In order to transform the batch procedure into an online one, the number of occurrences of each training example for each training dataset has to be estimated at its first occurrence, without the possibility to examine the rest of the training dataset.

Oza and Russell (2001) have observed that when the size of the training dataset tends to infinity; that is, when $N \rightarrow \infty$, the Binomial distribution tends to be more and more similar to the Pois(1) distribution which is given with:

$$Pois(\lambda) \sim \frac{e^{-1}}{k!} \text{ for } \lambda = 1,$$

where k is the number of occurrences of the discrete stochastic variable K. The formula gives us the means to compute the value of K for a given randomly chosen probability for each training example. At the same time, it applies perfectly to the online learning setup in which the size of the training set is unbounded and tends to infinity. Oza and Russell (2001) have proved that, if the batch and the online bagging algorithms use the same training set that grows to infinity, the distributions over the bootstrapped training sets will converge to the same distribution.

The generalized online bagging procedure used in our experiments is shown in Algorithm 5. The algorithm receives as input a set of initialized base models $H_M = \{h_1, ..., h_M\}$, i.e., one-leaf decision trees. For every training example e received from the data stream and for each base model h_i, the number of occurrences K of the example e in the online training set used to update h_i is set to $k = Poisson(1)$. In the next step, the example is given K times to the procedure which refines the corresponding hypothesis h_i. This procedure is repeated as long as there are training examples available to update the set of models.

Before every training phase, the base models are used to produce a prediction assuming that the value of e is unknown. The set of predictions are further aggregated by using a non-weighted voting rule in the case of online classification. However, for online regression the best method is to use the average $\overline{p}$. This method for estimating the accuracy of the base models, as well as of the ensemble, is known as the *test-then-train* or the *predictive sequential* methodology.

8.1.1.1 Online Bagging for Concept Drift Management

The *online bagging* meta-algorithm has been used by several authors and mainly for the task of online classification. Bifet et al. (2009b) have proposed two interesting algorithms for learning ensembles of Hoeffding trees for online classification: ADWIN Bagging and Adaptive-Size Hoeffding Tree (ASHT) Bagging. Both ADWIN Bagging and Adaptive-Size Hoeffding Tree (ASHT) Bagging have been developed to address the problem of learning under non-stationary distributions and perform some form of blind concept drift management.

Algorithm 5 *Online Bagging: An online bootstrap sampling procedure.* Pseudocode.

Input: λ - Poisson parameter ($\lambda = 1$), $H_M = \{h_1, ..., h_M\}$ - set of M base models.
Output: p - prediction from the current set of base models H_M.

```
for ∞ do
    e ← ReadNext()                          ▷ Reads the next example from the data stream.
    for i = 1 → M do
        p_i ← Predict(h_i, e)               ▷ Applies the hypothesis h_i on the instance e.
        k ← Pois(λ)
        for j = 1 → k do
            Update(h_i, e)                  ▷ Updates each hypothesis h_i with the training example e.
        end for
    end for
    p ← Aggregate(p_1, ..., p_M)            ▷ Computes an aggregate of M predictions.
end for
return p
```

ADWIN Bagging employs the *online bagging* procedure in order to learn an ensemble of incrementally induced Hoeffding trees. The ensemble is equipped with an explicit drift management mechanism that detects the worst performing classifier in the collection. A member whose performance was observed to drop significantly is immediately replaced with a new classifier that would be incrementally built using the new training instances. The name of the algorithm comes from the change detection method used to monitor the evolution of the average loss of each base classifier: This is the ADWIN algorithm, originally proposed by Bifet and Gavaldà (2007).

ASHT Bagging, on the other hand, builds a set of Hoeffding trees constrained in size by using the same *online bagging* procedure. Each tree is periodically reset (or regrown) using the most recent sample of training examples. The reset rate depends on the size of the tree. Trees limited to size s will be reset twice as often as the trees with a size limited to $2s$. The authors have studied different types of reset operations, varying between replacing oversized trees with new ones or pruning from the root.

The idea behind maintaining a set of Hoeffding trees with different height is based on the reasoning that smaller trees will be able to adapt more easily to changes which occur with a higher frequency, while larger trees will do better during periods with little or no concept drift. With this strategy, it is expected that the ability of the ensemble to respond to different types of concept drift will be improved. In addition, the approach is expected to produce a more diverse set of classifiers, which is an important factor for the overall performance. The prediction rule is based on a weighted voting scheme. The prediction of each classifier is weighted with the inverse of the squared error of that classifier.

The empirical study conducted by Bifet et al. (2009b) has included several variants of bagging, using different change detection algorithms (the DDM algorithm by Gama et al. (2004a) and the EDDM algorithm by Baena-García et al. (2006)), as well as several variants of *online boosting* and an algorithm for learning *Hoeffding Option Trees* by Pfahringer et al. (2007). The results suggested that best accuracy is obtained by online bagging of Hoeffding trees at the cost of an increased runtime and memory allocation.

8.1.1.2 Online Bagging for RandomForest

With the availability of an online bagging method and various online decision tree learning algorithms, a host of methods for learning online random forests have emerged (Abdulsalam et al., 2007, 2008; Bifet et al., 2010; Li et al., 2010). The online random forest method of Bifet et al. (2010) uses online bagging to create an ensemble of *Hoeffding Naive Bayes* trees.

A *Hoeffding Naive Bayes* tree is a leveraged Hoeffding tree that has Naive Bayes classifiers in the leaf nodes (Gama et al., 2003). When performing a prediction, the leaf will return the Naive Bayes prediction if it has been estimated as more accurate than the majority class. Otherwise, the leaf resorts to a majority class voting rule. In the online random forest method, the trees are grown using a modified version of the algorithm in which splits are allowed only on $\sqrt{m}$ attributes, randomly chosen at each node (here m is the number of attributes). Their experimental findings suggested that the online random forest performed faster, but was evaluated to have 5% lower accuracy than the bagged Hoeffding Naive Bayes trees.

Bifet et al. (2010) have further shown that the accuracy of the online random forest can be increased by setting the λ parameter to a value greater than 1. The effect of this adjustment is that the percentage of examples not included in the resampled dataset will be decreased. This, on the other hand, is expected to increase the diversity of the training datasets. The main result of this study was that sampling with $\lambda > 1$ can improve the accuracy of the proposed random forest method without adding additional members to the ensemble.

Another interesting idea in the work of Bifet et al. (2010) is to use random error-correcting output codes. The way that error-correcting output codes work is as follows. Every class is assigned a random binary string of length n, where each position in the string is learned by a single binary classifier. At prediction time, the testing instance is assigned the class whose code is closest to the combined prediction of the n classifiers. Since every classifier learns and predicts a different function the diversity of the ensemble is expected to increase.

8.1.2 Online Boosting

Like online bagging, the *online boosting* algorithm of Oza and Russell (2001) aims to transform the batch boosting technique into an online one. It has been designed to approximate the batch learning algorithm AdaBoost by Freund and Schapire (1997). The *online boosting* procedure is essentially the same as *online bagging*. The main difference is in the dynamic computation of the Poisson parameter λ, whose value in the case of online bagging is kept constant ($\lambda = 1$). In particular, the authors propose to adjust the value of λ according to the performance of the classifier that was updated last. When a base model incorrectly classifies a training example the value of the Poisson parameter λ associated with that example is increased for the next base model. In the case of a correct classification the value of λ will be decreased. Misclassified examples are given half the total weight in the next stage; the correctly classified examples are given the remaining half of the weight.

However, batch boosting is a sequential process that runs multiple times over the same training set, boosting the accuracy of the classifiers in each iteration based on the estimates from the previous run. The problem with online boosting is that the estimate used to determine the probability of presenting an example to the next model is based on the performance given the examples seen so far. This is completely different compared to the batch version, where the estimate is obtained on a fully trained base model. For that reason, the online boosting algorithm significantly underperformed as compared to batch boosting for small training sets. This was especially evident for the initial training sequence, when the estimates were fairly biased. However, priming online boosting with an initial batch training was shown to perform comparably to batch boosting over the whole course of learning. In the study of Bifet et al. (2009b), online boosting was also found to yield inferior accuracy compared to online bagging.

A slightly improved version of online boosting, called Online Coordinate Boosting (Pelossof et al., 2009), stems from an approximation to the AdaBoost's loss minimization, given a training set that grows one example at a time. While in Oza and Russell's online boosting

algorithm the update rule is broken down to two cases, one for the correctly and one for the incorrectly classified examples, in Online Coordinate Boosting a single update rule is used: This rule takes the average of the old weight and the newly updated weight that one would get by using AdaBoost's exponential re-weighting. In an experimental evaluation, it was show that by greedily minimizing the approximation error of each base classifier, Online Coordinate Boosting is able to approximate the batch AdaBoost performance better than Oza and Russell's online boosting algorithm.

Online boosting has also been used in the context of learning under concept drift. The *Fast and Light Boosting* method proposed by Chu and Zaniolo (2004) is based on a dynamic weight assignment scheme. The boosting process is performed over chunks of data with equal size. Each block is scanned twice: In the first pass each example is assigned a weight in the same way as in the *AdaBoost.M1* algorithm, while in the second pass, the classifier is retrained by using the weighted samples. The system keeps only the M most recent classifiers. As base models, standard batch C4.5 decision trees are used, whose number of terminal nodes is restricted to 32 (ranging from 2 to 32). The authors have shown that learning small trees over small sized data blocks enables a fast and light learning procedure, while a change detection algorithm triggers a complete rebuild of the ensemble from scratch.

8.2 Stacked Generalization with Restricted Hoeffding Trees

Stacked generalization or stacking (Wolpert, 1992) learns to combine multiple models by applying a higher level meta-learner in a space whose inputs are the predictions, confidences or distributions of the base models. There are many different ways to implement stacked generalization. The most common implementation combines the predictions of multiple models learned by different methods.

Stacking can also be used to improve the accuracy of a single learning algorithm. Breiman (1996b) demonstrated the success of stacked generalization in the setting of ordinary regression. In his study, he used as base models regression trees of different sizes or linear regression models on different numbers of variables. The predictions from the different regressors for each training instance along with the true response values comprise the training set for the meta-learning algorithm. Breiman (1996b) used least-squares linear regression under the constraint of non-negative regression coefficients. This constraint is necessary in order to guarantee that stacked regression improves predictive accuracy.

Like bagging, stacking is ideal for parallel computation. The construction of the base models can proceed independently, requiring no communication among the learning processes. While bagging usually requires a considerable number of base models, because it relies on the diversity of the training data used to induce the base models, stacking can work with only two or three base models (Ting and Witten, 1999). Therefore, it seems natural to think of stacking in the context of online learning, if one can determine a suitable online meta-learning algorithm. In this context, Bifet et al. (2012) proposed a method for learning an ensemble of restricted Hoeffding trees based on stacking. Each decision tree is learned in parallel from a distinct subset of attributes. Their predictions are further combined using perceptron classifiers, whose weights can be updated using an online update rule. The authors propose to enumerate all attribute-subsets of a given user-specified size k, and learn a Hoeffding tree from each subset. Then, by using the interleaved test-then-train procedure, the trees' predictions are gathered and used to train a perceptron classifier per class.

The computational complexity of this method obviously depends on the value of k, which defines the size of the ensemble. If there are m attributes in total, there are $\binom{m}{k}$ possible subsets of attributes. The authors propose $k = 2$ for datasets with a relatively large number of attributes. Acceptable runtimes can be also obtained with $k = 4$ for $m = 10$. Compared to online bagging of Hoeffding trees, this method was shown to have excellent classification

accuracy on most of the datasets used in the experimental evaluation. It was also shown that stacking can improve the accuracy of online bagging without increasing the number of trees in the ensemble. These results should be considered very carefully, having in mind that a change detection method is used for resetting the learning rates of the meta-classifiers, as well as for resetting ensemble members when their accuracy degrades significantly.

8.3 Online RandomForest for Any-time Regression

Randomization has been used extensively in the machine learning community as a method to generate more accurate ensembles of decision trees or classifiers. In particular, the RandomForest method of Breiman (2001) is widely considered as one of the most accurate ensemble learning algorithms. Due to the randomization introduced both in the selection of the training data and in the tree building process, ensembles of randomized trees are very robust to noise. They are particularly useful when the dataset is described by many input variables, each containing only a small amount of information. An important feature of randomized ensembles is their ability to reduce the computational complexity without sacrificing the predictive accuracy. The RandomForest algorithm, for example, is faster than Bagging and Boosting, its accuracy is as good as that of AdaBoost, it is simple to implement and can be easily parallelized. Hence, randomization is very suitable for learning ensembles from fast data feeds.

RandomForest improve the bias component of the error: One of the reasons for this is that the randomization helps in achieving a low correlation among the strong models in the ensemble. There are many ways to randomize the learning process. As Breiman (2001) noted, except bagging and random features, one can use random Boolean combinations of features, or similarly to the work of Amit and Geman (1997), one may grow shallow trees using random selection from a large number of geometrically defined features. We have considered the full spectrum of possibilities: online bagging, randomized trees, and a random selection of features per growing node. In this chapter, we extend the ideas of Breiman (2001) and introduce a random component in our online model tree learning algorithm FIMT-DD.

Learning an online random forest for regression requires the definition of two main components: a sampling strategy and an online algorithm for learning regression and model trees. In the previous section, we have described the sampling algorithm of Oza and Russell (2001), that draws samples from the distribution according to an approximated Poisson distribution with $\lambda = 1$. With this sampling strategy, we can achieve a diversification of the input space, while the diversification of the learning process can be achieved through a random selection of attributes.

We present the ORF (Online Random Forest) algorithm, which is an online representative of the Random Forest (Breiman, 2001) algorithm for regression. As our base learner, we will use a randomized version of the FIMT-DD algorithm described in Chapter 6, called R-FIMT-DD. The pseudocode of R-FIMT-DD is given in Algorithm 6. R-FIMT-DD proceeds as follows: every example e from the data stream is traversed to a leaf node using the $\mathit{Traverse}(\mathit{RootNode}, e)$ procedure. This is a standard procedure for traversing a tree, that starts with the corresponding $\mathit{RootNode}$. Whenever a new leaf is created, q attributes are chosen at random from the set of d possible attributes. For each of the chosen attributes, the required statistics $\{N, \sum y, \sum y^2\}$ per attribute value will be initialized to zero, and incrementally maintained until the moment of split construction. The leaf will contain an un-thresholded perceptron with a number of inputs equal to the number of numerical attributes plus the categorical ones which are previously transformed into binary.

The next best split will be evaluated after a minimum of n_{min} examples are observed in a leaf node. We will use the same probabilistic estimates, with a user-defined confidence level δ, as described in Chapter 6. The main difference is in the fact that only the previously

Algorithm 6 *R-FIMT-DD: Randomized FIMT-DD.* Pseudocode.

Input: *RootNode*, e - training instance, δ - confidence parameter, n_{min} - chunk size.
Output: p - prediction from the current hypothesis.

```
Leaf ← Traverse(RootNode, e)
                          ▷ Traverses the example e to the corresponding Leaf node.
Counter ← SeenAt(Leaf)
                          ▷ Computes the number of examples seen at the Leaf node.
if Counter = 0 then
    q ← SelectAttributesAtRandom(Leaf)
                                        ▷ Selects √d attributes at random.
    InitializePerceptron(Leaf)
            ▷ Initializes the perceptron weights to random values in the range [0, 1].
end if
Counter ← Counter + 1
p ← GetPrediction(Leaf, e)
UpdateStatistics(Leaf)
                          ▷ Updates the necessary statistics in the Leaf node.
UpdatePerceptron(Leaf)                  ▷ Updates the perceptron's weights.
if Counter mod n_min = 0 then
    for i = 1 → q do
        S_i = FindBestSplitPerAttribute(i)
                                        ▷ Computes the best split per attribute.
        S_a ← Best(S_1, ..., S_q)
        S_b ← SecondBest(S_1, ..., S_q)
        if S_b/S_a < 1 − √(ln(1/δ)/(2×Counter)) then
            MakeSplit(Leaf, S_a)
                                ▷ Creates a binary split by applying the split S_a.
        end if
    end for
end if
t ← GetTargetValue(e)
Δ ← ABS(t − p)                  ▷ Computes the absolute prediction error Δ.
BackPropagate(Δ)
                          ▷ Back-propagates the absolute error towards the root node.
return p
```

chosen q attributes will be considered as possible candidates in the split evaluation performed with the *FindBestSplit*(i) procedure, for $i = 1, ..., q$. In sum, each terminal node will have an independently and randomly chosen set of attributes which define the set of possible directions to search the hypothesis space. Since statistics will be maintained only for these q attributes, where $q = \sqrt{d}$, the memory allocation is significantly reduced.

Each node of the randomized model tree has an online change detection mechanism based on the Page-Hinkley test. The change detection test continuously monitors the accumulated absolute error given every example that goes through the node. This is performed through the *BackPropagate* procedure that takes as input the absolute error $\Delta = ABS(t - p)$ for a given example e, whose true target value is t, and the predicted value p is obtained with the *GetPrediction*$(Leaf)$ procedure. *GetPrediction*$(Leaf)$ returns either the estimated mean or the perceptron's output in the corresponding leaf node. The choice is made given the accuracy of the perceptron's predictions estimated with a weighted prequential squared error. As each tree is equipped with change detection units, the base models can be kept independently and automatically up-to-date with possible changes in the target function.

The pseudocode of the final algorithm ORF is given in Algorithm 7. The algorithm starts with an initialization of the set of base models $\{B_1, B_2, ..., B_{T_{max}}\}$. Given a training example e, ORF revisits each of its base models and updates them with $k \sim Poisson(1)$ copies of e using the R-FIMT-DD algorithm. The value of k is recomputed T_{max} times for each of the base models separately, which ensures the diversity of the training sets.

Each call of R-FIMT-DD returns a prediction for e. However, only the last one is considered in the aggregation procedure. The final prediction is computed by averaging the individual predictions $\{p_1, p_2, ..., p_{T_{max}}\}$ obtained from the constituent model trees. Naturally, each of these processes can be run simultaneously in parallel. Thus, the whole ensemble would represent an autonomous self-sustaining dynamic learning unit with a decentralized concept drift management.

Algorithm 7 *Algorithm ORF.* Pseudocode.

Input: T_{max} - number of base models, δ - confidence parameter, n_{min} - chunk size.
Output: p - prediction from the current Online Random Forest ensemble.

```
for i = 1 → T_max do
    B_i = InitializeTree()
                          ▷ Initializes T_max trees with the necessary statistics.
end for
for ∞ do
    e ← ReadNext()
    for i = 1 → T_max do
        k ← Pois(1)
        for j = 1 → k do
            p_i ← R-FIMT-DD(B_i, e, δ, n_min)
                  ▷ Computes the prediction from every tree B_i for the example e.
        end for
    end for
    p ← AVG(p_1, p_2, ..., p_{T_max})   ▷ Computes the average of T_max predictions.
return p
end for
```

Despite the fact than the ensemble has lower interpretability as compared to a single model, the adaptation to concept drift or changes in the target function is still performed in an informed way. The detected changes in the tree structure can be represented with a set of rules which can be matched for similarities or dissimilarities. We believe that, given the fact that each model can address a different aspect of the data, an ensemble of models has the ability to bring more value and return more information on the nature of changes than a single model tree.

8.4 Experimental Evaluation of Ensembles of Regression Trees for Any-Time Prediction

In this section, we will discuss the results of the empirical comparison performed to assess the relative performance of an ensemble of model trees with respect to an option tree for regression or a single model tree. We compare the FIMT-DD algorithm, the ORTO-A and ORTO-BT algorithms, online bagging of FIMT-DD model trees denoted as OBag, and the ORF algorithm. A single model tree has a different inductive bias from an option tree, which has a different inductive bias from an ensemble of model trees. Consequently, for each learning algorithm, the inductive inference process will be characterized with a different search trajectory in the space of hypotheses reaching different end points.

Table 16: Predictive performance of the FIMT-DD algorithm and the ensemble methods for online regression based on it. The averaged mean squared error from 10 runs of sampling for the last holdout evaluation on the UCI and Infobiotics datasets is given. The maximum tree depth is 5, the maximum number of trees is 10. Best results are shown in boldface.

Dataset	FIMT-DD	ORTO-BT	ORTO-A	OBag	ORF
Abalone	6.69E+0	6.25E+0	5.91E+0	6.84E+0	**5.68E+0**
Cal Housing	8.70E+9	5.24E+9	**5.14E+9**	7.82E+9	6.51E+9
Elevators	2.50E-5	2.40E-5	2.50E-5	2.30E-5	**2.20E-5**
House 8L	1.32E+9	1.18E+9	**1.11E+9**	1.30E+9	1.53E+9
House 16H	**1.56E+9**	1.58E+9	**1.56E+9**	1.57E+9	1.72E+9
Mv Delve	18.04E+0	18.33E+0	**17.12E+0**	17.20E+0	54.38E+0
Pol	273.63E+0	292.57E+0	**231.57E+0**	265.99E+0	1223.61E+0
Wind	48.89E+0	48.89E+0	48.89E+0	48.65E+0	**22.04E+0**
Wine Quality	**0.52E+0**	0.53E+0	**0.52E+0**	**0.52E+0**	0.58E+0
Infobiotics	29.76E+0	29.47E+0	**28.37E+0**	29.69E+0	31.56E+0

Table 17: The size of the tree models produced by the FIMT-DD algorithm and the ensemble methods for online regression based on it. The averaged number of tree leaves from 10 runs of sampling for the last holdout evaluation on the UCI and Infobiotics datasets is given. The maximum tree depth is 5, and the maximum number of trees is 10. Best results (smallest number of leaves) excluding FIMT-DD and ORTO-BT are shown in boldface.

Dataset	FIMT-DD	ORTO-BT	ORTO-A	OBag	ORF
Abalone	6.6	10.1	**42.5**	66.9	55.6
Cal housing	23.5	25.9	**75.3**	231.9	212.5
Elevators	16.9	16.7	**93.2**	162.1	185.6
House 8L	17.6	24.9	**79.9**	145.5	236.9
House 16H	19.6	22.0	**52.4**	192.7	209.7
Mv delve	30.9	30.9	**70.7**	313.7	287.7
Pol	19.8	20.3	**51.3**	199.8	148.3
Wind	7.6	7.6	**7.6**	81.6	126.5
Wine Quality	9.2	12.9	**48.8**	86.8	118.1
Infobiotics	31.3	31.5	**140.5**	310.3	302.7

Due to the fact that we are solving prediction tasks, the main quality measure is naturally the generalization power of the induced hypothesis, although the complexity of the models is also an important factor. Differently to the batch learning scenario, we are not only interested in the accuracy of the final hypothesis, but also in the evolution of the accuracy and the speed of convergence to the best possible model over the course of learning. The starting hypothesis is that an ensemble of model trees will have the highest accuracy among all of the algorithms. If that is true, we would like to know what is the price for the increased accuracy and what is its relative speed of convergence as compared to an option tree, which was shown to improve the any-time accuracy of online model trees.

8.4.1 Predictive Accuracy and Quality of Models

To assess the generalization power of each learning algorithm, we have performed an online holdout evaluation, which consists of a periodical evaluation of the induced model at some pre-defined time intervals, using a separate holdout set of testing examples. In Table 16, we give the measured mean squared error in the last hold-out evaluation, averaged over 10 rounds of training and testing. The size of the corresponding models in terms of the total number of leaves is shown in Table 17.

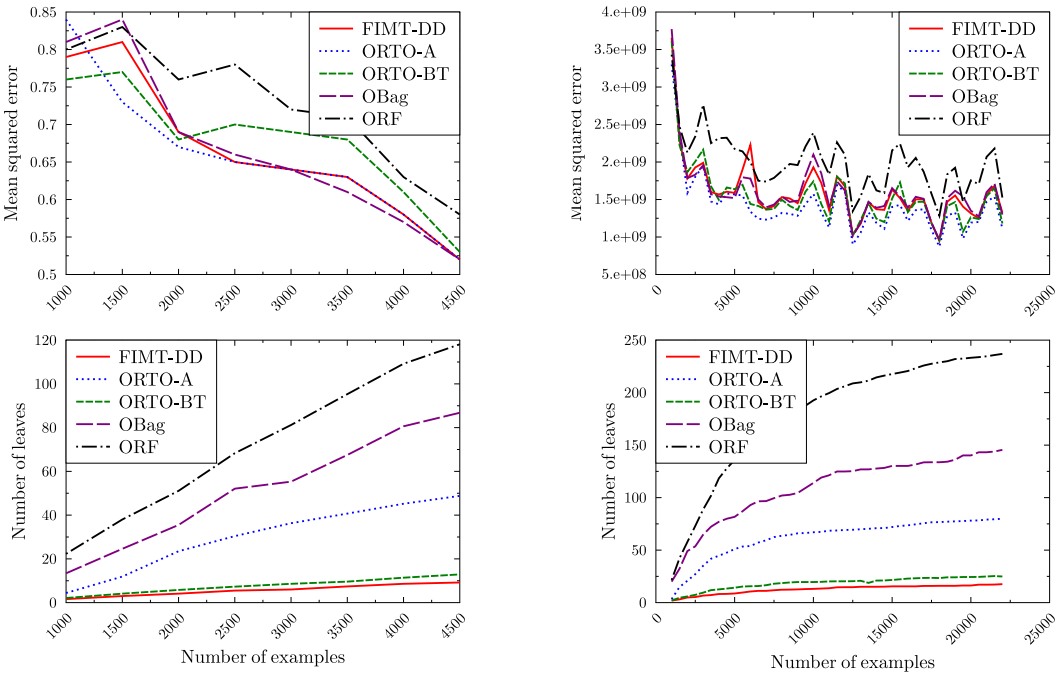


Figure 19: Mean squared error (upper) and number of leaves (lower) measured periodically using the holdout evaluation method for the Wine Quality (left) and House 8L (right) datasets.

For most of the problems, the best accuracy over the last holdout test set is achieved by the ORTO-A algorithm (option trees with averaging) and the ORF (online random forests) algorithm. The option nodes improve the accuracy of an online model tree for all of the datasets, except for the Wind dataset. At the same time, the option tree had the smallest number of leaves (predictive rules) among the ensemble methods. The online random forest method was able to improve the accuracy of a single tree only for 3 datasets. The randomization seems to harm the accuracy for the remaining datasets. From the results, we can see that online bagging performed more or less comparably to ORTO-A.

Due to the temporal non-homogeneity of the training data, the predictive performance can vary over time. We would expect that the option trees will perform best at the beginning of the induction process. Figure 19 depicts the mean squared error computed using the sliding window holdout evaluation method periodically performed every 500 examples on a test set of size 1000 examples, as well as the models' size for the Wine Quality dataset (left) and the House 8L dataset (right). We can see that the accuracy improves with the growth of the induced models. For these two particular datasets, the option tree seems to achieve a better accuracy at any-time with a substantially smaller number of rules as compared to the ensembles. Online bagging of FIMT-DD (OBag) trees also performs comparably, while the online random forest (ORF) performs worst. Since this is not always the case (we can see in Figure 20 left panel) that OBag has from time to time higher accuracy than ORF) we cannot draw strong conclusions on the advantage of one particular learning algorithm. It can be noted, however, that despite the expectations that ensembles of model trees would offer the best performance, this seems not to be the case for the chosen datasets and algorithms.

We have further performed an analysis of the distribution of errors over the range of the target attribute and over the range of the observed error values as well. We use error plots to visualize the results. The error plots are generated at each point of the sliding window holdout evaluation, each over the range of the target attribute. We give only the

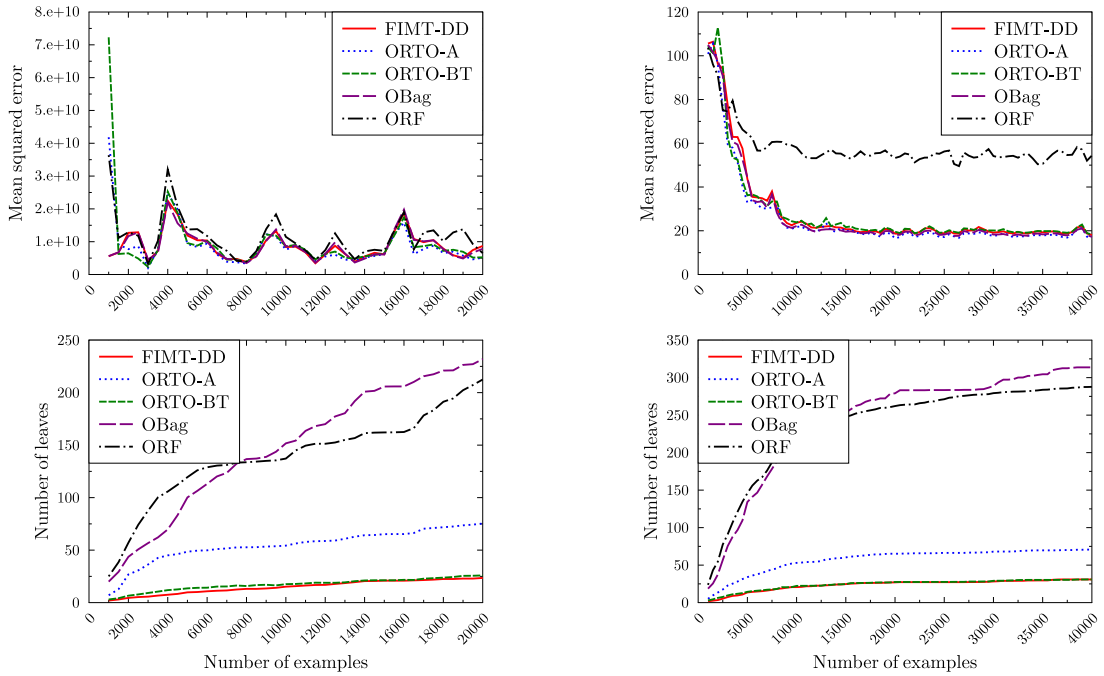


Figure 20: Mean squared error (upper) and number of leaves (lower) measured periodically using the holdout evaluation method for the Cal Housing (left) and Mv Delve (right) datasets.

plots from the last holdout evaluation. For each dataset, we first give graphs showing the distribution of the target values (counts per interval) over 10 intervals from the range of the target attributes. This graph gives information on the density of the output space. Ideally, we would like our algorithm to make the smallest errors for the most populated intervals, while larger errors would be allowed for the intervals with few instances.

Next, we plot the distribution of the averaged values of the errors (the difference between the true value and the predicted one) per interval over the range of the target attribute. An error is assigned to an interval if the true value of the target belongs to that interval. Finally, we plot the distribution of the standard deviation values of the errors over the same intervals. In this way, we can make a fair comparison and observe the magnitude of errors, and the average and standard deviation of these magnitudes per method. Figure **??** depicts the distribution of errors over the range of the target attribute obtained for the last holdout evaluation on the House 8L dataset. The corresponding figures for the remaining datasets are given in Appendix A subsection A.3 (A.34(a), A.34(b), A.36(a), A.36(b), A.35(a), A.35(b), A.35(c), and A.35(d)).

What we have observed is that, in general, the distribution of the average of errors inversely mirrors the distribution of target attribute values. This means that the loss is greater for those intervals of the range which are less populated. The standard deviation of the loss behaves similarly. ORF has, for most of the datasets, the highest standard deviation. On most of the datasets we also observed that, for the less populated intervals of the domain of the target attribute, different methods outperformed the leading one. For example, for the Cal Housing and House 16H datasets, ORTO-A has the smallest mean squared error on average, but FIMT-DD and OBag perform better for the intervals with fewer learning examples. For the House 8L and Pol datasets, again, the leading method ORTO-A is outperformed by ORTO-BT for the less populated intervals. On the Wind dataset, ORF performs best on average but its error is higher as compared to the rest of

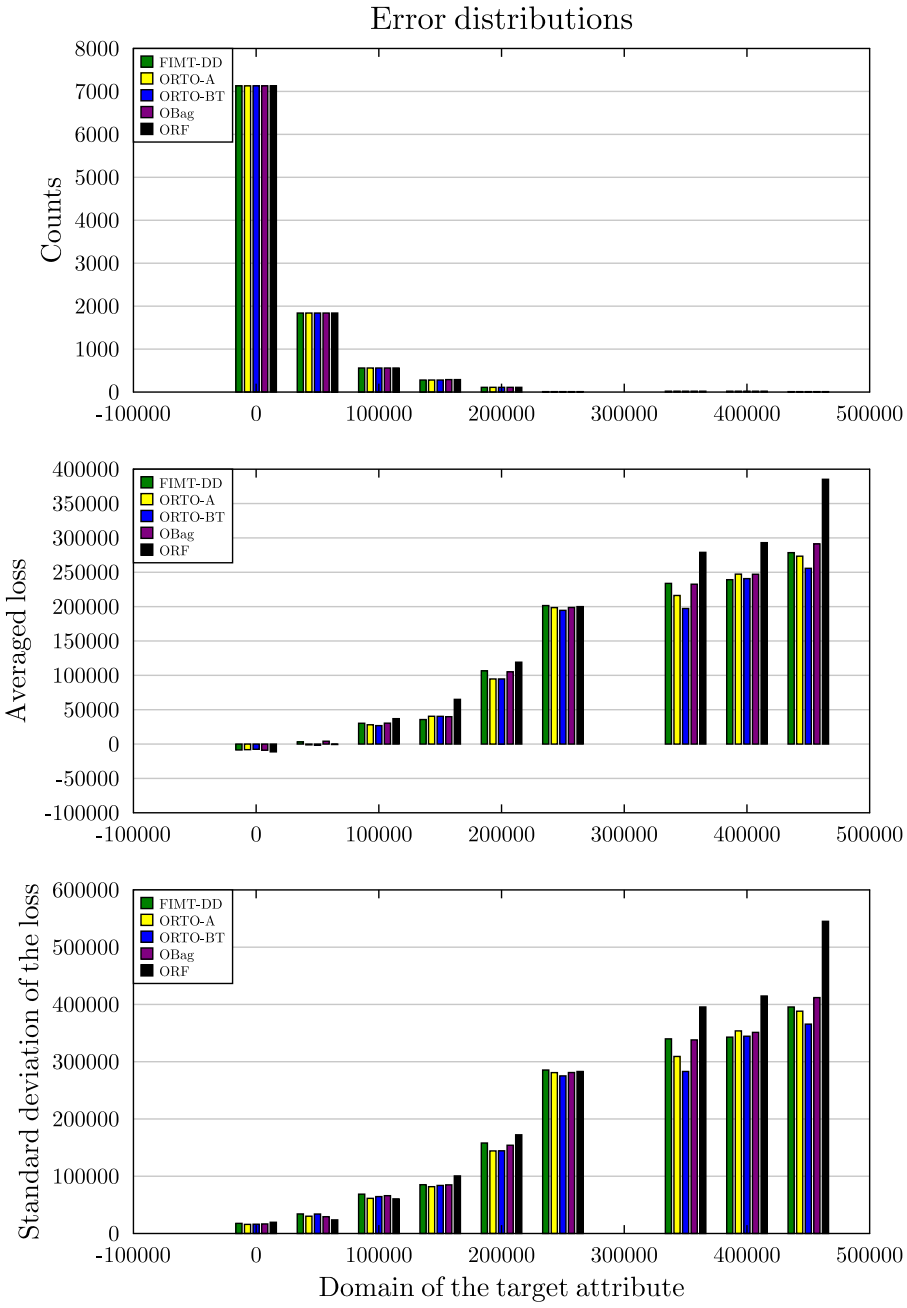


Figure 21: Distribution of errors over the range of the target attribute obtained for the last holdout evaluation on the House 8L dataset.

the methods in the less populated intervals of the domain, where OBag performs best. In addition, there is a pattern that we found to repeat across all of the datasets. Namely, the ORF method has the highest absolute error for the less populated intervals (except for the Abalone dataset).

8.4.2 Analysis of Memory and Time Requirements

Table 18: The memory used by the single-tree method FIMT-DD and each of the ensemble methods, shown as the average allocated memory in MB over 10 runs of sampling on the UCI and Infobiotics datasets. The best results (excluding the single-tree method FIMT-DD) are shown in boldface.

Dataset	FIMT-DD	ORTO-BT	ORTO-A	OBag	ORF
Abalone	3.26	10.69	**10.69**	34.10	11.80
Cal housing	26.20	130.29	130.43	267.20	**78.60**
Elevators	9.45	67.56	67.57	90.70	**28.30**
House 8L	41.55	181.22	181.29	425.40	**114.80**
House 16H	77.29	335.95	335.95	779.60	**235.80**
Mv delve	98.41	265.62	265.88	980.80	**249.70**
Pol	9.54	52.11	52.29	94.70	**29.60**
Wind	8.74	8.74	**8.74**	87.30	26.10
Wine Quality	3.47	16.52	16.52	31.70	**16.40**
Infobiotics	158.62	2192.16	2178.51	1581.80	**271.10**

Table 19: The learning time for the single-tree method FIMT-DD and each of the ensemble methods, given in seconds as the average from 10 runs of sampling on the UCI and Infobiotics datasets. The best results (excluding the single-tree method FIMT-DD) are shown in boldface.

Dataset	FIMT-DD	ORTO-BT	ORTO-A	OBag	ORF
Abalone	0.05	0.17	**0.15**	0.50	0.20
Cal housing	0.44	2.82	2.67	4.50	**2.30**
Elevators	0.61	4.40	4.50	6.10	**3.90**
House 8L	0.52	2.55	**2.53**	5.50	2.60
House 16H	1.62	5.55	**5.39**	16.20	7.30
Mv delve	1.24	3.55	**3.52**	12.20	5.70
Pol	1.42	3.44	**3.37**	14.00	8.00
Wind	0.22	0.22	**0.21**	2.20	1.00
Wine Quality	0.09	0.40	**0.39**	0.90	0.50
Infobiotics	2600.84	13526.00	13443.29	26508.70	**3993.50**

Next, we investigate and compare the space and time requirements of each algorithm. Table 19 gives the averaged learning time in seconds over 10 runs. The fastest learning algorithm, excluding the single-tree method FIMT-DD, is the ORTO-A algorithm. We can see that ORTO-BT is also slower than ORTO-A because of the necessity to search for the best path every time a prediction is needed. The ensemble methods are much slower, due to the sequential manner of updating each of their member hypotheses. However, if parallelized, their running time would be approximately 10 times shorter (here 10 is the number of trees in the ensemble).

Table 18 further shows the average allocated memory (in MB). We can see that for most of the datasets, the ORF algorithm allocates the least amount of memory, excluding the FIMT-DD algorithm which learns a single model. Figure 22 gives the allocation of memory for each algorithm over time for one of the datasets (Cal Housing) on the left panel, and the corresponding curve showing the learning time in seconds is shown on the right panel. Visibly, the most expensive algorithm is OBag. Due to the fact the memory

allocation is mainly dependent on the number of leaves in the tree, if an option tree has many option nodes nested one below another then its memory allocation might exceed the memory allocation of an ensemble, especially of the size of the trees in the ensemble is constrained to a pre-specified height.

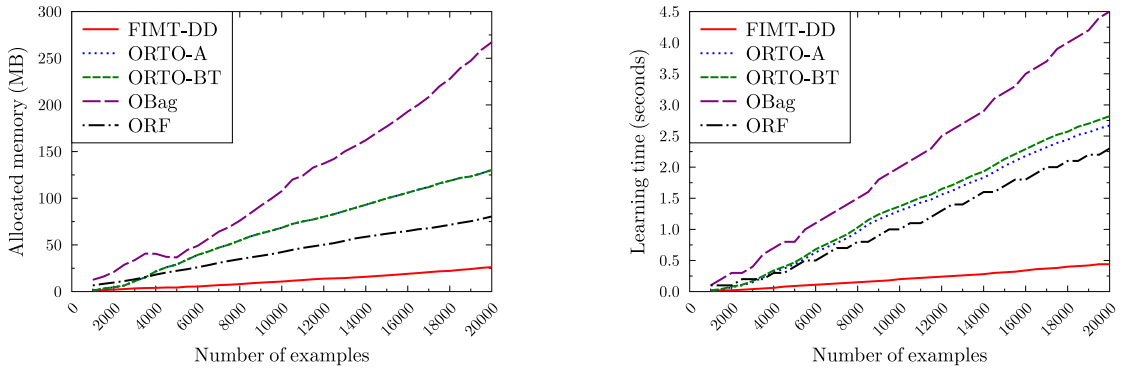


Figure 22: The allocated memory in MB (left) and learning time in seconds (right) evaluated periodically using a sliding window holdout evaluation method for the Cal Housing dataset. Note that in the left panel, the curves for ORTO-A and ORTO-BT coincide.

8.4.3 Bias-Variance Analysis

Interesting insights on the inherent properties of the different methods can be gained by using a bias-variance decomposition of the error. Table 20 shows the bias component of the error for the last sliding window holdout bias-variance decomposition, averaged over 10 runs of sampling. The corresponding variance component of the error is given in Table 21. Additional figures giving the variation of the bias and the variance over time for each of these datasets are given in Appendix A subsection A.4 (A.37(a), A.37(b), A.39(a), A.39(b), A.38(a), A.38(b), A.38(c), A.38(d) and A.40(a)).

The typical behavior for most of the datasets is the following: the bias component of the error is decreasing in time, while the variance and the covariance are increasing. This is natural, since as the tree grows, its predictions will improve until the tree reaches the optimal size, when it will be able to model the regression surface well enough. The larger the tree, however, the greater is the risk of overfitting. As a result, the variability of the predictions of the ensemble members increases. Also, with the growth, the trees become more similar, which increases the correlation between the predictions of the ensemble members. This is visible in most of the figures. However, one should keep in mind that for these datasets the density of the examples in the example space is not uniform over time, thus the buffered testing datasets might vary drastically over time.

The algorithms that performed best on individual datasets had in general the smallest bias component of the error. An exception is the ORTO-BT algorithm which showed a smaller bias component of the error in general, compared to the winning ORTO-A, but at the cost of a larger variance. The resulting difference in these deltas is such that ORTO-BT was outperformed by ORTO-A. However, this tells us that the option nodes indeed provide a better way to explore the search space and find the best tree over the most current data. The increased variability is due to the way the best tree is chosen in ORTO-BT. Namely, the selection procedure is based on a weighted (faded) mean squared error estimate obtained over the observed training examples at each option node. Therefore, there is a direct influence of the diversification of the training data due to the sampling method. On the other hand, in ORTO-A, the multiple predictions are combined by averaging, which

decreases the variability and obviously accounts for the best predictive performance.

Table 20: The bias component of the error for the FIMT-DD algorithm and the ensemble methods for online regression. Averages are given from 10 runs of sampling for the last holdout bias-variance decomposition on the UCI and Infobiotics datasets.

Dataset	FIMT-DD	ORTO-BT	ORTO-A	OBag	ORF
Abalone	6.39E+0	5.74E+0	5.59E+0	6.53E+0	**4.63E+0**
Cal Housing	7.25E+9	**4.47E+9**	4.75E+9	6.37E+9	4.54E+9
Elevators	2.60E-5	2.20E-5	2.40E-5	2.60E-5	**2.10E-5**
House 8L	1.15E+9	1.07E+9	**1.05E+9**	1.18E+9	1.25E+9
House 16H	1.39E+9	**1.37E+9**	1.42E+9	1.43E+9	1.42E+9
Mv Delve	13.42E+0	13.16E+0	13.86E+0	**11.96E+0**	36.44E+0
Pol	235.20E+0	207.75E+0	**191.08E+0**	227.20E+0	927.18E+0
Wind	41.35E+0	41.35E+0	41.35E+0	41.02E+0	**15.07E+0**
Wine Quality	0.49E+0	**0.48E+0**	0.49E+0	0.51E+0	0.52E+0
Infobiotics	28.45E+0	28.11E+0	**27.87E+0**	28.44E+0	29.62E+0

Online bagging was observed to have the smallest variability of the predictions, but its bias components are larger. While online bagging in general helps to reduce the variability of the predictions, the option nodes in ORTO and the randomization introduced in ORF help to reduce the bias component of the error, which can be understood as a method to improve the intrinsic ability of the learning algorithm to model the phenomenon under study. While the option tree is able to reduce the variance by averaging, the online bagging procedure used in ORF achieves this with greater success.

Table 21: The variance component of the error for the FIMT-DD algorithm and the ensemble methods for online regression. Averages are given from 10 runs of sampling for the last holdout bias-variance decomposition on the UCI and Infobiotics datasets.

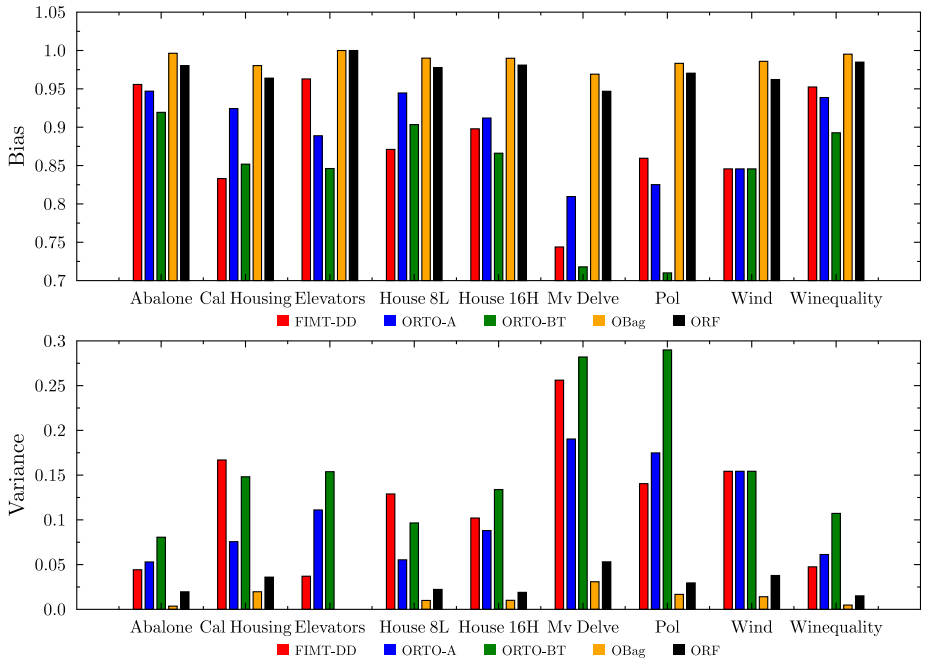
Dataset	FIMT-DD	ORTO-BT	ORTO-A	OBag	ORF
Abalone	0.29E+0	0.50E+0	0.31E+0	**0.02E+0**	0.09E+0
Cal Housing	1.45E+9	0.77E+9	0.39E+9	**0.13E+9**	0.17E+9
Elevators	0.10E-5	0.40E-5	0.30E-5	**0.00**	**0.00**
House 8L	0.17E+9	0.11E+9	0.62E+8	**0.12E+8**	0.28E+8
House 16H	0.16E+9	0.21E+9	0.14E+9	**0.15E+8**	0.28E+8
Mv Delve	4.62E+0	5.17E+0	3.26E+0	**0.38E+0**	2.04E+0
Pol	38.44E+0	84.82E+0	40.49E+0	**3.87E+0**	28.24E+0
Wind	7.54E+0	7.54E+0	7.54E+0	**0.59E+0**	**0.59E+0**
Wine Quality	0.02E+0	0.06E+0	0.03E+0	**0.00E+0**	0.01E+0
Infobiotics	1.31E+0	1.36E+0	0.50E+0	**0.11E+0**	0.17E+0

Figure 23 further illustrates the relative bias and variance components of the error, computed on the last holdout evaluation. The smallest relative bias for all of the datasets is observed for the ORTO-BT algorithm, while the smallest relative variance for all of the datasets is observed for the OBag algorithm. Also, we can see that, in general, a single model has a higher variance as compared to an ensemble of model trees.

8.4.4 Sensitivity Analysis

We now present the results from the tests performed in order to assess the influence of the parameters: maximum allowed tree depth l_{max}, and maximum number of trees in the ensemble T_{max}. Detailed results are given in Appendix A section A.5 (Tables A.34(a), A.34(b), A.35(a) and A.35(b)).

Figure 23: Relative bias and variance components of the error of the FIMT-DD algorithm and the different ensemble approaches computed on the last holdout evaluation.



When the ensemble size was fixed to $T_{max} = 10$, varying the maximal tree depth from level 5 to level 7 showed that allowing the tree to grow bigger can improve the accuracy only for some of the datasets. In particular, it was interesting to see that, by allowing the tree to grow one more level, the ORTO-A algorithm showed better accuracy compared to the FIRT-DD algorithm, especially on the House 16H dataset on which FIRT-DD was initially evaluated as better. For the remaining algorithms, we observed a significant increase of the accuracy only for the Mv Delve and Pol datasets. Contrary to this, for the Cal Housing, House 16H and Winequality datasets increasing the maximum tree level decreased the accuracy of OBag and ORF. In conclusion, for most of the datasets, the best results were obtained for a maximum level of depth $l_{max} = 6$.

With respect to the second parameter T_{max}, increasing the maximum number of trees of fixed depth resulted in slightly improved accuracy for the ORTO-A algorithm and for the ORF algorithm. While this trade-off is not so expensive for the ORTO-A algorithm, for the ORF algorithm the small increase of accuracy comes at a price of substantially higher memory allocation. This is one of the advantages of option model trees over ensembles of randomized model trees.

8.4.5 Responsiveness to Concept Drift

Finally, we examine the responsiveness of each learning algorithm to changes in the target function. For this task, we use only the last two datasets, i.e., the *City Traffic* and the *Airline* dataset, on which concept drift might be expected. We have chosen to evaluate the responsiveness to changes by observing the trend of the mean squared error, i.e., its variation over time.

Figure 24 shows on the left panel the evolution over time of the mean squared error and the total number of leaves of the different ensemble methods for the City Traffic dataset over time, and on the right panel the corresponding curves for the Airline dataset. In this respect, the best responsiveness to change was observed for the FIMT-DD and ORTO-A algorithms. The ORTO-BT and ORF algorithms were observed to behave in a highly unstable fashion,

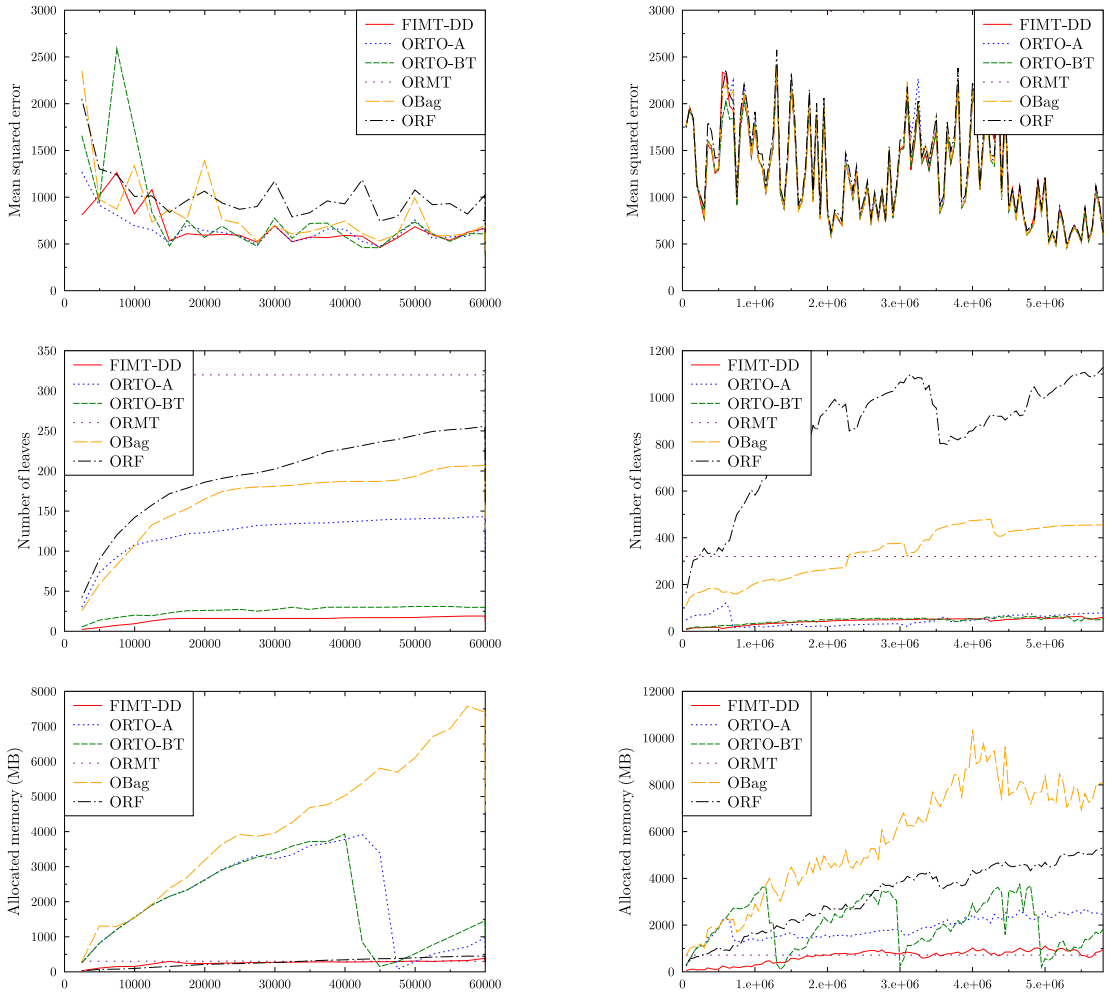


Figure 24: Mean squared error (upper), number of leaves (lower) and allocated memory (lowest, in MB) for the single-tree method FIMT-DD and the different ensemble methods measured periodically using a sliding window holdout evaluation method for the City Traffic (left) and Airline (right) datasets.

while OBag is somewhere in-between. None of the algorithms detected concept drift on the City Traffic dataset.

On the other hand, for the Airline dataset the algorithms detected a large number of changes, which resulted in constant adaptation of the models over time. This can be seen in Figure 24 on the lower plot, which shows the number of leaves for each method measured periodically. It is also visible that there is no clear winner, although the ORF performed worse than the rest in general. The memory allocation on these two datasets is given at the bottom of Figure 24. The OBag method has naturally the highest memory allocation.

8.5 The Diversity Dilemma

Diversity among the members of a team (ensemble) of classifiers is deemed to be a key issue when combining classifiers. However, measuring diversity is not straightforward, because it is not clear if any of the proposed measures is actually useful. In their study, Kuncheva and Whitaker (2003) have examined ten statistics which can measure diversity among binary classifier outputs, and investigated the relationship between the accuracy of the ensemble

and its diversity. It appeared that there is no clear relationship between diversity and the averaged individual accuracy, which is counter-intuitive. However, as they note, it is possible that in different circumstances, diversity and accuracy could exhibit a stronger relationship. The conclusion of their study ends with the sentence: "... *Our study suggests that the general motivation for designing diverse classifiers is correct, but the problem of measuring this diversity and so using it effectively for building better classifier teams is still to be solved.*" We devote this last section of our study to investigate the relationship between the diversity of the ensemble and its predictive performance. The question we wish to answer is: *Will an increase of the diversity of the ensemble of online model trees improve its overall accuracy and how can we achieve that?*

For this task, we have used the bias-variance-covariance decomposition of the error, which is applicable only for the ensemble-learning methods. The reason why this comparison does not apply to ORTO-A (and especially not to ORTO-BT) is that the decomposition requires a separate prediction from each base member for every example from the hold-out evaluation dataset. The option tree will not necessarily provide multiple predictions for every example, but only for those examples that belong to the example space covered by an option node. Further, the number of different predictions for different examples needs not to be equal. The ORTO-BT method should be considered as a single-tree method because it will always provide only a single prediction per example.

Here we give a comparison in terms of the decomposition of the mean squared error for the OBag and ORF algorithms. This analysis should provide some more insights into the stability of the models, and the level of correlation between the ensemble members, which can be related to the level of diversity. The decomposition of the error was performed according to Equation (14) given in Chapter 4. The obtained results on the bias, variance and covariance components of the error by using the corresponding Equations (15), (16), and (17) are given in Table 22. The corresponding figures are given in Appendix A subsection A.6 (A.41(a), A.43(a), A.42(a), A.41(b), A.42(b), A.43(b), A.44(a), and A.45(a)).

Due to the fact that the same building mechanism is used when creating the forest of randomized regression trees, the ORF algorithm is essentially a randomized version of the OBag algorithm. Thus, a comparison of their error components should give an insight into how the randomization actually affects the error. What we have observed is that the randomization method reduces the bias, but not for all of the datasets. On the other hand, the trees grown without randomization naturally have smaller variability of the predictions, thus randomization increased the variance. Further, while online bagging decreased the variance maximally, the online random forest method was shown to produce the least correlation among the predictions of the ensemble members. We can conclude that the randomization procedure in ORF actually increases the variance, but it also substantially decreases the covariance, which at the end results in more accurate predictions.

Recalling that the covariance is a measure of the strength of the correlation among the predictions from the different ensemble members, smaller covariance means smaller correlation. If we interpret diversity through the correlation between the models' predictions, the final conclusion would be that a randomization improves the diversity among the ensemble members, but it might not always lead to a better accuracy overall. Although randomization might not always improve the accuracy, it still has several advantages over plain online bagging: online random forests have smaller memory allocation, a shorter learning time, and accuracy comparable to bagging.

To understand how the choice of the number of randomly selected attributes will influence the predictive performance of the ensemble, we varied this parameter by increasing the starting value by one in a sequential manner. For each dataset, the initial value chosen was the square root of the number of descriptive attributes. What we have observed is that increasing the number of randomly chosen attributes would eventually harm the accuracy, due to the increased correlation of the predictions from the ensemble members, although

Table 22: Averaged bias, variance and covariance components of the error from 10 runs of sampling for the last holdout bias-variance-covariance decomposition on the UCI and Infobiotics datasets.

	Bias		Variance		Covariance	
Dataset	OBag	ORF	OBag	ORF	OBag	ORF
Abalone	168.86E+00	**121.04E+00**	**194.13E+00**	307.81E+00	8.95E+03	**5.43E+03**
Cal Housing	1.58E+11	**1.29E+11**	6.16E+11	6.16E+11	1.88E+13	**1.29E+13**
Elevators	6.94E-04	**5.79E-04**	**1.00E-03**	2.00E-03	**0.07E+00**	0.08E+00
House 8L	**2.37E+10**	2.51E+10	**8.49E+10**	1.08E+11	3.37E+12	**2.14E+12**
House 16H	**2.89E+10**	3.02E+10	**7.24E+10**	9.29E+10	1.98E+12	**1.21E+12**
Mv Delve	**374.59E+00**	1004.13E+00	**5.09E+03**	5.81E+03	2.28E+05	**1.11E+05**
Pol	**5.74E+03**	2.33E+04	**7.17E+04**	8.11E+04	3.55E+06	**5.89E+05**
Wind	9.31E+02	**4.67E+02**	5.04E+03	**2.54E+03**	2.30E+05	**6.61E+04**
Winequality	**1.18E+01**	1.28E+01	**1.57E+01**	2.71E+01	4.72E+02	**3.83E+2**
Infobiotics	**7.73E+2**	8.27E+02	7.49E+02	**7.30E+02**	2.89E+04	**1.84E+04**

the variance would be decreased. For some of the datasets, the increase of the number of attributes also harmed the bias component of the error. This analysis additionally confirmed the dependence between the predictive performance of the ensemble and its diversity, showing that a smaller correlation between the predictions of the ensemble members corresponds to a smaller mean squared error in total.

Bifet et al. (2010) have, on the other hand, studied the effect of randomizing the input and the output components of the data stream on the accuracy and the diversity of an ensemble of classifiers. Their findings are similar to ours: A randomization on the weights of the instances of the input stream gives an increase of the accuracy of the ensemble. Another interesting approach to introduce more diversity in an ensemble of classifiers is to use trees of controlled depth (Bifet et al., 2009b). Although the main motivation in this work was to address the problem of learning drifting concepts, the method was shown to increase the diversity which can improve the accuracy even when the concepts are stable

8.6 Summary

In this chapter we studied randomized methods for online learning ensembles of regression and model trees. Randomized algorithms are particularly useful when the dataset is described by many input variables, each containing only a small amount of information. One of the reasons for this is that the randomization helps in achieving low correlation among the strong models in the ensemble. We proposed and implemented two novel ensemble learning methods for regression on data streams: Online bagging with FIRT-DD trees, and an Online Random Forest in which we have used a randomized version of the FIRT-DD algorithm as a basic building block.

Within the study presented in this chapter, we evaluated the proposed algorithms along several dimensions, including predictive accuracy and quality of models, time and memory requirements, bias-variance and bias-variance-covariance decomposition of the error, and responsiveness to concept drift. The results from the empirical evaluation show that online bagging of FIMT-DD trees improves the accuracy of the base models by reducing the variability of their predictions. On the other hand, online random forests of randomized FIRT-DD trees improve the accuracy of the base models by improving the diversity of the ensemble, that is, the bias. The fastest learning algorithm, excluding the single-tree method FIMT-DD, is the ORTO-A algorithm. The ensemble methods are the slowest ones, although, if implemented through parallelization their running time would be significantly reduced. Increasing the maximum number of trees can improve the accuracy for both the option tree and the online random forests on the cost of an increased memory allocation. The memory allocation is clearly dependent on the number of leaves which store the necessary E-BST structures for numerical attributes and counts for the nominal attributes. While this trade-off is not so expensive for the ORTO-A algorithm, the small increase of accuracy for the ORF algorithm comes at a price of substantially higher memory allocation.

On the question of which is the best method to promote diversity in order to improve the accuracy, we can conclude that it depends on the type of the dataset we wish to analyze. With respect to the predictive accuracy, we compared online random forests with online bagging, where both use the same sampling method. While it did improve the accuracy on some of the datasets, randomization was not shown as the most convincing approach to obtain better accuracy. In the comparison with the option trees, we've observed that the ORTO-A algorithm was responsible for the best accuracy on most of the datasets, while the ensemble of randomized trees in general performed worse. However, there were datasets on which ORF had the best accuracy overall.

Noting that random forests (and randomized methods in general) are expected to improve performance when the dataset is described with many features, each associated loosely with the target attribute, is the key point. Namely, in such situations introducing options

will not be able to advance the search of the hypotheses space, mainly because the set of the ”best” options will not lead to a set of diverse and accurate hypotheses overall. A simple random choice brings more diversity when there are many hypotheses with equal or similar merit. A confirmation of this hypothesis is the fact that the datasets for which we could observe an improvement with randomization were those on which the highest gains in memory were achieved when constraining the option tree without loosing much accuracy. We can confirm the well known statement ”There is no single best method for every problem”. The choice should be made by close examination of the data under study.

9 Online Predictive Clustering Trees for Multi-Target Regression

Simplicity is the ultimate sophistication.

Leonardo da Vinci

Most research in data mining has focused on problems where the task to predict the value of a single target attribute. However, there exist many real life problems where the task is to predict not one, but several related target attributes. In the scenario of learning from data streams, one may think of many examples of correlated streams of data (e.g., temperatures in the same building, traces in the same road network, stock prices). Some interesting applications for multi-target learning are predicting the congestion at multiple road segments, tracking multiple-targets, predicting exchange rates of multiple currencies, and multi-robot learning, to name a few.

The problem of multi-target prediction can be decomposed and solved by modeling each target attribute independently. However, when the output space is large, simultaneous learning of many models can become prohibitively expensive. Interpreting the learned models becomes a problem as well: A large collection of single-target models is far less understandable than a single model that jointly predicts the values of all target attributes.

The problem of predicting multiple targets, both discrete and numerical, has been addressed in the framework of predictive clustering (Blockeel et al., 1998). In this chapter we study the applicability of the predictive clustering framework to the problem of online modeling of multiple numerical targets from data streams. We propose an approach to learning multi-target model trees from data streams and give some preliminary results and conclusions on some simulated and real-world datasets.

9.1 Online Multi-Target Classification

Like in batch learning, one may identify a class of streaming real-world problems which require the modeling of several targets simultaneously. Due to the dependencies among the targets, simultaneous modeling can be more successful and informative than creating independent models for each target. As a result, one may obtain a smaller model, able to simultaneously explain the relations between the input attributes and the targets. This problem has not been addressed previously in the streaming setting.

A promising approach, whose simplicity enables very low processing time per example while inducing small-sized models, is the predictive clustering framework proposed by Blockeel et al. (1998). Before discussing the details on how this framework can be extended to the online learning scenario, we will review the existing approaches for online learning of multi-target classification trees, or more precisely the methods for online multi-label classification. To the best of our knowledge, there exists no other work for online multi-target prediction in the regression domain. The only relevant work on online multi-label classification is presented as follows.

As stated previously, the problem of online multi-target prediction has been mostly studied in the context of multi-label classification. Given an input vector $\mathbf{x}$ and a set of possible labels $\mathscr{L}$, the task of multi-label classification is to predict which of those labels are relevant for a particular instance. The *label set* is typically represented as an L-vector $\mathbf{y} = [y_1, ..., y_L] = \{0,1\}^L$, where $y_j = 1$ iff the j-th label is relevant and $y_j = 0$ otherwise. Thus, in *multi-label* classification, each instance is assigned a subset of labels $S \subseteq L$. Typical real-world applications where the multi-label problem becomes relevant are text classification (Schapire and Singer, 2000), scene classification (Tsoumakas and Katakis, 2007) and gene function prediction (Barutcuoglu et al., 2006; Schietgat et al., 2010).

As with other multi-target prediction problems, a multi-label classification problem can be transformed into one or more single-label classification problems. There exist essentially three methods for transforming the multi-label problem:

1. **Binary Method.** With this method, a set of L binary classifiers are learned $B_0, ..., B_L$, where each B_i is responsible for predicting the association with the corresponding label $l_i \in \mathscr{L}$. The main advantage of using this method is that there are plenty off-the-shelf sophisticated binary classifiers available, which can be used to solve the binary classification problems obtained with the transformation. This is potentially beneficial for learning on data streams due to the existence of incremental single-label classifiers, such as Naive Bayes or k-Nearest Neighbor, whose time complexity is low and which can be easily parallelized. In particular, bagging of binary Hoeffding trees has been shown as a particularly successful method for online multi-label classification (Read et al., 2012).

2. **Ranking Method.** The ranking method relies on having a single-label classifier able to produce a probability distribution over the set of labels. At training time, all of the training examples are duplicated such that a separate single-label example is created for each relevant label. The prediction is obtained by applying a threshold on the obtained posterior probabilities. All class labels which are assigned a probability above a given threshold are considered as relevant. This method and the binary transformation method both suffer from the *label independence* assumption, failing to take advantage of any relationships between labels. However, they are both easily applicable in the streaming scenarios.

3. **Combination Method.** This method creates a single-label problem by treating each observed label combination (or label set) S as an atomic label l'_i. For example, the multi-label set $\{a, b, f\}$ would become a new single label abf. Hence, the set of all distinct label combinations as observed in the data becomes the new set of possible single labels L' which will be used to train a single-label classifier. In the literature methods that perform transformations of this type are known as Power Label-set (PL) methods (Tsoumakas and Katakis, 2007). While this method overcomes the *label independence* assumption problem, it suffers from an exponential expansion of the target space. A second crucial disadvantage is that it can only assign a particular label combination to a test example if that combination has been seen in some of the training examples. The evolution of the target space is actually a serious problem when extending these methods to the online learning setup.

In contrast to transforming a multi-label problem into one or several single-label problems, Read et al. (2012) propose an algorithm for learning multi-label Hoeffding trees which attacks the multi-label problem through the successful divide-and-conquer approach. The algorithm extends the well known Hoeffding trees in two directions. The first extension consists of using a multi-label measure for computing the information gain. The authors use the definition of the entropy for a set of multi-labelled examples proposed by Clare and

King (2001). For a set of examples S with N classes and probability $p(c_i)$ for each class c_i in S, the entropy is given with the following equation:

$$entropy_{ML}(S) = -\sum_{i=1}^{N} p(c_i)\log p(c_i) - \sum_{i=1}^{N}(1-p(c_i))\log(1-p(c_i)) \tag{56}$$

In this new definition, a new term $(1-p(c_i))\log(1-p(c_i))$ is added for each class c_i that represents the information needed to describe all the classes the example does not belong to. This multi-label entropy measure should quantify the amount of uncertainty in the dataset when all the classes an example belongs to are considered.

The second extension is the addition of multi-label classifiers in the leaves. Basically, any multi-label classifier can be used, having the *majority-label-set* classification rule as a default prediction. The multi-label classification process is enhanced with a batch method for pruning infrequently occurring label sets. The authors claim that this method helps to avoid much unnecessary and detrimental complexity and is able to improve the accuracy of the multi-label Hoeffding tree. This work is encouraging and provides some motivation for using the predictive clustering framework in solving the problem of online multi-target regression, since this framework is also founded on the divide-and-conquer idea. Following up on this matter, in the next section, we discuss the main ideas of our proposal.

9.2 Online Learning of Multi-Target Model Trees

One interesting approach to the multi-target prediction problem, which falls in the category of covariance-agnostic methods described in Chapter 3, is the predictive clustering methodology by Blockeel et al. (1998). Loosely stated, predictive clustering methods seek to construct clusters of examples which are similar to each other not only in the space of the target attributes, but also in the space of the descriptive attributes, while simultaneously associating a predictive model with each constructed cluster. Although we have already described the main ideas behind the predictive clustering methodology in Chapter 3, we will once again address the predictive clustering task here, but from the perspective of online learning.

The most intuitive way to describe the predictive clustering methodology is to consider a hierarchical type of a predictive model, like a decision or a regression tree, which partitions the space of descriptive (predictor) attributes into subspaces, each of which is associated with a node of the decision tree. Thus, each node corresponds to a cluster of examples which are assigned to it during the process of tree construction. Since the partitioning based on an evaluation function which strives to reduce the deviation of the target attribute values for the examples in a cluster, the examples in each cluster are expected to be similar in the space of the target attribute. However, if our prediction task involves multiple targets, not necessarily of the same type, an intuitive approach would be to construct a tree such that the examples grouped in a cluster would be similar in the more complex space of multiple target attributes. This basically means that one can apply the same principles for building decision or regression trees by defining a similarity measure in the space of the target attributes, just as in clustering, which is typically performed in the space of the descriptive attributes.

The most important assumption of predictive modeling is that there exist clusters for which the examples are similar, both in the space of descriptive attributes and in the space of the target attributes. Except for this assumption, the predictive clustering methodology does not make any other special assumptions, which are specific to the batch learning setup and different from the assumptions made when constructing a decision or a regression tree. Therefore, given an approach to incrementally learn decision and regression trees, we can extend it in a straightforward fashion to solve the problem of predicting multiple targets online. Under the previous assumption, the biggest challenge in an online learning setup

would be to track and adapt to changes in the target space, which now may evolve or drift in various complex ways. Before discussing the more advanced issues, we will first introduce the extensions we propose to enable learning multi-target regression trees under stationary data distributions.

9.2.1 Extensions to the Algorithm FIMT-DD

Using the predictive clustering methodology, we have designed and implemented an online model tree learning algorithm for simultaneous modeling of multiple numerical targets termed FIMT-MT. The algorithm extends the incremental single-target model tree algorithm FIMT-DD by incorporating a similarity measure defined in the space of the target attributes within the split selection criterion. In this work we have considered only numerical targets, although the same ideas would apply for the case of mixed types of targets (nominal and numerical), assuming that an appropriate similarity measure can be defined.

Given a cluster C of n_{min} examples observed over time at one of the leaf nodes of the model tree, the split selection criterion selects the split that would create two clusters C_1 and C_2 that correspond to the minimal sums of squared distances, given the examples that belong to the clusters, from the corresponding centers of mass. The center of mass c for a given cluster of examples C is called a prototype and is defined with a prototype function $c = Pro(C)$ specific for the task at hand, in our case multi-target regression. It is also important to weight the sums of squared distances with the number of examples falling in each cluster in order to take into account the density of each cluster. It is more beneficial to avoid creating very small but homogeneous clusters, thus the merit of the split is evaluated by comparing the homogeneity of the cluster per example. The resulting split evaluation function is given with:

$$\frac{1}{|C|}\sum_{y\in C} Dist(y, Pro(C))^2 - \frac{1}{|C_1|}\sum_{y_i\in C_1} Dist(y_i, Pro(C_1))^2 - \frac{1}{|C_2|}\sum_{y_i\in C_2} Dist(y_i, Pro(C_2))^2. \quad (57)$$

In order to compute the prototype for every candidate split and the corresponding distances, the online learning setup does not permit storing examples that were sorted to the leaves of the tree. The computation of the prototype and the distances has to be done on-the-fly and incrementally. This would pose some serious difficulties for arbitrarily complex tasks and target spaces, but for the task of predicting multiple numerical targets it is possible to define simple decomposable prototype and distance functions.

The prototype function for the case of multi-target regression will return the vector of mean values computed incrementally for each of the corresponding target attributes. The distance metric from the center of mass in this case can be the Euclidean distance, which can also be decomposed and computed on-the-fly.

At a high level, the FIMT-MT algorithm is similar to the FIMT-DD algorithm. The pseudo-code is given in Algorithm 8. The main difference, except for the split selection criterion is that, instead of a single prediction, the algorithm will return a set of predictions $\mathbf{p} = \{p_1, p_2, ..., p_T\}$ given with the current hypothesis. We have also extended the idea of using un-thresholded perceptrons for computing linear models on-the-fly in the space of multiple target attributes. This requires initializing and training a separate perceptron per target attribute. The details of the procedure are given in the following sections.

9.2.2 Split Selection Criterion

In the design of the FIMT-MT algorithm, we have chosen a combination of a prototype and a distance function that results in minimizing the weighted intra-cluster variance. The intra-cluster variance is the sum of the squared Euclidean distances between every example

Algorithm 8 *FIMT-MT: An extended FIMT-DD algorithm.* Pseudocode.

Input: *RootNode*, e - a training instance, δ - the confidence parameter, n_{min} - the chunk size, T - number of target attributes, *Pro* - a prototype function, *Dist* - a distance function.
Output: The set of predictions $\mathbf{p} = \{p_1, p_2, ..., p_T\}$ from the current hypothesis.

Leaf $\leftarrow$ *Traverse*(*RootNode*, e)
▷ Traverses the example e to the corresponding *Leaf* node.
Counter $\leftarrow$ *SeenAt*(*Leaf*)
if *Seen*(*Leaf*) $= 0$ **then**
  for $k = 1 \rightarrow T$ **do**
    InitializePerceptronPerTarget(Leaf, k)
  ▷ Initializes the weights of the perceptrons located in the leaf to random values in the range [0, 1].
  end for
end if
Counter $\leftarrow$ *Counter* $+ 1$
for $k = 1 \rightarrow T$ **do**
  $p \leftarrow$ *GetPredictionPerTarget(Leaf, e, k)* ▷ Collects multiple predictions each corresponding to a single target attribute.
end for
for $k = 1 \rightarrow T$ **do**
  $y_k^n \leftarrow$*NormalizeTargetValue(e, k)*
  ▷ Normalizes the target values by applying the studentized residual equation.
end for
UpdateStatistics(Leaf, e, y^n)
for $k = 1 \rightarrow T$ **do**
  UpdatePerceptronPerTarget(*Leaf*, k, y_k^n) ▷ Updates the weights of the perceptrons located in the leaf.
end for
if *Counter* $\bmod n_{min} = 0$ **then**
  for $i = 1 \rightarrow d$ **do**
    $S_i =$*FindBestSimilaritySplitPerAttribute(Pro, Dist, T, i)*
    ▷ Finds the best split point per attribute.
    $S_a \leftarrow$ *Best*($S_1, ..., S_d$)
    $S_b \leftarrow$ *SecondBest*($S_1, ..., S_d$)
    if $S_b/S_a < 1 - \sqrt{\frac{\ln(1/\delta)}{2 \times Counter}}$ **then**
      MakeSplit(*Leaf*, S_a) ▷ Creates a binary split by applying S_a.
    end if
  end for
end if
for $k = 1 \rightarrow T$ **do**
  $t_k \leftarrow$ *GetTargetValue(e, k)*
  $\Delta_k \leftarrow$ *ABS*($t_k - p_k$) ▷ Computes the absolute prediction error for every target attribute.
end for
BackPropagate($\{\Delta_1, ...\Delta_T\}$)
  ▷ Back-propagates a vector of absolute error values towards the root of the tree.
return $\{p_1, ..., p_T\}$

in the cluster C and the prototype $\overline{\mathbf{y}}$ (the center of mass). The prototype is the vector of the mean values of each target attribute y_k, that is, $\overline{\mathbf{y}} = \{\overline{y}_1, \overline{y}_2, ..., \overline{y}_T\}$, where T is the number of target attributes. The resulting intra-cluster variance is then given with:

$$IVar(C) = \frac{1}{|C|} \sum_{i=1}^{|C|} d(\mathbf{y_i}, \overline{\mathbf{y}})^2, \tag{58}$$

where $\mathbf{y_i} = \{y_{i_1}, y_{i_2}, ..., y_{i_T}\}$ is a vector of values for the target attributes of a given training example $(\mathbf{x_i}, \mathbf{y_i}) \in C$ and $d(\mathbf{y_i}, \overline{\mathbf{y}})$ is the Euclidean distance between the vector $\mathbf{y_i}$ and the prototype $\overline{\mathbf{y}}$. With a little algebra, by replacing the $d(\mathbf{y_i}, \overline{\mathbf{y}})$ with:

$$Dist(\mathbf{y_i}, \overline{\mathbf{y}})^2 = (y_{i_1} - \overline{y}_1)^2 + (y_{i_2} - \overline{y}_2)^2 + ... + (y_{i_T} - \overline{y}_T)^2 = \sum_{k=1}^{T} (y_{i_k} - \overline{y}_k)^2$$

we get the following formula:

$$IVar(C) = \sum_{k=1}^{T} \frac{1}{|C|} \sum_{i=1}^{|C|} (y_{i_k} - \overline{y}_k)^2 = \sum_{k=1}^{T} Var(y_k, C), \tag{59}$$

where $Var(y_k, C) = \sum_{i=1}^{|C|} (y_{i_k} - \overline{y}_k)^2$ denotes the variance of the target attribute y_k on the dataset C.

Algorithm 9 *FindBestSimilaritySplitPerAttribute: An extended FindBestSplitPerAttribute procedure.* pseudocode.

Input: *Pro* - a prototype function, *Dist* - a distance function, T - number of target attributes, *Att* - an attribute id
Output: h_{best} - the split that gives the maximal intra-cluster variance reduction
  if AttributeIsNumeric(Att) **then**
    $Node \leftarrow GetAnExtendedBSTreePerNumericalAttribute(Att)$
      ▷ An extended BST data structure is fetched for every numerical attribute.
  else
    $Node \leftarrow GetAnArrayPerNominalAttribute(Att)$
    ▷ The array has to be sorted in an increasing order according to the sum of variances $\sum_{k=1}^{T} Var(y_k)$
  end if
    ▷ Initializes the global variables
  for $k = 1 \rightarrow T$ **do**
    $sumC_1[k] = 0$
    $sumC_2[k] = Node \rightarrow sumLeft[k] + Node \rightarrow sumRight[k]$
    $sumSqC_1[k] = 0$
    $sumSqc_2[k] = Node \rightarrow sumSqLeft[k] + Node \rightarrow sumSqRight[k]$
    $countTotalC_2 = countTotal = Node \rightarrow countLeft + Node \rightarrow countRight$
  end for
  $maxSDR = 0$
  $h_{best} \leftarrow FindBestSplit(Node)$
  return h_{best}

The reduction of the weighted intra-cluster variance maximally improves the average cluster homogeneity. The split selection is thus defined with the maximal reduction of the inter-cluster variance:

$$ICVR(C_1, C_2) = IVar(C_1 \cup C_2) - \sum_{i=\{1,2\}} \frac{|C_i|}{|C|} IVar(C_i) \tag{60}$$

Algorithm 10 *FindBestSplit: An extended FindBestSplit procedure for multi-target attributes.* pseudocode.

Input: *Node* - the root of the extended binary search tree structure
Output: h - the split that gives the maximal intra-cluster variance reduction

```
if Node → leftChild ≠ ∅ then
    FindBestSplit(Node → leftChild)
end if
        ▷ Updates the sums and counts for computing the IVar differences for the split
for k = 1 → T do
    sumTotalC1[k] = sumTotalC1[k] + Node → sumLeft[k]
    sumTotalC2[k] = sumTotalC2[k] − Node → sumLeft[k]
    sumSqTotalC1[k] = sumSqTotalC1[k] + Node → sumSqLeft[k]
    sumSqTotalC2[k] = sumSqTotalC2[k] − Node → sumSqLeft[k]
    countTotalC2 = countTotalC2 − Node → countLeft
end for
if maxICVR < ComputeICVR(T) then
    maxICVR = ComputeICVR(T)
    h = Node → key
end if
if Node → rightChild ≠ ∅ then
    FindBestSplit(Node → rightChild)
end if
                ▷ Updates the sums and counts for returning to the parent node
for k = 1 → T do
    sumTotalC1[k] = sumTotalC1[k] − Node → sumLeft[k]
    sumTotalC2[k] = sumTotalC2[k] + Node → sumLeft[k]
    sumSqTotalC1[k] = sumSqTotalC1[k] − Node → sumSqLeft[k]
    sumSqTotalC2[k] = sumSqTotalC2[k] + Node → sumSqLeft[k]
    countTotalC2 = countTotalC2 + Node → countLeft
end for
return h
```

By replacing Equation (59) in Equation (60), we get:

$$ICVR(C_1,C_2) = \sum_{k=1}^{T} Var(y_k, C_1 \cup C_2) - \sum_{i=\{1,2\}} \frac{|C_i|}{|C|} \sum_{k=1}^{T} Var(y_k, C_i) \tag{61}$$

This definition can be reduced to the definition of variance reduction over a single target, proposed originally by Breiman et al. (1984), if the target vector contains only one attribute. As long as the split evaluation function can be computed on the fly, a straightforward application of the Hoeffding bound is possible, which requires the ratios of the two best splits to be computed after every training example. The variance can be easily computed incrementally by maintaining the sum of target-attribute values ($\sum y_k$), squared target-attribute values ($\sum y_k^2$) and a count of the observed examples for $k = 1,...,T$, by using Equation (29). An important step in computing the intra-cluster variance is the standardization of the values for each target attributes. The standardization is done by using Equation (47) in Chapter 6 in which we use the target specific standard deviation σ_k. This is necessary in order to ensure that the influence of each target attribute would is equal.

The pseudocode for the *FindBestSimilaritySplitPerAttribute* procedure is given in Algorithm 9. The procedure starts with acquiring the necessary statistics maintained per attribute. If the attribute is a numerical one, then the necessary sums and counts are maintained in an extended binary search tree (E-BST). If the attribute is a nominal one then

Algorithm 11 *ComputeICVR: An extended procedure for computing the reduction of the intra-cluster variance for multi-target attributes.* pseudocode.

Input: T - number of target attributes and access to the global variables $\{sumTotalC_1[k], sumTotalC_2[k], sumSqTotalC_1[k], sumSqTotalC_2[k], countTotal, countTotalC_2\}$ for $k = 1, ..., T$
Output: $ICVR(C_1, C_2)$ - the intra-cluster variance reduction.

$IVar = 0$
for $k = 1 \rightarrow T$ **do**
  $sumTotal[k] = sumTotal[k] + sumTotalC_1[k] + sumTotalC_2[k]$
  $sumSqTotal[k] = sumSqTotal[k] + sumSqTotalC_1[k] + sumSqTotalC_2[k]$
  $IVar = IVar + VAR(sumTotal[k], sumSqTotal[k], countTotal)$
    ▷ The function VAR() is computed using Equation (29)
end for
$IVarC_1 = 0$
$IVarC_2 = 0$
for $k = 1 \rightarrow T$ **do**
  $IVarC_1 = IVarC_1 + VAR(sumTotalC_1[k], sumSqTotalC_1[k], countTotal - countTotalC_2)$
  $IVarC_2 = IVarC_2 + VAR(sumTotalC_2[k], sumSqTotalC_2[k], countTotalC_2)$
end for
return $IVar - \frac{countTotal - countTotalC_2}{countTotal} \times IVarC_1 - \frac{countTotalC_2}{countTotal} \times IVarC_2$

the necessary sums and counts are maintained for every distinct symbolic value. In every split evaluation, the corresponding array of values has to be sorted in an increasing order of the sum of variances $S_{a_i} = \sum_{k=1}^{T} Var(y_k)$ per attribute value a_i, computed from the sums and the counts. For example, given an ordered set of nominal values $\{a, b, c\}$ and an ordering such that $S_b > S_c > S_a$ the new ordering of the array would be $\{b, c, a\}$. This is necessary to avoid the examination of every possible combination of splits in the symbolic domain of the nominal attribute. Details on the extended binary tree data structure for the case of a single target along with some additional ideas to reduce the memory allocation are given in Ikonomovska et al. (2010b). After the initialization of the global variables the procedure *FindBestSimilaritySplitPerAttribute* calls a recursive procedure *FindBestSplit* which will traverse the corresponding data structure starting from a given *Node* and will return the split h_{best} that gives the maximal intra-cluster variance reduction.

The pseudocode for the *FindBestSplit* procedure is given in Algorithm 10. *FindBestSplit* traverses the data structure from left to right, or in the case of a binary search tree in an in-order fashion, starting first with the left child, returning to the parent and visiting the right child. When moving to the left child of the node, the sums and counts per target attribute

$$\{sumTotalC_1[k], sumTotalC_2[k],$$
$$sumSqTotalC_1[k], sumSqTotalC_2[k],$$
$$countTotal, countTotalC_2\}$$

are being incremented with the corresponding sums and counts

$$\{Node \rightarrow sumLeft[k], Node \rightarrow sumRight[k],$$
$$Node \rightarrow sumSqLeft[k], Node \rightarrow sumSqRight[k],$$
$$Node \rightarrow countLeft, Node \rightarrow countRight\}$$

maintained for the current split point represented with the $Node \rightarrow key$. Whenever a new split

point is visited, the corresponding intra-cluster variance reduction is computed using the procedure *ComputeICVR*. If the new variance reduction is better than the current maximal variance reduction, the variable *maxICVR* is updated as well as the associated best split (kept in h). When returning back from the right child to the parent node, the corresponding sums and counts need to be decremented with the same quantities. This action is not necessary if the attribute is nominal. The pseudocode of the *ComputeICVR* procedure is given in Algorithm 11.

9.2.3 Linear Models for Multi-Target Attributes

In Chapter 6, we described a lightweight method for inducing linear models, based on the idea of online training of perceptrons. The trained perceptrons represent the linear models fitted separately in each leaf of the tree. We used the simplest approach: no attribute selection is performed. All the numerical attributes are included in the linear equation represented by a perceptron without an activation function.

Here, we extend the same idea by using a separate perceptron for each target attribute. For updating the weights, we use the *delta* rule (also known as the Widrow-Hoff rule), for which it is known that the learning process converges even in the case when the examples do not follow a linear model. The training procedure is described in Chapter 6. The values of the attributes should be standardized so that each of them will have the same influence during the process of training. We use again a variant of the studentized residual equation, where instead of dividing with one standard deviation we divide by three standard deviations.

An important issue when using the stochastic gradient descent methodology is setting an appropriate value for the learning rate. The learning rate determines the magnitude of the update. If chosen too large, the learning process might never converge. If chosen too small, converging to the best model will be too slow. A common strategy is to use a decaying learning rate. This is also very appropriate in the context of sequential training of the perceptrons through the hierarchy of the tree. The new leaves correspond to more homogeneous regions and thus, finer adaptation is required. When a split is performed, the linear models in the splitting node are passed down to the subsequent nodes, avoiding the need of training "from scratch": The learning continues in each of the new leaf nodes independently, and according to the examples that will be assigned to the leaf.

We use a separate *adaptive learning rate* for each target attribute. The value of the learning rate is set as a fraction (e.g., 1%) of the normalized standard deviation for the particular target attribute. The standard deviation is computed incrementally from the examples observed in the node. Therefore, it corresponds to the confidence in the prediction for that particular target. High confidence values correspond to smaller standard deviations, which means that the target is well predicted with its average. A higher standard deviation means that the average is not a good prediction and urges for a more "aggressive" training of the perceptron. An additional advantage of this idea is that there is no need of setting the learning rate for each specific problem separately. The adaptive learning rate rule can also be applied to the standard model trees induced with the FIMT-DD algorithm, for which the necessary statistics are already maintained in the leaf nodes.

During the initial stages of training, the perceptrons might not provide the most accurate predictions. Thus, the algorithm continuously revises a decision rule in order to decide whether to use the linear model (the perceptron) or to use the average as a prediction. Each perceptron is evaluated using the predictive sequential methodology with a fading factor applied when computing the mean squared error. If the perceptron has lower accuracy compared to the constant regressor (that is, the mean value of the target attribute computed incrementally), then its output will not be used for returning a prediction but only for updating its cumulated mean squared error. If, after a number of examples processed,

the perceptron is trained to approximate well the regression surface of its assigned target attribute, due to the utilization of a fading factor, its past errors will be diminished and the error will represent its performance only over a window of most recently observed instances. If smaller compared to the sum of squared residuals, over which a fading factor was also applied, a preference will be given to the perceptron's output. This evaluation is applied on all of the perceptrons that are trained in the leaves of the multi-target model tree.

9.3 Experimental Evaluation

In this experimental evaluation, we tried to assess the predictive accuracy and the quality of the models induced with the FIMT-MT algorithm. The datasets that were used in the experimental evaluation represent two real-world multi-target regression problems and four simulated multi-target regression problems. All of these datasets are described in detail in Chapter 5.

We have generated 3 artificial datasets using concepts from trees in which the targets are *related* (in one case) and *unrelated* (in the other case). In particular, a multi-target regression tree is generated at random, such that all of the splitting attributes and splitting thresholds are chosen randomly, as well as the numerical predictions per target in the leaf nodes. For every numerical target, a separate standard deviation is chosen, which is used for assigning the true value of the artificially generated training/testing examples. All of the values are chosen randomly with a standard normal distribution.

When targets are *related*, they are given constant but different values for each region of the input space. Since the trees are piece-wise constant, the targets practically behave in the same way, and are thus considered as related. When targets are said to be *unrelated* (in the case of Sim2 and Sim3) the prediction for each target is computed using a simple linear equation. By using a linear plane in the leaf nodes, the regression surface of the tree resembles a mesh of two-dimensional planes for each target. Due to the random choice of linear equations, the shape of the regression surface is not the equal for all of the targets: They behave differently in the same regions of the input space and are thus considered non-related.

The first dataset (Sim1) resembles a "perfect" regression tree of depth 5 (consisting of 5 levels), while the second dataset (Sim2) resembles a less discrete, but still non-smooth regression surface represented by a tree of the same depth. For both datasets, one of the target attributes has a higher standard deviation than the remaining targets. This would make the prediction and the modeling task much more difficult. The third artificial dataset (Sim3) is similar to Sim2 with two important differences: 1) the concepts are more complex (the synthetic random tree is much larger), 2) all of the targets have deviations which belong to the same range of values.

On one hand we performed a comparison with the problem decomposition approach which would give us an insight into what are the advantages when the multiple targets are jointly modeled over a separate modeling approach. On the other hand, we also performed a comparison with a batch-style learning algorithm developed within the *Clus* machine learning toolkit (Blockeel and Struyf, 2003). The CLUS algorithm learns the original predictive clustering trees, for which we have used parameter settings that are adequate for the regression task. In particular, the heuristic used is the variance reduction. The algorithm explores 6 values of the F-statistic parameter in CLUS 0.125, 0.1, 0.05, 0.01, 0.005, and 0.001, and uses 3-fold cross-validation for selecting the best of those parameter values.

The results from the experimental evaluation on real-world datasets with independent modeling of each target attribute are given in Table 23 (upper part). As expected, the batch learner achieves better accuracy for both of the problems, because the concepts are stationary and the algorithm is able to reuse all of the training examples at multiple levels of the tree. However, an important advantage of the incremental learner is that the variation of

the results per target is on average smaller than for the batch learner, which can be seen from the standard deviations. This is due to the fact that the models built using the incremental algorithms are more stable and less sensitive to the choice of training and testing examples.

Results on the complexity of the models (defined as the number of nodes in the tree) are given in Table 24 (upper part). As we can see, the size of the models induced with the incremental learners is significantly smaller. This is due to the lack of data for learning complex enough models, which also affects the accuracy.

The corresponding results when joint modeling was used are given in Table 23 (lower part) and Table 24 (lower part). For both of the real-world problems, joint modeling enables the induction of smaller models with the same accuracy as compared to the approach of independent modeling of each target.

Table 23: The root relative mean squared error (RRMSE) estimated with a 10-fold cross-validation on real-world problems with independent modeling of each target attribute (upper row) and with joint modeling (lower row). The results are averaged across all target attributes.

Dataset	FIRT	FIMT	CLUS-ST
Kras	0.68 ± 0.08	0.69 ± 0.08	0.61 ± 0.10
VegClus	0.87 ± 0.07	0.88 ± 0.07	0.81 ± 0.14
Kras	0.68 ± 0.08	0.69 ± 0.08	0.61 ± 0.09
VegClus	0.88 ± 0.07	0.87 ± 0.08	0.79 ± 0.11

Table 24: The size (number of nodes) estimated with a 10-fold cross-validation on real-world problems with independent modeling of each target attribute (upper row) and with joint modeling (lower row). The results are averaged across all target attributes.

Dataset	FIRT	FIMT	CLUS-ST
Kras	89.18 ± 7.32	89.18 ± 7.32	2606.25 ± 402.07
VegClus	45.01 ± 8.38	45.01 ± 8.38	1366.78 ± 564.02
Kras	79.4 ± 0.00	79.40 ± 0.00	2044.00 ± 0.00
VegClus	39.00 ± 0.00	39.00 ± 0.00	586.80 ± 0.00

The results from the evaluation on the artificial datasets, when independent modeling of each target attribute was used, are given in upper parts of Table 25 and Table 26. The corresponding results when joint modeling was used are given in lower parts of Table 25 and Table 26. There are several conclusions that can be drawn from these experiments.

With respect to the question whether joint modeling is better than independent modeling, we conclude that it mostly depends whether the targets behave similarly in the same regions of the input space, that is whether they are correlated. In general, on all of the simulated problems, the models induced by joint or independent modeling achieved the same accuracy. However, if we are interested in smaller models, joint modeling has an advantage for problems that have correlated target attributes. We have observed that, when a single model is built for all of the target attributes, the size of the model is significantly smaller as compared to the case when independent modeling is used.

It was interesting to see that when the targets are unrelated (Sim2), although the same concepts are used with the same number of attributes, the accuracy of all of the algorithms drops. In addition, the model tree is not able to improve the accuracy of a regression tree because the surface is not smooth. The situation is similar for the Sim3 dataset, with one major difference that brings us to the second conclusion: although the concepts are more complex for the Sim3 problem, the accuracy is better for all of the algorithms. Thus, when

Table 25: The root relative mean squared error (RRMSE) estimated with a 10-fold cross-validation on simulated problems with independent modeling of each target attribute.

Dataset	FIRT	FIMT	CLUS-ST
Sim1	0.08 ± 0.020	0.08 ± 0.020	0.060 ± 0.030
Sim2	0.12 ± 0.120	0.13 ± 0.120	0.100 ± 0.150
Sim3	0.05 ± 0.002	0.05 ± 0.002	0.004 ± 0.005
Fried*	0.38 ± 0.004	0.32 ± 0.003	0.270 ± 0.005
Fried**	0.29 ± 0.003	0.25 ± 0.002	-

Table 26: The size (number of nodes) estimated with a 10-fold cross-validation on simulated problems with independent modeling of each target attribute.

Dataset	FIRT	FIMT	CLUS-ST
Sim1	867.07 ± 21.99	867.07 ± 21.99	133.72 ± 10.41
Sim2	878.72 ± 26.60	878.72 ± 26.60	135.48 ± 20.79
Sim3	1379.76 ± 17.87	1379.76 ± 17.87	2779.72 ± 52.00
Fried*	1038.00 ± 5.93	1038.00 ± 5.93	21171.60 ± 139.22
Fried**	9597.00 ± 6.00	9597.00 ± 6.00	-

the targets have standard deviations in the same range, the accuracy of the predictions is significantly better. The Fried dataset does not have correlated targets. The results have confirmed that when there is no correlation among the targets, there is nothing special to be gained by joint modeling. Both independent or joint modeling resulted in models with equal accuracy and approximately equal complexity. This was also observed for the Sim2 dataset, although some gains in terms of the size of the trees were achieved with the joint modeling approach.

With the Fried dataset we aimed to examine whether the capacity of learning can be improved by allowing the incremental learner to observe a large enough training sample, which would enable it to induce complex enough models. We have generated a stream of several millions of examples which was fed to the incremental learners. The results given in the Tables 25 and 27 were obtained after processing approximately three million examples. The model trees managed to achieve an error of 0.25 which is less than the error of the model induced with the batch learner on the smaller dataset drawn from the same distribution, measured to be 0.27. At the same time, the model induced by FIMT-MT is of a size which is smaller than the model induced with CLUS-MT by an order of magnitude. The regression tree was also able to achieve an improved accuracy reaching an error of 0.29 which is a little bit higher than the error of the model induced with the batch algorithm CLUS-MT.

Table 27: The root relative mean squared error (RRMSE) estimated with a 10-fold cross-validation on simulated problems with joint modeling.

Dataset	FIRT-MT	FIMT-MT	CLUS-MT
Sim1	0.08 ± 0.020	0.08 ± 0.020	0.060 ± 0.030
Sim2	0.13 ± 0.120	0.13 ± 0.120	0.100 ± 0.150
Sim3	0.05 ± 0.001	0.05 ± 0.001	0.004 ± 0.005
Fried*	0.39 ± 0.003	0.32 ± 0.003	0.270 ± 0.005
Fried**	0.29 ± 0.003	0.25 ± 0.002	-

Table 28: The size (number of nodes) estimated with a 10-fold cross-validation on simulated problems with joint modeling.

Dataset	FIRT-MT	FIMT-MT	CLUS-MT
Sim1	241.00 ± 0.00	241.00 ± 0.00	72.20 ± 0.00
Sim2	236.00 ± 0.00	236.00 ± 0.00	100.00 ± 0.00
Sim3	1362.19 ± 0.00	1362.19 ± 0.00	2672.00 ± 0.00
Fried*	1039.60 ± 0.00	1039.60 ± 0.00	21143.20 ± 0.00
Fried**	9595.00 ± 0.00	9595.00 ± 0.00	-

In addition, as discussed previously the difference in the size of the models is substantial. This experiment has shown that for modeling complex relationships, the incremental learner can give better results both in terms of accuracy and size. We believe that this is achieved mostly due to the fact that it successfully avoids over-fitting by using probabilistic estimates which guard against overly "optimistic" splits.

The Fried dataset enables us to further examine whether linear model trees can have better accuracy when there is a nonlinear dependence between the target attributes and the descriptive attributes. For this dataset, it is known that the regression surface for each target attribute is smooth. The model tree achieves better accuracy as compared to the regression tree, regardless of which modeling approach was used (individual or joint modeling). The results confirm that online training of perceptrons in the leaves results in a model for which the crisp regions of the d-dimensional histogram are smoothed. This model is thus a better approximation to the true function under study.

9.4 Further Extensions

Statistical learning for structured output spaces represents today an important subfield of machine learning. With the emergence of new types of data, for which there is a demand of online monitoring and analysis of the complex relationships that exist in the output space, a new type of theoretical and practical challenges is now appearing. These include the prediction of multiple diverse target attributes with complex relationships.

With our study and results we have stumbled upon multiple interesting questions which arise within the problem of modeling multiple numerical targets. Among them, we would like to mention the problem of tracking the evolution of the correlations among the target attributes over time and its effect on the model tree. For example, one might imagine a situation in which the correlations have changed only under a specific set of conditions.

Having an online approach to the problem of learning from time-changing data streams, such that the change detection mechanisms are embedded within the learning algorithm, presents an interesting opportunity to study dynamic relationships in the (structured) output space. In particular, an interesting direction for extensions is to monitor the evolution of the error per target attribute and in every subregion of the input space according to the partitioning represented with the model tree. A simple approach would be to back propagate the set of absolute differences $\{\Delta_1, ...\Delta_T\}$ up to the root node for a bottom-up change detection method.

If a change is detected such that the predictions for a particular target or a group of targets have diverged significantly from the true values, one might opt for re-grouping and decomposing the model only locally. This can be useful for explaining the dynamics of temporal and local correlations among the target attributes. To the best of our knowledge, there exist no such work (or solution) in the batch or in the online learning setup.

Option trees and tree-ensembles for multi-target regression represent another interesting

line for extensions. Due to their ability to improve the inductive bias of model trees, option trees enable a more thorough exploration of the space of hypotheses. We anticipate that the averaging of multiple predictions can improve the accuracy and the robustness of the model. However, we would like to note that this approach will result in significantly increased requirements for memory and computational complexity due to combination of multiple options and a complex target space. On the other hand, if one is able to use a parallelization of the learning process, ensembles of predictive clustering trees might represent an interesting possibility to enhance the online predictive clustering methodology, without having to suffer an increase of the processing time per example.

9.5 Summary

Multi-target decision and regression trees represent an interesting class of tree-based algorithms, for their ability to provide predictions for multiple target variables by means of a single descriptive model. Using the predictive clustering methodology, we have designed and implemented an online multi-target model tree learning algorithm for regression named FIMT-MT. The FIMT-MT algorithm extends the algorithm FIMT-DD by incorporating a similarity measure over the space of the target attributes. In this work we have considered only numerical targets, although the same ideas would apply for the case of mixed types of targets (nominal and numerical), assuming that an appropriate similarity measure can be defined.

The experimental evaluation includes a comparison with a batch learning algorithm, over two real-world datasets and four simulated problems. In addition, we performed a comparison with an approach to decompose the multi-target prediction task into several single-target prediction tasks. The obtained experimental results have shown that, joint modeling is not expected to improve the accuracy as compared to the decomposition approach. What is expected to be gained is lower complexity, that is, models which are smaller in size and consequently more interpretable. On the simulated Fried dataset the online algorithm was shown to achieve better accuracy over the batch algorithm, producing models that are smaller by an order of magnitude (provided that enough data are available for training).

For better understanding of their advantages and disadvantages we intend to perform an extensive experimental evaluation using larger real-world multi-target datasets, in a truly streaming setup. To the best of our knowledge, this is the first attempt to address the task of multi-target regression in the online learning setup.

10 Conclusions

In this thesis, we proposed and empirically evaluated a range of algorithms for online regression from streaming data. The algorithms learn different types of tree-based models for regression. These include regression trees, linear model trees, option trees for regression, multi-target model trees, as well as ensembles of regression and model trees.

All algorithms proposed in this thesis make use of the general-to-specific ordering of possible hypotheses to perform more efficient online learning. This enables online exploration of the search space, through sequential refinements of the existing hypothesis. Each refinement is viewed as a selection problem, where one is basically interested in comparing one hypothesis over another, in terms of their expected performance over a distribution of instances. The goal is naturally to select the hypothesis with the highest expected performance.

When data are in abundance, the selection problem is reduced to evaluating a set of possible hypotheses with the aim to provide an answer to the question: *"How likely is that the estimated advantage of one hypothesis over another will remain truthful if more training examples were used?"* As noted in previous chapters, bounding the probability of failure in finding a hypothesis, which is approximately close to the best one in the set of all possible hypotheses, can be achieved by bounding the probability of failure in selecting the best refinement in the set of all possible refinements over the current hypothesis. Given a single reasonable selection assertion, the general agreement is that there is no single optimal method for ensuring this. The choice is predetermined with the set of trade-offs that a given technique has to offer. In that context, the Hoeffding bound has been successfully applied to various learning tasks and is used in all of the algorithms proposed in this thesis. Compared to other methods, such as the sequential probability ratio test or other distribution-specific statistical techniques, it offers the best quality-performance trade-off for learning from fast data feeds.

By combining the Hoeffding bound with the general-to-specific search in the space of possible hypotheses, the induction process is designed as *a sequence of probably approximately correct* inductive decisions. Each inductive decision of the learner will have a small probability of failure (bounded with a user-defined parameter) in estimating the advantage of the selected refinement over the rest: the absolute error value decreases as more instances are being used to support the decision. The algorithms designed in this thesis are thus successfully converging as close as possible to the best hypothesis. At any moment from the start of learning they are ready to output a current hypothesis or a prediction for a sequence of testing instances.

In the context of successful exploration of the search space, we proposed a technique based on introducing options in the hierarchy of splits. The intuition is that the selection decisions that govern the splits near the root of the tree have access to little or no information on the goodness (or the optimality) of the final model that would be reached if one of several alternative splits with similar estimated merits is chosen. A set of option splits, on the other hand, provides means for interactive exploration of multiple models, and backtracking to the best split after a sequence of selection decisions. Option trees thus allow us to combine multiple predictions and leverage the "knowledge" of multiple "experts" in a fashion similar to ensembles of multiple regression trees. Besides being able to successfully converge to

the best possible hypothesis, under the assumption that enough data have been provided for learning, the algorithms are also able to take into account possible changes in the data distribution or the target function, and adapt their current hypotheses accordingly.

In the remainder of this chapter, we summarize the original scientific contributions of the thesis. We then discuss how the proposed methods can be further extended and improved.

10.1 Original Contributions

The four main original contributions of this thesis are summarized as follows. The research presented in this thesis spans over three interesting areas of the field of machine learning, learning from data streams, regression trees and their variants, and ensemble learning methods. Each of these areas is covered in the thesis with an overview and a critical review of the important ideas that are relevant for the research we have conducted. We have further provided an extensive survey of the existing methods for learning decision and regression trees, option trees and multi-target trees, as well as methods for learning tree-based ensembles, both in the batch and in the online learning setup. Critical review of each particular topic has given us the necessary insights in the state-of-the-art methodology for the specific problem and the possibilities for advancements, which we have exploited and represent our original contributions to science.

Design, implementation and evaluation of an algorithm for online learning of model trees from non-stationary data distributions. The first main contribution of this thesis is the study of methods that enable learning under non-stationary data distributions. We presented the algorithm FIMT-DD for learning model trees from time-changing data streams. To the best of our knowledge, FIMT-DD is the first algorithm for learning model trees from time-changing data streams which employs explicit drift detection. The algorithm performs within the framework of sequential inductive learning, and uses a statistically sound sampling strategy, based on the Hoeffding bound, to support each inductive decision. This enables very fast and efficient learning, without having to store any of the training instances. The induced model tree is available for use at any time in the course of learning, while offering excellent processing and prediction time per example.

In terms of accuracy, FIMT-DD is competitive with batch learning algorithms even for medium sized datasets, while having a lower variance component of the error. It effectively maintains an up-to-date model, even in the presence of different types of concept drift. The most valuable characteristic of its drift management mechanism is the local change detection and adaptation. Besides the advantage of being able to avoid unnecessary costs for re-growing the whole model, local changes provide additional information on the set of conditions under which the current hypothesis can be regarded as invalid. The computational complexity of FIMT-DD is lower than the existing algorithms for online learning of model trees due to the linear complexity of the algorithm for online training of perceptrons and the linear complexity of the change detection method. The most computationally complex procedure of FIMT-DD is the split evaluation which is performed periodically on user-predefined time intervals.

Design, implementation and evaluation of an algorithm for online learning of model trees with options. The second main contribution of this thesis is the study of methods for enhancing the online tree learning process and improving its generalization performance without increasing computational complexity. Due to its ability to make valid selection and stopping decisions based on probabilistic estimates, the FIMT-DD algorithm enables the analysis of large quantities of data that go beyond the processing capabilities of batch learning algorithms. As discussed in previous chapters, the selection decisions made in the sequential inductive process are based on an estimation of the advantage of one candidate refinement of the current hypothesis over another. The specifics of the inductive inference problem require a large enough sample for sufficient statistical support. The required sample

size, on the other hand, depends on the difficulty of assessing this advantage and in the extreme case of equally good refinements might grow to infinity. Our contribution lies in using option splits (or options) as a natural and effective way to solve this problem and improve the learning progress. At the same time, options can improve the overall accuracy and stability, as well as reduce the ambiguity in the interpretation of the model. This comes as a consequence of improving the simple hill-climbing search in the space of possible hypotheses, which is known for its inability to avoid local minima.

We have proposed and implemented an online learning algorithm for regression trees with options named ORTO, which successfully resolves ambiguous selection decisions and improves the exploration of the space of possible hypotheses. Our experimental evaluation conducted over a set of 10 real-world benchmark datasets has empirically confirmed our hypothesis: option trees learned online were shown to reduce not only the variance in the predictions, but also the inductive bias (both were estimated when learning over different samples of the training data). The existence of option splits in the tree hierarchy pinpoints the ambiguity of a set of conditions, and thus alerts to the existence of several equally relevant models.

A very interesting result was obtained with the utilization of options to backtrack past selection decisions and search for the best single tree within the option tree. Our empirical comparison has shown that the best tree found within the option tree has a better accuracy (on most of the problems) than the single tree learned by FIMT-DD. The increased predictive performance and stability comes at the cost of a small increase of the processing time per example and a controllable increase in the allocation of memory. The increase in the computational complexity is due to the increased number of internal nodes being evaluated at any given point in time. The option tree has an additional increase in the computational complexity when computing the aggregate of multiple predictions for a single testing example due to the need to examine all of the options on the path from the root to the corresponding leaf node.

Design, implementation and evaluation of algorithms for online learning of tree-based ensembles for regression. In this thesis we have proposed and implemented two different methods for online learning of tree-based ensembles for regression from data streams, which constitutes our next contribution. While our main interest is in learning descriptive interpretable models, for completeness we have performed also an extensive experimental comparison of single model trees induced with the FIMT-DD algorithm, option trees with two different prediction rules induced with the ORTO algorithm, and ensembles of model trees. The later were induced by online bagging algorithm and consisted of model trees learned with the original FIMT-DD algorithm and a randomized version named R-FIMT-DD. This study enabled us to explore the idea of randomizing the learning process through diversification of the input space and the search trajectory and examine the validity of the statistical reasoning behind the idea for aggregating multiple predictions. It is expected that this would bring the resulting model closer to the optimal or *best* hypothesis, instead of relying only on the success of a greedy search strategy in a constrained hypothesis space. It was also interesting to perform a comparison with respect to the enhancements that an option tree brings to the learning process.

Using a set of diverse analytical tools, we performed a systematic evaluation and comparison of the mentioned approaches on a wide range of real-world and artificial datasets. The quality of the induced ensembles was assessed through comparisons with option trees and standard model trees induced on the same original samples of training data. The results from the empirical evaluation show that online bagging of incrementally induced model trees improves the accuracy of the base models by reducing the variability of their predictions. On the other hand, online random forests of incrementally induced model trees improve the accuracy of the base models by improving the diversity of the ensemble. Finally, online option trees with linear models in the leaves are a compromise between the efficiency

and interpretability of a single model tree and the improved accuracy and stability of an ensemble of multiple online model trees. With respect to the computational complexity, if run in parallel, the ensemble of regression or model trees would have similar complexity as compared to the FIMT-DD algorithm.

By using the bias-variance-covariance decomposition of the error, we investigated and presented an approach to measuring the diversity of an ensemble of models. Our experiments confirmed that randomization reduces the bias component and improves the ability to model the phenomenon under study. However, randomization introduces higher variability of the predictions, which can in the end cancel-out the positive effect introduced by decreasing the correlation. We also observed that larger trees tend to give more correlated predictions, which decreases the overall accuracy of the ensemble. Although we have not performed the same analysis for online option trees due to the lack of multiple predictions for every testing example, we anticipate that the type of diversification that option nodes enable might be more beneficial than simple randomization.

To this date and to the best of our knowledge there is no other work that implements and empirically evaluates online ensemble methods for regression. In that light, we found that the currently best method for online regression is online learning of option trees with averaging. Our results brought us to a set of conclusions which can be summarized as follows: 1) The option trees offer the best accuracy for most of the datasets, having time and memory requirements somewhere in the middle between those of a single model and an ensemble; 2) The bias-variance analysis of the error confirmed that the option nodes indeed provide a better way to explore the search space and find the best tree possible; 3) Averaging of multiple predictions is a strategy to reduce the variance and improve the final accuracy.

Design, implementation and evaluation of an algorithm for online learning of multi-target model trees. The last contribution of this thesis is the study of the possibilities to extend the proposed algorithms towards predicting structured outputs, in particular towards predicting multiple numerical targets. Multi-target decision and regression trees represent an interesting extension of the tree-structured paradigm to a multiple response setting, being able to provide predictions for multiple target variables by means of a single descriptive model. Using the predictive clustering methodology, we have designed and implemented an online model tree learning algorithm for simultaneous modeling of multiple numerical targets named FIMT-MT. The FIMT-MT algorithm extends the algorithm FIMT-DD by incorporating a similarity measure over the space of the target attributes in each selection decision. In the design, we have chosen a combination of a prototype and a distance function that results in minimal weighted intra-cluster variance which can be computed on the fly. This enables a straightforward application of the Hoeffding bound that preserves the online nature of the inductive inference process. To the best of our knowledge, this is the first attempt to address the task of multi-target regression in the online learning setup.

We performed an experimental evaluation and a comparison with a batch learning algorithm over two real-world datasets and four simulated problems. In addition, we performed a comparison with an approach in which the problem of predicting multiple targets is decomposed into several single-target prediction tasks. This study gives insights on the advantages and disadvantages of joint modeling with a single descriptive model. The obtained experimental results have shown that, while there are no gains in improving the accuracy if joint modeling is chosen over the problem decomposition approach, the induced models are smaller in size and thus more interpretable. On the simulated problems, the online algorithm was shown to achieve better accuracy over the batch algorithm, producing models that are smaller by an order of magnitude (provided that enough data are available for training).

The multi-target model trees are inherently characterized with an increased computational complexity and increased memory allocation. In particular, the computational com-

plexity of FIMT-MT is a multiple of the computational complexity of FIMT-DD, which is dependent on the number of target attributes represented with a single model. This applies to the learning procedure and the procedure for computing predictions.

10.2 Further Work

Let us conclude with some guidelines for further work. The thesis provides an extensive study and evaluation of several different methodologies for learning. Among them, the least explored ones are the methodology for online learning of ensembles of regression and model trees, and the methodology for online learning of multi-target decision and regression trees. In that context, we can determine two main directions for further work.

The first direction is towards exploring the possibilities to enhance the existing online ensemble learning methods. In this thesis, we have studied online bagging of standard and randomized model trees. The results from this study have showed that increasing the diversity can improve the accuracy of the ensemble, but it can also increase its variance. The diversity-variance trade-off opens several research questions, among which it seems crucial to determine a method to control the amount of diversity in the ensemble in order to achieve the best accuracy over time. Using the bias-variance-covariance decomposition of the error in order to control the amount of variance and covariance among the ensemble members might be a promising direction to start. This idea will be investigated further in our future work.

The second direction is towards the development of an efficient methodology for online multi-target learning. In the research presented in this thesis, we have considered only numerical targets, although the same ideas could be possibly applied to the problem of modeling a mixture of nominal and numerical targets, assuming that an appropriate similarity measure can be defined. We believe that defining computationally simple similarity measures over a structured output space will enable interesting applications of the online inductive inference framework studied in this thesis. It would be also beneficial to study online ensembles for multi-target predictions. Ensembles of predictive clustering trees have already been shown to provide good predictive accuracy in batch learning (Kocev et al., 2007b), thus the same can be expected in the online learning setup.

A third direction for further work is to address the problem of tracking changes in the correlation matrix of the target attributes. This presents an interesting opportunity to study dynamic relationships in the structured output space. In particular, an interesting idea which can be easily explored is to monitor the evolution of the error per target attribute in every subregion of the instance space. To the best of our knowledge, this is a problem which has not been studied previously.

We note that further experimental evaluation on large datasets and in truly streaming settings is intended for all the developed approaches. In that context, we would like to note that an extended evaluation in a realistic streaming setup of the algorithm for online learning of multi-target regression trees is especially needed.

Finally, the last direction for future work is on algorithms for learning relational decision and regression trees in a streaming setting. The task of online learning of relational decision and regression trees is very challenging. However, it promises many interesting research questions and a possibility to leverage the generalization power of tree-based models. Our idea is to develop a more general approach that would enable extensions of the algorithms to the tasks of multi-target and structured output prediction.

11 Acknowledgements

I cannot be thankful enough to my father, my sister, my aunt Tanja and the closest family for the endless support and love they have given me all of my life, but especially in the difficult times during my doctoral studies. I would have not made it to the end without their love and support.

I am thankful for the patience and the wisdom of my advisors, Prof. Dr. Sašo Džeroski and Prof. Dr. João Gama. Being an advisor is not an easy job. I appreciate the time and the effort they invested in this collaboration and our joint work. Their ideas and suggestions have guided my work and improved the quality of my research. I would also like to thank the members of my PhD committee for providing comments and useful feedback on the thesis.

My work and research was directly supported by several funding bodies. I wish to thank all of them for the financial support. These include the Slovenian Research Agency (young researcher grant number PR-03218), the European Commission (grant ICT-2010-266722 SUMO: Supermodeling by Combining Imperfect Models), and the grant "Knowledge Discovery from Ubiquitous Data Streams" (PTDC/EIA/098355/2008).

I would like to thank the colleagues at the Department of Knowledge Technologies for the pleasant working environment, but especially Mili Bauer, who helped me solve many of the administrative problems and made me feel less of a foreigner. I would also like to thank my office mates Bernard Ženko and Martin Žnidarsič for the many inspirational moments during the period of my doctoral studies.

A very special and warm thanks goes to my dear colleagues and friends, Dragi Kocev, Ivica Slavkov, Aleksandra Raškovska, Panče Panov and Daniela Stojanova. Thank you for the many beautiful moments we shared together during our PhD studies.

During the doctoral studies I had the pleasure to visit several research groups and meet many friendly professionals that enriched my experience as a researcher. I would like to thank the people in the LIAAD-INESC Porto L.A. for the pleasant discussions along the best coffee in the world. The several research visits to this lovely group will always bring warm memories. I thank the colleagues from the robotics laboratory (SwarmLab) at the Department of Knowledge Engineering (DKE), Maastricht University and from the machine learning (ML) group at the Catholic University of Leuven for the nice working environment.

I would like to thank the colleagues at the Database group of the Baskin's School of Engineering, University of California Santa Cruz, for giving me the opportunity to enrich my professional experience in a friendly but professional environment. I am specially thankful to Prof. Dr. Neoklis Polyzotis for the truly enjoyable joint research work and the friendly support during my research visit.

Last, but not least, I would like to thank, from the bottom of my heart, my remote advisor, friend and "father" Prof. Dr. Phokion Kolaitis, for the generous support, help and guidance he has given me during the last year of my doctoral studies. I think that I would have not been where I am today if I haven't met him.

This thesis is devoted to the memory of my beloved mother. She was my source of strength and taught me how to endure even in the most difficult moments. In many ways, I believe, I am her personal reflection. I can only hope to one day be the generous and beautiful human being she was.

12 References

Abdulsalam, H.; Skillicorn, D. B.; Martin, P. Streaming random forests. In: *Proceedings of 11th International Database Engineering and Applications Symposium*, 225–232 (IEEE Computer Society, Washington, DC, 2007).

Abdulsalam, H.; Skillicorn, D. B.; Martin, P. Classifying evolving data streams using dynamic streaming random forests. In: *Proceedings of 19th International Conference on Database and Expert Systems Applications*, 643–651 (Springer, Berlin, 2008).

Abramowitz, M.; Stegun, I. A. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* (Dover Books on Mathematics, Dover, New York, 1972).

Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, 1975).

Alippi, C.; Boracchi, G.; Roveri, M. Just in time classifiers: Managing the slow drift case. In: *Proceedings of 19th International Joint Conference on Neural Networks*, 114–120 (IEEE Press, Piscataway, NJ, 2009).

Alippi, C.; Roveri, M. Just-in-time adaptive classifiers - Part I: Detecting nonstationary changes. *IEEE Transactions on Neural Networks* **19**, 1145–1153 (2008).

Alon, N.; Gibbons, P. B.; Matias, Y.; Szegedy, M. Tracking join and self-join sizes in limited storage. In: *Proceedings of 18th ACM Symposium on Principles of Database Systems*, 10–20 (ACM, New York, 1999).

Amit, Y.; Geman, D. Shape quantization and recognition with randomized trees. *Neural Computation* **9**, 1545–1588 (1997).

Appice, A.; Džeroski, S. Stepwise induction of multi-target model trees. In: *Proceedings of 18th European conference on Machine Learning*, 502–509 (Springer, Berlin, 2007).

Armstrong, J. S.; Collopy, F. Error Measures for Generalizing about Forecasting Methods: Empirical Comparisons. *International Journal of Forecasting* **8**, 69–80 (1992).

Assche, A. V.; Blockeel, H. Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In: *Proceedings of 18th European conference on Machine Learning*, 418–429 (Springer, Berlin, 2007).

Atkeson, C. G.; Moore, A. W.; Schaal, S. Locally weighted learning for control. *Artificial Intelligence Review* **11**, 75–113 (1997).

Auer, P.; Warmuth, M. K. Tracking the best disjunction. *Machine Learning* **32**, 127–150 (1998).

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; Widom, J. Models and issues in data stream systems. In: *Proceedings of 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 1–16 (ACM, New York, 2002).

Babu, S.; Widom, J. Continuous queries over data streams. *ACM SIGMOD Record* **30**, 109–120 (2001).

Bacardit, J.; Krasnogor, N. The Infobiotics PSP benchmarks repository. http://www.infobiotics.org/PSPbenchmarks (Accessed: July, 2012)

Baena-García, M.; del Campo-Ávila, J.; Fidalgo, R.; Bifet, A.; Gavaldà, R.; Morales-Bueno, R. Early drift detection method. In: *Proceedings of 4th International Workshop on Knowledge Discovery from Data Streams*, 77–86 (AAAI Press, Palo Alto, CA, 2006).

Barbará, D.; DuMouchel, W.; Faloutsos, C.; Haas, P. J.; Hellerstein, J. M.; Ioannidis, Y. E.; Jagadish, H. V.; Johnson, T.; Ng, R. T.; Poosala, V.; Ross, K. A.; Sevcik, K. C. The New Jersey data reduction report. *IEEE Data Engineering Bulletin* **20**, 3–45 (1997).

Barutcuoglu, Z.; Schapire, R. E.; Troyanskaya, O. G. Hierarchical multi-label prediction of gene function. *Bioinformatics* **22**, 830–836 (2006).

Basseville, M.; Nikiforov, I. V. *Detection of Abrupt Changes: Theory and Application* (Prentice-Hall, Upper Saddle River, NJ, 1993).

Bechhofer, R. E. A single-sample multiple decision procedure for ranking means of normal populations with known variances. *The Annals of Mathematical Statistics* **25**, 16–39 (1954).

Bhuvanagiri, L.; Ganguly, S. Estimating entropy over data streams. *Lecture Notes in Computer Science* **4168**, 148–159 (2006).

Bifet, A.; Frank, E.; Holmes, G.; Pfahringer, B. Ensembles of restricted Hoeffding trees. *ACM Transactions on Intelligent Systems and Technology* **3**, 30:1–30:20 (2012).

Bifet, A.; Gavaldà, R. Learning from time-changing data with adaptive windowing. In: *Proceedings of 7th International Conference on Data Mining* (Omnipress, Los Angeles, 2007).

Bifet, A.; Gavaldà, R. Mining adaptively frequent closed unlabeled rooted trees in data streams. In: *Proceedings of 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 34–42 (ACM, New York, 2008).

Bifet, A.; Holmes, G.; Pfahringer, B. Leveraging bagging for evolving data streams. *Lecture Notes in Computer Science* **6321**, 135–150 (2010).

Bifet, A.; Holmes, G.; Pfahringer, B.; Gavaldà, R. Improving adaptive bagging methods for evolving data streams. In: *Proceedings of 1st Asian Conference on Machine Learning*, 23–37 (Springer, Berlin, 2009).

Bifet, A.; Holmes, G.; Pfahringer, B.; Kirkby, R.; Gavaldà, R. New ensemble methods for evolving data streams. In: *Proceedings of 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 139–148 (ACM, New York, 2009).

Blockeel, H.; De Raedt, L. Top-down induction of first-order logical decision trees. *Artificial Intelligence* **101**, 285–297 (1998).

Blockeel, H.; De Raedt, L.; Jacobs, N.; Demoen, B. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery* **3**, 59–93 (1999).

Blockeel, H.; Dehaspe, L.; Demoen, B.; Janssens, G.; Ramon, J.; Vandecasteele, H. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research* **16**, 135–166 (2002).

Blockeel, H.; Raedt, L. D.; Ramong, J. Top-down induction of clustering trees. In: *Proceedings of 15th International Conference on Machine Learning*, 55–63 (Morgan Kaufmann, San Francisco, CA, 1998).

Blockeel, H.; Struyf, J. Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Reseach* **3**, 621–650 (2003).

Blum, A.; Burch, C. On-line learning and the metrical task system problem. *Machine Learning* **39**, 35–58 (2000).

Bouckaert, R. R. Practical bias variance decomposition. In: *Proceedings of 21st Australasian Joint Conference on Artificial Intelligence*, 247–257 (Springer, Berlin, 2008).

Breiman, L. Bagging predictors. *Machine Learning* **24**, 123–140 (1996a).

Breiman, L. Stacked regressions. *Machine Learning* **24**, 49–64 (1996b).

Breiman, L. Arcing classifiers. *Annals of Statistics* **26**, 801–823 (1998).

Breiman, L. Randomizing outputs to increase prediction accuracy. *Machine Learning* **40**, 229–242 (2000).

Breiman, L. Random forests. *Machine Learning* **45**, 5–32 (2001).

Breiman, L.; Friedman, J.; Olshen, R.; Stone, C. *Classification and Regression Trees* (Wadsworth and Brooks, Monterey, CA, 1984).

Brown, G.; Wyatt, J.; Harris, R.; Yao, X. Diversity creation methods: A survey and categorization. *Journal of Information Fusion* **6**, 5–20 (2005a).

Brown, G.; Wyatt, J.; Sun, P. Between two extremes: Examining decompositions of the ensemble objective function. In: *Proceedings of 6th International Workshop on Multiple Classifier Systems*, 296–305 (Springer, Berlin, 2005b).

Brown, G.; Wyatt, J. L.; Tiňo, P. Managing diversity in regression ensembles. *Journal of Machine Learning Research* **6**, 1621–1650 (2005c).

Buntine, W. Learning classification trees. *Statistics and Computing* **2**, 63–73 (Chapman and Hall, London, 1992).

Castillo, G.; Gama, J. Bias management of Bayesian network classifiers. In: *Proceedings of 8th International Conference on Discovery Science*, 70–83 (Springer, Berlin, 2005).

Ceci, M.; Appice, A.; Loglisci, C.; Caruso, C.; Fumarola, F.; Valente, C.; Malerba, D. Relational frequent patterns mining for novelty detection from data streams. In: *Proceedings of 6th International Conference on Machine Learning and Data Mining in Pattern Recognition*, 427–439 (Springer, Berlin, 2009).

Chan, P. K.; Stolfo, S. J. Experiments on multistrategy learning by metalearning. In: *Proceedings of 2nd International Conference on Information and Knowledge Management*, 314–323 (ACM, New York, 1993).

Chapman, D.; Kaelbling, L. P. Input generalization in delayed reinforcement learning: an algorithm and performance comparisons. In: *Proceedings of 12th International Joint Conference on Artificial Intelligence* **2**, 726–731 (Morgan Kaufmann, San Francisco, CA, 1991).

Chaudhry, N.; Shaw, K.; Abdelguerfi, M. *Stream Data Management* (Springer, New York, 2005).

Chaudhuri, P.; Ching Huang, M.; Yin Loh, W.; Yao, R. Piecewise-polynomial regression trees. *Statistica Sinica* **4**, 143–167 (1994).

Chen, C.; Yan, X.; Zhu, F.; Han, J.; Yu, P. S. Graph OLAP: A multi-dimensional framework for graph data analysis. *Journal of Knowledge and Information Systems* **21**, 41–63 (2009).

Chen, Y.; Dong, G.; Han, J.; Wah, B. W.; Wang, J. Multi-dimensional regression analysis of time-series data streams. In: *Proceedings of 28th International Conference on Very Large Data Bases*, 323–334 (Morgan Kaufmann, San Francisco, CA, 2002).

Chow, G. Tests of equality between sets of coefficients in two linear regressions. *Econometrica* **28**, 591–605 (1960).

Chu, F.; Zaniolo, C. Fast and light boosting for adaptive mining of data streams. In: *Proceedings of 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 282–292 (Springer, Berlin, 2004).

Clare, A.; King R. D. Knowledge Discovery in Multi-label Phenotype Data. In: *Proceedings of 5th European Conference on Principles and Practice of Data Mining and Knowledge Discovery*, 42–53 (Springer, Berlin, 2001).

Cormode, G.; Garofalakis, M. Approximate continuous querying over distributed streams. *ACM Transactions Database Systems* **33**, 9.1–9.39 (2008).

Cormode, G.; Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *Algorithms* **55**, 58–75 (2005).

Crawford, S. L. Extensions to the CART algorithm. *International Journal of Man-Machine Studies* **31**, 197–217 (1989).

Dawid, P. Present position and potential developments: some personal views. Statistical theory: The prequential approach (with Discussion). *Journal of Royal Statistical Society A* **147**, 278–292 (1984).

De Raedt, L.; Džeroski, S. First order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence* **70**, 375–392 (1994).

Delve Repository. URL `http://www.cs.toronto.edu/~delve/data/datasets.html` (Accessed: July, 2012).

Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* **7**, 1–30 (2006).

Dietterich, T. G. Machine-learning research: Four current directions. *The AI Magazine* **18**, 97–136 (1998).

Dietterich, T. G. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning* **40**, 139–157 (2000).

Dobra, A.; Gehrke, J. Secret: a scalable linear regression tree algorithm. In: *Proceedings of 8th ACM SIGKDD International Conference on Knowledge discovery and data mining*, 481–487 (ACM, New York, 2002).

Domingos, P. Why Does Bagging Work? A Bayesian Account and its Implications In: *roceedings of 3rd International Conference on Knowledge Discovery and Data Mining*, 155–158 (AAAI Press, Palo Alto, CA, 1997).

Domingos, P. A unified bias-variance decomposition for zero-one and squared loss. In: *Proceedings of 17th National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 564–569 (AAAI Press, Palo Alto, 2000).

Domingos, P.; Hulten, G. Mining high-speed data streams. In: *Proceedings of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 71–80 (ACM, New York, 2000).

Dries, A.; De Raedt, L. Towards clausal discovery for stream mining. In: *Proceedings of 19th International Conference on Inductive Logic Programming*, 9–16 (Springer, Berlin, 2010).

Driessens, K.; Ramon, J. Relational instance based regression for relational reinforcement learning. In: *Proceedings of 20th International Conference on Machine Learning*, 123–130 (AAAI Press, Palo Alto, CA, 2003).

Driessens, K.; Ramon, J.; Blockeel, H. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In: *Proceedings of 12th European Conference on Machine Learning*, 97–108 (Springer, Berlin, 2001).

Džeroski, S.; De Raedt, L.; Driessens, K. Relational reinforcement learning. *Machine Learning* **43**, 7–52 (2001).

Dzeroski, S.; Kobler, A.; Gjorgjioski, V.; Panov, P. Using decision trees to predict forest stand heght and canopy cover from LANDSAT and LIDAR data. In: *Proceedings of 20th International Conference on Informatics for Environmental Protection*, 125-133 (Shaker Verlag, Aachen, 2006).

Džeroski, S.; Lavrač, N., editors *Relational Data Mining* (Springer, Berlin, 2001).

Earth System Research Laboratory. `http://www.esrl.noaa.gov/psd/` (Accessed: July, 2012).

Fan, W. Streamminer: a classifier ensemble-based engine to mine concept-drifting data streams. In: *Proceedings of 13th International Conference on Very Large Data Bases* **30**, 1257–1260 (Morgan Kaufmann, San Francisco, CA, 2004).

Fan, W.; Wang, H.; Yu, P. S.; Ma, S. Is random model better? On its accuracy and efficiency. In: *Proceedings of 3rd IEEE International Conference on Data Mining*, 51–58 (IEEE Computer Society, Washington, DC, 2003).

Fern, A.; Givan, R. Online ensemble learning: An empirical study. *Machine Learning* **53**, 71–109 (2003).

Fisher, D.; Hapanyengwi, G. Database management and analysis tools of machine induction. *Intelligent Information Systems* **2**, 5–38 (1993).

Frank, A.; Asuncion, A. UCI Machine Learning Repository (2010). URL `http://archive.ics.uci.edu/ml` (Accessed: July, 2012).

Freund, Y.; Schapire, R. E. Experiments with a new boosting algorithm. In: *Proceedings of 13th International Conference on Machine Learning*, 148–156 (Morgan Kaufmann, San Francisco, CA, 1996).

Freund, Y.; Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Computer and System Sciences* **55**, 119–139 (1997).

Friedman, J. H. Multivariate Adaptive Regression Splines, *The Annals of Statistics* **19**, 1–67 (1991).

Friedman, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association* **32**, 675–701 (1937).

Friedman, M. A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics* **11**, 86–92 (1940).

Fumarola, F.; Ciampi, A.; Appice, A.; Malerba, D. A sliding window algorithm for relational frequent patterns mining from data streams. In: *Proceedings of 12th International Conference on Discovery Science*, 385–392 (Springer, Berlin, 2009).

Gama, J. Probabilistic Linear Tree. In: *Proceedings of 14th International Conference on Machine Learning*, 134-142 (Morgan Kaufmann, San Francisco, CA, 1997)

Gama, J. Functional Trees. *Machine Learning* **55**, 219–250 (2004)

Gama, J. *Knowledge Discovery from Data Streams* (Chapman & Hall/CRC, Boca Raton, FL, 2010).

Gama, J.; Castillo, G. Learning with local drift detection. In: *Proceedings of 2nd International Conference on Advanced Data Mining and Applications*, 42–55 (Springer, Berlin, 2006).

Gama, J.; Medas, P. Learning decision trees from dynamic data streams. *Journal of Universal Computer Science* **11**, 1353–1366 (2005).

Gama, J.; Medas, P.; Castillo, G.; Rodrigues, P. Learning with drift detection. In: *17th Brazilian Symposium on Artificial Intelligence*, 286–295 (Springer, Berlin, 2004a).

Gama, J.; Medas, P.; Rocha, R. Forest trees for on-line data. In: *Proceedings of 19th ACM Symposium on Applied Computing*, 632–636 (ACM, New York, 2004b).

Gama, J.; Rocha, R.; Medas, P. Accurate decision trees for mining high-speed data streams. In: *Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 523–528 (ACM, New York, 2003).

Gama, J.; Sebastião, R.; Rodrigues, P. P. Issues in evaluation of stream learning algorithms. In: *Proceedings 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 329–338 (ACM, New York, 2009).

Ganguly, S.; Garofalakis, M.; Kumar, A.; Rastogi, R. Join-distinct aggregate estimation over update streams. In: *Proceedings of 24th ACM Symposium on Principles of Database Systems*, 259–270 (ACM, New York, 2005).

Gärtner, T.; Driessens, K.; Ramon, J. Graph kernels and Gaussian processes for relational reinforcement learning. In: *Proceedings of 13th International Conference on Inductive Logic Programming*, 146–163 (Springer, Berlin, 2003).

Gehrke, J.; Ganti, V.; Ramakrishnan, R.; ; Loh, W. Boat-optimistic decision tree construction. In: *Proceedings of 25th ACM-SIGMOD International Conference on Management of Data*, 169–180 (ACM, New York, 1999).

Gehrke, J.; Korn, F.; Srivastava, D. On computing correlated aggregates over continual data streams. *ACM SIGMOD Record* **30**, 13–24 (2001).

Gehrke, J.; Ramakrishnan, R.; Ganti, V. Rainforest: A framework for fast decision tree construction of large datasets. In: *Proceedings of 24th International Conference on Very Large Data Bases*, 416–427 (Morgan Kaufmann, San Francisco, CA, 1998).

Geurts, P. Dual Perturb and Combine Algorithm In: *Proceedings of 8th International Workshop on Artificial Intelligence and Statistics*, 196–201 (Society of Artificial Intelligence and Statistics, Key-West, FL, 2001).

Geurts, P.; Wehenkel, L.; d'Alché Buc, F. Kernelizing the output of tree-based methods. In: *Proceedings of 23rd International Conference on Machine learning*, 345–352 (ACM, New York, 2006).

Gibbons, P. B. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In: *Proceedings of 27th International Conference on Very Large Data Bases*, 541–550 (Morgan Kaufmann, San Francisco, CA, 2001).

Gilbert, A. C.; Kotidis, Y.; Muthukrishnan, S.; Strauss, M. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In: *Proceedings of 27th International Conference on Very Large Data Bases*, 79–88 (Morgan Kaufmann, San Francisco, CA, 2001).

Gillo, M. W. MAID, a Honeywell 600 program for an automatized survey analysis. *Behaviorial Science* **17**, 251–252 (1972).

Gillo, M. W.; Shelly, M. W. Predictive modeling of multi-variable and multivariate data. *Journal of the American Statistical Association* **69**, 646–653 (1974).

Gjorgjioski, V.; Džeroski, S.; White, M. *Clustering Analysis of Vegetation Data.* (Technical Report 10065, Jožef Stefan Institute, Ljubljana, 2003).

Goel, A.; Gupta, P. Small subset queries and bloom filters using ternary associative memories, with applications. *ACM SIGMETRICS Performance Evaluation Review* **38**, 143–154 (2010).

Golab, L.; Johnson, T.; Koudas, N.; Srivastava, D.; Toman, D. Optimizing away joins on data streams. In: *Proceedings of 2nd International Workshop on Scalable Stream Processing System*, 48–57 (ACM, New York, 2008).

Golab, L.; Özsu, M. T. Processing sliding window multi-joins in continuous queries over data streams. In: *Proceedings of 29th International Conference on Very Large Data Bases* **29**, 500–511 (Morgan Kaufmann, San Francisco, CA, 2003).

Ross, Gordon J., Adams, Niall M., Tasoulis, Dimitris K., Hand, David J. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters* **33** 191–198 (2012).

Gratch, J. *An Effective Method for Correlated Selection Problems.* (Technical report, University of Illinois, Champaign, IL, 1994).

Guha, S.; Koudas, N. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In: *Proceedings of 18th International Conference on Data Engineering*, 567–576 (IEEE Computer Society, Washington, DC, 2002).

Guo, H.; Viktor, H. L. Multirelational classification: a multiple view approach. *Knowledge and Information Systems* **17**, 287–312 (2008).

Hagquist, C.; Stenbeck, M. Goodness of fit in regression analysis : R2 and G2 reconsidered. *Quality and Quantity* **32**, 229–245 (1998).

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. The WEKA data mining software: An update. *SIGKDD Explorations* **11** (2009).

D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation* **100**, 78–150 (1992).

Hellerstein, J. M.; Haas, P. J.; Wang, H. J. Online aggregation. *ACM SIGMOD Record* **26**, 171–182 (1997).

Helmbold, D. P.; Long, D. D. E.; Sconyers, T. L.; Sherrod, B. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications* **5**, 285–297 (2000).

Herbster, M.; Warmuth, M. K. Tracking the best expert. *Machine Learning* **32**, 151–178 (1998).

Herbster, M.; Warmuth, M. K. Tracking the best linear predictor. *Journal of Machine Learning Research* **1**, 281–309 (2001).

Ho, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**, 832–844 (1998).

Hoeffding, W. Probability inequalities for sums of bounded random variables. *American Statistical Association* **58**, 13–30 (1963).

Holmes, G.; Kirkby, R.; Pfahringer, B. Stress-testing Hoeffding trees. In: *Proceedings of 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 495–502 (Springer, Berlin, 2005).

Hothorn, T.; Hornik, K.; Zeileis, A. Unbiased recursive partitioning: a conditional inference framework. *Journal of Computational and Graphical Statistics* **15**, 651–674 (2006).

Hou, W.; Yang, B.; Wu, C.; Zhou, Z. RedTrees: A relational decision tree algorithm in streams. *Expert Systems with Applications* **37**, 6265–6269 (2010).

Hu, X.-G.; Li, P.-p.; Wu, X.-D.; Wu, G.-Q. A semi-random multiple decision-tree algorithm for mining data streams. *Computer Science and Technology* **22**, 711–724 (2007).

Hulten, G.; Spencer, L.; Domingos, P. Mining time-changing data streams. In: *Proceedings of 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 97–106 (ACM, New York, 2001).

Hunt, E. B.; Marin, J.; Stone, P. J. *Experiments in Induction* (Academic Press, New York, 1966).

TomTom traffic prediction for intelligent GPS navigation. URL `http://tunedit.org/challenge/IEEE-ICDM-2010` (Accessed: July, 2012).

Ikonomovska, E.; Gama, J. Learning model trees from data streams. In: *Proceedings of 11th International Conference on Discovery Science*, 52–63 (Springer, Berlin, 2008).

Ikonomovska, E.; Gama, J.; Džeroski, S. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 128–168 (Springer, Berlin, 2011).

Ikonomovska, E.; Gama, J.; Sebastião, R.; Gjorgjevik, D. Regression trees from data streams with drift detection. In: *Proceedings of 12th International Conference on Discovery Science*, 121–135 (Springer, Berlin, 2009).

Ikonomovska, E.; Gama, J.; Ženko, B.; Džeroski, S. Speeding-up Hoeffding-based regression trees with options. In: *Proceedings of 28th International Conference on Machine Learning*, 537–544 (ACM, New York, 2011).

Iman, R. L.; Davenport, J. M. Approximations of the critical region of the Friedman statistic. *Communications in Statistics*, **A9** 571–595 (1980).

Indyk, P. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM* **53**, 307–323 (2006).

Kang, J.; Naughton, J.; Viglas, S. Evaluating window joins over unbounded streams. In: *Proceedings of 19th International Conference on Data Engineering*, 341–352 (2003).

Karalič, A. Employing linear regression in regression tree leaves. In: *Proceedings of 10th European Conference on Artificial Intelligence*, 440–441 (John Wiley & Sons, New York, 1992).

Kass, G. V. Signifcance testing in automatic interaction detection (a.i.d.). *Applied Statistics* **24**, 178–189 (1975).

Kass, G. V. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics* **29**, 119–127 (1980).

M. Kearns, R. Schapire, and L. Sellie. Toward eÂącient agnostic learning. *Machine Learning* **17**, 115–141 (1994).

Kifer, D.; Ben-David, S.; Gehrke, J. Detecting change in data streams. In: *Proceedings of 30th International Conference on Very Large Data Bases* **30**, 180–191 (Morgan Kaufmann, San Francisco, CA, 2004).

Kim, H.; Loh, W. Classification trees with unbiased multiway splits. *Journal of the American Statistical Association* **96**, 598–604 (2001).

Klinkenberg, R. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis* **8**, 281–300 (2004).

Klinkenberg, R.; Joachims, T. Detecting concept drift with support vector machines. In: *Proceedings of 17th International Conference on Machine Learning*, 487–494 (Morgan Kaufmann, San Francisco, CA, 2000).

Klinkenberg, R.; Renz, I. Adaptive Information Filtering: Learning in the Presence of Concept Drifts. In: *Workshop Notes on Learning for Text Categorization of 15th International Conference of Machine Learning*, 33–40 (AAAI Press, Palo Alto, CA, 1998).

Kocev, D.; Džeroski, S.; White, M. D.; Newell, G. R.; Griffioen, P. Using single and multi-target regression trees and ensembles to model a compound index of vegetation condition. *Ecological Modelling* **220**, 1159–1168 (2009).

Kocev, D.; Struyf, J.; Dzeroski, S. Beam search induction and similarity constraints for predictive clustering trees. In: *Proceedings of 5th International Workshop on Knowledge Discovery in Inductive Databases*, 134–151 (Springer, Berlin, 2007a).

Kocev, D.; Vens, C.; Struyf, J.; Džeroski, S. Ensembles of multi-objective decision trees. In: *Proceedings of 18th European conference on Machine Learning*, 624–631 (Springer, Berlin, 2007b).

Kohavi, R. Feature subset selection as search with probabilistic estimates. *AAAI Technical Report FS-94-02*, 122–126 (Stanford University, CA, 1994).

Kohavi, R. Scaling up the accuracy of naive-bayes classifiers: A decision tree hybrid. In: *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 202–207 (AAAI Press, Palo Alto, CA, 1996).

Kohavi, R.; Kunz, C. Option decision trees with majority votes. In: *Proceedings of 14th International Conference on Machine Learning*, 161–169 (Morgan Kaufmann, San Francisco, CA, 1997).

Kolter, J. Z.; Maloof, M. A. Using additive expert ensembles to cope with concept drift. In: *Proceedings of 22nd International Conference on Machine Learning*, 449–456 (ACM, New York, 2005).

Kolter, J. Z.; Maloof, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research* **8**, 2755–2790 (2007).

Kong, E. B.; Dietterich, T. G. Error-correcting output coding corrects bias and variance. In: *Proceedings of 12th International Conference on Machine Learning*, 313–321 (Morgan Kaufmann, San Francisco, CA, 1995).

Kramer, S. Structural regression trees. In: *Proceedings of 13th National Conference on Artificial intelligence* 812–819 (AAAI Press, Palo Alto, CA, 1996).

Krogh, A.; Vedelsby, J. Neural network ensembles, cross validation, and active learning. In: *Proceedings of 8th International Conference on Advances in Neural Information Processing Systems*, 231–238 (MIT Press, Cambridge, MA, 1995).

Kuncheva, L. I.; Whitaker, C. J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning* **51**, 181–207 (2003).

Lämmel, R. Google's mapreduce programming model - revisited. *Scientific Computer Programming* **68**, 208–237 (2007).

Langley, P. *Elements of Machine Learning* (Morgan Kaufmann, San Francisco, CA, 1994).

Lehmann, E. L.; Casella, George *Theory of Point Estimation, 2nd edition* (Springer, New York, 1998).

Lewis-Beck, M. S. *Applied Regression : An Introduction* (Sage Publications, Beverly Hills, CA, 1980).

Li, P.; Wu, X.; Hu, X.; Liang, Q.; Gao, Y. A random decision tree ensemble for mining concept drifts from noisy data streams. *Applied Artificial Intelligence* **24**, 680–710 (2010).

Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning* **2**, 285–318 (1988).

Littlestone, N.; Warmuth, M. K. The weighted majority algorithm. *Information and Computation* **108**, 212–261 (1994).

Liu, J.; Li, X.; Zhong, W. Ambiguous decision trees for mining concept-drifting data streams. *Pattern Recognition Letters* **30**, 1347–1355 (2009).

Loh, W. Y. Regression Trees with Unbiased Variable Selection and Interaction Detection. *Statistica Sinica* **12**, 361–386 (2002).

Loh, W. Y.; Shih, Y. S. Split selection methods for classification trees. *Statistica Sinica* **7**, 815–840 (1997).

Loh, W. Y.; Vanichsetakul, N. Tree-structured classification via generalized discriminant analysis. *Journal of American Statistical Association* **83**, 715–728 (1988).

RuleQuest research. URL `http://www.rulequest.com/cubist-info.html` (Accessed: July, 2012).

Malerba, D.; Appice, A.; Ceci, M.; Monopoli, M. Trading-off local versus global effects of regression nodes in model trees. In: *Proceedings of 13th International Symposium on Foundations of Intelligent Systems*, 393–402 (Springer, Berlin, 2002).

Manku, G. S.; Motwani, R. Approximate frequency counts over data streams. In: *Proceedings of 28th International Conference on Very Large Data Bases*, 346–357 (Morgan Kaufmann, San Francisco, CA, 2002).

Margineantu, D. D.; Dietterich, T. G. Pruning adaptive boosting. In: *Proceedings of 14th International Conference on Machine Learning*, 211–218 (Morgan Kaufmann, San Francisco, CA, 1997).

Maron, O.; Moore, A. Hoeffding races: Accelerating model selection search for classification and function approximation. In: *Proceedings of 7th International Conference on Advances in Neural Information Processing Systems*, 59–66 (Morgan Kaufmann, San Francisco, CA, 1994).

Maron, O.; Moore, A. W. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review* **11**, 193–225 (1997).

Martínez-Muñoz, G.; Suárez, A. Pruning in ordered bagging ensembles. In: *Proceedings of 23rd International Conference on Machine Learning*, 609–616 (ACM, New York, 2006).

Mehta, M.; Agrawal, R.; Rissanen, J. SLIQ: A fast scalable classifier for data mining. In: *Proceedings of 5th International Conference on Extending Database Technology*, 18–32 (Springer, Berlin, 1996).

Messenger, R.; Mandell, L. A modal search technique for predictable nominal scale multivariate analysis. *Journal of the American Statistical Association* **67**, 768–772 (1972).

Mitchell, T. M. *Machine Learning* (McGraw-Hill, New York, 1997).

Montgomery, D. *Introduction to Statistical Quality Control* 4th edition (John Wiley, New York, 2001).

Moore, A.; Lee, M. S. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research* **8**, 67–91 (1998).

Moore, A. W.; Lee, M. S. Efficient algorithms for minimizing cross validation error. In: *Proceedings of 11th International Conference on Machine Learning*, 190–198 (Morgan Kaufmann, San Francisco, CA, 1994).

Morgan, J.; Messenger, R. *THAID, a Sequential Analysis Program for the Analysis of Nominal Scale Dependent Variables* (Survey Research Center, Institute for Social Research, University of Michigan, Ann Arbor, MI, 1973).

Morgan, J. N.; Sonquist, J. A. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association* **58**, 415–434 (1963).

Mouss, H.; Mouss, D.; Mouss, N.; Sefouhi, L. Test of Page-Hinkley, an approach for fault detection in an agro-alimentary production system. In: *Proceedings of 5th Asian Control Conference* **2**, 815–818 (Electronic resource, Melbourne, 2004).

Muhlbaier, M. D.; Polikar, R. An ensemble approach for incremental learning in non-stationary environments. In: *Proceedings of 7th International Conference on Multiple Classifier Systems*, 490–500 (Springer, Berlin, 2007).

Musick, R.; Catlett, J.; Russell, S. J. Decision theoretic subsampling for induction on large databases. In: *Proceedings of 10th International Conference on Machine Learning*, 212–219 (Morgan Kaufmann, San Francisco, CA, 1993).

Muthukrishnan, S. Data streams: algorithms and applications. *Foundations and Trends in Theoretical Computer Science* **1**, 117–236 (2005).

Nemenyi, P. B. *Distribution-free Multiple Comparisons.* (Princeton University, NJ, PhD Thesis, 1963).

Neville, J.; Jensen, D.; Friedland, L.; Hay, M. Learning relational probability trees. In: *Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 625–630 (ACM, New York, 2003).

Oza, N. C.; Russell, S. J. Experimental comparisons of online and batch versions of bagging and boosting. In: *Proceedings of 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 359–364 (ACM, New York, 2001).

Page, E. S. Continuous inspection schemes. *Biometrika* **41**, 100–115 (1954).

Pan Pang, K.; Ting, K. M. Improving the centered CUSUMs statistic for structural break detection in time series. In: *Proceedings of 17th Australian Joint Conference on Artificial Intelligence*, 402–413 (Springer, Berlin, 2004).

Pelossof, R.; Jones, M.; Vovsha, I.; Rudin, C. Online coordinate boosting. In: *Proceedings of 12th International Conference on Computer Vision Workshops*, 1354–1361 (IEEE Conference Publications, Washington, DC, 2009)

Pfahringer, B.; Holmes, G.; Kirkby, R. New options for Hoeffding trees. In: *Proceedings of 20th Australian Joint Conference on Artificial Intelligence*, 90–99 (Springer, Berlin, 2007).

Pfahringer, B.; Holmes, G.; Kirkby, R. Handling numeric attributes in Hoeffding trees. In: *Proceedings of 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 296–307 (Springer, Berlin, 2008).

Piater, J. H.; Cohen, P. R.; Zhang, X.; Atighetchi, M. A randomized ANOVA procedure for comparing performance curves. In: *Proceedings of 15th International Conference on Machine Learning*, 430–438 (Morgan Kaufmann, San Francisco, 1998).

Potts, D.; Sammut, C. Incremental learning of linear model trees. *Machine Learning* **61**, 5–48 (2005).

Roberts, S. W. Conrol chart tests based on geometric moving averages. *Technometrics* **42**, 97-101 (1959).

Qin, S.; Qian, W.; Zhou, A. Approximately processing multi-granularity aggregate queries over data streams. In: *Proceedings of 22nd International Conference on Data Engineering*, 67–77 (IEEE Computer Society, Washington, DC, 2006).

Quinlan, J. R. Induction of decision trees. *Machine Learning* **1**, 81–106 (1986).

Quinlan, J. R. Learning with continuous classes. In: *Proceedings of 5th Australian Joint Conference on Artificial Intelligence*, 343–348 (World Scientific, Singapore, 1992).

Quinlan, J. R. *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Francisco, CA, 1993).

Quinlan, J. R.; Cameron-Jones, R. M. Over searching and layered search in empirical learning. In: *Proceedings of 14th International Joint Conference on Artificial Intelligence*, 1019–1024 (Morgan Kaufman, San Francisco, CA, 1995).

Raedt, L. D.; Laer, W. V. Inductive constraint logic. In: *Proceedings of 6th International Conference on Algorithmic Learning and Theory*, 80–94 (Springer, Berlin, 1995).

Ramon, J.; Driessens, K.; Croonenborghs, T. Transfer learning in reinforcement learning problems through partial policy recycling. In: *Proceedings of 18th European Conference on Machine Learning*, 699–707 (Springer, Berlin, 2007).

Read, J.; Bifet, A.; Holmes, G.; Pfahringer, B. Scalable and efficient multi-label classification for evolving data streams. *Machine Learning* **88**, 243–272 (2012).

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* **65**, 386–408 (1958).

Schapire, R. E.; Singer, Y. Improved boosting algorithms using confidence-rated predictions. In: *Proceedings of 11th Annual Conference on Computational Learning Theory*, 80–91 (ACM, New York, 1998).

Schapire, R. E.; Singer, Y. BoosTexter: A Boosting-based System for Text Categorization. *Machine Learning* **39**, 135–168 (2000).

Schietgat, L.; Vens, C.; Struyf, J.; Blockeel, H.; Kocev, D.; Dzeroski, S. Predicting gene function using hierarchical multi-label decision tree ensembles. *BMC Bioinformatics* **11**, 2 (2010).

Schlimmer, J. C.; Fisher, D. H. A case study of incremental concept induction. In: *Proceedings of 5th National Conference on Artificial Intelligence*, 496–501 (Morgan Kaufmann, San Francisco, CA, 1986).

Scholz, M.; Klinkenberg, R. Boosting classifiers for drifting concepts. *Intelligent Data Analalysis* **11**, 3–28 (2007).

Scott, A. J.; Knott, M. An approximate test for use with aid. *Applied Statistics* **25**, 103–106 (1976).

Sebastião, R.; Gama, J. Change detection in learning histograms from data streams. In: *Proceedings of 13th Portuguese Conference on Artificial Intelligence* 112–123 (Springer, Berlin, 2007).

Robert Sedgewick, Chapter 15: Balanced Trees *Algorithms* 199, (Addison-Wesley, 1983).

Segal, M. R. Tree-structured methods for longitudinal data. *American Statistical Association* **87**, 407–418 (1992).

Shafer, J.; Agrawal, R.; Mehta, M. Sprint: A scalable parallel classifier for data mining. In: *Proceedings of 22nd International Conference on Very Large Data Bases*, 544–555 (Morgan Kaufmann, San Francisco, CA, 1996).

Siddiqui, Z. F.; Spiliopoulou, M. Combining multiple interrelated streams for incremental clustering. In: *Proceedings of 21st International Conference on Scientific and Statistical Database Management*, 535–552 (Springer, Berlin, 2009).

Siegmund, D. *Sequential Analysis* (Springer, New York, 1985).

Song, X.; Wu, M.; Jermaine, C.; Ranka, S. Statistical change detection for multi-dimensional data. In: *Proceedings of 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 667–676 (ACM, New York, 2007).

StatLib System (1989). URL `http://lib.stat.cmu.edu/index.php` (Accessed: July, 2012).

Stojanova, D.; Panov, P.; Gjorgjioski, V.; Kobler, A.; Džeroski, S. Estimating vegetation height and canopy cover from remotely sensed data with machine learning. *Ecological Informatics* **5**, 256–266 (2010).

Stout, M.; Bacardit, J.; Hirst, J. D.; Krasnogor, N. Prediction of recursive convex hull class assignments for protein residues. *Bioinformatics* **24**, 916–923 (2008).

Street, W. N.; Kim, Y. A streaming ensemble algorithm (SEA) for large-scale classification. In: *Proceedings of 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 377–382 (ACM, New York, 2001).

Struyf, J.; Džeroski, S. Constraint based induction of multi-objective regression trees. In: *Proceedings of 4th International Conference on Knowledge Discovery in Inductive Databases*, 222–233 (Springer, Berlin, 2006).

Subramaniam, S.; Palpanas, T.; Papadopoulos, D.; Kalogeraki, V.; Gunopulos, D. Online outlier detection in sensor data using non-parametric models. In: *Proceedings of 32nd International Conference on Very Large Data Bases*, 187–198 (Morgan Kaufmann, San Francisco, CA, 2006).

Tan, S. C.; Ting, K. M.; Liu, T. F. Fast anomaly detection for streaming data. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 1511-1516 (AAAI Press, Palo Alto, CA, 2011)

Ting, K. M.; Witten, I. H. Issues in stacked generalization. *Journal of Artificial Intelligence Research* **10**, 271–289 (1999).

Torgo, L. Functional models for regression tree leaves. In: *Proceedings of 14th International Conference on Machine Learning*, 385–393 (Morgan Kaufmann, San Francisco, CA, 1997).

Tsoumakas, G.; Katakis, I. Multi label classification: An overview. *International Data Warehousing and Mining* **3**, 1–13 (2007).

Tsymbal, A.; Pechenizkiy, M.; Cunningham, P.; Puuronen, S. Dynamic integration of classifiers for handling concept drift. *Information Fusion* **9**, 56–68 (2008).

Ueda, N.; Nakano, R. Generalization error of ensemble estimators. In: *Proceedings of 6th IEEE International Conference on Neural Networks*, 90–95 (IEEE Computer Society, 1996).

Utgoff, P. E. An incremental ID3. In: *Proceedings of 5th International Conference on Machine Learning*, 107–120 (Morgan Kaufmann, San Francisco, CA, 1988).

Utgoff, P. E.; Berkman, N. C.; Clouse, J. A. Decision tree induction based on efficient tree restructuring. *Machine Learning* **29**, 5–44 (1997).

Valiant, L. A theory of the learnable. *Communications of the ACM* **27**, 1134–1142 (ACM New York, 1984).

Vens, C.; Struyf, J.; Schietgat, L.; Džeroski, S.; Blockeel, H. Decision trees for hierarchical multi-label classification. *Machine Learning* **73**, 185–214 (2008).

Viglas, S. D.; Naughton, J. F. Rate-based query optimization for streaming information sources. In: *Proceedings of 31st ACM SIGMOD International Conference on Management of Data*, 37–48 (ACM, New York, 2002).

Viglas, S. D.; Naughton, J. F.; Burger, J. Maximizing the output rate of multi-way join queries over streaming information sources. In: *Proceedings of 29th International Conference on Very Large Data Bases* **29**, 285–296 (Morgan Kaufmann, San Francisco, CA, 2003).

Vogel, D. S.; Asparouhov, O.; Scheffer, T. Scalable look-ahead linear regression trees. In: *Proceedings of 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 757–764 (ACM, New York, 2007).

Wald, A. Sequential Tests of Statistical Hypotheses. *Annals of Mathematical Statistics* **16**, 117–186 (1945).

Wald, A. *Sequential Analysis* (John Wiley and Sons, New York, 1947).

Y. Wang, I. H. Witten Induction of model trees for predicting continuous classes. In: *Poster papers of the 9th European Conference on Machine Learning* (Morgan Kaufmann, San Francisco, CA,1997).

Wang, H.; Fan, W.; Yu, P. S.; Han, J. Mining concept-drifting data streams using ensemble classifiers. In: *Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 226–235 (ACM, New York, 2003).

Widmer, G.; Kubat, M. Learning in the presence of concept drift and hidden contexts. *Machine Learning* **23**, 69–101 (1996).

Wirth, J.; Catlett, J. Experiments on the costs and benefits of windowing in ID3. In: *Proceedings of 5th International Conference on Machine Learning* (Morgan Kaufmann, San Francisco, CA, 1988).

Wolpert, D. H. Stacked generalization. *Neural Networks* **5**, 241–259 (1992).

Xu, Y.; Wang, K.; Fu, A. W.-C.; She, R.; Pei, J. Classification spanning correlated data streams. In: *Proceedings of 15th ACM International Conference on Information and Knowledge Management*, 132–141 (ACM, New York, 2006).

Yu, Y.; Zhou, Z.-H.; Ting, K. M. Cocktail ensemble for regression. In: *Proceedings of 7th IEEE International Conference on Data Mining*, 721–726 (IEEE Computer Society, Washington, DC, 2007).

Zhang, P.; Zhu, X.; Shi, Y. Categorizing and mining concept drifting data streams. In: *Proceedings of 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 812–820 (ACM, New York, 2008).

Zhang, R.; Koudas, N.; Ooi, B. C.; Srivastava, D.; Zhou, P. Streaming multiple aggregations using phantoms. *The VLDB Journal* **19**, 557–583 (2010).

Zhang, Y.; Jin, X. An automatic construction and organization strategy for ensemble learning on data streams. *ACM SIGMOD Record* **35**, 28–33 (2006).

Index of Figures

Index of Tables

List of Algorithms

Appendices

A Additional Experimental Results

A.1 Additional Results for Section 6.8.

Table 29: Results of 10-fold cross-validation for the regression tree learning algorithms on 10 real-world datasets.

		CART			M5'RT			FIRT		
Data	Size	RRSE%	#L	CC	RRSE	#L	CC	RRSE%	Leaves	CC
Abal.	4177	74.68	10.6	0.67	70.26	41.8	0.71	77.31	8.5	0.63
Ailer.	13750	58.67	9.6	0.81	46.31	246.7	0.89	56.73	23.4	0.82
Mv_D.	40769	27.51	8.0	0.96	4.76	312.1	0.99	8.97	98.8	0.99
Wind	6574	59.56	7.9	0.80	53.28	107.4	0.85	60.68	11.2	0.79
Cal_H.	20640	67.67	12.0	0.74	51.79	439.7	0.86	61.46	49.6	0.79
H_8L	22784	69.14	10.2	0.72	62.45	251.9	0.78	66.99	50.1	0.74
H_16H	22784	79.38	12.1	0.61	70.80	347.9	0.71	78.28	42.8	0.63
Elev.	16599	68.09	12.8	0.73	52.14	382.6	0.86	74.86	31.9	0.66
Pol	15000	36.76	7.0	0.93	22.57	264.9	0.97	29.18	27.1	0.96
Wine.	4898	86.27	6.5	0.51	82.85	51.9	0.56	86.62	12.0	0.50

Table 30: Additional Results of 10-fold cross-validation (RE%) for the model tree learning algorithms on 10 real-world datasets.

Dataset	CUBIST	M5'MT	FIMT_Const	FIMT_Decay	LR	BatchRD	BatchRA	OnlineRD	OnlineRA
Abalone	63.20	63.97	74.10	74.18	66.96	63.91	66.41	66.45	66.95
Ailerons	100.00	37.67	102.35	85.53	41.05	37.54	39.32	38.36	40.24
Mv_delve	0.70	0.54	4.83	8.49	38.32	0.06	1.19	0.40	2.05
Wind	44.60	45.98	53.77	60.48	46.90	43.53	44.02	44.48	44.32
Cal_housing	39.10	39.80	48.90	49.24	55.73	40.63	45.27	48.77	48.64
House_8L	50.30	51.39	65.37	69.61	75.10	51.86	55.48	62.31	56.89
House_16H	52.50	56.13	67.91	63.72	78.49	57.07	62.20		66.58
Elevators	49.90	34.78	82.50	80.35	43.52	34.25	39.01	36.87	39.76
Pol	6.80	6.83	16.20	18.09	71.50	7.06	63.34	43.99	30.38
Winequality	80.40	83.81	86.21	84.26	87.30	82.49	85.97	84.48	85.96

Table 31: Results of 10-fold cross-validation (learning time in seconds) for the model tree learning algorithms on 10 real-world datasets.

Dataset	CUBIST	M5'MT	FIMT_Const	FIMT_Decay	LR	BatchRD	BatchRA	OnlineRD	OnlineRA
Abalone	0.190	2.692	0.041	0.043	0.324	0.589	0.270	0.869	0.325
Ailerons	0.320	14.727	0.661	0.656	1.068	11.865	0.885	148.517	2.355
Mv_delve	1.290	135.125	0.907	0.911	15.449	12.968	11.799	13.974	12.526
Wind	0.450	8.179	0.138	0.135	0.663	1.797	0.657	4.114	0.744
Cal_housing	1.330	48.183	0.356	0.358	3.102	4.998	2.911	5.504	2.930
House_8L	1.50	29.931	0.357	0.352	1.671	4.907	1.921	4.099	1.962
House_16H	2.780	34.515	0.705	0.704	1.757	11.359	2.156	15.667	2.658
Elevators	0.540	11.1	0.367	0.366	0.736	4.905	0.732	14.41	1.203
Pol	1.480	8.24	0.606	0.610	0.771	7.125	0.820	117.468	5.737
Winequality	0.270	2.745	0.067	0.071	0.331	0.841	0.291	2.117	0.398

Table 32: Results of 10-fold cross-validation (size) for the model tree learning algorithms on 10 real-world datasets.

Dataset	CUBIST	M5'MT	FIMT_Const	FIMT_Decay	BatchRD	BatchRA	OnlineRD	OnlineRA
Abalone	11.5	8.4	8.5	8.5	7.1	2.0	2.7	2.0
Ailerons	1.0	5.8	23.4	23.4	13.5	4.7	3.9	2.5
Mv_delve	8.6	9.9	98.8	98.8	166.4	117.2	49.5	54.2
Wind	15.4	7.2	11.2	11.2	10.8	4.1	4.0	4.1
Cal_housing	41.8	214.0	49.6	49.6	120.3	24.8	7.1	9.8
House_8L	21.9	117.2	50.1	50.1	60.7	29.5	8.7	19.3
House_16H	29.6	171.3	42.8	42.8	81.4	33.8	4.8	18.9
Elevators	18.8	39.6	31.9	31.9	37.0	8.2	11.9	6.3
Pol	25.1	159.2	27.1	27.1	46.8	2.0	3.9	8.3
Winequality	33.0	34.5	12.0	12.0	11.8	2.2	3.3	2.4

Table 33: Results on the artificial datasets for the algorithms FIRT, CUBIST, FIMT_Const, FIMT_Decay, LR, BatchRD, BatchRA, OnlineRD, OnlineRA. The algorithms were run for 10 randomly generated pairs of train/test sets of size 1000K/300K correspondingly. Averages over the 10 runs of the RE, RRSE, number of leaves, running time and CC measures are given here.

Algorithm	Dataset	RE	RRSE	Leaves	Time (s)	CC
FIRT		0.31	0.32	2406.6	42.26	0.95
CUBIST		0.24	NA	58.2	192.47	0.97
FIMT_Const		0.26	0.27	2406.6	46.72	0.96
FIMT_Decay		0.26	0.27	2406.6	46.93	0.96
LR	Fried	0.50	0.52	1.0	2509.18	0.85
BatchRD		0.19	0.19	289.9	2341.83	0.98
BatchRA		0.20	0.20	160.8	1946.87	0.98
OnlineRD		0.19	0.20	120.7	2253.09	0.98
OnlineRA		0.20	0.21	143.8	2013.05	0.98
FIRT		0.09	0.11	2328.8	15.82	0.99
CUBIST		0.06	NA	27.5	59.19	0.99
FIMT_Const		0.01	0.02	2328.8	19.81	1.00
FIMT_Decay		0.01	0.02	2328.8	19.07	1.00
LR	Lexp	0.67	0.75	1.0	2785.64	0.67
BatchRD		0.01	0.04	43969.6	6411.54	0.99
BatchRA		0.92	0.88	9.1	2549.39	0.47
OnlineRD		0.004	0.02	13360.1	4039.49	0.99
OnlineRA		0.90	0.85	28.5	2645.06	0.53
FIRT		0.09	0.14	2621.3	16.16	0.99
CUBIST		0.09	NA	26.8	62.71	0.99
FIMT_Const		0.07	0.14	2621.3	14.81	0.99
FIMT_Decay		0.07	0.14	2621.3	14.80	0.99
LR	Losc	0.22	0.26	1.0	2111.03	0.96
BatchRD		0.02	0.07	37599.5	6951.18	0.99
BatchRA		1.00	0.99	1.0	2451.83	0.08
OnlineRD		0.11	0.17	6257.7	3006.89	0.98
OnlineRA		1.00	0.99	1.0	2423.58	0.08

A.2 Additional Learning Curves for Section 7.4

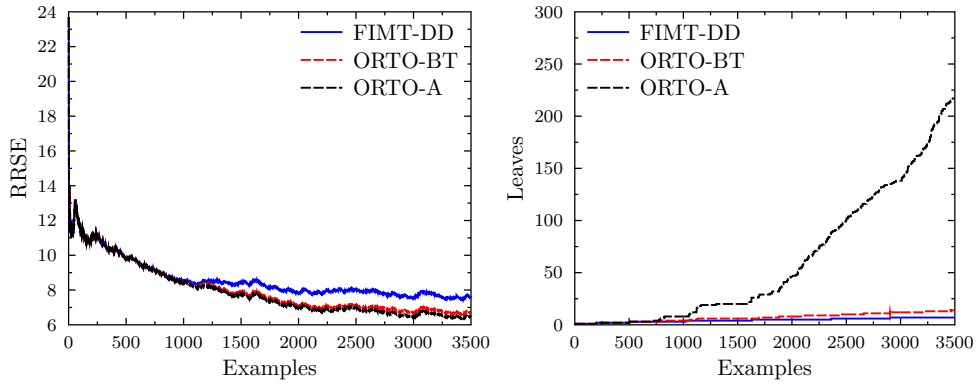


Figure 25: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Abalone dataset.

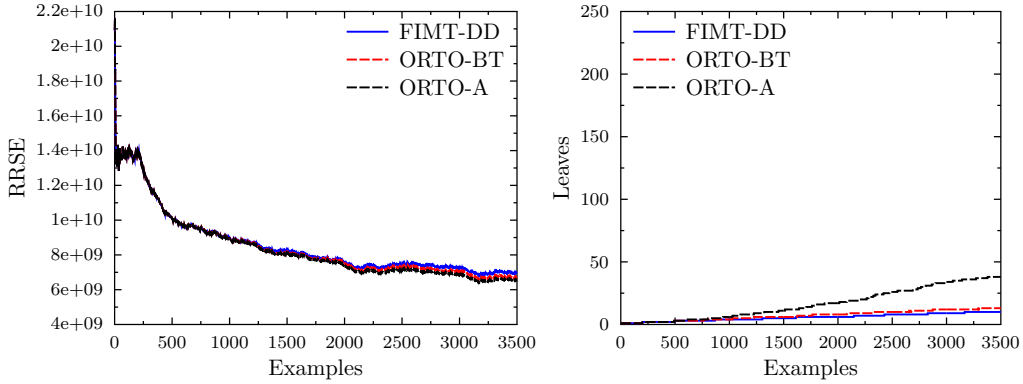


Figure 26: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Cal Housing dataset.

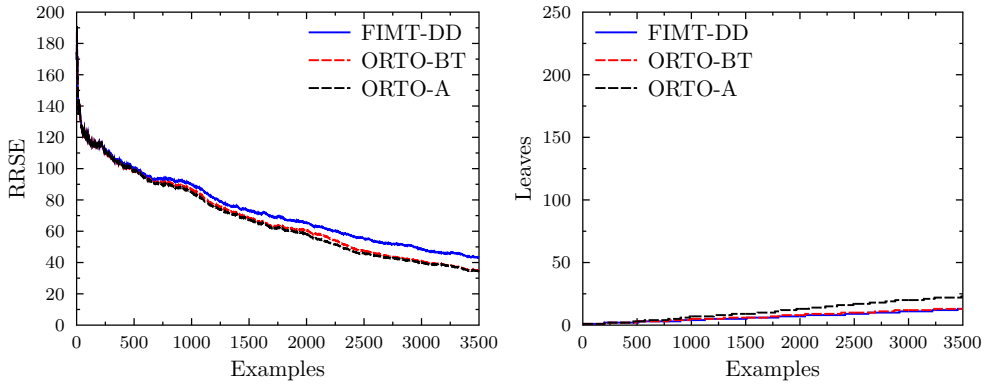


Figure 27: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Mv Delve dataset.

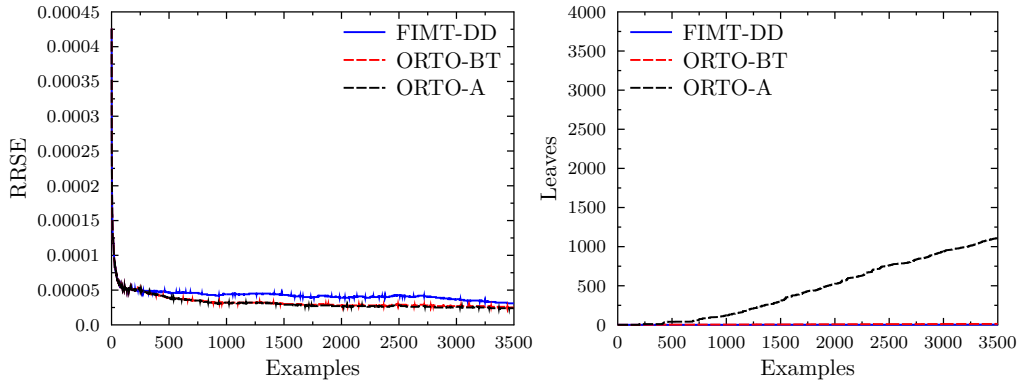


Figure 28: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Elevators dataset.

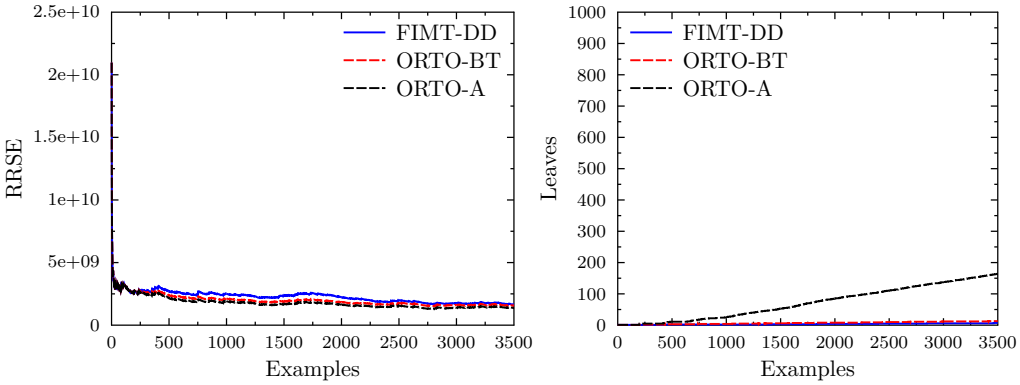


Figure 29: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the House 8L dataset.

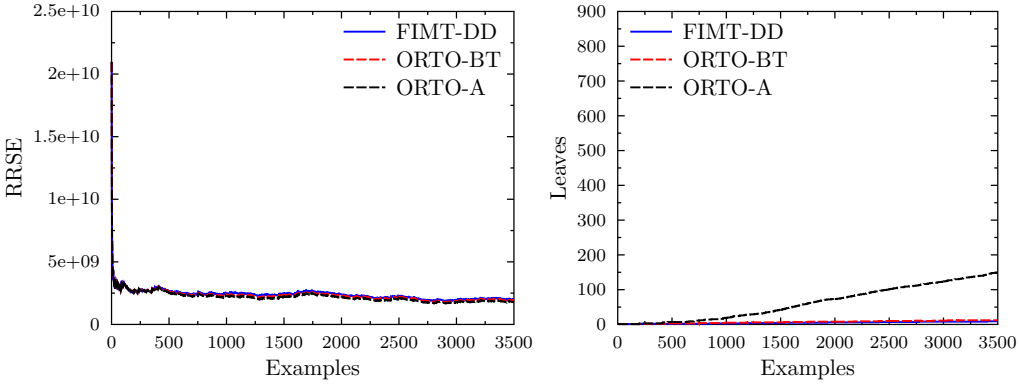


Figure 30: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the House 16H datasets.

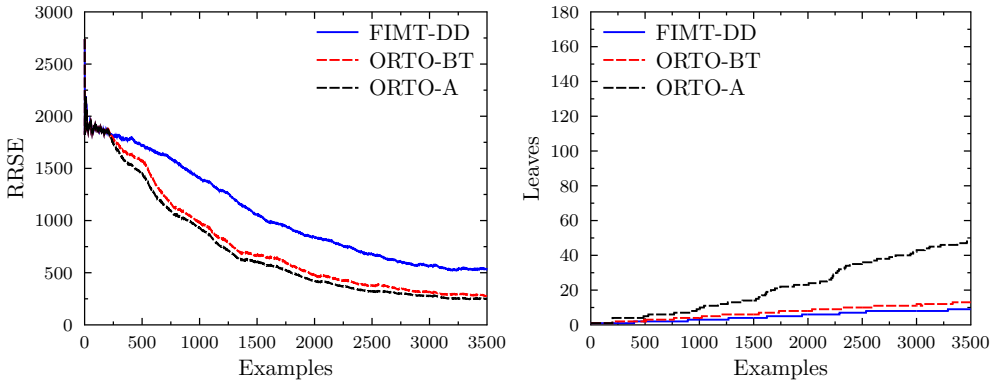


Figure 31: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Pol dataset.

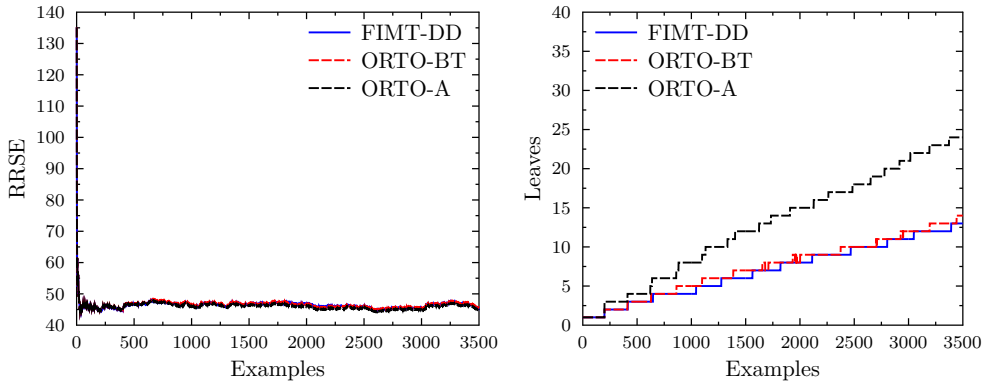


Figure 32: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Wind dataset.

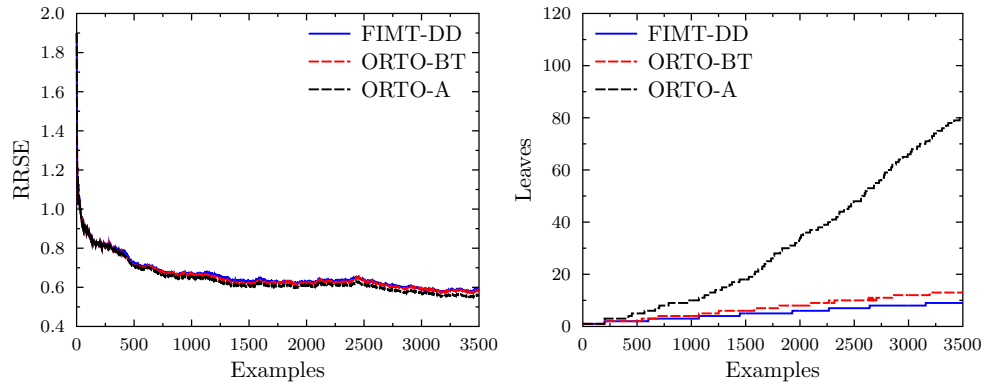
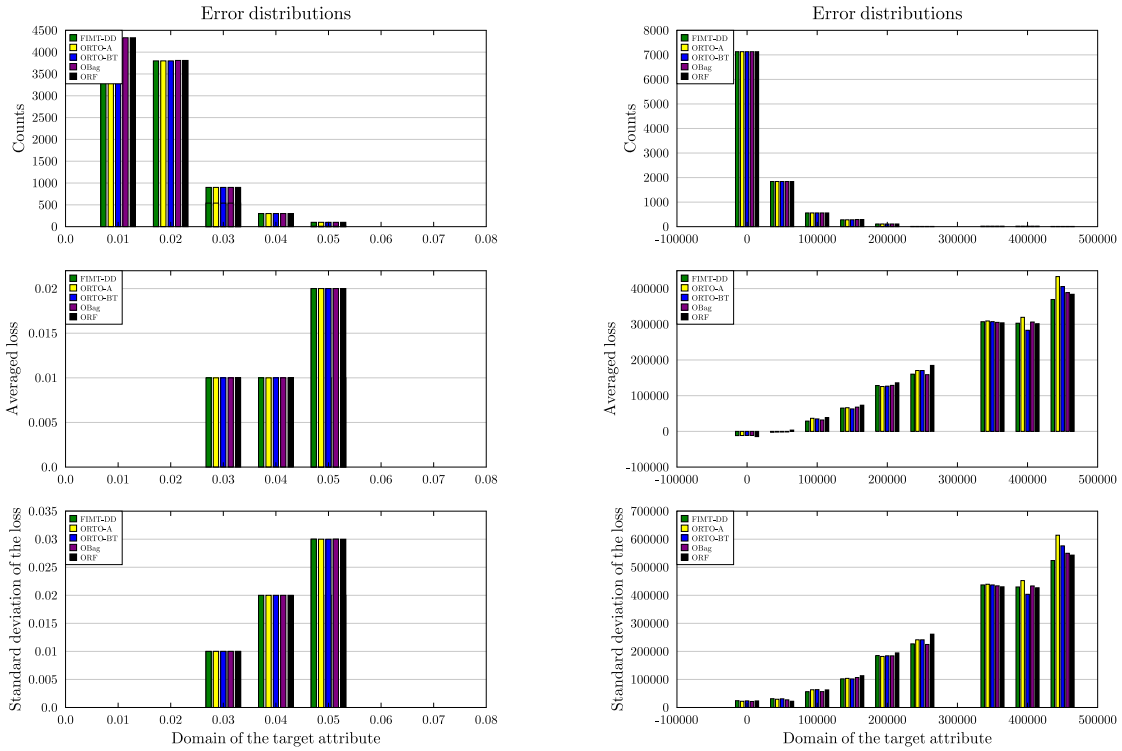


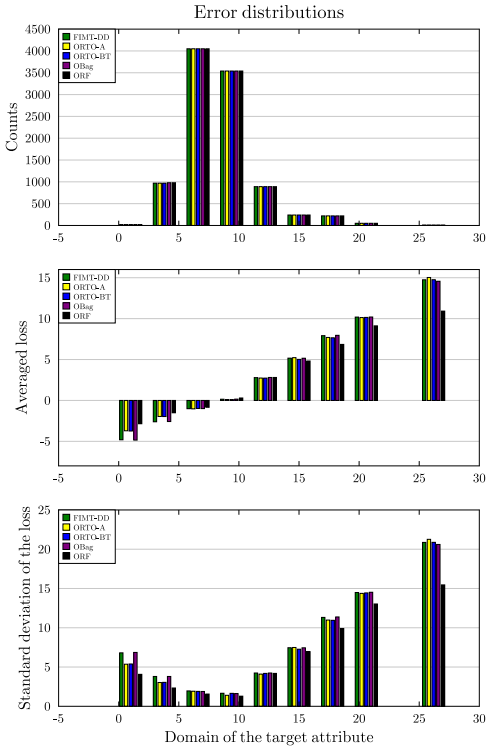
Figure 33: Weighted prequential mean squared error and number of leaves for FIMT-DD, ORTO-BT, and ORTO-A over the first 3500 examples of the Wine quality dataset.

A.3 Additional Error Bars for Section 8.4

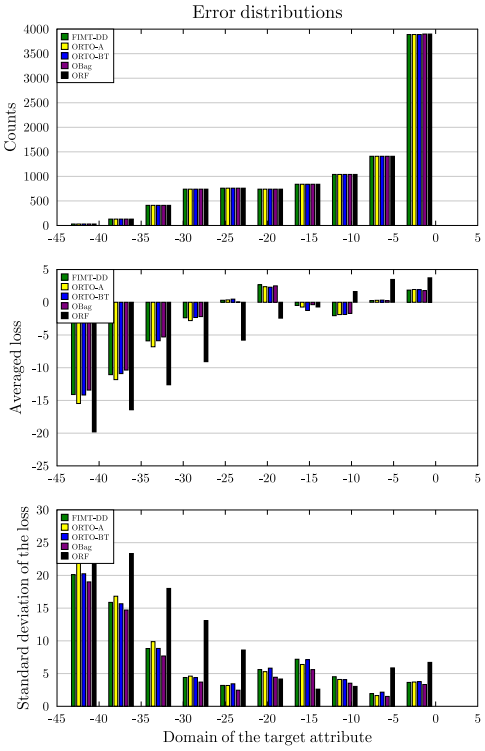


(a) Elevators (b) House 16H

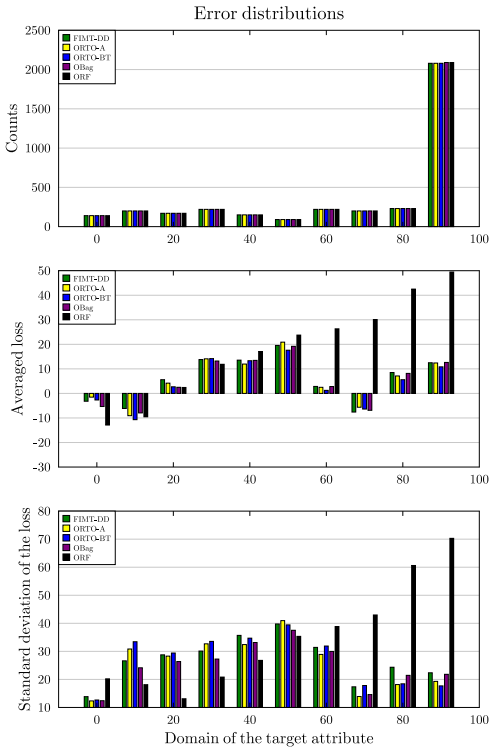
Figure 34: Distribution of errors of the single-tree method FIMT-DD and the different ensemble methods over the range of the target attribute obtained for the last holdout evaluation on the Elevators (upper left), and House 16H (upper right) dataset. For reference, we first give graphs showing the distribution of values for the target.



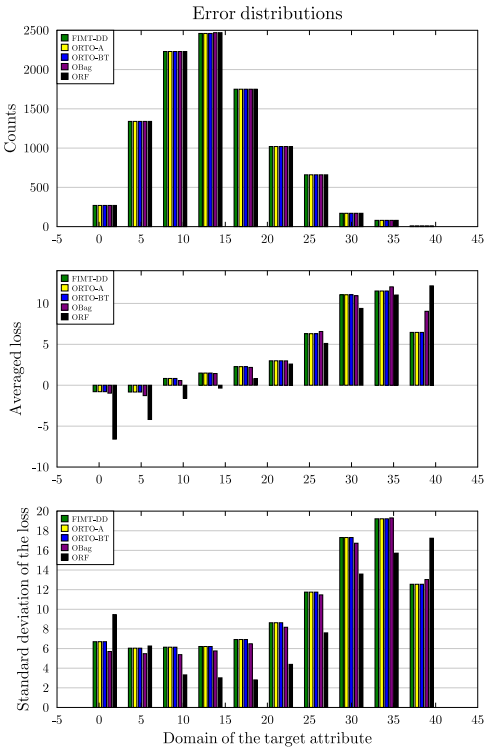
(a) Abalone



(b) Mv Delve

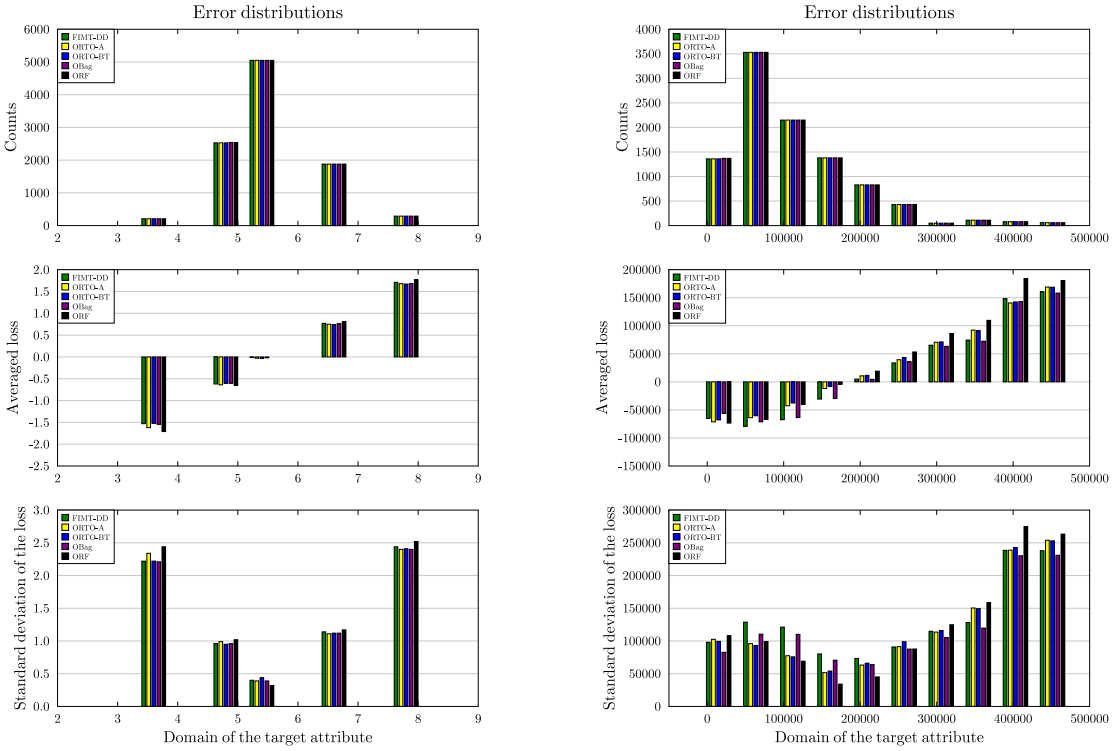


(c) Pol



(d) Wind

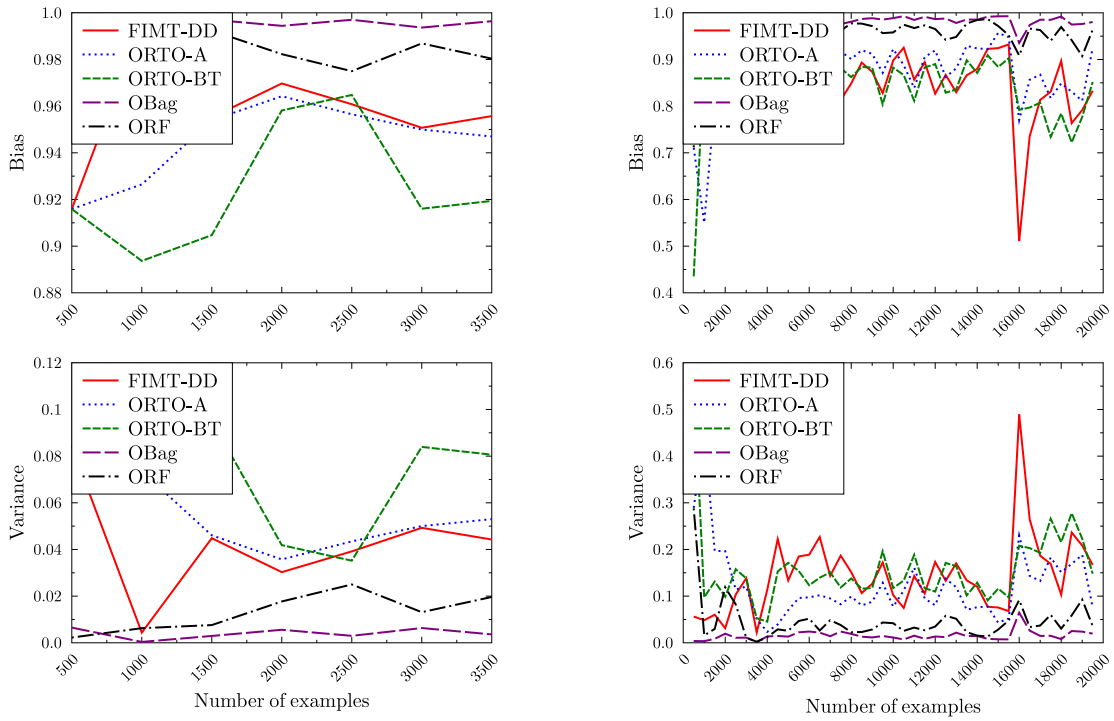
Figure 35: Distribution of errors of the single-tree method FIMT-DD and the different ensemble methods over the range of the target attribute obtained for the last holdout evaluation on the Abalone (upper left), Mv Delve (upper right), Pol (lower left), and Wind (lower right) dataset.



(a) Winequality (b) Cal Housing

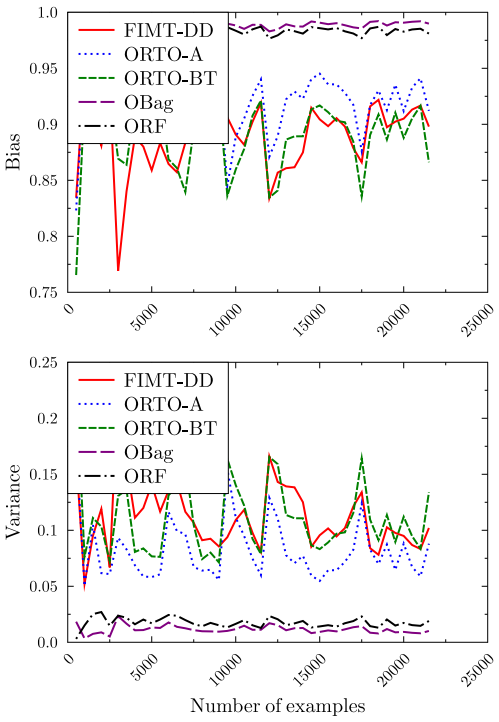
Figure 36: Distribution of errors of the single-tree method FIMT-DD and the different ensemble methods over the range of the target attribute obtained for the last holdout evaluation on the Winequality (lower left), and Cal Housing (lower right) dataset. For reference, we first give graphs showing the distribution of values for the target.

A.4 Additional Learning Curves for Section 8.4

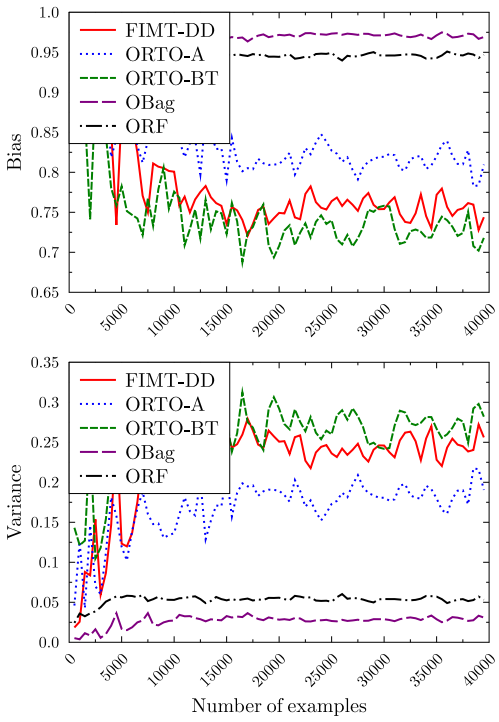


(a) Abalone (b) Cal Housing

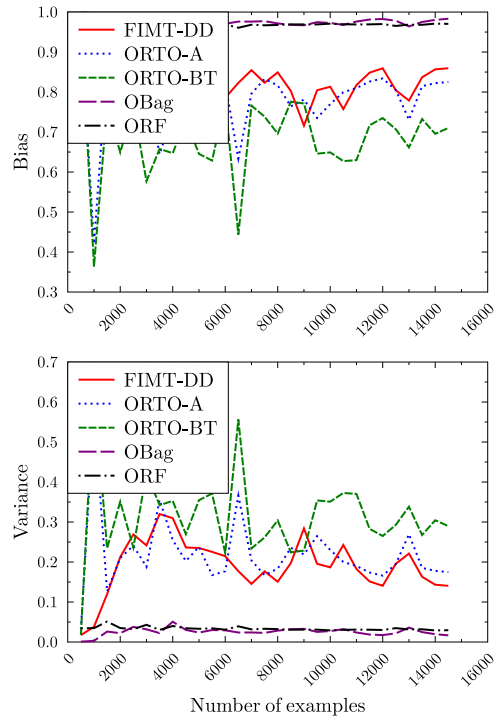
Figure 37: Relative bias and variance error components of the single-tree method FIMT-DD and the different ensemble methods computed using a sliding window bias-variance decomposition on the: (a) Abalone, and (b) Cal Housing.



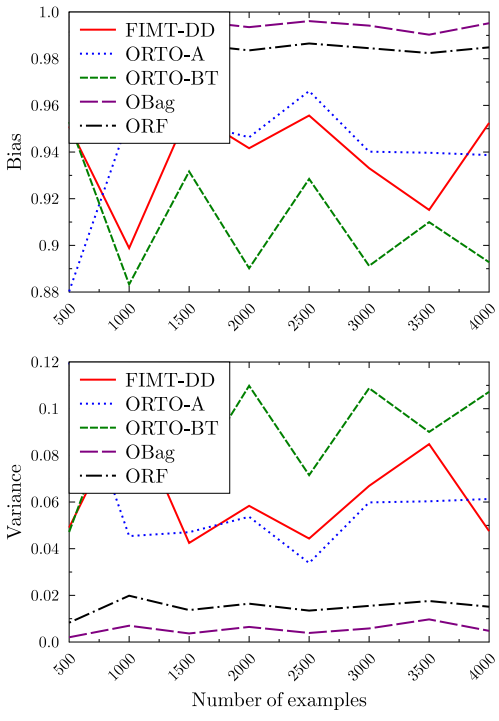
(a) House 16H



(b) Mv Delve

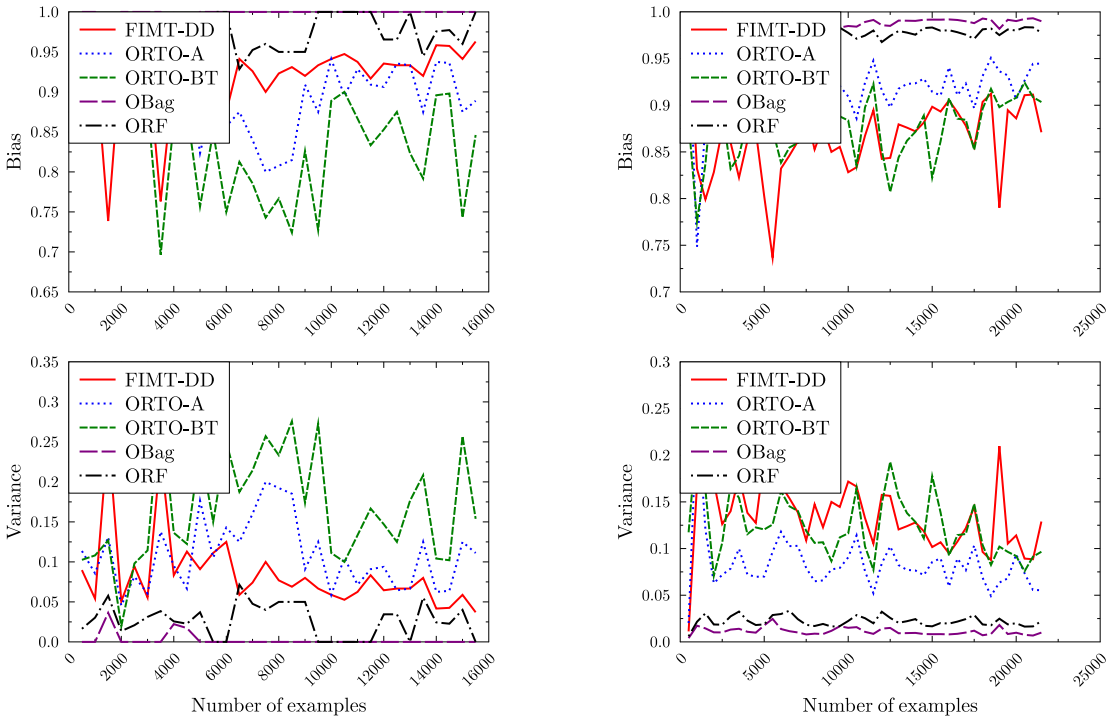


(c) Pol



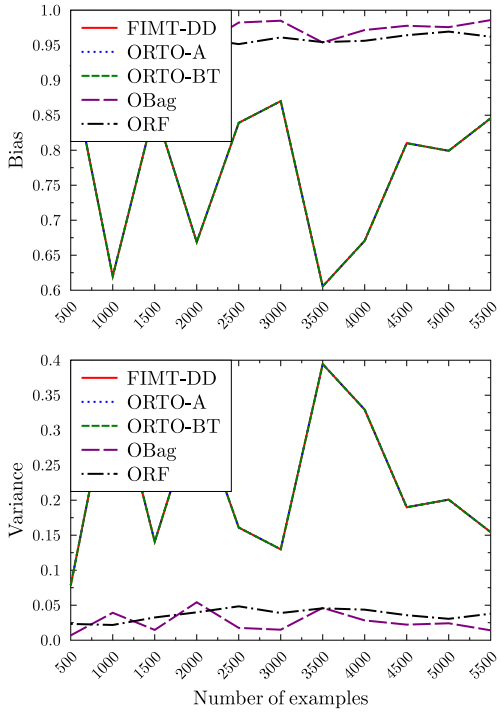
(d) Winequality

Figure 38: Relative bias and variance error components of the single-tree method FIMT-DD and the different ensemble methods computed using a sliding window bias-variance decomposition on the: (a) House 16H, (b) Mv Delve, (c) Pol and (d) Wine quality.



(a) Elevators (b) House 8L

Figure 39: Relative bias and variance error components of the single-tree method FIMT-DD and the different ensemble methods computed using a sliding window bias-variance decomposition on the: (a) Elevators, and (b) House 8L.



(a) Wind

Figure 40: Relative bias and variance error components of the single-tree method FIMT-DD and the different ensemble methods computed using a sliding window bias-variance decomposition on the Wind dataset.

A.5 Additional Results for Section 8.4

Table 34: Averaged mean squared error from 10 runs of sampling for the last holdout evaluation is given. The maximum number of trees is 10 and the maximum tree depth are: (a) 6 and (b) 7. Best results are shown in boldface.

(a) Mean squared error for maximal tree depth 6 and maximal number of trees 10

Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF
Abalone	6.69E+0	5.91E+0	6.25E+0	6.84E+0	**5.68E+0**
Cal Housing	7.08E+9	**5.03E+9**	5.14E+9	7.85E+9	6.55E+9
Elevators	2.50E-5	2.50E-5	2.40E-5	2.30E-5	**2.20E-5**
House 8L	1.26E+9	**1.06E+9**	1.10E+9	1.21E+9	1.52E+9
House 16H	1.59E+9	**1.45E+9**	1.46E+9	1.59E+9	1.71E+9
Mv Delve	8.73E+0	**8.21E+0**	8.71E+0	8.30E+0	48.84E+0
Pol	239.11E+0	**189.27E+0**	226.16E+0	232.51E+0	1140.89E+0
Wind	48.31E+0	48.31E+0	48.31E+0	48.50E+0	**21.17E+0**
Winequality	0.52E+0	**0.51E+0**	0.54E+0	0.52E+0	0.59E+0

(b) Mean squared error for maximal tree depth 7 and maximal number of trees 10

Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF
Abalone	6.69E+0	5.91E+0	6.25E+0	6.84E+0	**5.68E+0**
Cal Housing	12.12E+9	**5.63E+9**	7.00E+9	10.35E+9	7.67E+9
Elevators	2.50E-5	2.50E-5	2.40E-5	2.30E-5	**2.20E-5**
House 8L	1.21E+9	**1.11E+9**	1.27E+9	1.21E+9	1.60E+9
House 16H	1.73E+9	**1.47E+9**	1.57E+9	1.78E+9	1.75E+9
Mv Delve	5.15E+0	**4.28E+0**	4.95E+0	4.83E+0	43.78E+0
Pol	255.17E+0	**176.13E+0**	201.37E+0	254.34E+0	1080.68E+0
Wind	47.79E+0	47.79E+0	47.79E+0	48.72E+0	**21.31E+0**
Winequality	0.52E+0	**0.51E+0**	0.53E+0	0.52E+0	0.59E+0

Table 35: Averaged mean squared error from 10 runs of sampling for the last holdout evaluation is given. The maximum tree depth is 5 and the maximum number of trees is: (a) 20 and (b) 30. Best results are shown in boldface.

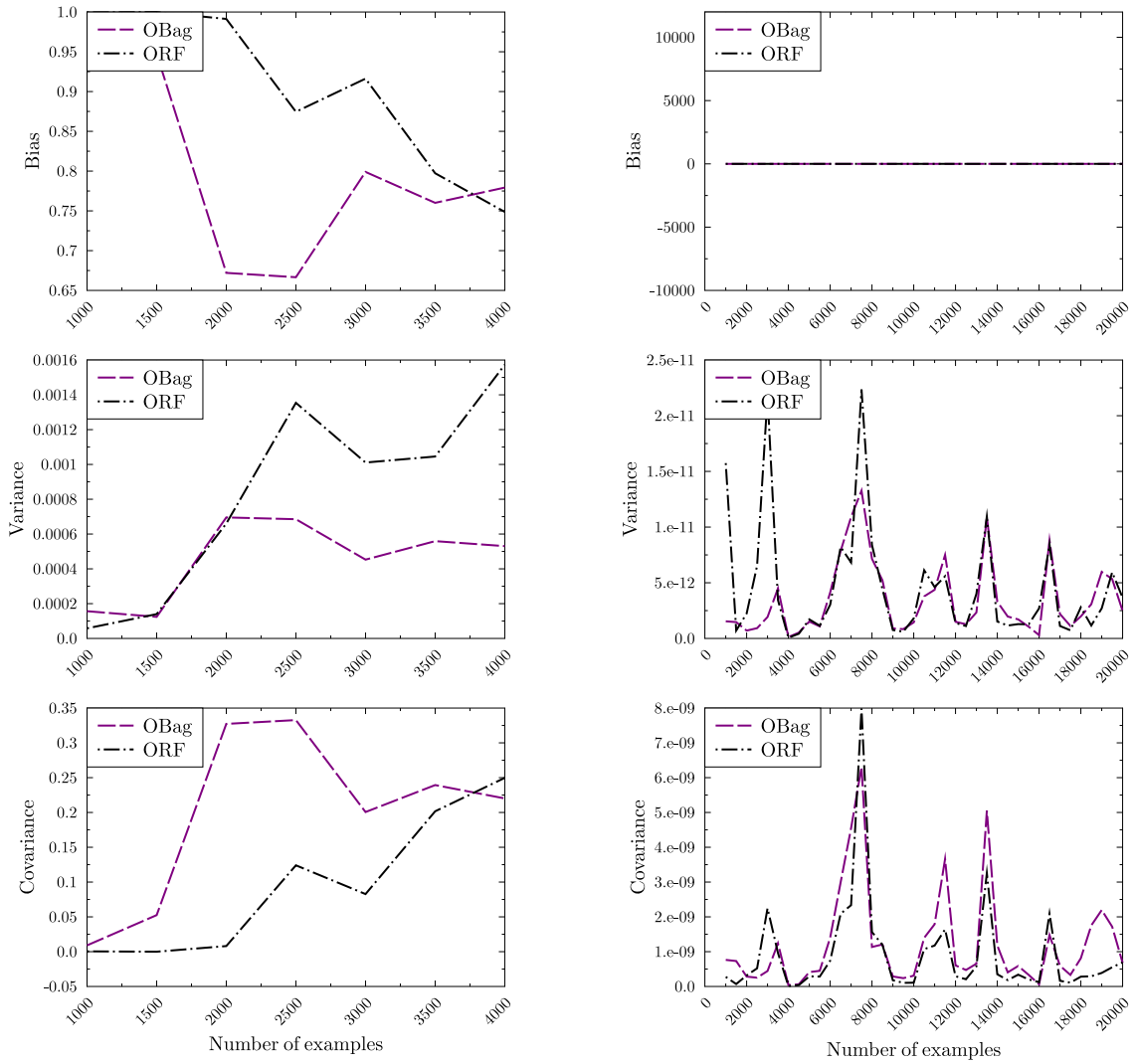
(a) Mean squared error for maximal tree depth 5 and maximal number of trees 20

Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF
Abalone	6.69E+0	5.65E+0	5.92E+0	6.82E+0	**5.62E+0**
Cal Housing	8.70E+9	**4.95E+9**	5.00E+9	7.82E+9	6.49E+9
Elevators	2.50E-5	2.50E-5	2.40E-5	2.30E-5	**2.20E-5**
House 8L	1.32E+9	**1.10E+9**	1.16E+9	1.29E+9	1.52E+9
House 16H	1.56E+9	**1.57E+9**	1.60E+9	1.56E+9	1.75E+9
Mv Delve	18.04E+0	**17.22E+0**	18.26E+0	17.45E+0	53.54E+0
Pol	273.63E+0	**220.29E+0**	233.52E+0	269.76E+0	1192.42E+0
Wind	48.89E+0	48.89E+0	48.89E+0	48.69E+0	**21.82E+0**
Winequality	**0.52E+0**	0.72E+0	0.74E+0	**0.52E+0**	0.58E+0

(b) Mean squared error for maximal tree depth 5 and maximal number of trees 30

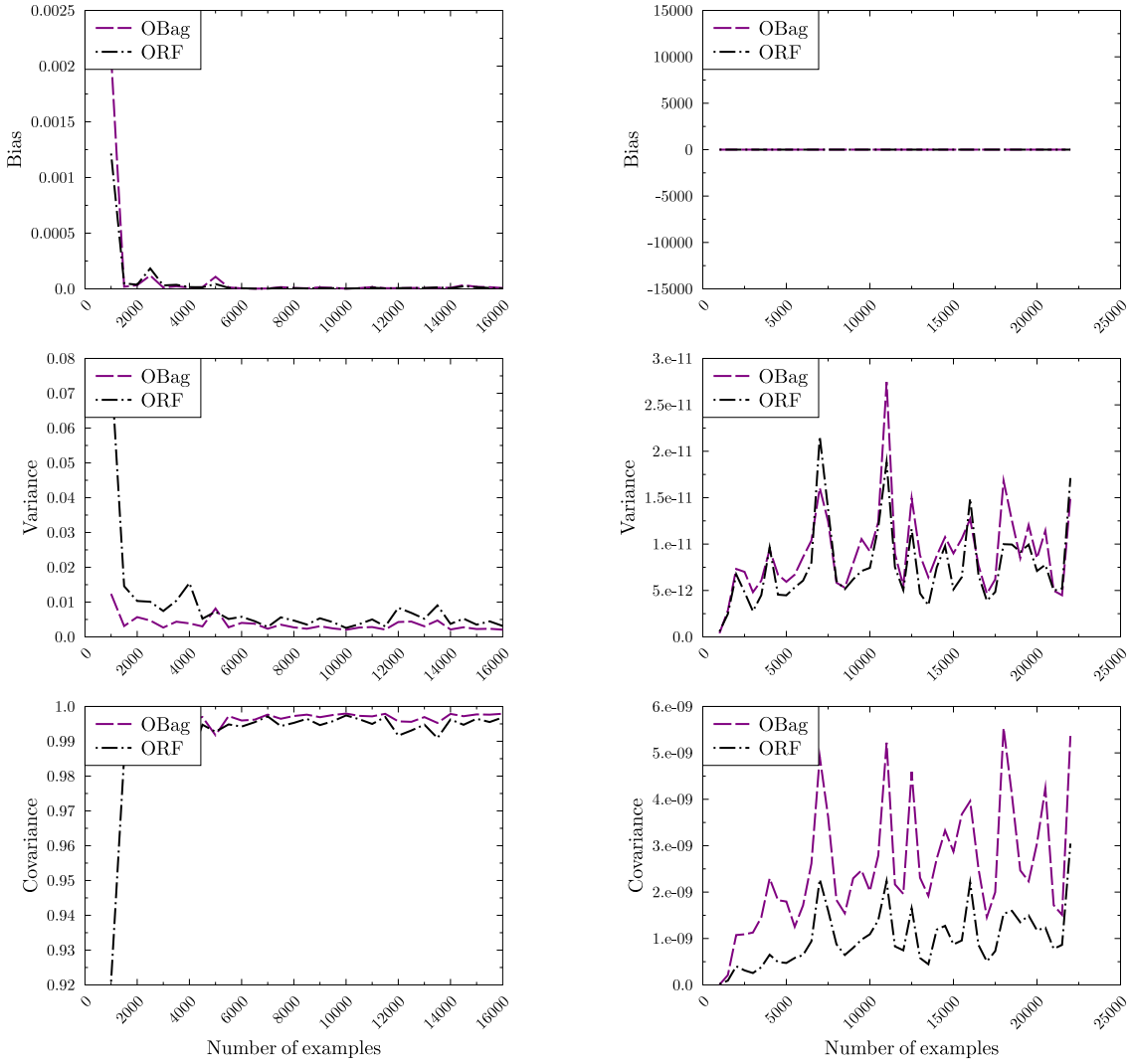
Dataset	FIMT-DD	ORTO-A	ORTO-BT	OBag	ORF
Abalone	6.69E+0	**5.57E+0**	6.07E+0	6.81E+0	5.62E+0
Cal Housing	8.70E+9	**4.74E+9**	5.32E+9	7.82E+9	6.48E+9
Elevators	2.50E-5	2.50E-5	2.40E-5	2.30E-5	**2.20E-5**
House 8L	1.32E+9	**1.14E+9**	1.23E+9	1.29E+9	1.53E+9
House 16H	1.56E+9	**1.57E+9**	1.59E+9	1.56E+9	1.75E+9
Mv Delve	18.04E+0	**17.22E+0**	18.26E+0	17.64E+0	53.99E+0
Pol	273.63E+0	**249.65E+0**	250.07E+0	269.13E+0	1178.83E+0
Wind	48.89E+0	48.89E+0	48.89E+0	48.70E+0	**21.91E+0**
Winequality	**0.52E+0**	**0.52E+0**	0.53E+0	**0.52E+0**	0.58E+0

A.6 Additional Learning Curves for Section 8.4



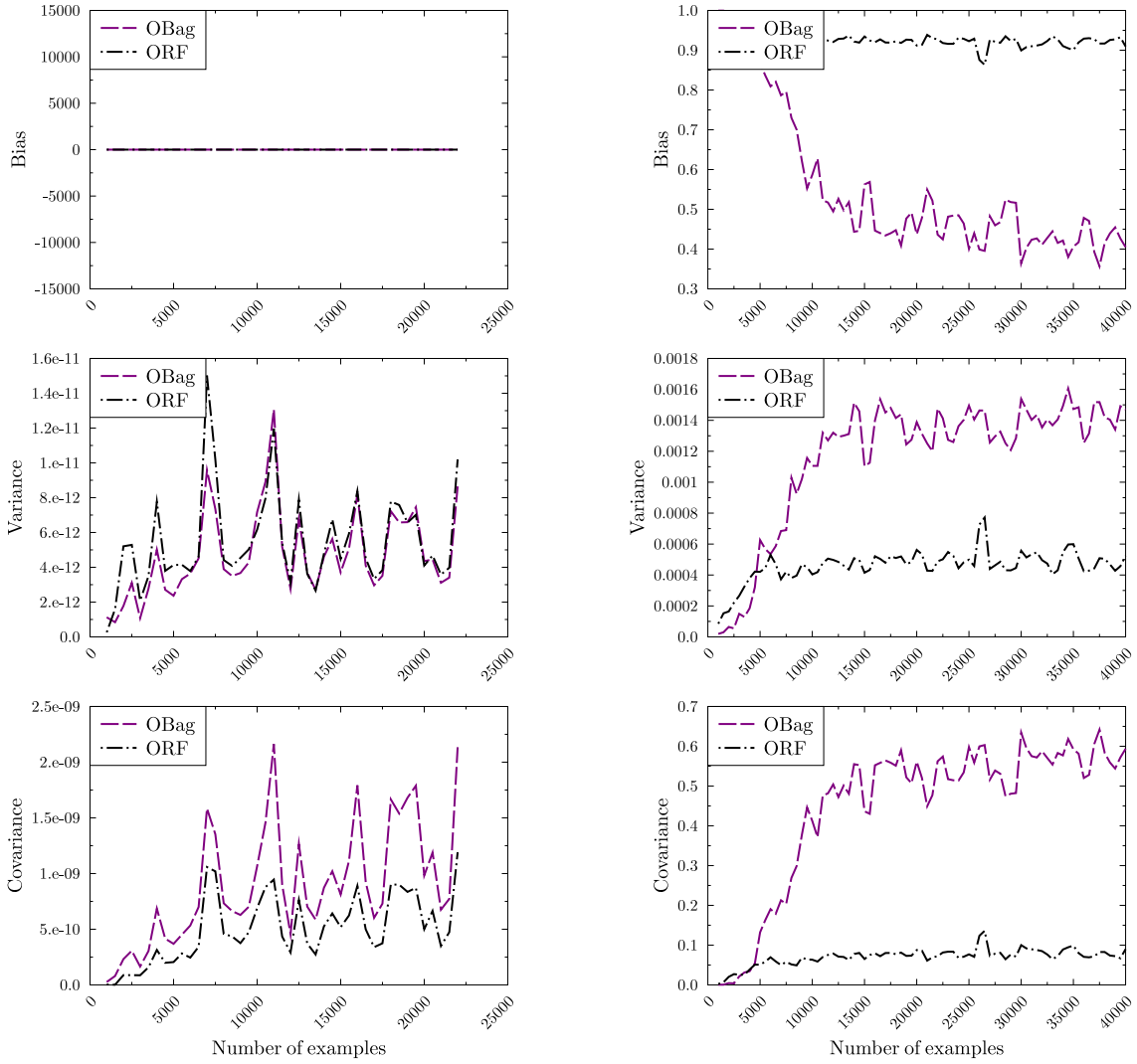
(a) Abalone (b) Cal Housing

Figure 41: Relative bias, variance and covariance error components computed using a sliding window bias-variance-covariance decomposition on the datasets: (a) Abalone, (b) Cal Housing.



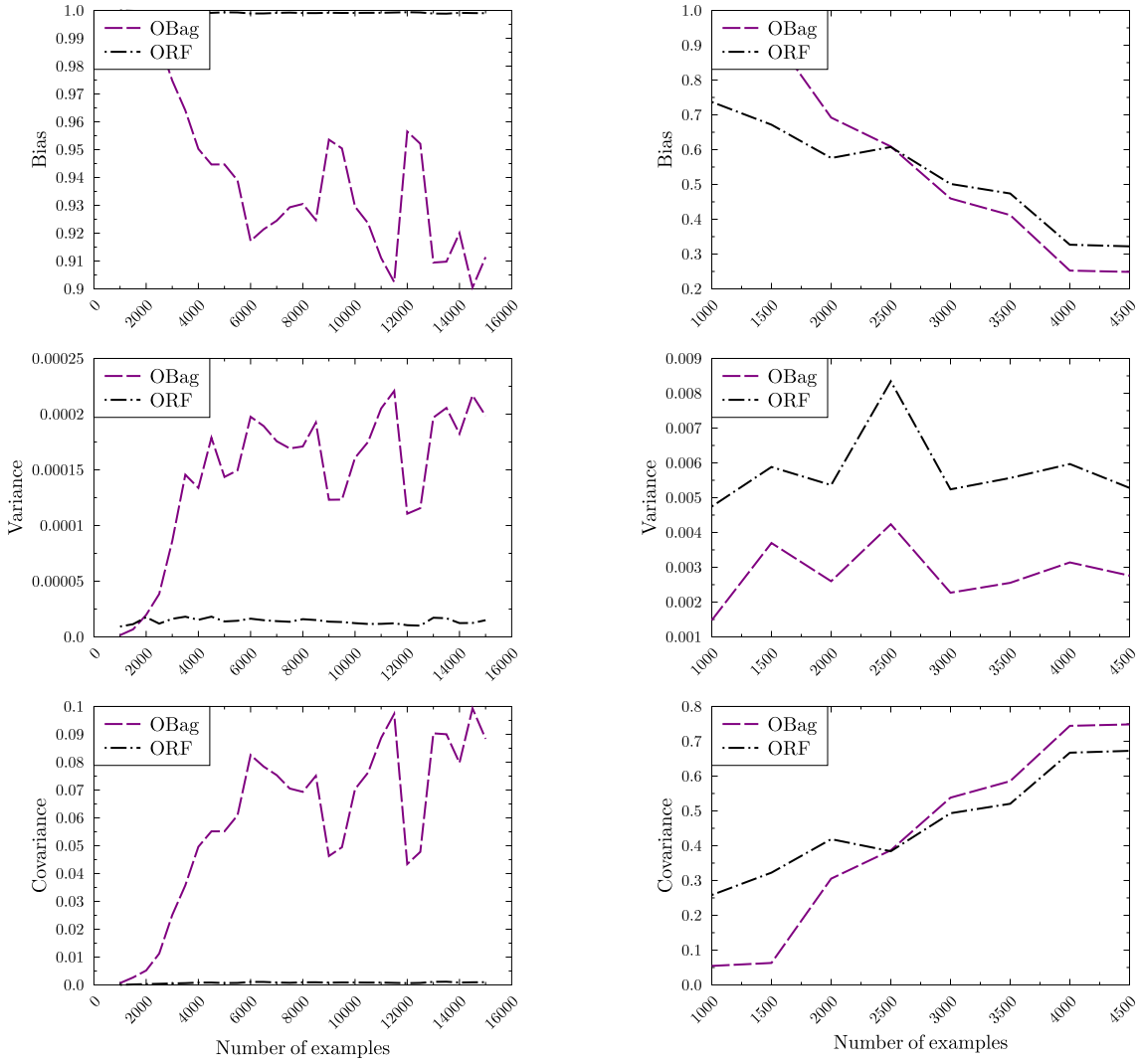
(a) Elevators (b) House 8L

Figure 42: Relative bias, variance and covariance error components computed using a sliding window bias-variance-covariance decomposition on the datasets: (a) Elevators, and (b) House 8L.



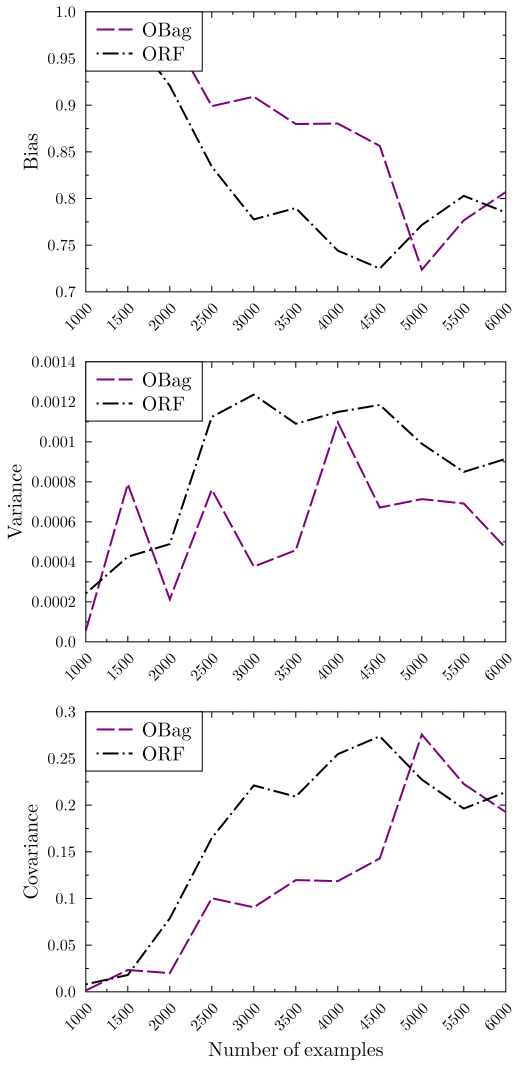
(a) House 16H (b) Mv Delve

Figure 43: Relative bias, variance and covariance error components computed using a sliding window bias-variance-covariance decomposition on the datasets: (a) House 16H, (b) Mv Delve dataset.



(a) Pol (b) Winequality

Figure 44: Relative bias, variance and covariance error components computed using a sliding window bias-variance-covariance decomposition on the datasets: (a) Pol and (b) Winequality.



(a) Wind

Figure 45: Relative bias, variance and covariance error components of the single-tree method FIMT-DD and the different ensemble methods computed using a sliding window bias-variance-covariance decomposition on the Wind dataset.

B Bibliography

B.1 Publications Related to the Thesis

B.1.1 Original Scientific Articles

1. Ikonomovska, E.; Gama, J. Learning model trees from data streams. *Lecture Notes in Computer Science*, 5255: 52-63 (2008).

2. Ikonomovska, E.; Gama, J.; Sebastião, R.; Gjorgjevik, D. Regression trees from data streams with drift detection. In: *Lecture Notes in Computer Science*, 5808: 121-135 (2009).

3. Ikonomovska, E.; Gama, J.; Džeroski, S. Learning model trees from evolving data streams. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1): 128-168 (2011). JCR IF = 1.545.

B.1.2 Published Scientific Conference Contributions

1. Ikonomovska, E. Regression on evolving multi-relational data streams. In: *Proceedings of the 2011 Joint EDBT/ICDT Ph.D. Workshop*, 1–7 (ACM, New York, 2011).

2. Ikonomovska, E.; Driessens, K.; Džeroski, S., Gama, J. Adaptive windowing for online learning from multiple inter-related data stream. In: *Proceedings of Data Mining Workshops - 11th IEEE International Conference on Data Mining* 697–704 (IEEE Computer Society, Washington, DC, 2011).

3. Ikonomovska, E.; Gama, J.; Džeroski, S. Incremental multi-target model trees for data streams. In: *Proceedings of 26th ACM Symposium on Applied Computing*, 988–993 (ACM, New York, 2011).

4. Ikonomovska, E.; Gama, J.; Ženko, B.; Džeroski, S. Speeding up Hoeffding-based regression trees with options. In: *Proceedings of the 28th International Conference on Machine Learning* 537–552 (Omnipress, Madison, WI, 2011).

5. Ikonomovska, E.; Gama, J.; Džeroski, S. Incremental option trees for handling gradual concept drift. In: *Proceedings of the ECML/PKDD International Workshop on Handling Concept Drift in Adaptive Information Systems* ,17–27 (Polytechnic University of Catalonia, Barcelona, 2010).

6. Ikonomovska, E.; Loskovska, S.; Gjorgjevik, D. A survey of stream data mining. In: *Proceedings of 8th National Conference on Electronics, Telecommunications, Automatics and Informatics, with International Participation* **16**, 1–4 (Združenie za elektronika, telekomunikacii, avtomatika i informatika na Republika Makedonija, Skopje, Makedonija, 2007).

B.2 Publications not Related to the Thesis

B.2.1 Published Scientific Conference Contributions

1. Ikonomovska, E.; Chorbev, I.; Gjorgjevik, D.; Mihajlov, D. The adaptive tabu search and its application to the quadratic assignment problem. In: *Proceedings of 9th International Multiconference Information Society*, 27–30 (Jožef Stefan Institute, Ljubljana, 2006).

2. Ikonomovska, E.; Gjorgjevik, D. A data mining approach to performance refinement of a heuristic algorithm. In: *8th National Conference with International Participation* **16**, 5–10 (Združenie za elektronika, telekomunikacii, avtomatika i informatika na Republika Makedonija, Skopje, Makedonija, 2007).

3. Madžarov, G.; Gjorgjevik, D.; Ikonomovska, E. Stebla od mašini so nosečki vektori za prepoznavanje na primeroci. In: *8th National Conference with International Participation* **16**, 11-14 (Združenie za elektronika, telekomunikacii, avtomatika i informatika na Republika Makedonija, Skopje, Makedonija, 2007).

4. Ikonomovska, E.; Delev, T.; Dimov, Z.; Gjorgjevik, D. Web sistem za elektronsko podnesuvanje i recenziranje na trudovi na naučna konferencija. *8th National Conference with International Participation* **17**, 5–10 (Združenie za elektronika, telekomunikacii, avtomatika i informatika na Republika Makedonija, Skopje, Makedonija, 2007).

B.3 Articles Pending for Publication Related to the Thesis

1. Ikonomovska, E.; Gama, J.; Džeroski, S. Option trees and tree-based ensembles for regression on data streams. Journal of Machine Learning Research. [Preparing a revised version. To be resubmitted for review.]

2. Ikonomovska, E.; Zelke, M. Algorithmic techniques for processing data streams. In: *Dagstuhl Follow-Up Series*. [Under review.]

C Biography

Elena Ikonomovska is born on 30.11.1982 in Bitola, R. Macedonia. She attended secondary school in Resen and finished gymnasium majoring in natural sciences and mathematics. In 2001 she started her undergraduate studies at the Faculty of Electrical Engineering and Information Technologies, University "Ss. Cyril and Methodius" in Skopje, R. Macedonia. She was enrolled in a 9 semester BSc program in the area of Computer Engineering, Information Science and Automatics. She finished her undergraduate study in 2006 with an average grade of 9.81 (with 10 being the highest grade) and third best in her generation. She defended her BSc thesis titled "Algorithm Tabu Search" in July 2006, under the supervision of Prof. Dr. Dragan Mihajlov. The next semester, in 2006 she started her postgraduate studies at the same faculty. She was enrolled in a 4 semester MSc program in the area of Computer Technology and Informatics. She finished her postgraduate studies in 2009 with an average grade of 10.00 (with 10 being the highest grade). She defended her MSc thesis titled "Efficient Incremental Learning of Regression Trees and Model Trees from Data Streams" in July 2009, under the supervision of Prof. Dejan Gjorgjevik and co-supervision of Prof. João Gama. In the fall of 2009, she started her doctoral studies at the Jožef Stefan International Postgraduate School, in Ljubljana, R. Slovenia. She is enrolled in the third-level PhD program entitled "Information and Communication Technologies" under the supervision of Prof. Dr. Sašo Džeroski and co-supervision of Prof. Dr. João Gama, from the LIAAD-INESC Porto L.A. Lab.

During the preliminary, secondary, undergraduate and postgraduate studies she continuously held a state scholarship for talented students (awarded by the Ministry of Education of Macedonia). During her undergraduate studies she successfully participated at several math, physics, and informatics competitions. She won a 1st place at the National competition in physics in 1998, 3rd place at the National competition in physics in 1999, several recognitions at national competitions in math and informatics, and several 1st places at regional competitions in physics.

In the period between January 2006 and July 2006 she was working as a junior teaching assistant at the Faculty of Electrical Engineering and Information Technologies in Skopje, Macedonia. From October 2006 till October 2009 she held a new position as a junior teaching and research assistant at the same faculty. Currently she holds a research assistant position at the Department of Knowledge Technologies (E-8) at the Jožef Stefan Institute, Ljubljana, Slovenia, funded by the Slovenian Research Agency (ARRS).

Her research interests focus on mining massive and streaming data, and include approximate aggregate queries on data streams, on-line regression, learning from evolving data streams, multi-relational data streams, and learning from multiple information sources. She is currently involved in several research projects: Advanced Machine Learning Methods for Automated Modeling of Dynamic Systems, Knowledge Discovery from Ubiquitous Data Streams (PTDC/EIA/098355/2008) and Large Scale Information Extraction and Integration Infrastructure for Supporting Financial Decision Making (FIRST - http://project-first.eu/).

In the period from January 2008 to May 2008 she spent three and a half months visiting the LIAAD-INESC Porto L.A. lab under the supervision of Prof. Dr. João Gama. During

this stay she was involved in the European Coordinated Action KDubiq (IST/FET 021321) aimed at discovering new challenges for knowledge discovery (data mining) in the fields of ubiquitous computing. In 2010 she spent one month on a research stay at the LIAAD-INESC Porto L.A. Lab. as part of her doctoral studies. In 2010 she visited the Catholic University of Leuven and in 2011 she spent one month on a research stay at the robotics laboratory (SwarmLab) at the Department of Knowledge Engineering (DKE), Maastricht University, where she initiated collaboration on common research interests.

During her postgraduate and doctoral studies she attended the 2nd European Summer School on Knowledge Discovery for Ubiquitous Computing held at the University of Porto, 2-9 March 2009 in Porto, and the Advanced International Doctoral School on Data Exchange, Integration and Streams held in Dagstuhl, Germany, from 7 - 12 November 2010. She has attended and given talks at several conferences: IMCL 2011, EDBT/ICDT 2011, ACM SAC 2011, ECML-PKDD 2010, DS 2009, DS 2008, ETAI 2007.

She was in the organization committee of the 8th and the 9th National conferences with international participation (ETAI07 and ETAI09). She has also served as PC member for several international workshops and symposiums: KDBI@EPIA2011, HIMoA 2011, ACM SAC 2011 and SAC 2012 Data Streams Track, and UDM 2010.