

Tampereen teknillinen yliopisto. Julkaisu 1015  
Tampere University of Technology. Publication 1015

Timo Aho

## **Steps on Multi-Target Prediction and Adaptability to Dynamic Input**

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 27<sup>th</sup> of January 2012, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology  
Tampere 2012

ISBN 978-952-15-2725-8  
ISSN 1459-2045

# Abstract

How should we react to dynamically changing inputs in various areas of computer science? This is one of the main questions we discuss in this thesis. The problem is present both in machine learning environment coping with massive amount of data available today and on the low level programming of computers.

One of the hot topics currently in machine learning are so called ensemble methods. An ensemble model is a collection of multiple divergent, often simple, base models. The variance of base models has been shown to give clear benefit to the predictive power over using a single model. Not surprisingly, ensemble methods also give new possibilities for coping with dynamic online inputs; we can simply reweight the base models to adapt.

However, in this thesis we are especially interested in ensemble methods in a specific framework. In many practical problems, we have multiple related attributes that need to be predicted. For example, predicting the growth of flora or biological composition of water are tasks that can be presented with multiple attributes that clearly relate to each other. Recently there has been some progress on methods that gain both smaller and more accurate overall models by making use of relations between the predicted attributes. In this thesis, we show that we can achieve both small and accurate models with a rule based ensemble method FIRE. The method is extensively evaluated experimentally.

We also pull some strings together by showing how similar problems have been solved in separate areas of computer science. In machine learning, the problem of dynamically changing input has been studied under a term of concept drift. Similarly in algorithm and data structure analysis a notion of locality of reference has been present for long. We introduce a general framework that covers both of the problems and briefly go through the work done on both of the areas. We hope that by giving pointers to a bridge over the gap between the fields, researchers in both areas could be able to pick up some fruits on the other side.



# Preface

My work on this thesis has been supported in many ways and by many people. First of all I acknowledge the guidance and advice of my supervisor Prof. Tapio Elomaa at Tampere University of Technology (TUT). In addition, during my visit to Jožef Stefan Institute (JSI), Slovenia I got invaluable supervising from Prof. Sašo Džeroski. I also want to thank the opponent Juho Rousu and the thesis reviewers Prof. Hendrik Blockeel and Tapio Pahikkala for insightful and encouraging comments.

I am also very grateful for the help and support offered by my co-authors and colleagues at TUT and JSI. I want to especially thank Jussi Kujala, Bernard Ženko and Martin Žnidaršič for all kind of supporting discussions. Moreover, I want to thank Janne Lautamäki, Henri Hansen, Panče Panov, Prof. Tommi Mikkonen, my brother Eero and many others for similar reasons.

I am deeply indebted to Kirsti and Sami for help I did not consider possible. Finally, I thank my dear wife Leena and our lovely daughters Aleksandra, Sonja and Nadja for giving meaning to all this.

Timo Aho, 15.12.2011, Tampere, Finland

*Dedicated to my children, and to the child.*

*χαρις τω θεω επι τη ανεκδιηγητω αυτου δωρεα*

## Some Further Acknowledgments

The scientific work that led to this Ph.D. thesis was primarily supported by Tampere Doctoral Programme in Information Science and Engineering (TISE) and multiple Academy of Finland projects: Intelligent Online Data Structures (INTENTS), Approximation and Learning Algorithms (ALEA) and Reactive learning and mining (REALM). Additionally Department of Software Systems in Tampere University of Technology helped in economic and practical ways.

Additional funding came from smaller scholarships by following instances: Finnish Foundation for Technology Promotion, Tampere University of Technology Grant for Scientific Visit, Industrial Research Fund at the Tampere University of Technology—Tuula and Yrjö Neuvo Foundation, Ulla Tuominen Foundation and Nokia Foundation.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>Contents</b>	<b>v</b>
<b>List of Publications</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 The Structure of the Thesis . . . . .	2
1.2 Summary of the Main Contribution . . . . .	4
<b>2 Background</b>	<b>7</b>
2.1 Areas of Machine Learning . . . . .	7
2.1.1 Error and Loss Functions . . . . .	10
2.1.2 Multi-Target Loss . . . . .	12
2.1.3 Model Families for Machine Learning . . . . .	14
2.2 Dynamic Input . . . . .	16
2.2.1 Concept Drift in Online Learning . . . . .	16
2.2.2 Locality of Reference on Data Structures . . . . .	17
<b>3 Regularization in Convex Optimization</b>	<b>19</b>
3.1 Regularization . . . . .	20
3.2 Gradient Directed Optimization . . . . .	22
3.3 Efficient Regularization in Support Vector Machines . . . . .	24
3.3.1 Background . . . . .	24
3.3.2 Reweighting Ridge Regularized SVM . . . . .	25

---

<b>4</b>	<b>Rule Based Ensembles</b>	<b>27</b>
4.1	On the Background of the Rule Ensembles . . . . .	27
4.2	Rule Ensembles in Single Target Regression . . . . .	28
4.3	Learning Rule Based Ensembles for Multi-Target Regression . . .	32
4.3.1	Generation of Base Models . . . . .	34
4.3.2	Optional Linear Terms . . . . .	37
4.3.3	Gradient Directed Weight Optimization . . . . .	38
4.4	Empirical Evaluation . . . . .	41
4.4.1	Experimental Setting . . . . .	41
4.4.2	Results . . . . .	44
4.4.3	Analysis of Normalization and the Use of Linear Terms . .	51
4.5	Conclusions on the Multi-Target Rule Ensemble Method . . . . .	55
<b>5</b>	<b>Binary Search Trees in Online Context</b>	<b>57</b>
5.1	Splay Trees in Theory . . . . .	58
5.2	Practical Efficiency of Splay Trees . . . . .	60
5.3	Conditional Adaptation of a BST . . . . .	62
5.4	A Dynamic Version of WSPLAY . . . . .	63
<b>6</b>	<b>Working Sets as Drifting Concepts</b>	<b>67</b>
6.1	Working Sets and Locality Phases . . . . .	67
6.2	Concept Drift . . . . .	69
6.3	The Simple Generalized Problem . . . . .	71
6.3.1	Relation to LR and CD . . . . .	72
6.3.2	More Previous Work on the Problem . . . . .	73
6.4	Further Generalizations . . . . .	77
6.5	On Locality of Reference and Drifting Concepts . . . . .	78
<b>7</b>	<b>Conclusions</b>	<b>81</b>
	<b>Bibliography</b>	<b>83</b>

# List of Publications

This thesis is a compound of the following publications. The publications are referred to in text with Roman numerals [I]–[V]. The permissions of the copyright holders of the original publications to reprint them in this thesis are hereby acknowledged.

- [I] Jussi Kujala, Timo Aho, and Tapio Elomaa. A walk from 2-norm SVM to 1-norm SVM. In Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *Proceedings of the Ninth International Conference on Data Mining (ICDM 2009)*, pages 836–841, 2009. IEEE Computer Society, Los Alamitos, CA.
- [II] Timo Aho, Bernard Ženko, and Sašo Džeroski. Rule ensembles for multi-target regression. In Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *Proceedings of the Ninth International Conference on Data Mining (ICDM 2009)*, pages 21–30, 2009. IEEE Computer Society, Los Alamitos, CA.
- [III] Timo Aho, Bernard Ženko, Sašo Džeroski, and Tapio Elomaa. Learning Multi-Target Regression with Rule Ensembles. Technical Report 18, Department of Software Systems, Tampere University of Technology, Tampere, Finland, 2011.
- [IV] Timo Aho, Tapio Elomaa, and Jussi Kujala. Reducing splaying by taking advantage of working sets. In Catherine C. McGeoch, editor, *Proceedings of the Seventh International Workshop on Experimental Algorithms (WEA 2008)*, volume 5038 of LNCS, pages 1–13, 2008. Springer, Berlin/Heidelberg, Germany.

- [V] Timo Aho, Tapio Elomaa, and Jussi Kujala. Unsupervised classifier selection based on two-sample test. In Jean-François Boulicaut, Michael R. Berthold, and Tamás Horváth, editors, *Proceedings of the 11th International Conference on Discovery Science (DS 2008)*, volume 5255 of LNAI, pages 28–39, 2008. Springer, Berlin/Heidelberg, Germany.

# Chapter 1

## Introduction

An often repeated slogan in computer science community is the increased volume of information in multiple areas of life. For example, the amount of network traffic and the number of computers, web pages, IP traffic, published scientific papers, news reports, mobile network devices has been growing acceleratingly. In many cases, it is not possible to go through all the data manually and we, hence, need automated services that compress out the essential information.

One of the research areas to find out meaningful *patterns* in an ocean of data is machine learning. Originally, the idea of machine learning was to automatically create rules for the intelligent behavior of algorithms. However, nowadays machine learning is an umbrella term consisting of varying kinds of methods. One of its most studied subareas is predictive learning on which we mostly concentrate in this thesis.

In addition to learning, we are studying a separate but related area of computer science: adaptive, or self-adjusting, *data structures*. We now illustrate the content of both the areas with an example from mining industry. Assume we are given a bunch of data which is represented by an ore block. We want to extract some information, precious minerals, out of it.

With the predictive machine learning methods, we aim to figure out the information based on the observable nature of the data. That is, by detecting the mineral veins on the surface of the ore we predict if the block is rich enough for further processing. The reasoning is based on our prior deep and thorough analysis of similar ore blocks.

On the other hand, in the adaptive online data structure framework we are interested in somewhat different things. For instance, we want to sort and

store the ore blocks based on their most common material. Thus, we can later easily get the blocks that have a lot platinum. In an online environment, we do not specifically know the ore usage beforehand. Instead, the decision of needed materials is made simultaneously with mining. Thus, we need adaptive solutions to keep the storage in good order in all situations.

## 1.1 The Structure of the Thesis

As mentioned, in this thesis we concentrate mostly on some predictive machine learning methods and adaptive online data structures. We start the thesis in Chapter 2 by introducing the basic terms of the areas.

In predictive learning, we are trying to predict some unknown attributes for new data instances based on our prior knowledge. In our quest for solving learning problems, we have a couple of subtasks: for example, we have to choose a suitable method, its parameters and make some more implementation level choices like how possible optimization is executed. All this affects the type of resulting model, its accuracy and understandability.

In fact, in addition to predictions, we can often use the learned model itself as a source of information. Nevertheless, to gain accurate predictions the model has to find and compress out some essential relationships inside the data. Thus, the aim in learning can be two-fold: predictive accuracy and presentation of complex relationship information. Unfortunately, two aims seem often to be contradictory. The most accurate predictive models tend to be quite complex and not easily presentable to a human reader.

Because of this, we have to choose our method carefully based on our needs. One of the modern accurate approaches is the so-called *support vector machine* (SVM) [Vapnik, 2000]. It is based on finding a maximal margin linear function between two separate sets in some carefully chosen space. SVM is a good example for case, where study of different optimization types gives still fruitful results [e.g., Chapelle, 2007; Hsieh et al., 2008; Lin et al., 2007; Masnadi-Shirazi and Vasconcelos, 2010]. In addition to efficiency, we are interested in the properties of the resulting model. One way to affect the behavior and size of the model is to choose the regularization of optimized function. In Chapter 3 we present, our study on creating more static, and thus possibly more reliable models without losing in efficiency. More precisely, we study a technique of getting approximately L1 regularized results by running efficient L2 regularized SVM a couple of times in a row.

---

Even small SVM models are usually considered quite unreadable for a layman. Thus, if we want to base our decision making on the information collected by a machine learning method, SVM is not usually the right choice. Fortunately, we have other techniques for creating more understandable models. Decision rule sets are commonly considered one of the most interpretable ones. However, the traditional methods leave some hope for accuracy. Nevertheless, more accurate versions called rule ensemble methods have been presented [Dembczyński et al., 2008b; Friedman and Popescu, 2005, 2008]. In Chapter 4 we introduce and evaluate our rule ensembles method which is applicable to *multi-target regression* problems. That is, we are simultaneously predicting multiple numeric values instead of the traditional single one.

In machine learning the used computational time is not irrelevant. However, in the area of data structures the problem has been even more present for a long time. This is stressed when the amount of information is vast in comparison with the time we have for processing it. For example, in sublinear time algorithms we are not touching all the data when it is processed. More generally we often can not go through all the past data when a new piece of information is achieved. In our mining example it is clear that handling all the previous ore blocks when storing a new one is out of question. There are plenty of frameworks for studying this kind of situation, where data is inserted and accessed piece by piece. This is different from the traditional offline framework where we have all the material available from the beginning.

One of the older frameworks, called online algorithms [Albers, 2006], limits the time usage to linear. This means that we should not use more than constant time processing each of the data units. Or, better yet, we should touch each unit only once during our scan. However, in the newer data stream framework we are aiming for even sublinear time and storage usage [Muthukrishnan, 2005].

If the data is achieved in online fashion, the distribution of the data may change during the progress. This happens even if the problem we are trying to solve stays intact. This change may happen either abruptly or gradually. In our mining example this could mean reaching a new rock layer in mining or slow change in ore composition.

In the area of algorithms and data structures we call this change with terms like *locality of reference* or appearance of *working sets* [Denning, 2005]. In solving even the basic problems like data storage, searching or sorting efficiently, it could benefit us to notice the existence of this phenomenon. That is, we should realize that our model of the incoming distribution is not perfect forever and some online adaptivity to input is needed. In our example it is clear that detecting an increase in amount of platinum concentrated blocks allows us to

sort them out more efficiently. Also it is natural that all the stored platinum ore blocks are used at once and thus they define a working set. In Chapter 5, we present our binary search tree algorithm that is designed to cope with occurring working sets.

A similar phenomenon has been detected in machine learning. Here we are talking about *concept drift* [Tsymbal, 2004]. In this case, the phenomenon may be even more important to notice because instead of just slowing down the process, failure leads to the decay of the current model. This, on the other hand, means we are failing in our both predictive learning aims. We would find ourselves making decisions based on outdated data. If we do not notice the concentration of mined ore changing, it may be very disastrous for our search for economically interesting materials. The analogy between concept drift and locality of reference is an interesting problem in its own right. We cover the similarities and study made in the areas in Chapter 6.

Finally, we end with some concluding remarks and future work ideas.

## 1.2 Summary of the Main Contribution

In short, the main contribution of this thesis can be listed as follows:

- In [I] we inspect theoretically and experimentally a previously known method for simulating L1 regularization with clearly more efficient L2 regularization in case of support vector machines. We also give convergence proofs for the method. This work is also examined briefly in Chapter 3.
- In [II, III] and in Chapter 4 we develop and extensively experimentally evaluate a novel rule ensemble algorithm FIRE which is applied to predicting simultaneously multiple target attributes.
- We present an adaptive binary search tree WSPLAY in [IV]. The algorithm is developed to cope with and adapt to online input where the dynamically changing locality of reference occurs. The work is also narrowly touched in Chapter 5.
- On the other hand, in Chapter 6 we examine the similarities of concept drift and locality of reference. In a survey like manner, we show that lots of common work has been done in these completely distinct areas.

- Finally, [V] introduces a metric based method for weighting or selecting machine learning predictive models in a collection. The metric is computed on features gathered during the training phases of models. In some often used cases, e.g. by using polynomial or Gaussian kernels, the weighting can even be approximated much more efficiently than the traditional method. The publication is covered in Chapter 6.

In publications [II, III, IV] and [V] the author gave most of the contribution. However, in [I] the author had mostly a supportive role.



# Chapter 2

## Background

In this chapter, we go through some background in the two areas mentioned in the first chapter.

### 2.1 Areas of Machine Learning

Machine learning is an umbrella term covering techniques for distinct tasks. We now go through a common categorization for it.

In this thesis, we are mainly considering *supervised learning*. In this case, our task is to predict a vector valued function or probability distribution for  $T$  target attributes  $\mathbf{y} = (y_1, y_2, \dots, y_T)^\top \in \mathbb{Y}$  based on known vector  $\mathbf{x} = (x_1, x_2, \dots, x_K)^\top \in \mathbb{X}$  of  $K$  descriptive attributes. The process is divided in two separate phases. In the training phase, we are given a set of training examples  $\mathbb{E}$  of the form  $(\mathbf{x}, \mathbf{y})$ . Our predictive model is created based on these examples. After this, in the test phase we are given only the descriptive vector  $\mathbf{x}$ . Our task is to choose an answer that best estimates and generalizes the original data set  $\mathbb{E}$  distribution. For instance in regression estimation our target attribute prediction  $\hat{\mathbf{y}}$  is a function of the descriptive vector  $\hat{\mathbf{y}} = \mathbf{f}(\mathbf{x})$ . In the traditional supervised learning environment the target vector  $\mathbf{y}$  is scalar and is often referred to as a *label*.

Slightly more formally, Vapnik [1999] defines supervised learning as follows.

**Problem Definition 1.** *Supervised learning has three components:*

1. random vectors  $\mathbf{x}$  drawn independently from a (for the time being) fixed, but unknown distribution  $D(\mathbf{x})$ ,

2. a (hypothetic) supervisor returning target vector  $\mathbf{y}$  for every  $\mathbf{x}$ , according to a (for time being) fixed, but unknown conditional distribution  $D(\mathbf{y}, \mathbf{x})$ , and
3. a set of model functions  $\mathbf{f}(\mathbf{x}; \boldsymbol{\beta})$ , where  $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots)$  is a meta-parameter that identifies the function in the family.

Now our task is to find  $\boldsymbol{\beta}$  so that function  $\mathbf{f}(\mathbf{x}; \boldsymbol{\beta})$  approximates the supervisor's answers as well as possible. This is done by minimizing the error estimate for the function  $f$  over some test set  $\mathbb{E}_{test} \subseteq \mathbb{X} \times \mathbb{Y}$ :

$$ERR(\boldsymbol{\beta}, \mathbb{E}_{test}) = \frac{1}{|\mathbb{E}_{test}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathbb{E}_{test}} L(\mathbf{y}, \mathbf{f}(\mathbf{x}; \boldsymbol{\beta})). \quad (2.1)$$

Here  $|\mathbb{E}|$  is the size of set  $\mathbb{E}$  and  $L$  is a preselected loss function. Low loss function values imply a better model.

Sometimes the distributions  $D(\mathbf{x})$  and  $D(\mathbf{y}, \mathbf{x})$  may be dynamic in nature. We will cover this topic later in Section 2.2.1 and Chapter 6. Moreover, the choice of loss function depends on our goal. Some often used loss functions are introduced in Section 2.1.1.

So, we now have the problem definition: we have to find a good  $\boldsymbol{\beta}$  for some family of functions. How is this done in practice? First of all, we have to choose a suitable family of functions based on the data and especially on the need we have on the task. There are many function family alternatives with different properties like accuracy, efficiency, simplicity etc. depending on the problem at hand. After that, we select a loss function and use some heuristic or optimization method to find the  $\boldsymbol{\beta}$  value that gives the best approximation in the family for the target function. We will cover the procedure in more detail later.

Let us go on to the next category of machine learning. In *unsupervised learning* we work on unlabeled data. That is, we do not have the target vector  $\mathbf{y}$  at all. In this case, we are trying to track the features of data with tools like *clustering* [Berkhin, 2006; Xu and Wunsch II, 2005] and *independent component analysis* [Hyvärinen and Oja, 2000]. In this categorization we usually mention semi-supervised and reinforcement learning. In *semi-supervised learning* [Chapelle et al., 2010] we have some labeled examples but have also hold on a greater set of unlabeled ones. On the other hand, in *reinforcement learning* [Buşoniu et al., 2008] the feedback given by a supervisor is only partial and may, for example, consist only of hints of how well we are doing. In this thesis,

we are mostly addressing the supervised learning as in [I, II, III, V]. However, the method presented in [V] is also available for unlabeled data and thus for unsupervised or semi-supervised learning.

Our goal in all these learning problems can usually be divided in two parts. By modeling the given data, we try to create predictions on it or describe it. As mentioned in Section 1.1, even if we are aiming for the prediction of the unknown target attributes, the model usually also describes the underlying relationships of the data. This makes the descriptive modeling possible. Moreover, the latter aim is very evident in unsupervised learning. The relationship between description and prediction is especially investigated in the predictive clustering framework [Blockeel, 1998; Blockeel and De Raedt, 1998; Blockeel et al., 1998; Ženko, 2007; Ženko et al., 2006].

After selecting the function family, our task is to find a suitable  $\beta$  value by minimizing the error estimate for Equation 2.1 on the facing page. Let us now introduce how this subproblem is usually solved. Our task is to find a value  $\beta^*$  defined by

$$\beta^* = \arg \min_{\beta} \text{ERR}(\beta, \mathbb{E}_{\text{test}}). \quad (2.2)$$

If we can not find an analytic solution, we need to use optimization methods of some kind for this.

Unfortunately the answers given by Equation 2.2 are known to be unstable especially if the sample size  $|\mathbb{E}_{\text{test}}|$  is small in comparison with the number of parameters related to  $\beta$ . Thus, we usually add some static *penalty*, or *regularization* part  $\lambda|\beta|^\alpha$  that stabilizes the results. That is, we rather solve the problem

$$\beta^* = \arg \min_{\beta} \text{ERR}(\beta, \mathbb{E}_{\text{test}}) + \lambda \sum_i |\beta_i|^\alpha. \quad (2.3)$$

Here with  $\lambda \geq 0$  we can tune the effectiveness of regularization. In addition,  $\alpha \geq 0$  affects the type of regularization and, hence, the characteristic of solution  $\beta^*$ .

Thus the optimization problem usually consists of two parts: the actual loss function  $L$  and the regularization part. For regularization popular values for  $\alpha$  include  $\alpha = 2$  (called L2 or ridge regularization) [Hoerl and Kennard, 1970; Vapnik, 2000] and  $\alpha = 1$  (L1 or lasso regularization) [Donoho and Johnstone, 1994; Tibshirani, 1996]. The values have been shown to have interesting effects on the resulting model. With regularization, we can affect, for example, the *overfitting* nature of the model. Overfitting roughly means that the model is based too deeply in details of training data and thus it loses the ability to generalize the predictions outside the set. We will cover the regularization more

in detail in Chapter 3. However, we discuss the loss functions in slightly more detail in the next few sections.

### 2.1.1 Error and Loss Functions

In an optimization task the loss function, as in Equation 2.3 on the preceding page, maps a solution candidate to its quality. Thus, in optimization our duty is to choose a loss function that gives small value to good solutions and larger one for bad solutions. The loss function, hence, substantially affects the optimization result.

A significant feature of a loss function is its possible *convexity*.<sup>1</sup> For convex loss functions every local minimal point is also a global minimal point. However, it is worth noting that there may be also non-convex functions with a similar property.

A simple proof of convexity for twice differentiable function  $f$  is to show that its second derivative is always non-negative:  $f''(x) \geq 0$ . An optimization problem with convex functions can be solved with simple and efficient local optimization methods; whenever we end up in local minimum, we also have a global minimum. Nevertheless, for many methods also differentiability is needed. There are also convex functions like  $|x|$  that are not differentiable. For more background on convex optimization, please see any optimization book [e.g., Jarre and Vavasis, 2010].

We now go through some very often used examples of loss functions for a couple of tasks. In supervised learning a common task is that of regression estimation. In this case, the supervisor in Problem 1 on page 7 is simply a real valued function  $\mathbf{y} = \mathbf{F}(\mathbf{x}) \in \mathbb{R}^T$  which may or may not belong to our set of model functions  $\mathbf{f}(\mathbf{x}; \boldsymbol{\beta})$ . In the latter case, we are trying to get as good an approximation as possible.

Let us first assume that the supervisor function and model functions are scalar valued, and mark them with  $y = F$  and  $f$  in this order.

The most widely used loss function for regression tasks is the squared loss

$$L_{\text{squared}}(y, f(\mathbf{x}; \boldsymbol{\beta})) = \frac{1}{2}(y - f(\mathbf{x}; \boldsymbol{\beta}))^2.$$

Especially for an interesting and common class of models, namely those that depend linearly on  $\boldsymbol{\beta}$ , the squared loss function has the nice property of being

---

<sup>1</sup>To be exact, we should talk about error function convexity, because error is the quantity we are minimizing. However, the properties are equivalent for most errors.

convex. The convexity over any weight  $\beta_j$  can be seen from the following:

$$\frac{\partial^2 L_{\text{squared}}(y, f(\mathbf{x}; \boldsymbol{\beta}))}{\partial^2 \beta_j} = \underbrace{\left( \frac{\partial f(\mathbf{x}; \boldsymbol{\beta})}{\partial \beta_j} \right)^2}_{\geq 0} + (y - f(\mathbf{x}; \boldsymbol{\beta})) \underbrace{\frac{\partial^2 f(\mathbf{x}; \boldsymbol{\beta})}{\partial^2 \beta_j}}_{=0} \geq 0.$$

Squared loss is known to be vulnerable to extreme, possibly defective, data points. Thus, robust losses have been presented [Huber, 1964].

Moreover, clear deficiency in error based on squared loss is that the value depends on the scale of variables: if the target variable  $y$  has values in a narrow interval like  $y \in [0, 1]$  we can not compare the loss function value to another target value that has values from a wider interval. This property makes squared error values in Equation 2.1 on page 8 incomparable between tasks.

As a solution, error functions like relative root squared error (RRSE), also called relative root mean squared error (RRMSE), have been presented. Let us assume that error is computed over the set  $\mathbb{E} \subseteq \mathbb{X} \times \mathbb{Y}$ . In this case RRSE is defined by

$$\text{ERR}_{\text{RRSE}}(\boldsymbol{\beta}, \mathbb{E}) = \sqrt{\frac{\sum_{(\mathbf{x}, y) \in \mathbb{E}} (y - f(\mathbf{x}; \boldsymbol{\beta}))^2}{\sum_{(\mathbf{x}, y) \in \mathbb{E}} (y - \bar{y})^2}}.$$

Here  $\bar{y}$  is the average of attribute  $y$  over the set  $\mathbb{E}$ . RRSE compares the squared error of a model with a constant model that always predicts the mean value of the target attribute  $y$ . Thus, RRSE values over 1 mean that our model is worse than a constant prediction. Because of these features, RRSE values are somewhat more expressive and are often used for error reporting [II, III].

Another common task in supervised learning is classification. The only difference with regression estimation is that now the supervisor function is nominal valued instead of real scalar. The function co-domain may be, e.g., the set of integers  $y = F(\mathbf{x}) \in \mathbb{Z}$ . In this thesis, we concentrate mostly on regression but cover briefly a common special case of binary function here. From this, there are plenty of generalizations to multiple label case [Elisseeff and Weston, 2001; Fürnkranz et al., 2008]. In binary classification the target function  $y = F(\mathbf{x})$  may have only two different values from set  $\{-1, 1\}$ . In these cases, 0/1-loss, which simply computes the amount of right answers, is traditionally used:

$$L_{0/1}(y, f(\mathbf{x}; \boldsymbol{\beta})) = \frac{1}{2} \left( 1 - y \frac{f(\mathbf{x}; \boldsymbol{\beta})}{|f(\mathbf{x}; \boldsymbol{\beta})|} \right).$$

For predictive function  $f(\mathbf{x}; \boldsymbol{\beta})$  we simply interpret all positive values as label 1 and negative values as  $-1$ .

Unfortunately, this loss function is not convex and thus losses like the hinge loss

$$L_{\text{hinge}}(y, f(\mathbf{x}; \boldsymbol{\beta})) = \max\left(0, \frac{1}{2}|1 - yf(\mathbf{x}; \boldsymbol{\beta})|\right),$$

have been put forward. Also other convex classification loss functions like logarithmic loss exist.

### 2.1.2 Multi-Target Loss

As mentioned, in the traditional machine learning setting one predicts the value of a single target attribute; categorical or a numeric one. A natural generalization of this setting is to predict multiple target attributes simultaneously [Blockeel et al., 1998; Caruana, 1997]. A typical example from the environmental sciences, is the task of predicting species distribution or community structure [Demšar et al., 2006], where we are interested in predicting the abundances of a set of different species living in the same environment. These species represent the target attributes, which might, but need not be related. Examples from other areas, ranging from natural language processing to bioinformatics and medicine [Bickel et al., 2008; Jeong and Lee, 2009; Liu et al., 2010] are also plentiful.

If our only goal is to achieve high predictive accuracy, a collection of single target models may suffice to solve the problem. However, if we are also interested in the interpretability of the model, the collection of single target models is much more complex and harder to interpret than a single model that jointly predicts all target attributes [Blockeel, 1998; Suzuki et al., 2001; Ženko and Džeroski, 2008]. An additional benefit of the *multi-target* models is that they are frequently more accurate than the corresponding collections of single target models [Blockeel, 1998; Caruana, 1997; Kocev et al., 2007; Suzuki et al., 2001].

Several standard learning methods such as neural networks [Caruana, 1997], decision trees [Blockeel et al., 1998], model trees [Appice and Džeroski, 2007], classification rules [Suzuki et al., 2001; Ženko and Džeroski, 2008], random forests [Kocev et al., 2007] and regression rule ensembles [II, III] have been extended towards multi-target prediction.

An approach related to multi-target learning is *multi-task learning* [Argyriou et al., 2008; Caruana, 1997; Chapelle et al., 2011]. In multi-task learning our aim is to learn multiple single-target learning tasks with different training sets (and even features) at the same time. This way, like multi-target prediction, multi-task learning should be able to benefit from relationships between tasks.

As a result from multi-task training we get a separate trained model for each of the tasks.

There are some clear differences between multi-target and multi-task learning. The most obvious one between the problem domains is the amount of trained models: a separate model for each of the tasks or single model trained for the whole problem. Moreover, we recall from Section 2.1 the parts of a learning problem. Multi-task learning aims to predict the target features and also describe their relationship with the descriptive features. However, the multi-task model does not necessary aim to describe the relationships between the target features. The last aim, nevertheless, is exactly one of the main points in multi-target learning.

Nevertheless, it is true that the domains have some common background. This can be seen, for instance, in the similar decision practices of the learning algorithms in multi-target [II, III] and multi-task [Chapelle et al., 2011] domains. Anyway, the areas have to be considered separate and we concentrate on multi-target learning in the rest of the thesis.

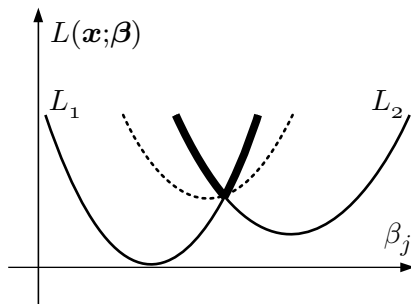


Figure 2.1: Illustration of aggregate losses max (bold line) and avg (dashed line) in case of two targets.

Back to multi-target learning and loss functions applicable to it. All the loss functions presented in Section 2.1.1 are intended for scalar valued target functions only. In case of multiple targets, we have to use some kind of aggregate. To keep the aggregate of loss functions convex, we have at least two simple possibilities, namely taking mean or maximum over single target loss functions. Let us denote with  $L_t$  a loss function for the  $t$ -th coordinate  $y_t$  of  $T$  dimensional target vector  $\mathbf{y}$ . The situation is presented in Figure 2.1. Now, the proposed

aggregate loss functions would take the form

$$L_{\text{avg}}(\mathbf{y}, \mathbf{f}(\mathbf{x}; \boldsymbol{\beta})) = \frac{1}{T} \sum_{t=1}^T L(y_t, f_t(\mathbf{x}; \boldsymbol{\beta}))$$

$$L_{\text{max}}(\mathbf{y}, \mathbf{f}(\mathbf{x}; \boldsymbol{\beta})) = \max_{t=1 \dots T} L(y_t, f_t(\mathbf{x}; \boldsymbol{\beta})).$$

We could assume the first aggregation  $L_{\text{avg}}$  to be more suitable. First of all, even if  $L_{\text{max}}$  is may be continuous and piecewise differentiable, it is not usually differentiable. Thus, the behavior of many optimization methods, especially those based on the gradient, is unpredictable near the discontinuity points. Those are naturally candidates as minimum. In practice this would cause oscillation near the optimums. Moreover, in case of the gradient based methods we have to compute explicit loss function values in addition to the gradients. For  $L_{\text{avg}}$  gradient computation is always enough.

The aggregates also treat the targets somewhat differently. If one of the targets is more difficult to optimize than the others—i.e., has greater loss—the optimization with  $L_{\text{max}}$  is totally dominated by it. This could, of course, be the desired behavior. Nevertheless,  $L_{\text{avg}}$  gives some effect on all the targets even if a more difficult one may still be dominating. Naturally the domination for both loss functions can be prevented with a suitable normalization of data.

In any case, we want to report errors over multi-target data sets with RRSE. For this we have two possibilities: On one hand, we can treat each of the target attributes of all data sets as an independent measurement. The argument against this option is that target attributes within one data set are probably not independent and as a result our statistical test will show more significant differences than there actually are. On the other hand, we can compute the average over all targets within each data set and consider such averages as independent measurements. The argument against the second option is that by computing target averages we are actually “summing apples and oranges” and the resulting average is probably not a valid quantity. In the absence of a better available solution, our suggestion is to present statistical tests for both options. This is what we did in our experiments [II, III].

### 2.1.3 Model Families for Machine Learning

In Section 1.1, we mentioned the existence of differing kinds of machine learning methods. The choice of method type will set our possible models, that is, the model set  $\mathbf{f}(\mathbf{x}; \boldsymbol{\beta})$ . In this section, we will briefly mention a couple of methods that will be used later in the thesis.

---

One of the modern accurate approaches is the so-called *support vector machine* (SVM) [Vapnik, 2000]. It is based on finding a maximal margin hyperplane between two sets in some carefully chosen space. In Chapter 3, we present our study on creating less overfitting models without losing in the efficiency of the optimization process. This is done by simulating a lasso regularization by running a more efficiently solvable problem with ridge regularization.

As already stated in Section 2.1, in addition to predictions, we can often use the learned model itself as a source of information. In this thesis, we are especially interested in one property of the model family  $f(\mathbf{x}; \boldsymbol{\beta})$ : understandability or interpretability for the human reader. That is, when a predictive model has been trained, how well a layman can understand the reasoning behind the predictions. Naturally, the size of the model has an effect on this but the model type is an even greater factor.

Unfortunately the most accurate predictive models like SVM tend to be quite complex and not easily presentable to a human reader: even small SVM models can be considered quite unreadable for a layman. SVM model consists of weights for summation over inner products. Thus, if we want to base our decision making on the information collected by a machine learning method, SVM is not usually the right choice.

Fortunately, we have other techniques for creating more understandable models. Decision rule sets [Flach and Lavrač, 2003] are commonly considered one of the most interpretable ones. The decision rules are based on simple **IF** <condition> **THEN** <prediction> type conditional rules that give the prediction if the conditions are met, that is, if rule *covers* the example. The rule conditions are created, for example, by splitting the descriptive attribute space based on the maximum gain. In this case the averages of target values in split intervals could form the predictions.

However, the traditional decision rule methods [like Michalski, 1969] are not as accurate as one would hope. Fortunately, more accurate versions called rule ensembles have been presented [Dembczyński et al., 2008b; Friedman and Popescu, 2005, 2008]. In Chapter 4 we introduce and evaluate our rule ensembles method which is applicable to multi-target regression environment mentioned in Section 2.1.2.

## 2.2 Dynamic Input

In the traditional perspective of computer science, we are treating the given data as a static input. This way the whole data set is available all the time and also prior to executing the algorithm. This framework creates *offline* methods. On the other hand, *online* algorithms have been under increasing study [Albers, 2006]. In the online framework, the data is processed, e.g., sample by sample while arriving. There are a couple of important problems we are facing specifically in online environment: for example the efficiency of the algorithm for on-the-fly processing and preparation for sudden changes in the nature of input. There is clearly a practical need for such research in form of internet and mobile phone applications.

### 2.2.1 Concept Drift in Online Learning

One of the additional benefits of using ensembles of models instead of single model learning is that they can often easily adapt to dynamically changing input, i.e. *concept drift*. In concept drift the input distribution is changing due to unknown reasons [Klinkenberg, 2004; Kolter and Maloof, 2007].

A good—and unfortunately often daily—example of concept drift are attributes of junk e-mail. The senders of this kind of mail try to pass the filters designed to eliminate unwanted e-mails. Thus, the junk mail is under continual change when filters try to adapt to new kind of threat. In this situation, the filtering task does not change: we try to decide if incoming mail is junk or not. However, the features that reveal a mail to be junk transform.

But how is the ensemble learning related to this? First of all, the advantage of an ensemble in this case is partly related to the fact that ensembles are known to less likely overfit on the training data set [Blockeel, 1998; Caruana, 1997]. Second, like Tsymbal [2004] mentions, three approaches to handling concept drift can be separated: instance selection, instance weighting, and weighting base models in ensembles [e.g., Kolter and Maloof, 2007; Scholz and Klinkenberg, 2007; Wang et al., 2003]: The idea of instance selection is to select only those instances that are relevant in current situation. Instance weighting is a more general way of doing the previous—we now weight the instances based on their relevance. Both of these approaches need, more or less, to tune or train the learned models again on the modified data set.

On the other hand, the ensemble weighting approach changes the base model weights based on their relevance to the current input. Thus, it is possible to modify only the weights of predictors without necessarily going back to the

---

training phase. In our paper [V] we have studied a solution of weighting the base models based on the similarity of current data input with the training data sets. The idea of the proposed method MMDSEL is to use a metric called *maximum mean discrepancy* (MMD) by Smola et al. [2007] and Gretton et al. [2007a,b] to compare the training data sets with current input. To achieve significant improvements in time efficiency we proposed the use of approximation methods for some versions of MMD.

Concept drift and dynamic input in machine learning environment are discussed more in detail in Chapter 6.

### 2.2.2 Locality of Reference on Data Structures

In online algorithm and data structures research a long perceived *locality of reference* effect very similar to concept drift can be found. This phenomenon has been modeled e.g. with *working sets* and recurring *locality phases* [Albers et al., 2005; Shen et al., 2007].

There are a couple of reasons why the study of this kind of phenomenon has been going on so long [Albers, 2006]: First, like in machine learning we are interested in creating algorithms that work effectively on real applications. Second, there are some very intriguing examples where our traditional worst case techniques fail to give us the needed information.

The best known example is about a method called *competitive analysis* which compares the algorithm with a hypothetical optimal one. In paging problem competitive analysis unfortunately fails to separate the basic “first in first out” algorithm from some more advanced ones like “last recently used” which in practice are more efficient. This is due to the worst case nature of the competitive analysis. For more information, we guide the reader to the survey by Albers [2006].

We also studied the problem of getting use of the working set model in splay tree data structure [IV]. The introduced WSPLAY algorithm is presented more in detail in Chapter 5.

There is a clear analogy between locality of reference and concept drift. In fact, both research problems can be expressed as special cases of a more general model of change. Connecting the results of different areas could give us new ideas on how to solve the problem. The analogy is covered in Chapter 6.



# Chapter 3

## Regularization in Convex Optimization

In this chapter, we discuss convex optimization more in detail and in particular the role of regularization in it. Based on [1], we also introduce some results on support vector machine (SVM) regularization. In this chapter, we assume the single target setting. The multi-target case will be covered in the next chapter.

In Section 2.1, we gave most of the needed background terms. However, before we go to details we make one more assumption. In this chapter we limit our family of models  $f(\mathbf{x}; \boldsymbol{\beta})$  to the weighted sums of base predictors:

$$f(\mathbf{x}; \boldsymbol{\beta}) = f(\mathbf{x}; \mathbf{w}) = w_0 + \sum_{i=1}^M w_i p_i(\mathbf{x}), \quad (3.1)$$

where  $p_i$  are *base functions* or *base models* dependent only on the descriptive attributes  $\mathbf{x}$ . This collection covers both SVM and rule ensembles mentioned in Section 2.1.3.

Recall that an optimization problem is defined by equations like

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \frac{1}{|\mathbb{E}|} \sum_{(\mathbf{x}, y) \in \mathbb{E}} L(y, f(\mathbf{x}; \mathbf{w})) + \lambda \sum_{i=1}^M |w_i|^\alpha, \quad (3.2)$$

where we are searching for good approximate solution  $\mathbf{w}^*$ . In this chapter, we assume that the loss function is convex as defined in Section 2.1.1. We already discussed the left hand part of the optimization problem in Equation 3.2. We now examine the regularization part  $\lambda \sum_{i=1}^M |w_i|^\alpha$  more in depth.

### 3.1 Regularization

As mentioned we use the regularization part in optimization problems mostly to stabilize the results. This way the optimization result is not as dependent on the natural variance of randomly drawn sample  $|\mathbb{E}|$ . In other words, the background idea is to add some static penalty that is independent of the current set. In addition to stability, the regularization tends to prevent the optimized variable values  $w_i$  from getting values unnecessarily far from zero.

In the regularization term  $\lambda \sum_i |w_i|^\alpha$  we have two parameters. With  $\lambda$  we can tune the strength of regularization. In one extreme with  $\lambda = 0$  we get a completely unregularized problem with great variance. In the other end  $\lambda = \infty$  we get a completely static result  $\mathbf{w} = \mathbf{0}$  which does not consider the actual loss function at all.

Generally strategies like cross-validation are used to find out suitable value for  $\lambda$ . In  $K$ -fold cross-validation, the training set is split into  $K$  equal size subsets. The algorithm is repeatedly trained on  $K - 1$  subsets and validated with the remaining one. By uniting the results of all  $K$  validation cases we can make assumptions on the overall behavior.

However, in the regularization term the parameter  $\alpha$  interests us more. It affects the type of regularization and, hence, the characteristic of the solution  $\mathbf{w}^*$ . Popular values for  $\alpha$  include  $\alpha = 2$  (L2 or ridge regularization) [Hoerl and Kennard, 1970; Vapnik, 2000] and  $\alpha = 1$  (L1 or lasso regularization) [Donoho and Johnstone, 1994; Tibshirani, 1996]. The values have been shown to have interestingly different effect on the resulting model:

First of all, ridge regularization tends to result in many nonzero weights  $w_i$ . This happens because with lower absolute values of  $|w|$  we lose smaller amount when updating. For example, let  $\eta$  be the amount of change and  $\zeta, \omega$  some weights with order  $0 \leq \zeta < \omega$ , i.e.  $\zeta/\omega < 1$ . Now we notice the effect by computing how much the regularization sum is affected by updating one of the two weights:

$$|(\zeta + \eta)^2 - \zeta^2| = |2\zeta\eta + \eta^2| = \left| 2\frac{\zeta\omega\eta}{\omega} + \eta^2 \right| < |(\omega + \eta)^2 - \omega^2|.$$

Thus, ridge regularization encourages the change of the variable values near zero and ends up having roughly equal weights for highly correlating variables.

On the other hand, lasso regularization treats weights more equally in this sense. Namely, Tibshirani [1996] showed that lasso tends to lead to weights of highly correlated base functions being set to zero. This, then usually means simpler models and should help against overfitting.

Figure 3.1 illustrates the effect of L1 (lasso) and L2 (ridge) regularization. We come back to RW in Section 3.3. The shapes expose in two dimensions the distances where the regularization penalty is equal. For lasso regularization, we are more likely to find such solutions to the problem that have more variables set to zero while ridge regularization has the reverse effect. However, as Friedman and Popescu [2004] state regularization should always be chosen based on the assumed best possible solution. Often the best solution lies between these two extremes [Lounici et al., 2009; Rakotomamonjy et al., 2011]. If we have no prior knowledge about it, we can again use model selection techniques like cross-validation.

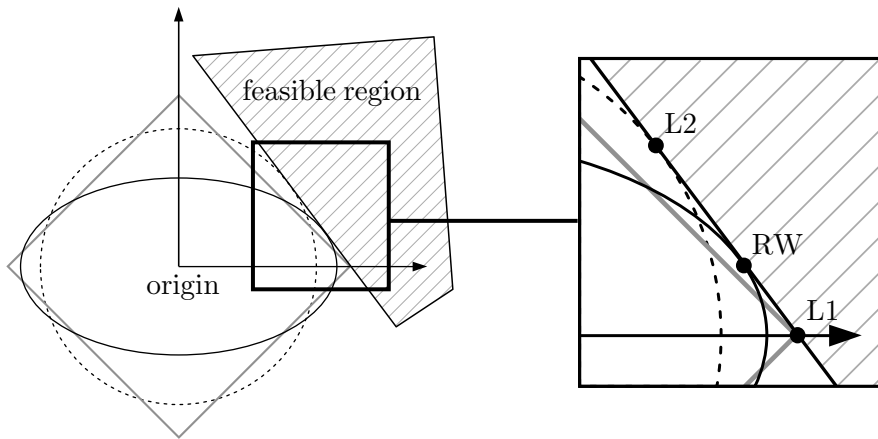


Figure 3.1: Effect of L1 (lasso—gray square), L2 (ridge—circle in dashed line) and their combined (RW—ellipsoid) regularization on optimization result in two dimensional case. For each regularization, the points in shapes have equal penalty on the optimization problem.

## 3.2 Gradient Directed Optimization

In this section, we go through a gradient directed optimization method introduced by Friedman and Popescu [2004]. It finds a local minimum of the optimization problem. Thus, to gain a globally optimal result, the problem has to be convex. In addition, because the method is based on computing gradients, the loss function has to be differentiable.

The method, as the authors claim, roughly generalizes such known methods as gradient descent, partial least squares regression [Wold et al., 1984], ridge regression [Hoerl and Kennard, 1970], least angle regression [Efron et al., 2004], and lasso regression [Tibshirani, 1996]. These optimization methods have a variety of properties that are comparable to what was mentioned in the previous section about regularization. Friedman and Popescu [2004] introduce how the optimization method can be used efficiently with squared loss and robust Huber [1964] loss function in linear regression and classification.

The gradient directed optimization method is based on the basic *gradient descent*, or steepest descent, where we always take a step toward the negative gradient. That is, if  $\mathbf{g}(\mathbf{x}) = \nabla L(y, f(\mathbf{x}; \mathbf{w}))$  is the gradient we follow it in infinitesimally small steps. In practice the step size is controlled with variable  $\gamma$  and the step on each iteration is of form  $-\gamma\mathbf{g}(\mathbf{x})$ .

However, in the gradient directed optimization we rather take step toward a tangent vector  $\mathbf{h}(\mathbf{x})$  that corresponds to *some* direction that lowers the error. Simply put, we take steps that lower the error but are not necessarily directed to the steepest slope. Formally this condition is met if the chosen direction, tangent, projects positively on the negative gradient:  $\mathbf{h}^\top(\mathbf{x})(-\mathbf{g}(\mathbf{x})) > 0$ . Thus, Friedman and Popescu [2004] choose the tangent direction to be

$$\mathbf{h}(\mathbf{x}) = -(l_0(\mathbf{x})g_0(\mathbf{x}), l_1(\mathbf{x})g_1(\mathbf{x}), \dots, l_M(\mathbf{x})g_M(\mathbf{x}))^\top,$$

where components  $l_j$  of vector  $\mathbf{l}(\mathbf{x})$  are scaling factors. The tangent condition is followed at least when scaling factors are non-negative  $l_j \geq 0$  as we see by following:  $\mathbf{h}^\top(\mathbf{x})(-\mathbf{g}(\mathbf{x})) = \mathbf{l}^\top(\mathbf{x})\|\mathbf{g}(\mathbf{x})\|^2$ . In this case, the new weights  $\mathbf{w}'$  are formed with small steps  $\mathbf{w}' = \mathbf{w} + \gamma\mathbf{h}$ .

Setting all  $l_j$  factors to some constant value causes the gradient directed approach to be equivalent to the original gradient descent method. However, Friedman and Popescu [2004] show that by adjusting the diversity of the scaling factors  $\mathbf{l}$  we can control the optimization process in a way similar to regularization. They claim that with enough possibilities for  $\mathbf{l}$ , infinitesimal size steps to tangent direction  $\mathbf{h}(\mathbf{x})$  essentially cover the possible paths of all the optimization methods mentioned at the beginning of this section. Thus, we can find

all the optimal points reachable by those algorithms with the gradient directed method.

However, we still have not defined the actual form of the function  $\mathbf{l}$  that would allow such diversity. Friedman and Popescu [2004] suggest

$$l_j(\mathbf{x}) = I \left[ |g_j(\mathbf{x})| \geq \tau \max_{0 \leq k \leq M} |g_k(\mathbf{x})| \right],$$

where  $\tau \in [0, 1]$  is a gradient threshold parameter that affects the diversity. Furthermore,  $I[\cdot]$  is an indicator function that equals one if the statement is true and zero otherwise. In practice the scaling factors grant that the tangent vector  $\mathbf{h}(\mathbf{x})$  is only affected by those gradients  $g_j$  that have high enough absolute value, relative to  $\tau$ . Thus, only those weights  $w_j$  are modified. Clearly with larger values of  $\tau$  we get more diversity to the optimization process by updating less weights at each iteration.

Hence, for the gradient directed optimization the regularization part of the optimization problem in Equation 3.2 on page 19 is not computed. Instead we explicitly control the number of weights that get changed on each optimization iteration. With  $\tau = 0$ , we are changing all the weights on each iteration, resulting in a behavior similar to ridge regularization ( $\alpha = 2$ ). On the other hand, if  $\tau = 1$ , only one gradient on each iteration is modified and the behavior is similar to lasso regularization ( $\alpha = 1$ ). We want to select the best  $\tau$  value based on our prior knowledge on the problem.

Even if we try out different  $\tau$  values with cross-validation, the procedure is very effective especially with the squared loss regression. In this case, we can update both the gradients  $\mathbf{g}$  and weights  $\mathbf{w}$  efficiently because they are affected only by the non-zero tangent vector  $\mathbf{h}$  coordinates. Because of the definition of scaling vector  $\mathbf{l}$  this is usually only a small subset of gradients  $\mathbf{g}$ . Hence, most of the optimization time, namely  $|\mathbb{E}|M^2/2$ , is spent on computing the covariances between the weights. In addition, we do not have to compute the covariances for zero weighted predictive terms at all. [Friedman and Popescu, 2004] Thus, most of the resources are usually used for optimization with lower values of  $\tau$  because that is when most of the weights are affected. For example, in our experiments, the case  $\tau = 0$  seems to take at least half of the computing time alone.

We come back to the gradient directed optimization in Section 4.3.3. We now move on to consider regularization more deeply.

### 3.3 Efficient Regularization in Support Vector Machines

In this section, we present our study on support vector machines (SVM) regularization [I].

#### 3.3.1 Background

SVMs [Cortes and Vapnik, 1995; Vapnik, 2000] are a vastly studied method for classification and regression. Their clear benefit is the stable theoretical background backing them up. SVMs basically aim to find a maximal margin hyperplane between two sets.

The original linear SVM is only able to separate two sets linearly in classification setting. Thus, in this section we are talking only about binary classification. However, the usefulness of SVMs greatly improved when it was noticed that in the so-called dual form the optimization problem uses only inner products between training and testing instances. Thus, an implicit mapping to another suitable inner product space can be used. With this kind of trick also nonlinear set separators are available. In this case, the inner products are replaced with the so-called kernel functions. The study of kernel functions has been fruitful area of research—see, e.g., [Shawe-Taylor and Cristianini, 2004] for more information.

Nevertheless, in this section we stick to linear SVM with L2 regularization. The optimization problem is usually expressed as:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{(\mathbf{x}, y) \in \mathbb{E}} L(y, f(\mathbf{x}; \mathbf{w})).$$

However, this usual form is slightly different from the one we used in Equation 3.2 on page 19, and thus we instead use the equivalent form

$$\min_{\mathbf{w}} \frac{1}{|\mathbb{E}|} \sum_{(\mathbf{x}, y) \in \mathbb{E}} L(y, f(\mathbf{x}; \mathbf{w})) + \lambda \|\mathbf{w}\|^2. \quad (3.3)$$

In classification we usually use as a loss function either hinge loss (L1-SVM) presented in Section 2.1.1 or its square (L2-SVM). The regularization is either the ridge (2-norm), as in Equation 3.3, or lasso (1-norm) [Zhu et al., 2004]:  $\sum_i^M |w_i| = \|\mathbf{w}\|_1$ . The effect on regularization function has been analyzed at

the beginning of this chapter. As mentioned, using lasso regularization tends to give sparser and more stable results. Thus, it would often be preferable.

Nevertheless, it seems like ridge regularized SVM is more easily solvable. At least we have multiple optimization problem solver implementations with many desirable abilities: for example solvers with great performance [Hsieh et al., 2008; Shalev-Shwartz et al., 2007], scalability, stability, and availability of different objective functions (like L1-SVM and L2-SVM) minimizations can be mentioned. For lasso regularized SVM there has been less success with efficient solvers.

In [I] we consider a reweighting algorithm (RW) that approximates lasso SVM with multiple runs of ridge SVM. In addition, the optimization process and result keeps most of the characteristics of the original 2-norm solver. Thus, we aim to get the best parts of both 1-norm and 2-norm solving. Prior to our work, it was already known that such a reweighting method converges to the lasso optimization result [Grandvalet and Canu, 1999]. However, the rate of convergence is still unknown. Fortunately, we are able to give a lower bound per iteration for the progress. The result also gives some intuition to the process of reweighting. For example, we notice that the convergence is slow when coordinates are near zero.

In addition, in [I] we experimentally show that even a few iterations of the reweighting algorithm may give a solution that is more accurate and less overfitting than either the original ridge or the target lasso solution. However, according to Schmidt et al. [2007] we may need hundreds of iterations to actually converge to the lasso solution. This supports the statement by Friedman and Popescu [2004]: the most accurate solution is often found somewhere between the two regularization extremes (lasso and ridge).

### 3.3.2 Reweighting Ridge Regularized SVM

In this section, we go through the reweighting algorithm. However, let us first define an element-wise product (Hadamard product) of two vectors  $\mathbf{w}$  and  $\mathbf{v}$  with  $\mathbf{w} \otimes \mathbf{v} = (w_1v_1, w_2v_2, \dots, w_Mv_M)^\top$ .

The reweighting algorithm is presented in Algorithm 3.1 on the next page. On each iteration, the descriptive attributes  $\mathbf{x}$  are weighted with  $\mathbf{v}$ . The weights  $\mathbf{v}$ , however, are iteratively computed based on previous SVM solutions so that for iteration  $t > 1$  the used weight in coordinate  $i$  is  $v_{(t,i)} \leftarrow \sqrt{|w_{(t-1,i)}v_{(t-1,i)}|}$ , where  $w_{(t-1,i)}$  is the corresponding coordinate of the ridge regularized SVM solution of iteration  $t - 1$ .

**Input:** training set  $\mathbb{E}$  and number of iterations  $N$   
**Output:** weight vector  $\mathbf{w}$

- 1:  $\mathbf{v}_1 \leftarrow \mathbf{1}$
- 2: **for**  $t = 1$  to  $N$  **do**
- 3:    $\mathbb{E}'_t \leftarrow \{(\mathbf{x} \otimes \mathbf{v}_t, y) \mid (\mathbf{x}, y) \in \mathbb{E}\}$
- 4:    $\mathbf{w}_t \leftarrow$  solve ridge SVM with examples  $\mathbb{E}'_t$
- 5:   **for** each coordinate  $i$  **do**
- 6:      $v_{(t+1,i)} \leftarrow \sqrt{|w_{(t,i)} v_{(t,i)}|}$
- 7:   **end for**
- 8: **end for**
- 9:  $\mathbf{w} \leftarrow \mathbf{w}_N \otimes \mathbf{v}_N$ .
- 10: **return**  $\mathbf{w}$

Algorithm 3.1: Algorithm for reweighting the ridge regularized SVM

See Figure 3.1 on page 21 for an illustration of effect of reweighting on the result. The shapes expose in a two-coordinate case the areas where the regularization penalty is equal. Every reweighting of ridge SVM solution kind of squeezes the result into an ellipse which is nearer to lasso solution. The same is now expressed slightly more formally.

Let us consider reweighting algorithm iteration  $t$  with our current optimization weight  $\mathbf{w}_t$ . In addition, we assume that loss function is convex and there is a weight vector  $\mathbf{w}_{(t+1)}^*$  with lower lasso regularized objective function as presented in Equation 3.3 on page 24. Let us also assume that  $\mathbf{w}_t$  has no zero values in coordinates, where  $\mathbf{w}_{(t+1)}^*$  has non-zero values. This can be forced by changing zeros to small constant values. Now we state the following.

**Theorem 1.** *An iteration of the reweighting algorithm decreases the lasso regularized objective function value. That is, the weight  $\mathbf{w}_{(t+1)}$  has lower lasso objective function value than  $\mathbf{w}_t$ .*

The details of the theorem are presented in [1].

In our experiments we moreover showed that our reweighting algorithm would be the most accurate choice after only a couple of iterations. More precisely, in the linear classification SVM problems our reweighting algorithm was usually more accurate than the ridge or lasso regularized versions.

# Chapter 4

## Rule Based Ensembles

This chapter is about rule ensembles. We first introduce them generally and then go through our multi-target rule ensemble method, FIRE, introduced in [II, III]. We also present some new experimental results on a slightly modified version of FIRE. Finally, we conclude on which of all the versions should be preferred.

First we go through some notation. Like in Chapter 3, we are mostly addressing the weighted sums of base functions. If we are only weighting the readily given base functions, we use the form

$$\mathbf{f}(\mathbf{x}; \mathbf{w}) = \mathbf{f}(\mathbf{x}) = w_0 \mathbf{avg} + \sum_{i=1}^M w_i \mathbf{p}_i(\mathbf{x})$$

in the more general multi-target case. Here  $\mathbf{avg}$  is a constant offset vector. However, if we want to stress that we are still searching for the base functions, we use the form of

$$\mathbf{f}(\mathbf{x}; \boldsymbol{\beta}) = \sum_{i=0}^M \mathbf{p}_i(\mathbf{x}; \beta_i).$$

### 4.1 On the Background of the Rule Ensembles

Rule sets [Flach and Lavrač, 2003] are, together with decision trees, one of the most expressive and human readable model representations. They are frequently used when an interpretable model is desired. The majority of rule learning methods are based on the sequential covering algorithm [Michalski, 1969], originally

designed for learning ordered rule lists for binary classification domains. This is also the case with the existing methods for learning multi-target rules [Ženko, 2007; Ženko and Džeroski, 2008] introduced in Section 2.1.2. As the reader may recall, in the multi-target case we are predicting multiple target values at the same time instead of the traditional single value. The accuracy of multi-target rules on classification domains is comparable to other classification methods like decision trees. Unfortunately, on both single target and multi-target *regression* problems, the accuracy of rule sets that are learned by the sequential covering approach is considerably worse than that of other regression methods like regression trees [for an empirical comparison see Ženko, 2007].

An alternative approach to rule learning is called rule ensembles [Friedman and Popescu, 2005, 2008]. Strictly speaking, any set of (unordered) rules can be called a rule ensemble like, e.g., Indurkha and Weiss [2001] do. In this thesis, however, a rule ensemble is understood to be a set of unordered rules whose predictions are combined through weighted voting, which is the approach originally introduced by RULEFIT [Friedman and Popescu, 2005, 2008], and used by REGENDER [Dembczyński et al., 2008a,b] as well as in our multi-target rule ensemble method FIRE [II, III].

In the next few sections, we discuss some rule ensembles approaches for single target and multi-target regression problems.

## 4.2 Rule Ensembles in Single Target Regression

We first go through the single target rule ensemble methods in regression problems. So, we need to get a weighted sum of unordered rules for approximating some scalar valued function.

First of all, the name of rule ensembles is a bit misleading: in addition to rules, for example in RULEFIT, we can also use the simple linear functions of numeric descriptive attributes and add them to the initial set of rules. Thus, the final prediction for a given example is obtained by a weighted voting of all linear terms and those rules that apply (i.e., cover the example). The resulting model can, hence, be written as:

$$\hat{y} = f(\mathbf{x}) = w_0 + \sum_{i=1}^N w_i r_i(\mathbf{x}) + \underbrace{\sum_{j=1}^K w_{(N+j)} x_j}_{\text{optional}} \quad (4.1)$$

where  $w_0$  is the baseline prediction, the first sum is the correction value obtained

from the  $N$  rules, and the second sum is the correction value obtained from the (optional)  $K$  linear terms. The rules  $r_i$  are here represented as indicator functions mentioned in Section 3.2: they have a value of 1 for all examples that they cover, and 0 otherwise. For REGENDER, adding linear terms is not possible.

**Input:** training examples  $\mathbb{E}$

**Constants:** the desired size of the ensemble  $M$ , subsample size  $\eta$  and memory or shrinkage parameter  $\nu$

**Output:** members of the ensemble  $\mathbf{p}$

- 1:  $f_0(\mathbf{x}) \leftarrow \arg \min_w \sum_{(\mathbf{x}, y) \in \mathbb{E}} L(y, w)$
- 2: **for**  $m = 1$  **to**  $M$  **do**
- 3:   Draw randomly  $\mathbb{E}_m \subseteq \mathbb{E}$  with  $|\mathbb{E}_m| = \eta$ .
- 4:    $\beta_m \leftarrow \arg \min_{\beta} \sum_{(\mathbf{x}, y) \in \mathbb{E}_m} L(y, f_{m-1}(\mathbf{x}) + p(\mathbf{x}; \beta))$
- 5:    $p_m(\mathbf{x}) \leftarrow p(\mathbf{x}; \beta_m)$
- 6:    $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + \nu p_m(\mathbf{x})$
- 7: **end for**
- 8: **return**  $(f_0(\mathbf{x}), p_1(\mathbf{x}), p_2(\mathbf{x}), \dots, p_m(\mathbf{x}))^\top$

Algorithm 4.1: General ensemble generation.

REGENDER [Dembczyński et al., 2008a] is based on the idea first suggested by Friedman and Popescu [2003, 2005] and shown in Algorithm 4.1. In short, REGENDER seeks one rule and its weight (both on line 4) at a time and adds the predictor to the ensemble.

Let us discuss the solution by Dembczyński et al. [2008a] in more detail. First of all, they only have rules:  $p_i(\mathbf{x}; \beta) = w_i r_i(\mathbf{x}; \beta')$ . Note, that  $\beta$  and  $\beta'$  are not exactly equivalent because  $\beta$  also holds the information about the weight.

The optimization problem for adding a rule on line 4 of Algorithm 4.1 is computationally too hard to be practical, but Dembczyński et al. [2008a] demonstrate greedy approximation methods for a couple of loss functions and techniques. The precise form of the minimization depends on the used loss function and minimization technique. We give the gradient boosting technique for squared loss here because this resulted in the most accurate models. They solve the problem in two parts.

First, the authors find the conditional part of the rule  $r_i(\mathbf{x}; \beta')$  by minimizing the loss greedily single elementary expression at time. In this case Dembczyński et al. [2008a] give the form of

$$\beta'_m = \arg \min_{\beta'} - \frac{\left| \sum_{(\mathbf{x}, y) \in \mathbb{E}_m \wedge r(\mathbf{x}; \beta')=1} (y - f_{m-1}(\mathbf{x})) \right|}{2\sqrt{\sum_{(\mathbf{x}, y) \in \mathbb{E}_m} r(\mathbf{x}; \beta')}} \quad (4.2)$$

for minimization by gradient boosting.

Second, the weight  $w_m$  should be found by solving the line-search problem

$$w_m = \arg \min_w \sum_{(\mathbf{x}, y) \in \mathbb{E}_m} L(y, f_{m-1}(\mathbf{x}) + wr(\mathbf{x}; \beta'_m)).$$

However, for squared loss  $L_{\text{squared}}$  an analytical solution can be found:

$$w_m = - \frac{\left| \sum_{(\mathbf{x}, y) \in \mathbb{E}_m \wedge r(\mathbf{x}; \beta'_m)=1} (y - f_{m-1}(\mathbf{x})) \right|}{\sqrt{\sum_{(\mathbf{x}, y) \in \mathbb{E}_m} r(\mathbf{x}; \beta'_m)}}.$$

Interestingly, the solution is nearly identical to the rule condition optimization in Equation 4.2.

The possibility of analytical solution for weights with this technique-loss combination is a clear benefit in REGENDER approach. On the other hand, the greedy rule searching procedure may not find the best possible rules.

Friedman and Popescu [2005, 2008] also note this possibility of creating a rule ensemble directly with the general method mentioned in Algorithm 4.1 on the previous page. However, they point out that solving the optimization problem on line 4 directly is impractical.

The approach used by their RULEFIT method is roughly presented in Algorithm 4.2 on the facing page: Friedman and Popescu [2005, 2008] exploit the existing efficient algorithms for producing decision *tree* ensembles. Thus, RULEFIT starts by generating a set of decision trees  $\{d_h(\mathbf{x})\}_{h=1}^H$  in much the same way as ensembles are generated by methods like bagging [Breiman, 1996] and random forests [Breiman, 2001]. In fact they use their own more general ensemble framework ISLE [Friedman and Popescu, 2003] which is a generalization of the previously mentioned ones. This generation is executed on lines 1–7.

**Input:** training examples  $\mathbb{E}$

**Constants:** desired size of the initial tree ensemble  $H$ , subsample size  $\eta$  and memory or shrinkage parameter  $\nu$

**Output:** members of the rule ensemble  $\mathbf{p}$  with their weights  $\mathbf{w}$

```

1:  $f_0(\mathbf{x}) \leftarrow \arg \min_w \sum_{(\mathbf{x}, y) \in \mathbb{E}} L(y, w)$ 
2: for  $h = 1$  to  $H$  do
3:   Draw randomly  $\mathbb{E}_h \subseteq \mathbb{E}$  with  $|\mathbb{E}_h| = \eta$ .
4:    $\beta_h \leftarrow \arg \min_\beta \sum_{(\mathbf{x}, y) \in \mathbb{E}_h} L(y, f_{h-1}(\mathbf{x}) + d(\mathbf{x}; \beta))$ 
5:    $d_h(\mathbf{x}) \leftarrow d(\mathbf{x}; \beta_h)$ 
6:    $f_h(\mathbf{x}) \leftarrow f_{h-1}(\mathbf{x}) + \nu d_h(\mathbf{x})$ 
7: end for
8:  $\mathbf{r} \leftarrow \text{ConvertTreeNodesToRules}((d_1, d_2, \dots, d_H)^\top)$ 
9:  $\mathbf{p} \leftarrow (r_1, r_2, \dots, r_N, x_1, x_2, \dots, x_K)^\top$  {Optionally  $x_j$ }
10:  $\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} \sum_{(\mathbf{x}, y) \in E} L\left(y, w_0 + \sum_{i=1}^N w_i r_i(\mathbf{x}) + \sum_{j=1}^K w_{(N+j)} x_j\right)$ 
11: return  $(\mathbf{p}, \mathbf{w})$ 

```

Algorithm 4.2: Another, RULEFIT, approach for rule ensemble generation.

All the trees are then transcribed into a collection of rules on line 8, and an optimization procedure (line 10) is used to select a subset of rules and to determine their weights. As a result, they get a set of weighted rules. In addition to rules, it is possible to use the simple linear functions of descriptive attributes, add them to the initial set of rules, and likewise select them and determine their weights in the optimization step. The linear terms part of the model is global, that is, it covers the entire example space. Note that this is different from model trees [Karalič, 1992; Quinlan, 1992; Wang and Witten, 1997], where we have local linear models in separate leaves, which only cover specific examples. The overall model is then like the one presented in Equation 4.1 on page 28.

The most difficult task is to find the optimal weights on line 10. However, Friedman and Popescu [2008] point out that using the gradient directed optimization method by Friedman and Popescu [2004] this can be done quite efficiently. The method was already discussed briefly in Section 3.2. As the reader may recall, one of the main insights in the approach is the way of simulating different regularization techniques efficiently. We come back to the optimization method when considering our own multi-target rule ensemble method FIRE in Section 4.3.3.

### 4.3 Learning Rule Based Ensembles for Multi-Target Regression

In the previous sections, we introduced some background on rule ensembles and two single target rule ensembles methods. We next cover our suggestion for the first multi-target rule ensemble method.

Our algorithm for learning rule based ensembles for multi-target regression problems (which we call FIRE: Fitted rule ensembles) [III, II] is based on the RULEFIT method [Friedman and Popescu, 2005, 2008]. The top level of the FIRE algorithm is outlined in pseudo code of Algorithm 4.3. It starts by generating a set of diverse regression trees, which are converted to rules. Because linear dependencies are known to be difficult to approximate with rules, in [III] we proposed to optionally add linear terms (simple linear functions) of all numeric descriptive attributes to the collection. The original algorithm in [II] did not include this possibility.

**Input:** training examples  $\mathbb{E}$   
**Constants:** gradient limit step *step* and overfitting parameter *threshold*  
**Output:** members of the rule ensemble  $\mathbf{p}$  with their weights  $\mathbf{w}$

- 1:  $\mathbf{d} \leftarrow \text{GenerateSetOfTrees}(\mathbb{E})$
- 2:  $\mathbf{r} \leftarrow \text{ConvertTreesToRules}(\mathbf{d})$
- 3:  $\mathbf{p} \leftarrow \text{AddLinearTerms}(\mathbb{E}, \mathbf{r})$  {Optional linear terms}
- 4:  $ERR_{\min} \leftarrow \infty$
- 5: **for**  $\tau = 1.0$  **to**  $0.0$  **with** *step* **do**
- 6:    $(\mathbf{w}_\tau, ERR_\tau) \leftarrow \text{OptimizeWeights}(\mathbf{p}, \mathbb{E}, \tau)$
- 7:   **if**  $ERR_\tau < ERR_{\min}$  **then**
- 8:      $(\mathbf{w}_{\text{opt}}, ERR_{\min}) \leftarrow (\mathbf{w}_\tau, ERR_\tau)$
- 9:   **else if**  $ERR_\tau > \text{threshold} \cdot ERR_{\min}$  **then**
- 10:     **break**
- 11:   **end if**
- 12: **end for**
- 13:  $(\mathbf{p}', \mathbf{w}') \leftarrow \text{RemoveZeroWeightedTerms}(\mathbf{p}, \mathbf{w}_{\text{opt}})$
- 14: **return**  $(\mathbf{p}', \mathbf{w}')$

Algorithm 4.3: The algorithm FIRE for learning rule ensembles for multi-target regression.

FIRE then optimizes the weights of rules and linear terms with a gradient directed optimization algorithm. This optimization procedure depends on a gradient threshold parameter  $\tau$  and we repeat the optimization for different values of  $\tau$  in order to find a set of weights with the smallest validation error. In the end, we remove all the rules and linear terms whose weights are zero.

The resulting rule ensemble is a vector function  $\mathbf{f}$ ; given an unlabeled example  $\mathbf{x}$  it predicts a vector  $\hat{\mathbf{y}}$  consisting of the values of all target attributes:

$$\hat{\mathbf{y}} = \mathbf{f}(\mathbf{x}) = w_0 \mathbf{avg} + \sum_{i=1}^N w_i \mathbf{r}_i(\mathbf{x}) + \underbrace{\sum_{t=1}^T \sum_{j=1}^K w_{(t,j)} \mathbf{x}_{(t,j)}}_{\text{optional}}. \quad (4.3)$$

The first term includes a constant vector  $\mathbf{avg}$ , whose components are the average values for each of the targets. The first sum is the contribution of the  $N$  rules: each rule  $\mathbf{r}_i$  is a vector function that gives a constant prediction, if it covers the example  $\mathbf{x}$ , or returns a zero vector otherwise.

The double sum in Equation 4.3 is the contribution of optional linear terms. There is a term for each combination of target and numeric descriptive attributes, thus the total number of linear terms is the number of numeric descriptive attributes  $K$  times the number of target attributes  $T$ . A linear term  $\mathbf{x}_{(t,j)}$  is a vector that corresponds to the influence of the  $j$ -th numerical descriptive attribute  $x_j$  on the  $t$ -th target attribute; its  $t$ -th component is equal to  $x_j$ , while all other components are zero:

$$\mathbf{x}_{(t,j)} = (0, \dots, \underset{t-1}{0}, \underset{t}{x_j}, \underset{t+1}{0}, \dots, 0)^\top.$$

The values of all weights  $w$  are determined during the optimization phase, and because of interpretability one of our goals is to have as many weights equal to zero as possible.

**Example 1.** *Let the problem domain have eight descriptive attributes  $\mathbf{x} = (x_1, \dots, x_8)$  and three target attributes  $\mathbf{y} = (y_1, y_2, y_3)$ . A hypothetical rule ensemble that predicts all the target values of this domain simultaneously could*

be:

$$\begin{aligned}
 \hat{\mathbf{y}}^T &= \mathbf{f}^T(\mathbf{x}) = 0.95(16.2, 6.0, 21.1) \\
 &\quad + 0.34 [\text{IF } (x_8 > 3.8) \& (x_6 > 7.2) \text{ THEN } (15.9, 36.2, 14.4)] \\
 &\quad + 0.21 [\text{IF } (x_3 \leq 12.1) \text{ THEN } (6.3, 50.0, -14.3)] \\
 &\quad + 0.80(x_2, 0, 0) + 0.11(0, 0, x_2) + 0.17(0, x_5, 0) + 0.22(0, 0, x_5) \\
 &= (15.4 + 0.80 x_2, 5.7 + 0.17 x_5, 20.0 + 0.11 x_2 + 0.22 x_5) \\
 &\quad + [\text{IF } (x_8 > 3.8) \& (x_6 > 7.2) \text{ THEN } (5.4, 12.3, 4.9)] \\
 &\quad + [\text{IF } (x_3 \leq 12.1) \text{ THEN } (1.3, 10.5, -3.0)]
 \end{aligned}$$

It comprises a constant vector, two rules and four linear terms (of attributes  $x_2$  and  $x_5$ ), but can also be simplified in a sum of a vector of linear equations and two rules.

Now, given a descriptive attribute vector, e.g.,

$$\mathbf{x}^\dagger = (0.0, 1.0, 9.0, 6.4, 2.0, 5.5, 0.0, 4.2)^T,$$

we get the corresponding prediction

$$\begin{aligned}
 \mathbf{f}^T(\mathbf{x}^\dagger) &= (15.4 + 0.8, 5.7 + 0.34, 20.0 + 0.11 + 0.44) \\
 &\quad + (1.3, 10.5, -3.0) \\
 &= (17.5, 16.54, 17.55).
 \end{aligned}$$

So far, we have only briefly mentioned two important aspects of our algorithm, the generation of the initial collection of trees, rules and linear terms, and the weight optimization procedure. We describe them in detail in the next two subsections.

### 4.3.1 Generation of Base Models

The basic decision tree learning method used within the GenerateSetOfTrees procedure of FIRE (Algorithm 4.3 on page 32) is the predictive clustering tree learning method [Blockeel et al., 1998] that can learn multi-target regression trees. As a starting point, we use the implementation of this paradigm within the system CLUS [Blockeel and Struyf, 2002], which can learn multi-target regression trees [Struyf and Džeroski, 2006]. A set of diverse trees is generated with the multi-target implementation of the random forest ensemble method [Kocev et al., 2007], modified to generate trees of limited depth.

It is well known that variance of constituent base models is essential for good accuracy of ensembles [Dietterich, 2000]. In order to increase the tree (and, thus, rule) variance, the RULEFIT method [Friedman and Popescu, 2008] limits the depth of a particular tree in a randomized fashion. We also adopt this approach as detailed in [III].

All regression trees generated with the above procedure are transcribed into rules with the ConvertTreesToRules procedure and each leaf of each tree is converted to a rule. The weights of these rules are later computed with the gradient directed optimization. However, before optimizing the rule weights, it is necessary to normalize their predictions.

Namely, the optimization problem we are solving is not invariant to the scaling of rule predictions. Nevertheless, as in [III], we would like to put all the rules and targets on the same line when the optimization error is considered.

Unfortunately, the approach used by Friedman and Popescu [2008, Section 3.2] can not be used for multi-target problems since setting the rule predictions for all targets to 1 would delete all the information on relations between the targets. In addition to equalizing the initial importance of rules, we also have to equalize the effect of different targets during the optimization. Otherwise, targets with large scales would dominate the rule selection.

In order to equalize the importance of different rules and different targets we proceed with three separate steps: Firstly, we simply zero-center all the target dimensions. Secondly, we scale each rule with a factor that represents the prior magnitude of the rule. This should equalize the effect of the rules on the optimization process. Thirdly, we normalize the differing scales of target spaces away temporarily. This last step is omitted after optimization. The first and last step of the process are trivial and repeated in most of the optimization processes.

In the first step, we make all the rule target predictions  $\mathbf{r}'$  zero-centered by subtracting the average *avg* from each of the original rule predictions  $\mathbf{r}''$ :  $\mathbf{r}' = \mathbf{r}'' - \mathbf{avg}$ . The average *avg* contains the average values of target attributes on the learning set.

In the second, more complex, step we scale the predicted values  $r'_t$  of each target attribute  $t$  by dividing them with some factor  $\gamma$ :

$$r_t = \frac{r'_t}{\gamma}. \tag{4.4}$$

However, we have a couple of alternatives for the scaling factor.

Let  $\sigma_t$  be the standard deviation of a target attribute  $t$  over the entire learning set. Assuming a normal distribution, dividing a zero-centered target attribute  $t$  by  $2\sigma_t$  should put 95% of all values within the  $[-1, 1]$  interval.

Knowing this, there are a couple of ways to scale the rule target predictions by defining the factor  $\gamma$  for normalization. We can, for instance, normalize with the average of target prediction values or with the maximum absolute value when standard deviations are eliminated. That is, we would make all the rules have equal average of target predictions or equal maximal target prediction.

The first option would mean normalization by average values:

$$\gamma_{\text{avg}} = \frac{1}{T} \sum_{i=1}^T \frac{r'_i}{2\sigma_i}.$$

This form of  $\gamma$  might be more intuitive because rules end up having equal average of targets. This is discussed more in Section 4.4.3.

The second option was used in [II, III] and would mean computing the maximum target value  $r'_m$  where the index  $m \in \{1, \dots, T\}$  has following property:

$$m = \arg \max_t \left| \frac{r'_t}{2\sigma_t} \right|.$$

Simply put,  $r'_m$  is the largest target prediction when the scale of targets is omitted. In this maximum based case the normalization factor  $\gamma$  for Equation 4.4 on the previous page would be

$$\gamma_{\text{max}} = \frac{r'_m}{2\sigma_m}.$$

An interesting note is that in this latter option we can bound the predictions  $r_t^*$ . After this second stage of normalization, it holds that  $r_m^* = 1$  and  $|r_t^*| = |\sigma_m/r'_m r'_t/\sigma_t| \leq 1$  for all other targets  $t$  by the definition of  $m$ . This has analogy with the solution of Friedman and Popescu [2008] in the single target case.

Either way, at the second stage of normalization the target predictions  $r_t$  in a certain rule  $\mathbf{r}$  are scaled by a factor  $\gamma$  that represents the prior degree of the mentioned rule. Thus, both ways of normalization roughly equalize the rules before the optimization phases and affect the rules of the final model. After these two normalization steps our rule predictions are of form:

$$\mathbf{r} = \frac{\mathbf{r}'}{\gamma} = \frac{\mathbf{r}'' - \mathbf{avg}}{\gamma}.$$

Finally, we are in the last, temporarily affecting, step of normalization. In this step, we equalize the scaling differences between different target attribute spaces. However, clearly our intention is to use this normalization only temporarily during optimization. Otherwise the resulting model would no more be applicable to real world data. We do this simply by dividing the target attribute prediction values  $r_t$  by two times their standard deviations  $2\sigma_t$ :

$$r_t^* = \frac{r_t}{2\sigma_t} = \frac{r'_t}{2\sigma_t\gamma} = \frac{r''_t - avg_t}{2\sigma_t\gamma}.$$

### 4.3.2 Optional Linear Terms

We recall from Equation 4.3 on page 33 that in addition to rules we can optionally add linear terms to the rule ensemble. As already mentioned, a single linear term is defined as

$$\mathbf{x}_{(t,j)} = (0, \dots, \underset{t-1}{0}, \underset{t}{x_j}, \underset{t+1}{0}, \dots, 0)^\top,$$

which depicts the influence of the descriptive attribute  $x_j$  on the target attribute  $t$ . We add linear terms for all possible combinations of numeric descriptive attributes and target attributes.

The linear terms are normalized in a somewhat similar way as rules. However, unlike rules, linear terms are affected by *two* attributes and, thus, *two* attribute space scales: that of the  $j$ -th feature space and that of  $t$ -th target space. In addition, linear terms with their single non-zero coordinate are multi-target only in principle. There is no sense in scaling with the average or maximum coordinate of linear terms as was done in the second normalization stage. These differences have to be taken into account.

We again shift the linear terms by the average  $\bar{x}_j''$  of the  $j$ -th descriptive attribute  $\mathbf{x}'_{(t,j)} = (0, \dots, x_j'' - \bar{x}_j'', \dots, 0)^\top$ . Here  $x_j''$  is the original attribute  $x_j$  value. However, we continue by normalizing the terms to the target attribute scale

$$\mathbf{x}_{(t,j)} = \mathbf{x}'_{(t,j)} \frac{\sigma_t}{\sigma_j}.$$

Here we note the effect of two separate attribute spaces.

There would be an option to scale linear terms by multiplying them with  $T$ . Intuitively, this could bring their effect to the same level with rules because of the  $T - 1$  zero-valued coordinates in linear terms. However, in practice this seemed to stress linear terms too much and cause reduction in accuracy. We discuss this briefly in the experimental section.

Linear terms normalized like this appear in the final rule ensemble model. A similar normalization of linear terms is also performed in the RULEFIT method [Friedman and Popescu, 2008, Section 5].

However, analogously to the third stage of rule normalization we also scale the target dimension space out temporarily:

$$\mathbf{x}_{(t,j)}^* = \frac{\mathbf{x}_{(t,j)}}{2\sigma_t} = \frac{\mathbf{x}'_{(t,j)}}{2\sigma_j}.$$

This is, again, only intended to equalize the terms of different target attributes during the optimization procedure.

### 4.3.3 Gradient Directed Weight Optimization

The weights from Equation 4.3 on page 33 are determined within the OptimizeWeights procedure presented in Algorithm 4.4 on the facing page. As mentioned in Equation 3.2 on page 19, the optimization problem we are solving is typically written as

$$\arg \min_{\mathbf{w}} \sum_{\mathbf{x} \in \mathbb{E}} L(\mathbf{y}, \mathbf{f}(\mathbf{x}; \mathbf{w})) + \lambda \sum_{i=1}^M |w_i|^\alpha, \quad (4.5)$$

where  $L$  is the loss function and the last term is the regularization part. In our case, because of interpretability, we aim to keep the model as small as possible with the regularization. However, we are not solving this optimization problem directly: rather, we are using the approach proposed by Friedman and Popescu [2004], also used by the RULEFIT method and discussed in Section 3.2. They propose a gradient directed optimization method with a squared loss

$$L_t(y_t, f_t(\mathbf{x})) = \frac{1}{2}(f_t(\mathbf{x}) - y_t)^2.$$

As in Section 2.1.2, we denote the target dimension with index  $t$ .

We are aiming to use a gradient directed optimization method for our weight optimization. Thus, as we found out in Section 2.1.2, a suitable generalization of loss function for our multi-target problem is taking the average over the squared loss of all  $T$  target attributes:

$$L(\mathbf{y}, \mathbf{f}(\mathbf{x})) = L_{\text{avg}}(\mathbf{y}, \mathbf{f}(\mathbf{x})) = \frac{1}{T} \sum_{t=1}^T L_t(y_t, f_t(\mathbf{x})). \quad (4.6)$$

**Input:** base models  $\mathbf{p}$ , training examples  $\mathbb{E}$  and gradient threshold  $\tau$   
**Constants:** limit for iterations, number of allowed nonzero weights and overfitting parameter *threshold*  
**Output:** weights  $\mathbf{w}$  and an error estimate *ERR*

```

1:  $\mathbf{w}_0 = \mathbf{0}$ 
2:  $(\mathbb{E}_t, \mathbb{E}_v) \leftarrow \text{SplitSet}(\mathbb{E})$  {Training and validation sample}
3: for  $i = 0$  to Maximum number of iterations do
4:   do every 100 iterations
5:      $ERR_i \leftarrow \text{Error}(\mathbb{E}_v, \mathbf{p}, \mathbf{w}_i)$ 
6:     if  $ERR_i > \text{threshold} \cdot \min_{j < i} ERR_j$  then
7:       goto line 19
8:     else if  $ERR_i < \min_{j < i} ERR_j$  then
9:        $\text{StoreWeights}(\mathbf{w}_i, ERR_i)$ 
10:    end if
11:  end do
12:   $\mathbf{g} \leftarrow \text{ComputeGradients}(\mathbb{E}_t, \mathbf{p}, \mathbf{w}_i)$ 
13:  if Limit of allowed nonzero weights is reached then
14:     $\mathbf{g} \leftarrow \{g_k \in \mathbf{g} \mid w_{(i,k)} \neq 0\}$ 
15:  end if
16:   $\mathbf{g}_{\max} \leftarrow \{g_j \in \mathbf{g} \mid |g_j| \geq \tau \max_k |g_k|\}$ 
17:   $\mathbf{w}_{i+1} \leftarrow \text{ChangeWeightsWithStep}(\mathbf{g}_{\max}, \mathbf{w}_i)$ 
18: end for
19:  $(\mathbf{w}, ERR) \leftarrow \text{WeightsWithSmallestError}(\mathbb{E}_v, \mathbf{p})$ 
20: return  $(\mathbf{w}, ERR)$ 

```

Algorithm 4.4: Procedure OptimizeWeights for the gradient directed optimization.

We recall from Section 3.2 that in this gradient directed optimization method we do not explicitly compute the regularization part of the optimization problem. Instead we change only the largest gradients (in terms of absolute values) of each iteration. The limit of modified gradients is tuned by the parameter  $\tau$ . In our case, lasso regularization seems to be the best choice, because it has been shown to lead to many weights being set to zero [Tibshirani, 1996], which means simpler and more interpretable models with fewer rules.

However, in practice it is hard to know in advance which value of  $\tau$  will result in the most accurate model. Both theoretical and experimental results suggest that differing regularization suit better different data types [Lounici et al., 2009;

Rakotomamonjy et al., 2011]. Thus, the most suitable value of  $\tau$  depends on the properties of the learning data. We overcome this problem by trying a set of different values of  $\tau$  (Algorithm 4.3 on page 32, line 5) and estimating their accuracies on a separate validation set, which is the same for all  $\tau$  values. In the end, the model with the smallest validation error is selected.

Our aim is to efficiently learn a rule ensemble model that is both small and accurate. Therefore, we start with a  $\tau$  value that creates a small model,  $\tau = 1$ , and then iteratively decrease the value of  $\tau$  until it reaches zero. We stop the loop if the validation error increases above a threshold, since it is unlikely that trying smaller values would result in a more accurate model. It is possible that we are stopping in only a local optimum and the result can be a suboptimal model. However, not evaluating the lower  $\tau$  values does not seem to lower the accuracy significantly in practical experiments. Also, this procedure is very effective, because, as Friedman and Popescu [2004] point out, most of the optimization time is spent on computing the covariances between the weights. Thus, as discussed in Section 3.2, the lower  $\tau$  values use the main part of computer time.

We now go through the optimization procedure `OptimizeWeights` presented in Algorithm 4.4 on the previous page in more detail. The procedure starts by initializing all the weights to zero and splitting the entire learning set  $\mathbb{E}$  into an internal training set  $\mathbb{E}_t$  and a validation set  $\mathbb{E}_v$ . Within the loop, we iteratively compute the gradients  $g_k$  for each of the weights (line 12) and then change the selected weights  $w_j$  in the most promising direction for a predefined step size (line 17). The step size is automatically computed constant which is based on the theoretically optimal step. See our publication [III, Appendix A] for a more detailed description of the step size computation.

In addition to this basic idea, there are some additional details implemented. First, on every 100-th iteration we stop the optimization if we are overfitting (line 6), that is, if the validation error starts to increase. Second, we can define a maximum number of nonzero weights in advance (lines 13–15), which makes a suitable parameter for setting the accuracy vs. simplicity trade-off.

We already have an experimental evaluation of the previous versions of the algorithm presented in our papers [II, III]. However, in the following sections we present some novel results: among others, the results of the FIRE with average normalizing discussed in Section 4.3.1.

## 4.4 Empirical Evaluation

In this section, our primary target is to evaluate the discussed new normalization for FIRE and compare it with previous versions. Because the novel average normalization affects only multi-target behavior of FIRE, we skip the single target data sets presented in [III]. In addition, to avoid needless redundancy we only briefly mention the equivalent test settings and parameters. The details can be found in the paper.

In the experimental evaluation, we investigate three issues. First, we compare FIRE with other multi-target methods. As reference algorithms, we use three other multi-target variants of popular tree based methods: regression trees [Blockeel et al., 1998], random forests [Koccev et al., 2007], and model trees [Appice and Džeroski, 2007]. In the comparison, we focus on the accuracy and size of the learned models. The model size is used to indicate the interpretability of the model. As described in Section 4.3, FIRE has a parameter that can be used to limit the total number of nonzero weights, that is, the number of rules and linear terms. We use this parameter in the second part of the evaluation to investigate how the size of a rule ensemble influences its accuracy.

Thirdly, based on all the experiments we evaluate the two presented normalization techniques, average and maximum based, and the effect of linear terms. In this part, we try to find out whether the novel more intuitive normalization is sometimes more accurate. We also want to know when the use of linear terms benefits us.

Our preliminary experiments showed that, in some cases, we can significantly reduce the model size with only a marginal decrease in accuracy. This is the reason why in the first evaluation we also include results for FIRE models with an arbitrary limit of 30 rules and terms (denoted as “Max 30” in the results). Another optional setting of the FIRE algorithm is whether to include linear terms in the model or not. We present both cases in all evaluation scenarios; models with linear terms are denoted as “+ linear”.

### 4.4.1 Experimental Setting

In this section, we present the algorithms and their parameter settings, evaluation methodology and data sets used.

Whenever possible, the parameters of FIRE are set to the values used by Friedman and Popescu [2008] in the RULEFIT method. When generating the initial set of trees (Algorithm 4.3 on page 32, line 1) we use 100 random trees with an average depth of 3 (in practice, the average number of terminal nodes in

our implementation is about 7). The optimization procedure (Algorithm 4.3 on page 32, line 5) is run with the gradient threshold parameter  $\tau$  values ranging from 1 to 0 in 0.1 decrements. In the OptimizeWeights procedure (Algorithm 4.4 on page 39), the initial set  $\mathbb{E}$  is split into 2/3 for training ( $\mathbb{E}_t$ ) and 1/3 for validation ( $\mathbb{E}_v$ ). The maximum number of optimization iterations is 10,000. The threshold for detecting error increase (Algorithm 4.3 on page 32, line 9 and Algorithm 4.4 on page 39, line 6) is 1.1 and the step size is computed automatically based on the optimal step size as described in [III] appendix. Our algorithm, as well as regression trees [Blockeel et al., 1998] and random forests [Kocev et al., 2007] used in our experiments are implemented in the CLUS<sup>2</sup> predictive clustering system [Blockeel and Struyf, 2002]. All the parameters for regression trees and random forests are set to their default values; regression trees use reduced error pruning, random forests consist of 100 trees.

We cover the other algorithm parameters only briefly here. The detailed parameter values of the algorithms are presented in [III].

For experiments with model trees, we use the multi-target implementation MTSMOTI by Appice and Džeroski [2007] with the recommended settings. Experiments with the rule ensemble methods RULEFIT and REGENDER were performed with the original implementations of their authors and with the settings that lead to the best performance in the original papers [Dembczyński et al., 2008a; Friedman and Popescu, 2004, 2008].

The data sets used in the experiments, together with their properties, are presented in Table 4.1 on the next page. Publicly available multi-target data sets are, unfortunately, scarce. In addition to a single public data set from UCI repository [Frank and Asuncion, 2011], we have collected ten previously analyzed data sets. The references are given in the table.

The accuracy of the learned regression models is estimated for each target attribute by computing the RRSE values  $ERR_{RRSE}$  defined in Equation 2.1.1 on page 11. The size of tree based models (regression trees, random forests, and MTSMOTI) is measured as the number of leaves in all the trees. The size of rule ensemble models (FIRE, RULEFIT, and REGENDER) is measured as the number of rules or the sum of the number of rules and linear terms, if linear terms are used. All the above measures are estimated using 10-fold cross-validation, where the folds for each data set are the same for all the algorithms.

---

<sup>2</sup>Available at <http://www.cs.kuleuven.be/~dtai/clus> under the GNU General Public License.

DATA SET	# EXS	% MISS VALS	# NOM ATTS	# NUM ATTS	# TAR ATTS	# ALL ATTS	SOURCE
COLLEMBOLAN	393	20.4	8	40	3	51	[KAMPICHLER ET AL., 2000]
EDM	154	0.0	0	16	2	18	[KARALIČ AND BRATKO, 1997]
FOREST KRAS	60,607	0.0	0	160	11	171	[DŽEROSKI ET AL., 2006]
FOREST SLIVNICA	6,219	0.0	0	149	2	151	[STOJANOVA, 2009]
META LEARNING	42	27.9	0	56	10	66	[DŽEROSKI ET AL., 2002]
MICROARTHROPODS	1,944	0.1	0	142	3	145	[DEMŠAR ET AL., 2006]
SIGMEA REAL	817	0.0	0	4	2	6	[DEMŠAR ET AL., 2005]
SIGMEA SIMULATED	10,368	0.0	2	8	3	13	[DŽEROSKI ET AL., 2005]
SOLAR FLARE	323	0.0	10	0	3	13	UCI
VEGETATION	29,679	0.0	0	64	11	75	[GJORGJIOSKI ET AL., 2008]
WATER QUALITY	1,060	0.0	0	16	14	30	[DŽEROSKI ET AL., 2000]

TABLE 4.1: THE EXPERIMENTAL EVALUATION DATA SETS, THEIR PROPERTIES AND REFERENCES.

To test whether any of the observed differences in accuracy and size between the algorithms are significant, we followed the methodology suggested by Demšar [2006]: First, we use the Friedman test to check if there are any statistically significant differences between the compared algorithms. If the answer is positive, we additionally use the Nemenyi post-hoc test to find out what these differences are, and present them by average ranks diagrams. These diagrams show all the compared algorithms in the order of their average ranks; the best are on the right and the worst are on the left side of the diagram. The algorithms that differ by less than a critical distance for a  $p$ -value = 0.05 are connected with a horizontal bar and are not significantly different. We perform such significance testing for each of the RRSE and model size metrics. Recall from Section 2.1.1 we show the RRSE values for two cases: for each target separately and averages over targets of single data sets. The results of the experimental evaluation are presented in the next section.

## 4.4.2 Results

The results of the algorithms are mostly the same as the ones presented in [III]. Only the used FIRE version has changed a little bit.

### 4.4.2.1 Multi-Target Regression

For multi-target data, the Friedman test shows that the RRSE values of algorithms are significantly different with a  $p$ -value  $< 2.2 \cdot 10^{-16}$ , if we treat each target separately, and with a  $p$ -value =  $3.3 \cdot 10^{-5}$ , if we compare target averages over each data set. The model sizes are different with a  $p$ -value =  $7.9 \cdot 10^{-11}$ . The average ranks and results of the Nemenyi test are given in Figure 4.1 on the next page for RRSE evaluated on separate targets (a), RRSE evaluated on target averages within data sets (b), and model size (c).

The ranking of algorithms in Figure 4.1 on the facing page is nearly equivalent to the results in [III]. The only notable difference is in the per data set target average evaluation (b), where the linear term version of FIRE is slightly and non-significantly winning over the random forest. Thus, we achieve at most only a little by changing our normalization style from the average based to the new maximum based.

In diagram (a) random forests and both unlimited FIRE versions are more accurate than regression trees, the limited versions of FIRE, and MTSMOTI. Due to small critical distance, the three more accurate algorithms are significantly better than the rest. Evaluation over target averages within data sets (b)

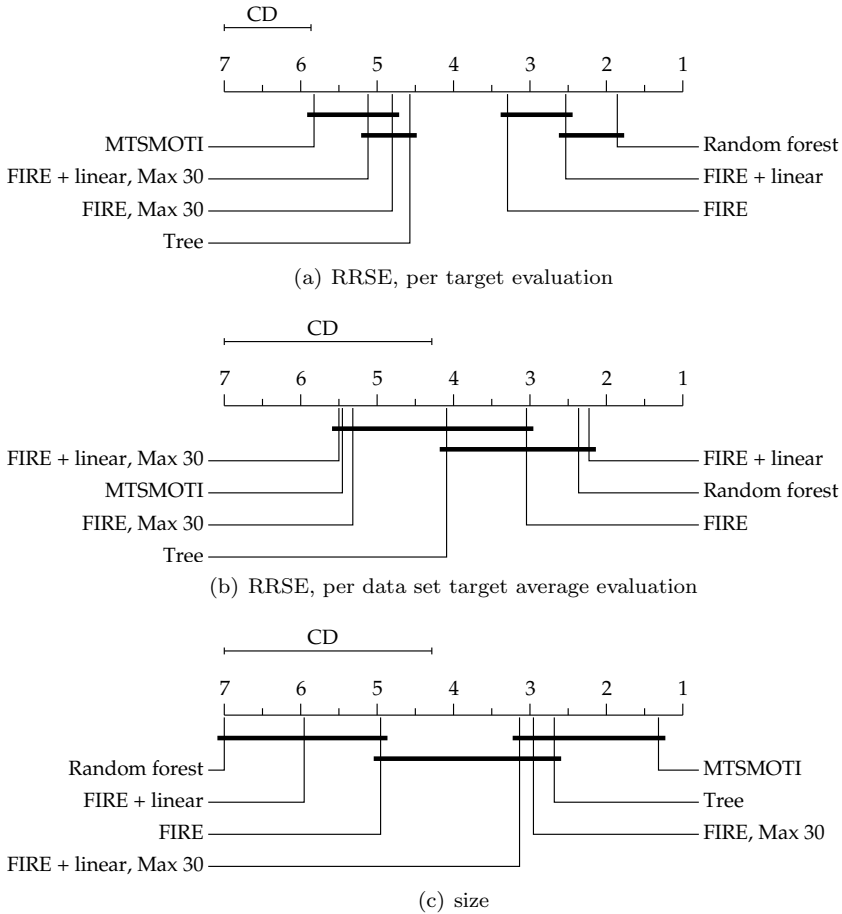


Figure 4.1: The average ranks diagrams on *multi-target* data for *RRSE* evaluated on *separate targets* (a), on *target averages within data sets* (b) and *model size* (c). Better algorithms are on the right-hand side, the ones that do not differ significantly by the Nemenyi test ( $p$ -value = 0.05) are connected with a horizontal bar. CD is the critical distance.

shows a similar picture, but because the sample size is smaller (11 data sets vs. 63 targets), the critical distance is larger and differences are less significant: Only FIRE with linear terms and random forests are significantly better than MTSMOTI and the limited versions of FIRE. Now linear terms seem to increase the accuracy of unlimited FIRE considerably but for limited versions the effect seems to be negative. The regression trees are usually in the middle class. Additional examination showed that we need a limit of 60–70 for FIRE to overtake regression trees. The diagram for size (c) is almost identical to the previous version of FIRE: The limited FIRE, regression trees and MTSMOTI are significantly smaller than the two most accurate models.

The detailed results of the algorithm comparison in Table 4.2 on page 50 are also almost identical to the ones achieved in [III] on multi-target regression data: the difference in average sizes between the random forest and the unlimited FIRE versions is huge but the accuracy differences are rather small. When both size and accuracy are taken into account, the unlimited version of FIRE with linear terms seems to perform very well on multi-target regression problems.

An interesting detail we should notice is that the average proportion of linear terms in the unlimited model is 24%, but it drops to only 3% if we limit the model size to 30. For some reason the linear terms do not seem very useful in small models. Only a small number of linear terms are kept in the resulting model and they seem to have mostly negative effect on accuracy. Also in our additional experiments we stressed the linear terms by multiplying them with the number of target values  $T$  as we proposed in Section 4.3.2. This modification increased the average amount of linear terms to 50% for limited FIRE, but only reduced the accuracy. From this, we could conclude that linear terms are a useful addition only if the training data is already well covered with rules.

It is also interesting that the proportion of linear terms highly depends on the data set. This suggests that for some data sets rules are not enough for high accuracy. For example, FOREST SLIVNICA and SIGMEA REAL seem to be quite linear in nature, because the unlimited FIRE version with linear terms prevails in these when compared with the one without them.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MtSMOTI	
	TARGET ATTRIBUTES	RRSE	#	RRSE	#	RRSE	#	RRSE	# [ %]	RRSE	#	RRSE	# [ %]	RRSE
COLLEMBOLAN	0.97	3	<b>0.92</b>	13,145	0.94	547	0.94	643 [ 15]	0.95	30	0.95	30 [ 0]	1.04	16
SPECIES-NB	0.98		<b>0.94</b>		0.96		0.96		0.97		0.97		1.01	
ABUD-TOTAL	0.96		<b>0.93</b>		0.95		0.95		0.96		0.96		1.10	
ABUD-F.QUAD	0.96		<b>0.89</b>		0.90		0.90		0.91		0.91		1.01	
EDM	0.72	11	0.69	2,923	<b>0.68</b>	617	<b>0.68</b>	645 [ 5]	<b>0.68</b>	30	<b>0.68</b>	30 [ 0]	0.85	2
D-FLOW	0.68		0.66		<b>0.64</b>		0.65		0.67		0.67		0.92	
D-GAP	0.75		0.71		0.71		0.71		<b>0.69</b>		<b>0.69</b>		0.77	
FOREST KRAS	0.62	1,142	<b>0.55</b>	1,530,421	0.65	830	0.64	2,590 [ 68]	0.72	30	0.72	30 [ 0]	0.69	10
CC	0.53		<b>0.47</b>		0.56		0.54		0.61		0.61		0.57	
FSH	0.49		<b>0.42</b>		0.53		0.52		0.61		0.61		0.59	
DELVEG	0.53		<b>0.47</b>		0.55		0.54		0.60		0.60		0.56	
VPV1-HMX	0.60		<b>0.52</b>		0.63		0.62		0.72		0.72		0.68	
VPV1-H99	0.58		<b>0.51</b>		0.62		0.61		0.71		0.71		0.68	
VPV1-H95	0.57		<b>0.50</b>		0.61		0.60		0.70		0.70		0.67	
VPV1-H75	0.57		<b>0.51</b>		0.61		0.60		0.70		0.70		0.67	
VPV1-H50	0.60		<b>0.54</b>		0.64		0.63		0.72		0.72		0.70	
VPV1-H25	0.66		<b>0.60</b>		0.70		0.70		0.76		0.76		0.75	
VPV1-H10	0.76		<b>0.69</b>		0.78		0.78		0.83		0.83		0.82	
VPV1-H05	0.83		<b>0.76</b>		0.83		0.83		0.87		0.88		0.86	
FOREST SLIVNICA	0.54	321	0.49	227,198	0.50	705	<b>0.47</b>	1,014 [ 26]	0.58	30	0.58	30 [ 10]	0.51	4
HEIGHT	0.56		0.51		0.53		<b>0.49</b>		0.61		0.62		0.52	
COVER	0.52		0.48		0.47		<b>0.45</b>		0.54		0.55		0.50	

TABLE 4.2: CONTINUED ON THE NEXT PAGE.

CONTINUED FROM THE PREVIOUS PAGE.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MTSMOTI		
	TARGET ATTRIBUTES	RRSE	#	RRSE	#	RRSE	#	RRSE	# [%]	RRSE	#	RRSE	# [%]	RRSE	#
META LEARNING		1.36	2	0.92	1,415	<b>0.88</b>	464	<b>0.88</b>	974 [52]	0.94	17	0.94	18 [ 0]	1.20	2
LTREE		1.47		0.93		<b>0.87</b>		0.88		0.95		0.95		1.20	
C50-RULES		1.46		0.92		<b>0.84</b>		<b>0.84</b>		0.93		0.93		1.17	
LINDISCR		1.12		0.87		<b>0.82</b>		0.83		0.89		0.89		0.96	
MLCIB1		1.49		0.95		0.89		<b>0.87</b>		0.95		0.95		1.28	
MLCNB		1.29		<b>0.92</b>		<b>0.92</b>		0.95		0.98		0.97		1.08	
RIPPER		1.33		0.93		0.85		<b>0.84</b>		<b>0.84</b>		<b>0.84</b>		1.13	
CLEM-RBFN		1.11		<b>0.89</b>		0.92		0.92		0.99		0.99		1.20	
C50-TREE		1.47		0.93		<b>0.87</b>		<b>0.87</b>		0.94		0.94		1.24	
CLEM-MLP		1.22		0.94		0.93		<b>0.91</b>		0.98		0.97		1.54	
C50-BOOST		1.51		0.95		0.90		<b>0.89</b>		0.98		0.97		1.20	
MICROARTHROPODS		0.77	52	<b>0.74</b>	12,411	0.75	642	<b>0.74</b>	1,059 [39]	0.84	30	0.84	30 [ 0]	0.83	18
ACARI		0.76		<b>0.71</b>		0.73		0.72		0.82		0.82		0.79	
COLLEMBOLAN		<b>0.74</b>		<b>0.74</b>		<b>0.74</b>		<b>0.74</b>		0.81		0.81		0.75	
SH-BIODIV		0.81		<b>0.75</b>		0.77		0.77		0.90		0.90		0.94	
SIGMEA REAL		<b>0.61</b>	12	0.65	22,506	0.68	158	0.66	180 [ 4]	0.74	17	0.77	30 [ 6]	0.62	7
MFO		<b>0.62</b>		0.67		0.73		0.72		0.77		0.82		0.64	
MSO		0.61		0.62		0.63		<b>0.59</b>		0.71		0.72		<b>0.59</b>	
SIGMEA SIMULATED		0.03	146	<b>0.02</b>	78,750	0.04	658	0.03	676 [ 3]	0.08	30	0.08	30 [13]	0.05	17
DISP-RATE		0.03		<b>0.02</b>		0.04		0.04		0.09		0.09		0.05	
DISP-SEEDS		0.03		<b>0.02</b>		0.03		0.03		0.07		0.07		0.04	

TABLE 4.2: CONTINUED ON THE NEXT PAGE.

CONTINUED FROM THE PREVIOUS PAGE.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MtSMOTI	
	RRSE	#	RRSE	#	RRSE	#	RRSE	# [%]	RRSE	#	RRSE	# [%]	RRSE	#
SOLAR FLARE	<b>0.99</b>	2	1.05	3,974	1.00	499	1.00	499 [ 0]	1.01	25	1.01	25 [ 0]	1.02	14
C-CLASS	<b>0.99</b>		1.03		1.01		1.01		1.00		1.00		1.03	
M-CLASS	<b>0.97</b>		1.04		0.99		0.99		0.98		0.98		1.00	
X-CLASS	<b>1.01</b>		1.07		<b>1.01</b>		<b>1.01</b>		1.03		1.03		1.04	
VEGETATION	0.82	349	<b>0.72</b>	1,102,508	0.80	826	0.79	1,541 [46]	0.88	30	0.88	30 [ 0]	0.89	8
NUMBER-SPP	0.80		<b>0.67</b>		0.74		0.73		0.85		0.85		0.94	
DEM	0.68		<b>0.52</b>		0.65		0.63		0.79		0.79		0.81	
TWI	0.70		<b>0.60</b>		0.67		0.66		0.74		0.74		0.80	
SOLAR	0.99		<b>0.97</b>		0.99		0.98		0.99		0.99		0.99	
THINVK	0.96		<b>0.88</b>		0.94		0.93		0.98		0.98		0.96	
THK	0.96		<b>0.88</b>		0.94		0.94		0.98		0.98		0.96	
EFF-RAIN	0.68		<b>0.53</b>		0.66		0.64		0.79		0.79		0.81	
MINT-JUL	0.72		<b>0.58</b>		0.69		0.67		0.82		0.82		0.83	
MAX-FEB	0.68		<b>0.53</b>		0.67		0.65		0.83		0.83		0.85	
GRND-DPTH	0.81		<b>0.71</b>		0.78		0.77		0.87		0.87		0.89	
SALINITY	0.94		<b>0.89</b>		0.92		0.91		0.97		0.97		0.97	

TABLE 4.2: CONTINUED ON THE NEXT PAGE.

CONTINUED FROM THE PREVIOUS PAGE.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MTSMOTI	
	TARGET ATTRIBUTES	RRSE	#	RRSE	#	RRSE	#	RRSE	# [%]	RRSE	#	RRSE	# [%]	RRSE
WATER QUALITY	0.96	5	<b>0.90</b>	41,568	0.94	802	0.94	995 [19]	0.94	30	0.94	30 [ 0]	0.96	8
CLAD-SP	0.99		<b>0.94</b>		0.96		0.96		0.96		0.96		0.98	
GONG-INC	1.00		<b>0.97</b>		1.00		0.99		0.99		0.99		1.00	
OEDO-SP	1.00		<b>0.94</b>		0.97		0.96		0.97		0.97		0.99	
TIGE-TEN	0.95		<b>0.88</b>		0.93		0.93		0.92		0.92		0.95	
MELO-VAR	0.98		<b>0.90</b>		0.93		0.93		0.94		0.95		0.97	
NITZ-PAL	0.90		<b>0.83</b>		0.87		0.87		0.86		0.86		0.89	
AUDO-CHA	0.98		<b>0.96</b>		0.98		0.98		0.97		0.97		0.98	
ERPO-OCT	0.97		<b>0.91</b>		0.94		0.94		0.93		0.93		0.97	
GAMM-FOSS	0.93		<b>0.80</b>		0.83		0.83		0.89		0.89		0.91	
BAET-RHOD	0.97		<b>0.89</b>		0.94		0.94		0.95		0.95		0.96	
HYDRO-SP	0.98		<b>0.90</b>		0.95		0.95		0.96		0.96		0.97	
RHYA-SP	0.96		<b>0.90</b>		0.92		0.92		0.93		0.93		0.94	
SIMU-SP	0.99		<b>0.94</b>		0.97		0.97		0.99		0.99		1.00	
TUBI-SP	0.89		<b>0.84</b>		0.91		0.91		0.87		0.87		0.91	
AVERAGE – TARGETS	0.87		<b>0.75</b>		0.79		0.78		0.83		0.83		0.88	
AVERAGE – DATA SETS	0.76	185.9	<b>0.70</b>	276,075	0.71	613.5	0.71	983.3 [25]	0.76	27.2	0.76	28.5 [2.6]	0.79	9.6

TABLE 4.2: COMPARISON OF *RRSE* FOR *multi-target* REGRESSION. FOR EACH DATA SET, WE FIRST GIVE THE *average RRSE over all targets*, AND THEN THE *RRSE for each target separately*. IN EACH ROW, THE SMALLEST ERROR IS TYPESET IN BOLD. SIZE IS GIVEN EITHER AS THE NUMBER OF RULES AND LINEAR TERMS OR AS THE NUMBER OF TERMINAL NODES IN TREES. BESIDES THE MODEL SIZES ( $\#$ ), WE ALSO GIVE IN BRACKETS THE PERCENTAGE OF LINEAR TERMS WITHIN THE TOTAL MODEL SIZE ( $[\%]$  IF PRESENT). TWO FINAL ROWS GIVE THE AVERAGE RRSEs OVER ALL TARGETS, OVER DATA SET TARGET AVERAGES AND AVERAGE MODEL SIZE OVER ALL DATA SETS.

#### 4.4.2.2 Model Size Limitation

Experiments presented in the previous section considered two size limit options for the FIRE algorithm, one with the maximum model size set to 30, and one without any model size restrictions. In this subsection, we present experiments with different values of the maximum model size parameter. These hopefully show how the model size limit influences the accuracy of models. In addition to the unlimited version, we use values between 10 and 100.

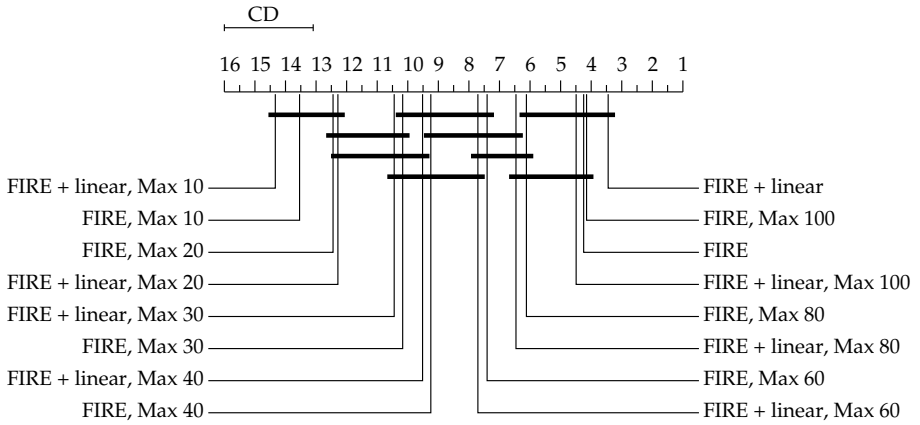
The average ranks diagrams comparing different maximum model size parameters are presented in Figure 4.2 on the following page. Diagrams (a) and (b) show RRSE on multi-target data for per-target evaluation and for per-data set target average evaluation, respectively. Because of a larger sample, there are more significant differences in (a) than in (b), however, what is common to both diagrams is the trend that smaller models are also less accurate. The size limitation parameter can therefore be used as an accuracy for simplicity (and interpretability) trade-off setting.

Moreover, it is interesting that linear terms usually have only small negative effect on the accuracy. The sole exception to this is the unlimited version where the linear term version surpasses. This result on average based normalization differs from the behavior of maximum based normalization [III]. In case of maximum based normalization, the FIRE versions with the limit higher than 40 terms got some benefit from the linear terms. For smaller ensembles, the effect was similar to the one we got on average normalization. Nevertheless, the unlimited FIRE with linear terms is now also the most accurate model. However, the difference with some other versions is not great.

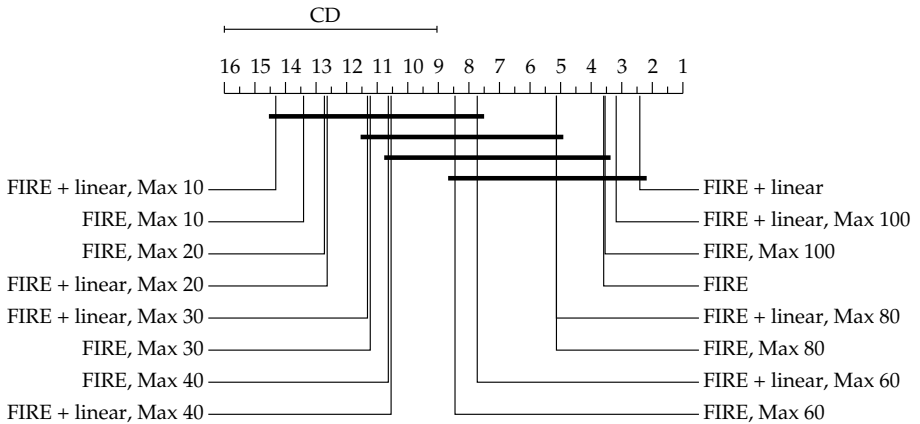
#### 4.4.3 Analysis of Normalization and the Use of Linear Terms

We also wanted to examine how the alternative rule normalization technique affects the rule ensemble prediction. The meaning of rule normalization is to scale both base models and targets so that they have somewhat similar impact on the loss function in optimization. The problem clearly more generally touches the field of multi-target prediction; nevertheless, it is more explicit for weighted rule ensembles. For single target prediction the problem is trivial. Friedman and Popescu [2008], for example, just make all the rule predictions equal to 1.

For normalization between targets, we have a clear solution, used also in [III]: scale the targets with their standard deviation. However, it is more problematic to scale each of the rules so that they have more or less the same impact on the



(a) per target evaluation



(b) per data set target average evaluation

Figure 4.2: The average ranks diagrams of FIRE with different model size limitations for  $RRSE$  on multi-target data evaluated on *separate targets* (a), and on *target averages within data sets* (b). Better algorithms are on the right side, the ones that do not differ significantly by the Nemenyi test ( $p$ -value = 0.05) are connected with a horizontal bar. CD is the critical distance.

loss function when all targets are considered. Our proposition in [III] was to scale the rules according to their maximal target prediction and ended up with all predictions on the interval  $[-1, 1]$ .

Nevertheless, perhaps such normalization stresses too much the rules with many approximately equal valued targets: In one extreme we may end up with a rule predicting 1 for all the targets or with a rule predicting 1 for a target and zeros for the rest. In the previous case the average of all the  $T$  predictions is 1 and in the latter  $1/T$ . As in Section 2.1.1, our loss function is an average aggregation of single target loss functions. Thus, the average of rule predictions should give a good impression of the effect of rule on the loss function.

Hence, in this thesis we proposed an alternative way of normalization: average based technique. Instead of scaling the predictions based on the maximal value, we rather do it on the average of the absolute values of predictions. Thus, all the rules are bound to have the same average of 1 and roughly the same impact on the loss function. Nevertheless, in this case we can not give any guarantees for the single prediction values as we did for maximum based normalization.

Based on the experiments presented in [III] and in the previous section the differing effect of two normalization variances on the resulting model seems to be quite small. The average model size for unlimited FIRE seems to be 10–15% larger for average based than for maximum based normalization. There is no clear difference in average accuracies, but diagram (b) in Figure 4.1 on page 45 suggests some statistically insignificant improvement in average normalization for the unlimited version.

However, when we compare multiple model size limitations for both the FIRE normalizations, we get a differing result. The Nemenyi test comparison is presented in Figure 4.3 on the following page. In the figure, we denote by “MAX” and “AVG” the maximum and average based normalization respectively.

First we note that the maximum based normalization is nearly always statistically non-significantly more accurate than the analogous average based normalization. Sometimes the difference is even notable, e.g. with unlimited versions. The only exceptions are the smallest versions with a maximum term limit of 30. However, further experimentation on small limit sizes shows that this is barely an exception and maximum based normalization prevails also on small limits. Also based on results in [III] we note that linear terms are helpful in multi-target evaluation usually only with term limit 50 or greater.

In addition to normalizing the rules, we have to solve the same problem for multi-target linear terms. We have now normalized the linear terms only by scaling most of the values on the interval  $[-1, 1]$ . Nevertheless, in the average



based normalization if we want to make the linear terms equal to rules in terms of impact, we should take care that the averages over the absolute values of target predictions are similar. To achieve this we could simply multiply the single nonzero prediction of linear terms with the number of target values  $T$ . Like proposed in Section 4.3.2 we also tried this in our initial experiments, but surprisingly this seemed to end up in more inaccurate models. Probably this put too much stress on linear terms in the optimization.

## 4.5 Conclusions on the Multi-Target Rule Ensemble Method

In this chapter, we went through some rule ensembles methods for regression learning. We also presented our algorithm FIRE for generalizing rule ensembles to multi-target prediction.

FIRE has a simple parameter for limiting the size (the number of rules and linear terms) of the learned model. This enables us to trade accuracy for size (and interpretability) of the learned models. We experimentally evaluated its behavior with different normalization techniques, limit parameter values, and whether optional linear terms are added to the ensemble.

We got two important conclusions on FIRE. First of all, which one of the normalization tactics should we use? Roughly speaking, it does not seem to matter much. If we have preferences of some kind on the relation between the target prediction values, it can easily be used. However, because of somewhat smaller models and slightly increased accuracy the maximum based normalization presented in [III] should maybe be the default choice.

Second, as a general suggestion we could conclude that for larger, with term limit 50 or greater, multi-target rule ensembles adding linear terms should benefit. On the other hand, for smaller rule ensembles we should select the version without them. However, for single target rule ensembles the linear terms seem to help all the time as seen in [III].



# Chapter 5

## Binary Search Trees in Online Context

In this chapter, we change our focus temporarily from machine learning to the world of data structures. However, in the next chapter we will show how much similarity can be found in these seemingly distinct areas of computer science.

A *binary search tree* (BST) is a data structure where the integer valued keys are stored in a binary tree like structure in symmetric order. A tree consists of *nodes* which include a key, two references to left and right *child* and possibly additional constant sized fields. If  $y$  is a child of  $x$ , then  $x$  is the *parent* of  $y$ . If a node  $y$  can be reached with one or more parent-child steps from the node  $x$ ,  $y$  is a *descendant* of the node  $x$  and  $x$  is called an *ancestor* of  $y$ . For any node  $x$  in a BST, its left (resp. right) descendants have key values less than<sup>3</sup> (resp. greater than) the key value of  $x$ . Any node  $x$  and its descendants form a *subtree* where  $x$  is considered the *root*. Because of the symmetric order, there cannot be cycles in the BST. Thus, for any subtree there is an unambiguous root. If no subtree is mentioned, we refer to the root of the whole BST. The *depth* of a node is the amount of parent-child steps from the root to the node. Thus, the root has a depth of 0. The *height* of the tree is defined to be the depth of the deepest node. An example binary search tree is presented in Figure 5.1 on the next page.

---

<sup>3</sup>Sometimes multiple elements with equal key value are allowed. Then e.g. the left subtree may include these. However, only one of the elements can be accessed with normal BST access.

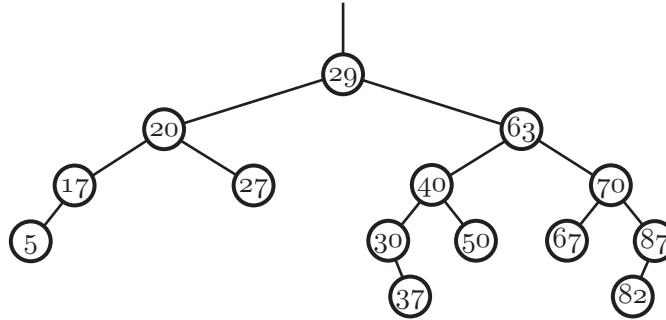


Figure 5.1: A binary search tree of height 4.

Clearly the form of BST is flexible: a BST of size  $n$  can be either a singly-linked list of length  $n$  or a perfectly *balanced* BST with a height of  $\lceil \log(n+1) \rceil$ .<sup>4</sup> Because the access time in BST depends on its height, there is plenty of work on minimizing it, that is, balancing the tree. Traditionally local modifications called *rotations* are used for this [Cormen et al., 2009]. Often BSTs have a balance invariant independent of the access sequence. The invariant is maintained by the rotations during every access.

## 5.1 Splay Trees in Theory

*Splay tree* [Sleator and Tarjan, 1985] is a BST which keeps balance by splay operation during every access. Splay trees do not have provably logarithmic worst-case bounds of individual operations, but still behave well under amortized analysis. When splaying, the accessed item is elevated to the root of the tree using specific rotations. These operations keep the tree pretty well in balance. Because no other balancing is enforced, a splay tree does not contain any additional information and, thus, does not require extra storage space. Because of its strategy, a splay tree also keeps recently accessed items very near the root.

Splay trees are most well known for their vast theoretical properties. Following results are reported for  $m$  size access sequence  $x_1, x_2, \dots, x_m$  in amortized asymptotic sense. Amortized bounds could be called worst-case bounds over sequences, not single accesses. Bădoiu et al. [2007], for instance, list at least the following properties:

<sup>4</sup>We use the base 2 for the logarithm:  $\log = \log_2$ .

- 
- **Balance bound** states that if no information about the access distribution is known, splay tree is as efficient as any possible offline or online BST [Sleator and Tarjan, 1985]. More formally, the access of element  $x$  is bound by  $O(\log(n + 1))$  in amortized sense. There are other BSTs like red black tree [Bayer, 1972; Cormen et al., 2009] that achieve this in worst-case for single access.
  - **Entropy bound** or **static-optimality bound** states that splay trees are in amortized sense as efficient as any offline BST that knows beforehand the access frequencies of elements [Sleator and Tarjan, 1985]. Note, that splay tree achieves this without knowing the frequencies. More formally, let  $p(x)$  be the access frequency for element  $x$  in the BST. The splay tree access time for the element is bound by  $O(1 + \log[1/p(x)])$ .
  - Another interesting bound is so called **static-finger bound** which denotes that for any fixed key  $f$  the access time of element  $x$  is relative to the difference of their key values, or *rank distance*  $d(f, x)$  [Sleator and Tarjan, 1985]. Rank distance  $d_i(y, z)$  is defined by the number of elements in BST between  $y$  and  $z$  in time  $i$ . Formally the bound states that splay tree accesses element  $x$  in  $O(\log[d_i(x, f) + 2])$  amortized time. Here the  $+2$  term is included by Bădoiu et al. [2007] to make the time positive. What makes the bound interesting is the fact that splay tree achieves it again without knowing the  $f$  at all, that is, for all alternatives of  $f$  simultaneously.
  - The **working set bound** is more general than any of the previous ones and thus implies all the previously mentioned in amortized sense [Iacono, 2001b]. It roughly states that recently accessed elements are still efficiently accessible. Formally, let  $w_i(x)$  be the number of different elements accessed before time  $i$  since the last access of item  $x$ . In this case, splay tree is able to access element  $x$  in  $O(\log[w_i(x) + 2])$  time [Sleator and Tarjan, 1985].
  - The **dynamic-finger bound**, however, is independent from the working set bound. It roughly states that elements that are near, in terms of rank difference, the last accessed elements are efficiently accessible. Formally, it states that the access  $x_i$  takes at most  $O(\log[d_i(x_{i-1}, x_i) + 2])$  time. The amortized access time bound was conjectured for splay trees by Sleator and Tarjan [1985] and proved by Cole [2000] and Cole et al. [2000].

Iacono [2001a] and Bădoiu et al. [2007] point out that separately working set and dynamic finger properties fail at least on sequences like

$$1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, 3, \frac{n}{2} + 3, \dots, \frac{n}{2}, n, 1, \frac{n}{2} + 1, \dots .$$

Both the working set and dynamic finger bounds give only  $O(m \log n)$  bound for access. The authors unite the two bounds in a strictly more general one called the **unified bound**. It roughly states that elements that are near the previously accessed elements are efficiently accessible. Formally let  $S_i$  be the set of elements in BST before access  $i$ . Then the unified bound can be defined by

$$O\left(\min_{y \in S_i} \log[w_i(y) + d_i(x_i, y) + 2]\right).$$

Bădoiu et al. [2007] also conjecture that splay tree accesses are bounded by this in amortized sense, but no BST has yet been proved to achieve the unified bound. The authors also introduce a non-BST data structure having this property in the worst case. Nevertheless, there are other attempts to prove nearly the unified bound for BSTs in amortized sense [Bose et al., 2010; Derryberry and Sleator, 2009].

Behind all these bounds, we are naturally aiming for the **dynamic optimality bound** which states that no other search tree algorithm is asymptotically more efficient than the considered one for any access sequence. This was conjectured in amortized sense for splay trees by Sleator and Tarjan [1985] and there has been some progress on this. For example, Demaine et al. [2007] present an algorithm that is a factor of  $\log \log n$  far from it.

An interesting theoretical examination of splay trees has been presented by Subramanian [1996]. He proposed a more general group of trees possessing similar properties as splay trees. The splay heuristic has additionally been applied in other tree structures [Badr and Oommen, 2005].

## 5.2 Practical Efficiency of Splay Trees

In practical use, the splay trees are problematic because of the amount of expensive rotations during each access. The need for splaying can be reduced by randomizing the decision of whether to splay or not in connection of an operation [Albers and Karpinski, 2002; Fürer, 1999] as well as by heuristic limit-splaying algorithms [Lai and Wood, 1991; Sleator and Tarjan, 1985; Williams et al., 2001].

---

Also, the theoretical bounds mentioned before would suggest that splay trees should work well in environments with a high locality of reference. However, some empirical studies [Bell and Gupta, 1993; Heinz and Zobel, 2002; Williams et al., 2001] have indicated that, when related to other data structures, they could actually be at their best in highly dynamic environments, where the focus of locality drifts over time. Moreover, despite careful implementation basic splay tree variations have empirically been observed to be less efficient than red black trees (RBs), standard BSTs, and hashing at least in some situations [Bell and Gupta, 1993; Williams et al., 2001]. Randomized adaptive data structures can do better [Albers and Karpinski, 2002; Williams et al., 2001], but only heuristic limit-splaying has been competitive in practice [Williams et al., 2001]. However, some more recent studies [Lee and Martel, 2007; Pfaff, 2004] have demonstrated that in some settings splay trees may be more efficient than other BSTs. Motivated by these, we studied what could be achieved by splaying only when a need for an update is noticed [IV].

The idea of our algorithm WSPLAY [IV] is to track the changes of the access sequence in a sense of working set property. Randomized splay trees achieve some practical savings without giving in on asymptotic efficiency. However, they do not pay attention to the request sequence and we achieved some improvement with this.

In particular, we examine access request sequences that exhibit locality of reference in the form of *working sets* [Denning, 1980]. This means that, at any time interval, most accesses refer only to a small portion of all keys—the current working set. Real-world situations often conform with this assumption. Of course, we may not afford to implement a too complicated request sequence monitoring method.

The WSPLAY algorithm maintains a (discounted) counter to monitor the depths of recent searches in the tree. Low average search depth indicates that a working set exists near the root of the tree and is being actively used. Occasional deep searches do not change the situation. Only when the searches are constantly deep, is there a need to update the splay tree.

Independent of us Lee and Martel [2007] proposed an algorithm very similar to ours [IV] for cache efficient splaying. Their algorithm uses a sliding window of accesses. The algorithm executes splaying if a too large portion of the accesses is deeper than a predefined limit depth. They also present experiments somewhat similar to ours. However, their test setting is a lot more static and, thus, gives a more optimistic view on the proposed algorithms.

### 5.3 Conditional Adaptation of a BST

Before introducing the BST algorithm intended to cope with working sets, we give the basic philosophy of the algorithm: how to identify a working set and its change.

First of all we should execute splay operations only when the active working set is not near the root or when the whole tree is unbalanced. Thus, unnecessary splaying is avoided in accessing a single item outside the working set as well as an item of the working set already near the root.

In order to gain knowledge of the input we simply maintain a discounted depth counter, which gives us information about the relation between on-going access operations and the current working set. If the last few access operations are deep, we can conclude that the working set has changed or the tree is not in balance. Thus, there is a need to splay to correct the situation.

For our counter, we need an approximate value for the size of working sets  $b$ . See, e.g., [Denning, 1980; Dhodapkar and Smith, 2002] for techniques on approximating  $b$ . With this value, we can calculate the limit depth  $limit_w$ , which represents the acceptable average depth for access operations in working sets. We set  $limit_w = a \log_2(b + 1)$ , where the multiplier  $a$  is a constant chosen suitably for the current environment.

The value of the *condition counter* is updated in connection of an access operation to *depth* as follows:

$$counter \leftarrow d \cdot counter + depth - limit_w.$$

Here the *discounting factor*  $d$ ,  $0 \leq d < 1$ , is a constant regulating the impact of the history of access operations on the current value. The difference  $depth - limit_w$  tells us how much (if at all) below the limit depth we have reached. If the value of *counter* is non-positive, we may assume that splaying is not required. On the other hand, a positive value suggests that the operation is needed.

Taking discounted history into account ensures that isolated accesses outside the working set do not restructure the tree needlessly. On the other hand, giving too much weight to earlier accesses makes the data structure slow to react to changes in the working set.

We analyzed in [IV] the case where the  $limit_w$  value is a constant. Using simple calculus we can prove that the average depth of non-splayed phases is bound by  $2limit_w = 2a \log(b + 1)$ . Thus, we get the amortized bound of  $O(a \log b + \log n) = O(limit_w + \log n)$ . By restricting  $limit_w = O(\log n)$  we clearly get overall amortized bound  $O(\log n)$  like for splay tree.

However, we could as well let the value of  $limit_w$  change during the execution of the algorithm. In fact, in the empirical evaluation of [IV] we use a dynamically changing  $limit_w$ . We can clearly keep the time bounds by giving maximal value for the dynamic  $limit_w$ . In the next section, we introduce the version of WSPLAY that was used in the experiments, that is, the version where  $limit_w$  changes during the execution.

There are alternatives for our approach. We could, for example, get rid of the whole discounting philosophy and use individual access counters for the keys. These counters should be included either in the nodes of the tree or kept in a separate data structure. However, e.g., Lai and Wood [1991] have already inspected the first approach with splay tree. Also Seidel and Aragon [1996] introduced randomized search trees and Cheetham et al. [1993] conditional rotating based on this approach. Because keeping the nodes free from additional information is an essential part of splay trees, we would, nevertheless, like to find another way.

On the other hand, if the counters were kept in a separate data structure, updating this information would be problematic. We are, anyway, mostly interested in only the last access operations. Thus, too static counters would not react quickly enough to the altering working set. Hence, we should have a technique to decrease the significance of old access operations. Lee and Martel [2007] solve the problem by counting only the amount of deep accesses, that is, the accesses that are deeper than the limit. Thus their solution loses information about the access depths. Another possibility would be to have a circular buffer which includes the depths of accesses in a window. With this solution, the average depth of accesses could be easily computed. However, the use of discounted depth computation does not force us to set an explicit history horizon after which depths are forgotten.

## 5.4 A Dynamic Version of Wsplay

Let us now introduce in more detail an algorithm based on the information collecting approach described above. WSPLAY in Algorithm 5.1 on the following page simply splays whenever the condition counter implies that the working set is changing.

The main functionalism of WSPLAY algorithm is on the lines 12–22. Because of simplicity, this was also the version presented in [IV]. Function BSTACCESS implements the standard BST access and returns the depth of the accessed item. As we want to execute the more efficient top-down version of splaying, we splay

**Constants:** the factors for discounting  $d$ , and  $limit_w$  adjustment  $c$   
**Parameters:** initial value for  $limit_w$ , and the target portion of splaying  $p$

```

1: {A starting window without splaying.}
2: if number of accesses <  $1/p$  then
3:   BSTACCESS( $x$ )
4:   return
5: end if
6: {Adjust  $limit_w$  to get portion  $p$  of splaying.}
7:  $\Delta limit_w \leftarrow c(\text{number of splays}/\text{number of accesses} - p)$ 
8: if Splayed last access and  $\Delta limit_w > 0$  or
   Non-splayed last access and  $\Delta limit_w < 0$  then
9:    $limit_w = limit_w + \Delta limit_w$ 
10: end if
11: {The rest code lines include the original static WSPLAY.}
12: if counter > 0 then
13:    $depth \leftarrow \text{SPLAY}(x)$ 
14:    $counter \leftarrow depth - limit_w$ 
15: else
16:    $depth \leftarrow \text{BSTACCESS}(x)$ 
17:    $counter \leftarrow counter \cdot d + depth - limit_w$ 
18:   if counter > 0 then
19:      $depth \leftarrow \text{SPLAY}(x)$ 
20:      $counter \leftarrow depth - limit_w$ 
21:   end if
22: end if

```

Algorithm 5.1: The dynamic WSPLAY algorithm.

if *counter* indicates the need for splaying at the beginning of access at line 12. This is the case when the previous access operation, which necessarily included splaying, was deep. Thus, we avoid doing unnecessary BST access before every top-down splay. Otherwise, on lines 16–21, we access the item, update the condition counter, and splay if the updated counter value indicates a need for it. Observe that, after each execution of the SPLAY function, on lines 14 and 20 the history of access depths is erased.

The problem in evaluating WSPLAY is to control its two parameters  $d$  and  $limit_w$ . The rest of the code addresses this. Examining all value combinations would be a massive task. However, moderate changes in the value of parameter

$d$  do not affect the results much in practice. Hence, we can easily use a constant value near 1 for it. In [IV] we used  $d = 0.9$ .

The parameter  $limit_w$  is more problematic because with different values the results vary greatly. However, we are mostly interested in the efficiency of splay reduction by monitoring. Thus, it would be natural to compare its efficiency against data structures that reduce splaying without such monitoring. Randomized splay trees [Albers and Karpinski, 2002; Fürer, 1999] do it at random. The data structure of Albers and Karpinski [2002], referred here as RSPLAY, matches WSPLAY perfectly in a sense that it contains no other modifications to basic splay trees than reduced splaying.

RSPLAY needs as a parameter the probability  $p$ . With the probability  $1 - p$  standard BST access is executed instead of splaying. We modify WSPLAY to execute the same amount of splaying by including simple adaptation for the variable  $limit_w$ . We adjust the limit in the beginning of every access by adding a value proportional to difference between  $p$  and amount of executed splay operations. This is done on lines 7 and 9 of the algorithm. For this we need yet another constant  $c$  to tune the agility of limit depth adjustment: we use  $c = 0.1 \log(n)$ .

Nevertheless, oscillation around the optimal limit depth is clearly a possibility in the adjustment. To prevent this we allow the modification of  $limit_w$  only if the direction of change  $\Delta limit_w$  agrees with the information given in the last access. Thus, if last access is splayed, it is possible that  $limit_w$  is too low and it might have to be increased and vice versa. This is done on the lines 8–10 of the algorithm.

Moreover, one more practical thing is that we need to prevent the algorithm from executing all the available splaying in the beginning. This could happen because of the great influence of first access depths on the  $limit_w$  adaptation. Thus, on lines 2–5 we start the sequence with a margin of length  $1/p$  where no splaying is allowed.

With this version it is possible to relate RSPLAY and WSPLAY with different values of  $p$ .

In the experiments with highly dynamic data reported in [IV] the gains were contradictory. On one hand, both the average access depth and time were reduced when compared with randomized splaying. This was also the case in comparison with splay with high skewness. On the other hand, splay trees excelled with a high locality of reference. Also red black trees were still usually more efficient in these experiments.

However, as mentioned in the previous section, Lee and Martel [2007] got more optimistic results with a similar algorithm, but using more static data set.



# Chapter 6

## Working Sets as Drifting Concepts

Recall from Section 2.2, locality of reference considered in Chapter 5 has much to do with the concept drift occurring in machine learning. In this chapter, we point out the analogies and try to find a common framework for both of the phenomena. Our aim in the work is to present solutions used for one field that could be useful for the other one.

We first start by illustrating the characteristics of each of the fields. Then we continue with introducing our own proposal for a more general problem statement that includes both of the areas. Moreover, we explain the progress made on both sides. In this, we are almost exclusively referring to work done by others.

### 6.1 Working Sets and Locality Phases

In the area of process scheduling and memory paging a very long known regularity called *locality of reference* (LR) is observed. The effect is also called *principle of locality*. Denning [1968, 1980] presented *working sets* to model this phenomenon. He defined a working set to be the set of referenced memory pages by a process during some time interval. An essential property of his working set definition is that we expect past page references to be a good prediction for immediate future references. A working set can also be presumed to roughly indicate the process using it [Dhodapkar and Smith, 2002]. The information

can, thus, be used to more generally predict the behavior of currently executed processes. An interesting glance to the history of LR is given by Denning [2005].

In theoretical discussion, modeling LR makes efficiency estimation more relevant in practice [Albers et al., 2005; Angelopoulos et al., 2008; Dorrigiv and López-Ortiz, 2008]. More directly, knowledge about working sets can be used to increase the theoretically proved performance of various algorithms: e.g., by analyzing the amount of cache misses when predicting referenced pages or in data structure analysis [Bose et al., 2010; Bădoiu et al., 2007; Iacono, 2001a; Sleator and Tarjan, 1985].

LR is often split into two distinct categories: In case of *temporal* locality, a reference is often reused in the near future. In *spatial* locality, references near each other, in a sense of memory address location, are often used simultaneously. In addition, more complex patterns can be formed. Especially in spatial dimension references often form a linearly increasing order.

The current research within this area is often based on indicating recurring *phases* [Denning, 1980; Shen et al., 2007] instead of analyzing the change. A phase is the maximum interval over which the working set remains more or less constant. When a phase that has appeared previously is indicated again, we restore the information gathered about it. In practice restoring may mean selecting and moving the corresponding reference pages to the more efficient memory in advance. This process is called *refetching* [Gramacy et al., 2003].

When addressing working set phases, we identify at least the following concrete tasks:

- Like Dhodapkar and Smith [2002] mention, the working set can be indicated by collecting its *identity* and *size*. Because not all the accessed elements are actually accessed multiple times (we could talk about noise) some robustness is needed in identity tracking. Otherwise we are identifying too many working set phases. For example in case of refetching, we are bound to be overwhelmed by the cost of reference page data movement. The identification could include information about the working set size, a fingerprint and the interval of reaccessing instances, called *reuse distance* [Ding and Zhong, 2003].
- Traditionally the methods for tracking the phases use either a sliding window or a series of non-overlapping windows following each other. However, also the use of multiple simultaneous windows has been tried [Nagpurka et al., 2006]. A straightforward way to discover similarities with different working set sequences  $W_1$  and  $W_2$  is presented by Dhodapkar and Smith

[2002] as a concept of *working set distance*:

$$\delta(W_1, W_2) = \frac{|W_1 \cup W_2| - |W_1 \cap W_2|}{|W_1 \cup W_2|}. \quad (6.1)$$

The working set distance is a symmetric metric. It can be used both to identify the working set and as a way to discover a change in the working set phase by a sudden increase.

- On the other hand, the change of phase is often discovered by the radical increase in the cache misses or other changes in the model statistics of current references gathered in online manner. Thus, the metric for indicating current distance to the previous noticed working set phase can vary greatly.

## 6.2 Concept Drift

A usual problem in real-world machine learning applications is the existence of unknown *hidden context* that affects the examined concept or topic. A change in the hidden context causes inexplicable alteration in examined concepts which should be adapted to in learning progress. This process of, at least partially, unpredictable change is generally called *concept drift* (CD). The change in the hidden context may be repetitive. In addition, the change may appear abruptly or gradually. Tsymbal [2004] gives an interesting survey about CD in machine learning.

An example of CD with recurrent states is the behavior of consumers buying products [Tsymbal, 2004]. The behavior alters due to daytime and time of the month and year. However, also nonrecurring events, like opening a new shop nearby, occur.

As pointed out in Section 2.2.1 the examination of CD can be separated in three approaches [Tsymbal, 2004]: First, we may have some selected instances as a current model for the concept. In practice this is executed with a fixed or an adaptive size sliding window [e.g., Widmer and Kubat, 1996]. Second, we could weight instances according to their freshness [e.g., Klinkenberg, 2004]. However, this approach is useful only if the learning method used can exploit weighed instances. The third alternative is to have a set of basic concept descriptions. The current concept description could be a mixture of these basic ones. In particular the instance selection mentioned first, clearly has analogy with tracking down the current working set in LR.

However, as Tsymbal [2004] mentions, in practice some differences can still be observed in the type of CD. For example, CD may be *virtual* and consist on seeing only part of the data set. In this case real CD on the data set does not really exist, but similar phenomenon is observed. In practice, usually experiments are done with virtual CD and it is not possible to differentiate it from a real one. Another categorization is that the CD may be either gradual or instant in nature. In gradual CD, the concept is changing little by little and our model of the current concept is decaying slowly.

For CD we, as in Section 6.1, find the following tasks:

- First we should have some way to indicate and model the current concept. We suggested the use of a training set fingerprint in [V]. The solution is simply a generalization of histograms as suggested by Sebasti ao and Gama [2007]. The fingerprint can consist on either a sample of the current model or some of its precomputed statistics. However, traditionally we can often use the very trained machine learning model as a fingerprint. The model is itself a collection of statistics of the current sequence. One more thing to notice is that extreme instances, the so called *outliers*, should not affect our model too much. This problem is analogical to the robustness of working set identification mentioned for LR.
- Change of concept is usually discovered by the radical decrease in the accuracy of a model in predicting future instances [e.g., Widmer and Kubat, 1996]. Thus, as a metric indicating concept we could use the amount of wrong predictions. However, a fingerprint can again be used for this [V].
- For the distance between concepts Widmer and Kubat [1996] present us the computational learning theoretical relative error between the concept sequences  $W_1$  and  $W_2$  as a *drift extent*:

$$\delta(W_1, W_2) = |W_1 \cup W_2| - |W_1 \cap W_2|.$$

This can be used as a symmetric metric that indicates both the reoccurring states and change in the current concept. Actually, this is just an unnormalized case of the metric presented for LR in Equation 6.1 on the previous page.

Traditionally, CD is considered as an online learning task, where the feedback of the correction is given right after the prediction. Thus, instances are considered arriving an instance at a time. However, recently also getting instances

in *batches* for simplicity has been considered [Klinkenberg, 2004]. Furthermore, [Gao et al., 2007] even present that CD should be considered in data stream model.

### 6.3 The Simple Generalized Problem

As our aim is to find common elements in two different fields with their own terminologies, our first task is to define the united terms for the common problem. We are not informed about any other similar attempts for the generalized problem.

We simulate the online environment, similar to Klinkenberg [2004], with input given in *batches*. Single input element is called an *instance*:

**Problem Definition 2.** *Let  $D_1, D_2, D_3, \dots$  be a sequence of (possibly i.i.d.) distributions. Let  $x_{(i,1)}, x_{(i,2)}, x_{(i,3)}, \dots, x_{(i,m_i)}$  be a sequence of instances  $x \in X$  drawn from the distribution  $D_i$ . The total input sequence is*

$$\begin{aligned} &x_{(1,1)}, x_{(1,2)}, x_{(1,3)}, \dots, x_{(1,m_1)}, \\ &x_{(2,1)}, x_{(2,2)}, x_{(2,3)}, \dots, x_{(2,m_2)}, \dots \end{aligned}$$

A sequence  $x_{(i,\cdot)}$  is called batch  $i$  and it forms a concept.

Our task is, possibly in online fashion, to:

- (A) Track the current distribution  $D_i$  down and get its features, potentially even the density function  $p_i : X \rightarrow [0, 1]$ . In other words, we need to answer the questions “How important the instance is to our algorithm?” and “What are the specific characteristics of the current distribution?”.
- (B) Notice when the current distribution changes.
- (C) Store the information we have about the current distribution. We call this stored information a model of the concept. Also we need to decide if storing is worthwhile.
- (D) Decide if it is useful to restore some previous model. This is the case if the situation reoccurs. In addition, we have to organize the restoring.
- (E) Adapt the restored model or train a new one for the current distribution.

We mention the online environment in the definition with the intention of restricting the processing time to at most constant per instance. It is also

worth noting that overlap between distributions during shifts is not allowed. Implicitly in this definition, there are some often met characteristics of the tasks. First, Task A often includes separating acceptable variance from real distribution change [e.g., Widmer and Kubat, 1996]. Also the task can be difficult to answer even in hindsight because we may never have the real density function explicitly available. Thus, we are searching for the most efficient use of this information regardless of the “true answer”. In addition we have to notice that the Tasks C and D are useful only in case of reoccurring distributions.

We do not necessarily assume that instances in a batch  $i$  are drawn i.i.d. from the distribution  $D_i$ . Thus, in this definition, the probability of drawing an instance may also relate to previous ones in the same batch. This, on the other hand, allows for example gradual change or spatial locality in references. Nevertheless, it could be acceptable to make the i.i.d. assumption inside batches like Klinkenberg [2004] does. In any case, we need to note that between the batches i.i.d. surely can not be assumed. This affects the evaluation of the algorithms because techniques like cross-validation in CD should not be used.

We now introduce how the features of LR and CD correspond to the generalized problem definition.

### 6.3.1 Relation to LR and CD

In the framework of LR, our instances  $x$  are computer program accesses. These may be either memory address references for low level programming or element key numbers for data structures. If the instance is in a working set of an algorithm, the instance is probably referenced again soon. In addition, the concepts relate to working sets or, better yet, to locality phases.

Now the Task A is about tracking down how relevant the instance is in terms of working sets [e.g., Sleator and Tarjan, 1985]. The task could, however, be simplified to finding out the most accessed instances by having a threshold  $\alpha \in [0, 1]$  and saying that  $x$  is in the working set if  $p(x) \geq \alpha$ . Thus, the problem is simplified to asking if the instance is in the working set and we are just tracking down a binary function  $p_{\text{bin}} : X \rightarrow \{0, 1\}$ . However, an exact nature of working sets is not necessarily what we are searching for.

On the other hand, the Task B and possibly also Tasks C–E appear more clearly in the framework of locality phases [Denning, 1980; Shen et al., 2007]. In this framework, we are tracking down the phases where the input distribution  $D_i$  is in a constant state. Here it is usually assumed that the shift between distributions is abrupt. Even if this is not the case, concentrating on the static phases instead of the change may be a good practice.

Dhodapkar and Smith [2003] and Nagpurka et al. [2006] put forward and experimentally examine some phase detection techniques. Nagpurka et al. [2006] test multiple locality phase detection algorithms on virtual machines. Thus, their findings of the usefulness are somewhat independent of the underlying environment.

In the area of CD, the instances are either labeled or unlabeled examples and we are interested in a couple of things. Task A includes, among others, tracking down outliers and mislabeled examples in the machine learning data. Also with the probability function we get a lot of information about the relevance of the instance. Relevance can be defined by frequency or simply the importance of right prediction for the example. A clear example of differing importances can be found in spam mail filtering: deleting non-spam mail has much more effect than leaving some spam [Delany et al., 2005].

The last Tasks C–E, on the other hand, include modifying the model to better cope with current input distribution. These tasks have interest in somewhat broader area on machine learning. For example, areas such like *transfer learning* [Pan and Yang, 2010; Taylor and Stone, 2009] and, to some extent, *meta-learning* [Smith-Miles, 2009; Vilalta and Drissi, 2002] are specifically interested in this. In the latter, we are usually interested in finding the best machine learning algorithm, instead of the best model, for the current batch.

### 6.3.2 More Previous Work on the Problem

In this section, we discuss some previously introduced ways to solve Problem 2 on page 71 in the mentioned frameworks.

There is some previous work on finding connections between the areas of machine learning and algorithms. At least Blum [1998] has applied the methods of machine learning to online problems. However, he only narrowly discusses drifting concepts.

Plenty of research has been done for all of the Tasks A–E separately. However, often only small part of the whole problem has been solved. Thus, many applicable results in some other areas may not be relevant to the whole problem solution because of different backgrounds. Especially the online environment is a limitation here.

### 6.3.2.1 The Tasks

Let us now go through tasks one by one.

The Task A has received a lot of research: Finding frequent item sets is a much studied problem [Agrawal and Srikant, 1994; Cheng et al., 2008; Goethals, 2002; Yahia et al., 2006], but in our setting it should be done in online fashion like, e.g., Cheng et al. [2008] do. A very natural approach is to use a window that is, a sequence of consequent instances. The window can be either a sliding one [Widmer and Kubat, 1996] or one that takes steps of constant length. The size of the window may be either static or dynamic. [Tsymbol, 2004] Also the use of multiple windows has been shown to give more information about the type of CD [Lazarescu et al., 2004]. The same approaches have been used for LR problems also [Angelopoulos et al., 2008; Denning, 1968; Nagpurka et al., 2006]. In addition, the discounting method mentioned in Section 5.3 is also available [Gramacy et al., 2003].

As mentioned before, we have a couple of alternatives to tracking down the current sequence properties. We can either select appropriate instances or weight them according to their age or importance to the current concept [e.g., Gramacy et al., 2003]. The selection can be done with either using a window [e.g., Widmer and Kubat, 1996] or more rarely more freely choosing a subset of previously occurred instances. For example, Klinkenberg [2004] proposed selecting those past batches that, roughly speaking, exhibit good prediction accuracy on the current model. This way we can use the previously available information about reoccurring concepts. In the information filtering domain it seems like both instance selection methods overtake the weighting methods in case of support vector machine learners in experiments Klinkenberg [2004]. However, the results may depend on the machine learning method and the data.

Task B concerns noticing the change. Klinkenberg and Renz [1998] divide the indicators of CD to three categories: those that are based on the performance measurements (e.g., accuracy) of the model, those that consider directly the properties (e.g., features like size, complexity) of the model, and those that are grounded only on the statistics of the input data. The two first ones rely on the fact that the model at the same time identifies the current concept. The classical, and not necessarily the optimal [Fung et al., 2004], way of measuring the performance in CD is simply the accuracy. This corresponds roughly to, e.g., counting the cache misses on LR [Gramacy et al., 2003]. The second one, directly handling the model properties, is a highly model type specific solution. For example, we can easily check the most important decision rules on rule ensembles mentioned in Chapter 4. In LR we can similarly track down the

---

size and identity information of the working set model [Dhodapkar and Smith, 2002].

Nonetheless, we have a lot of different and generalizable solutions based on the last possibility: the properties of the input data. One notable drawback in this approach is that in many solutions we are implicitly assuming that the instances  $x_{(i,\cdot)}$  are drawn i.i.d. from the distribution  $D_i$ . As Klinkenberg [2004] notes and we already mentioned in Section 6.3 this may not be wise. Nevertheless, the methods are clearly quite generalizable to both LR and CD. For instance, Song et al. [2007] present a method for using statistical tests to solve Task B. They are explicitly using the density estimation on this. On the other hand, Castillo et al. [2003] show that Statistical Quality Control (SQC) methods called P-Charts can also be used for CD detection. The main idea behind SQC is to monitor the stability of one or more quality characteristics in a production process. In case of a transition, the variance or average should be changing.

If we restrict Task A to finding the subset with density over some threshold value, we get more possibilities. We can now solve Task B by finding if the current batch does not seem to correspond to last noticed concept; that is, if the current concept candidate  $A$  is distinct from the previous real one  $W$ . For this we need a metric  $\delta(A, W) \sim P(A \approx W)$  for which a growing value indicates a change of concept. In previous sections, we already had some proposals for this by Dhodapkar and Smith [2002] and Widmer and Kubat [1996]. We will return to the subject in the next section.

Widmer and Kubat [1996] suggest that in Task C the right time to store the concept information is when the distribution has reached a stable state and we have enough information about it. Their proposal is to use a metric  $\delta$  to indicate that a concept has been tracked, that is, when  $\delta$  has continually low values.

Task D addresses the storage of the current static concept model. In addition, the task covers restoring the model when the concept is discovered as a reappearance. Clearly, the restoration should be done when we have noticed a new batch starting [Widmer and Kubat, 1996]. One more subtask in this case is to decide *if* restoring a previous model is useful at all. This is only the case if the situation, the concept, reoccurs in a very similar way. If there is even some distortion, it may be more beneficial to learn everything from scratch. The distance metric  $\delta$  may also be used for this.

Ali and Smith [2006] have conducted an interesting study on what kind of learning algorithm types are good for which kind of data. With this kind of information we could select the best algorithm type for current data if there is no previously trained model. This way meta-learning is also related to the problem.

In case of ensemble learning, Tasks D and E take somewhat different form. We are not restoring any previous models as themselves, but instead reweighting the models in the ensemble. There are plenty of examples of this [Kolter and Maloof, 2007; Wang et al., 2003]. Nevertheless, it could be claimed that static ensembles are themselves a way to cope with CD. Different base models in the ensemble could be used for different types of input distribution. However, Ko et al. [2008] show that dynamic ensemble weighting gives a better result than choosing a single ensemble to cope with the whole input.

### 6.3.2.2 A Couple of Metrics

As has been seen, one way to solve the Tasks B and D and even E is to use proper distance metric for concept candidates. Optimal metric would take into account the instances, their frequencies and their order. The last one may be very useful with spatial locality or any other similar patterns.

However, even the metrics based only on the instances might be enough. If we restrict the problem like this, we are very close to the problem of two sample test. In the two sample problem, we have we two instance sets and want to know if they are drawn from the same distribution. This was our motivation to solve part of the Task D, choosing the best previous model, with our MMDSEL algorithm in machine learning environment [V]. The key idea of our algorithm is to store a fingerprint of each model in a pool and decide the best model for restoring based on the fingerprints. In practice, we use samples of the training data instances. Thus, we can compare these samples with the current batch with suitable metrics like two sample tests.

Our algorithm used a two sample test metric called *maximum mean discrepancy* (MMD) by Smola et al. [2007] and Gretton et al. [2007a,b] which runs in quadratic time by default. However, with approximations we can compute MMDSEL in some common cases much faster, that is, roughly in linear time. In addition to the efficiency, there are a couple of further points in the approach: for example, the method represents an alternative measurement for selecting the best classifier.

Somewhat similarly Pan et al. [2008, 2009] show how MMD can be used for solving Task E. Namely they consider how regression models can be tuned if the

test distribution is slightly different from the training distribution. Other similar change detection methods have also been presented. For example, Harchaoui et al. [2009] present a technique based also on the so-called kernel functions. Hido et al. [2008] study a so called *change analysis* where also the type of change is tracked down.

Furthermore, Duesterwald et al. [2003] also study Task D on LR. They are representing metrics derived from hardware counters on microarchitectures. With the metrics, they are predicting future concepts. Naturally periodicity, reoccurring concepts, is assumed in this context. They found out that different metrics find the same periods and that their techniques really increase the efficiency. Similarly Shen et al. [2007] study the problem of predicting locality phases based on locality analysis and signal processing techniques. In addition, according to Ding and Zhong [2003] we can make somewhat more general assumptions on a program based on its behavior on some input.

## 6.4 Further Generalizations

In the definition of Problem 2 on page 71, we were interested only in the steady phases and assumed abrupt changes between them. The steady situation means that the detected concept has reached a static state. The change states could be treated as separate phases if needed.

However, if we want explicitly to model the *gradual* change we need *transition instances* between the batches. This kind of transition has been presented, e.g., by Fung et al. [2004]. Thus, next we present a problem statement with a gradual change of concepts.

**Problem Definition 3.** *Let  $D_1, D_2, D_3, \dots$  be a sequence of distributions. Let  $x_{(i,1)}, x_{(i,2)}, x_{(i,3)}, \dots, x_{(i,m_i)}$  be a sequence of instances  $x \in X$  drawn from the distribution  $D_i$ . The total input sequence is*

$$\begin{aligned} &x_{(1,1)}, x_{(1,2)}, x_{(1,3)}, \dots, x_{(1,m_1)}, \\ &y_{(1,1)}, y_{(1,2)}, y_{(1,3)}, \dots, y_{(1,l_1)}, \\ &x_{(2,1)}, x_{(2,2)}, x_{(2,3)}, \dots, x_{(2,m_2)}, \dots, \end{aligned}$$

where the  $y_{(i,j)} \in X$  are transition instances that are defined as follows. Let  $w_i$  be the density function of distribution  $D_i$ . The density function for  $y_{(i,j)}$  is defined by

$$w(y_{(i,j)}) = (1 - g(j))w_i(y_{(i,j)}) + g(j)w_{i+1}(y_{(i,j)}),$$

where the ending points of  $g : N \rightarrow [0, 1]$  are bound by  $g(1) \approx 0$  and  $g(l_j) \approx 1$ .

Most of the problem definition and the tasks are the same as in the simpler one. Redundancy is omitted. Roughly speaking, in the transition phase the effect of distribution  $D_i$  gradually descends while the one of  $D_{i+1}$  increases. Often the function is simplified to  $g(j) = j/l_j$  where  $l_j$  is the length of the examined batch [e.g., Fung et al., 2004]. A more general simplification would be to assume  $g$  to be monotonically increasing. This could give us some more power to use in the analysis.

In addition to gradual change, we have plenty of options for the problem definition. For example, we can assume that between the static phases we have just uniform noise from which no concepts can be found. Also the noise can be present during the static phases, which should be noticed. In case of CD, this noise could be, e.g., mislabeled instances. Like we noticed in Problem 2 on page 71, also normal variance can be seen as noise. For LR, in addition to the variance, occasional nonrecurring accesses should maybe be left outside the model.

## 6.5 On Locality of Reference and Drifting Concepts

As discussed in this chapter, important analogy exists on two virtually separate areas of theoretical computer science: namely drifting concepts [Tsymbal, 2004] of machine learning and locality of reference [Denning, 2005] in algorithm analysis especially on lower level programming. In addition, nearly identical problems have been solved in such areas like transfer learning [Pan and Yang, 2010], meta-learning [Vilalta and Drissi, 2002], and change analysis [Hido et al., 2008]. Somehow the similarity should not come as a big surprise: the basic common factor in most cases is dynamically changing input and tracking it down. This, however, is something that is natural to find in multiple areas.

Nevertheless, the subjects are quite distant in many ways: In machine learning we often consider abstract methods that are distinguished from the underlying hardware and low level solutions. On the other hand study of LR is usually present exactly on the low level algorithm analysis even if it is also often platform independent. Nevertheless, common work has not, to our knowledge, been introduced save the mentioned survey by Blum [1998].

What is our motivation for generating such a united framework that includes all the problems? An example of benefits for this kind of knowledge is our algorithm MMDSEL [V] for selecting machine learning models. The basic idea

of using model fingerprints came from solutions like the one by Dhodapkar and Smith [2002]. They present a method for identifying the current concept, the working set, by comparing their hash function values. Thus, even if the solutions in one framework were not directly applicable to another one, the philosophy behind them could be. In addition, it could be assumed that the more abstract machine learning methods could in some cases directly be applicable to the locality of reference problems.

All in all, in this chapter we build some foundation for the bridge between these various areas of study by proposing a shared problem definition. Even if the research on some problems, namely in locality of reference, is not very active or at least changing its shape, we hope that this knowledge of similarities would result in some innovative solutions.



# Chapter 7

## Conclusions

In this thesis, we have studied various methods that cope with dynamically changing input. In machine learning, imperfect information about the world is present by definition; that is exactly the reason why learning is needed. However, in addition we addressed unknown changes in algorithmic analysis.

At first we introduced some background knowledge about general optimization problems. These are often used in different machine learning methods. We also discussed the regularization of optimization, which plays an important role in the quality of the optimization result. A method [I] for efficiently solving lasso type regularized support vector machines was introduced.

Then we introduced how suitable weights for a multi-target rule ensemble can be found with gradient directed optimization [II, III]. In the multi-target framework, we are predicting multiple targets simultaneously instead of a single one. With this strategy, we might be able to get both smaller and more accurate machine learning models. We presented both a theoretical and an experimental evaluation of this FIRE method and gave guidelines for its parameter values.

There are some interesting ways to further develop the ensemble method. First of all, we already know ways to increase the robustness of the method by using the solutions introduced by RULEFIT. In addition, especially on multi-target prediction it would be very interesting to try out how random subspace selection on target attributes increases the diversity of base models. That is, in addition to selecting the descriptive attributes during the induction, as random forests do, we could do the same for target attributes. This solution is not limited to rule ensemble methods; it could first be tried to multi-target random forests.

After this, we moved to somewhat different area of locality of reference and binary search trees. We introduced a method, WSPLAY, that aims to adapt itself wisely to the properties of the input [IV]. As a basis, it uses the adaptive binary search tree method of splay trees. The main modification is that the amount of costly rotations is decreased if they do not seem to benefit us.

In the last part of the thesis, we examined two distinct areas of computer science in a survey like manner and tracked down similarities between them. In fact, we presented a general framework that contains the essential parts of problems called concept drift and locality of reference. Our hope is that the brief examination of varying solutions would bring more knowledge to the researchers in both the fields and end up with fruitful new ideas.

On the whole, in this thesis we introduced and deeply examined a method for accurate but still interpretable prediction of multi-target data. In addition, we gave some solutions to the question of how dynamic online input can be coped with.

# Bibliography

- Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 487–499, 1994. Morgan Kaufmann, San Francisco, CA.
- Susanne Albers. Online algorithms. In Dina Goldin, Scott A. Smolka, and Peter Wegner, editors, *Interactive Computation: The New Paradigm*, pages 143–164. Springer, Berlin/Heidelberg, Germany, 2006.
- Susanne Albers and Marek Karpinski. Randomized splay trees: Theoretical and experimental results. *Information Processing Letters*, 81(4):213–221, 2002.
- Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2):145–175, 2005.
- Shawkat Ali and Kate A. Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.
- Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. List update with locality of reference. In Eduardo Laber, Claudson Bornstein, Loana Nogueira, and Luerbio Faria, editors, *Proceedings of the Eighth Latin American Theoretical Informatics Symposium (LATIN 2008)*, volume 4957 of *LNCS*, pages 399–410, 2008. Springer, Berlin/Heidelberg, Germany.
- Annalisa Appice and Sašo Džeroski. Stepwise induction of multi-target model trees. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, volume 4701 of *LNCS*, pages 502–509, 2007. Springer, Berlin/Heidelberg, Germany.

- Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.
- Ghada Hany Badr and B. John Oommen. Self-adjusting of ternary search tries using conditional rotations and randomized heuristics. *The Computer Journal*, 48(2):200–219, 2005.
- Rudolf Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, 1972.
- Jim Bell and Gopal Gupta. An evaluation of self-adjusting binary search tree techniques. *Software: Practice and Experience*, 23(4):369–382, 1993.
- P. Berkhin. A survey of clustering data mining techniques. In Jacob Kogan, Charles Nicholas, and Marc Teboulle, editors, *Grouping Multidimensional Data*, pages 25–71. Springer, Berlin/Heidelberg, Germany, 2006.
- Steffen Bickel, Jasmina Bogojeska, Thomas Lengauer, and Tobias Scheffer. Multi-task learning for HIV therapy screening. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 56–63, 2008. ACM, New York, NY.
- Hendrik Blockeel. *Top-down Induction of First Order Logical Decision Trees*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, 1998.
- Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
- Hendrik Blockeel and Jan Struyf. Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research*, 3:621–650, 2002.
- Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. In Jude W. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, pages 55–63, 1998. Morgan Kaufmann, San Francisco, CA.
- Avrim Blum. On-line algorithms in machine learning. In Amos Fiat and Gerhard Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*, Chapter 14. Springer, 1998.

- 
- Prosenjit Bose, Karim Douieb, Vida Dujmović, and John Howat. Layered working-set trees. In Alejandro López-Ortiz, editor, *Proceedings of the Tenth Latin American Theoretical Informatics Symposium (LATIN 2010)*, volume 6034 of *LNCS*, pages 686–696, 2010. Springer, Berlin/Heidelberg, Germany.
- Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- Mihai Bădoiu, Richard Cole, Erik D. Demaine, and John Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theoretical Computer Science*, 382(2):86–96, 2007.
- Lucian Buşoniu, Robert Babuška, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008.
- Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.
- Gladys Castillo, João Gama, and Pedro Medas. Adaptation to drifting concepts. In Fernando Moura-Pires and Salvador Abreu, editors, *Proceedings of the Tenth Portuguese Conference on Artificial Intelligence (EPIA 2003)*, volume 2902 of *LNCS*, pages 279–293. Springer, Berlin/Heidelberg, Germany, 2003.
- Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.
- Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2010.
- Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Boosted multi-task learning. *Machine Learning*, 85(1):149–173, 2011.
- Robert P. Cheetham, B. John Oommen, and David T.H. Ng. Adaptive structuring of binary search trees using conditional rotations. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):695–704, 1993.
- James Cheng, Yiping Ke, and Wilfred Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems*, 16(1):1–27, 2008.

- Richard Cole. On the dynamic finger conjecture for splay trees. Part II: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.
- Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting log n-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, third edition, 2009.
- Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- Sarah Jane Delany, Pádraig Cunningham, Alexey Tsymbal, and Lorcan Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4–5):187–195, 2005.
- Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrăşcu. Dynamic optimality—almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.
- Krzysztof Dembczyński, Wojciech Kotłowski, and Roman Słowiński. Solving regression by learning an ensemble of decision rules. In Leszek Rutkowski, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *Proceedings of the Ninth International Conference on Artificial Intelligence and Soft Computing (ICAISC 2008)*, volume 5097 of *LNCS*, pages 533–544, 2008a. Springer, Berlin/Heidelberg, Germany.
- Krzysztof Dembczyński, Wojciech Kotłowski, and Roman Słowiński. Maximum likelihood rule ensembles. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 224–231, 2008b. ACM, New York, NY.
- Damjan Demšar, Marko Debeljak, Claire Lavigne, and Sašo Džeroski. Modelling pollen dispersal of genetically modified oilseed rape within the field. In *Abstracts of the 90th ESA Annual Meeting*, page 152, 2005. The Ecological Society of America, Montreal, Canada.
- Damjan Demšar, Sašo Džeroski, Thomas Larsen, Jan Struyf, Jørgen Axelsen, Marianne Bruus Pedersen, and Paul Henning Krogh. Using multi-objective classification to model communities of soil microarthropods. *Ecological Modelling*, 191(1):131–143, 2006.

- 
- Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- Peter J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.
- Peter J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, 6(1):64–84, 1980.
- Peter J. Denning. The locality principle. *Communications of the ACM*, 48(7):19–24, 2005.
- Jonathan C. Derryberry and Daniel D. Sleator. Skip-splay: Toward achieving the unified bound in the BST model. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Proceedings of the 11th International Symposium on Algorithms and Data Structures (WADS 2009)*, volume 5664 of *LNCS*, pages 194–205, 2009. Springer, Berlin/Heidelberg, Germany.
- Ashutosh S. Dhodapkar and James E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA 2002)*, pages 233–244, 2002. IEEE Computer Society, Los Alamitos, CA.
- Ashutosh S. Dhodapkar and James E. Smith. Comparing program phase detection techniques. In *Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO 2003)*, pages 217–227, 2003. IEEE Computer Society, Los Alamitos, CA.
- Thomas Dietterich. Ensemble methods in machine learning. In Josef Kittler and Fabio Roli, editors, *Proceedings of the First International Workshop on Multiple Classifier Systems (MCS 2000)*, volume 1857 of *LNCS*, pages 1–15, 2000. Springer, Berlin/Heidelberg, Germany.
- Chen Ding and Yutao Zhong. Predicting whole-program locality through reuse distance analysis. *ACM SIGPLAN Notices*, 38(5):245–257, 2003.
- David L. Donoho and Jain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.
- Reza Dorrigiv and Alejandro López-Ortiz. On certain new models for paging with locality of reference. In Shin ichi Nakano and Md. Saidur Rahman,

- editors, *Proceedings of the Second International Workshop on Algorithms and Computation (WALCOM 2008)*, volume 4921 of *LNCS*, pages 200–209, 2008. Springer, Berlin/Heidelberg, Germany.
- Evelyn Duesterwald, Călin Cașcaval, and Sandhya Dwarkadas. Characterizing and predicting program behavior and its variability. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT 2003)*, pages 220–231, 2003. IEEE Computer Society, Los Alamitos, CA.
- Sašo Džeroski, Ljupčo Todorovski, and Hendrik Blockeel. Relational ranking with predictive clustering trees. In *Proceedings of the Workshop on Active Mining (in ICDM 2002)*, pages 9–15, 2002. IEEE Computer Society, Los Alamitos, CA.
- Sašo Džeroski, Andrej Kobler, Valentin Gjorgjioski, and Panče Panov. Using decision trees to predict forest stand height and canopy cover from LANSAT and LIDAR data. In Klaus Tochtermann and Arno Scharl, editors, *Proceedings of the 20th International Conference on Informatics for Environmental Protection (EnviroInfo 2006)*, pages 125–133, 2006. Shaker, Aachen, Germany.
- Sašo Džeroski, Damjan Demšar, and Jasna Grbović. Predicting chemical parameters of river water quality from bioindicator data. *Applied Intelligence*, 13(1):7–17, 2000.
- Sašo Džeroski, Nathalie Colbach, and Antoine Messéan. Analysing the effect of field character on gene flow between oilseed rape varieties and volunteers with regression trees. In Antoine Messéan, editor, *Proceedings of the Second International Conference on Co-existence between GM and non-GM based Agricultural Supply Chains*, pages 207–211, 2005. Agropolis Productions, Montpellier, France.
- Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.
- André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS 2001)*, pages 681–687, 2001. MIT Press, Cambridge, MA.
- Peter Flach and Nada Lavrač. Rule induction. In Michael R. Berthold and David J. Hand, editors, *Intelligent Data Analysis*, pages 229–267. Springer, Berlin/Heidelberg, Germany, 2003. Second edition.

- 
- Steven A. Frank and Arthur Asuncion. UCI machine learning repository, 2011. URL <http://archive.ics.uci.edu/ml/>.
- Jerome H. Friedman and Bogdan E. Popescu. Importance sampled learning ensembles. Technical report, Stanford University, Stanford, CA, 2003.
- Jerome H. Friedman and Bogdan E. Popescu. Gradient directed regularization for linear regression and classification. Technical report, Stanford University, Stanford, CA, 2004.
- Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. Technical report, Stanford University, Stanford, CA, 2005.
- Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008.
- Gabriel Pui Cheong Fung, Jeffrey Xu Yu, and Hongjun Lu. Classifying text streams in the presence of concept drifts. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2004)*, volume 3056 of *LNCS*, pages 373–383. Springer, Berlin/Heidelberg, Germany, 2004.
- Martin Fürer. Randomized splay trees. In *Proceedings of the Tenth Annual Symposium on Discrete Algorithms (SODA 1999)*, pages 903–904, 1999. SIAM, Philadelphia, PA.
- Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153, 2008.
- Jing Gao, Wei Fan, and Jiawei Han. On appropriate assumptions to mine data streams: Analysis and practice. In *Proceedings of the Seventh International Conference on Data Mining (ICDM 2007)*, pages 143–152, 2007. IEEE Computer Society, Washington, DC.
- Valentin Gjorgjioski, Sašo Džeroski, and Matt White. Clustering analysis of vegetation data. Technical Report 10065, Jožef Stefan Institute, Ljubljana, Slovenia, 2008.
- Bart Goethals. Survey on frequent pattern mining. Technical report, HIIT Basic Research Unit, University of Helsinki, Helsinki, Finland, 2002.

- Robert B. Gramacy, Manfred K. Warmuth, Scott A. Brandt, and Ismail Ari. Adaptive caching by refetching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS 2002)*, pages 1465–1472, 2003. MIT Press, Cambridge, MA.
- Yves Grandvalet and Stéphane Canu. Outcomes of the equivalence of adaptive ridge with least absolute shrinkage. In Michael S. Kearns, Sara A. Solla, and David A. Cohn, editors, *Advances in Neural Information Processing Systems (NIPS 1998)*, pages 445–451, 1999. Morgan Kaufmann, San Francisco, CA.
- Arthur Gretton, Karsten M. Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel method for the two-sample-problem. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS 2006)*, pages 513–520, 2007a. MIT Press, Cambridge, MA.
- Arthur Gretton, Karsten M. Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel approach to comparing distributions. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 1637–1641, 2007b. AAAI Press, Menlo Park, CA.
- Zaid Harchaoui, Francis Bach, and Eric Moulines. Kernel change-point analysis. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems (NIPS 2008)*, pages 609–616, 2009. MIT Press, Cambridge, MA.
- Steffen Heinz and Justin Zobel. Performance of data structures for small sets of strings. *Australian Computer Science Communications*, 24(1):87–94, 2002.
- Shohei Hido, Tsuyoshi Idé, Hisashi Kashima, Harunobu Kubo, and Hirofumi Matsuzawa. Unsupervised change analysis using supervised learning. In Takashi Washio, Einoshin Suzuki, Kai Ting, and Akihiro Inokuchi, editors, *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008)*, volume 5012 of *LNCIS*, pages 148–159. Springer, Berlin/Heidelberg, Germany, 2008.
- Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In

- 
- Andrew McCallum and Sam Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 408–415, 2008. ACM, New York, NY.
- Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.
- A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4–5):411–430, 2000.
- John Iacono. Alternatives to splay trees with  $O(\log n)$  worst-case access times. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA 2001)*, pages 516–522, 2001a. SIAM, Philadelphia, PA.
- John Iacono. *Distribution-sensitive data structures*. PhD thesis, Rutgers, The State University of New Jersey, New Brunswick, New Jersey, 2001b.
- Nitin Indurkha and Sholom M. Weiss. Solving regression problems with rule-based ensemble classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001)*, pages 287–292, 2001. ACM, New York, NY.
- Florian Jarre and Stephen Vavasis. Convex optimization. In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook, second edition, Volume 1: General Concepts and Techniques*, Chapman & Hall/CRC Applied Algorithms and Data Structures series, chapter 32. Chapman and Hall/CRC, 2010.
- Minwoo Jeong and Gary Geunbae Lee. Multi-domain spoken language understanding with transfer learning. *Speech Communication*, 51(5):412–424, 2009.
- Christian Kampichler, Sašo Džeroski, and Ralf Wieland. Application of machine learning techniques to the analysis of soil ecological data bases: relationships between habitat features and collembolan community characteristics. *Soil Biology and Biochemistry*, 32(2):197–209, 2000.
- Aram Karalič. Employing linear regression in regression tree leaves. In B. Neumann, editor, *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI 1992)*, pages 440–441, 1992. John Wiley & Sons, New York, NY.
- Aram Karalič and Ivan Bratko. First order regression. *Machine Learning*, 26(2-3):147–176, 1997.

- Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.
- Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. In *Notes of the Workshop on Learning for Text Categorization (in ICML/AAAI-98)*, pages 33–40, 1998. AAAI Press, Menlo Park, CA.
- Albert H.R. Ko, Robert Sabourin, and Alceu Souza Jr. Britto. From dynamic classifier selection to dynamic ensemble selection. *Pattern Recognition*, 41(5):1718–1731, 2008.
- Dragi Kocev, Celine Vens, Jan Struyf, and Sašo Džeroski. Ensembles of multi-objective decision trees. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, volume 4701 of *LNCS*, pages 624–631, 2007. Springer, Berlin/Heidelberg, Germany.
- J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, 2007.
- Tony W. Lai and Derick Wood. Adaptive heuristics for binary search trees and constant linkage cost. In *Proceedings of the Second Annual Symposium on Discrete Algorithms (SODA 1991)*, pages 72–77, 1991. SIAM, Philadelphia, PA.
- Mihai M. Lazarescu, Svetha Venkatesh, and Hung H. Bui. Using multiple windows to track concept drift. *Intelligent Data Analysis*, 8(1):29–59, 2004.
- Eric K. Lee and Charles U. Martel. When to use splay trees. *Software: Practice and Experience*, 37(15):1559–1575, 2007.
- Hsuan-Tien Lin, Chih-Jen Lin, and Ruby Weng. A note on Platt’s probabilistic outputs for support vector machines. *Machine Learning*, 68(3):267–276, 2007.
- Qi Liu, Qian Xu, Vincent W. Zheng, Hong Xue, Zhiwei Cao, and Qiang Yang. Multi-task learning for cross-platform siRNA efficacy prediction: an in-silico study. *BMC Bioinformatics*, 11(1):181–196, 2010.

- 
- Karim Lounici, Massimiliano Pontil, Alexandre B. Tsybakov, and Sara A. van de Geer. Taking advantage of sparsity in multi-task learning. In *Proceedings of the 22nd Conference on Learning Theory (COLT 2009)*, 2009.
- Hamed Masnadi-Shirazi and Nuno Vasconcelos. Risk minimization, probability elicitation, and cost-sensitive SVMs. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 759–766, 2010. Omnipress, Madison, WI.
- Ryszard S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing (FCIP 1969)*, volume A3, Switching Circuits, pages 125–128, 1969. Bled, Yugoslavia.
- S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.
- Priya Nagpurka, Michael Hind, Chandra Krintz, Peter F. Sweeney, and V.T. Rajan. Online phase detection algorithms. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO 2006)*, pages 111–123, 2006. IEEE Computer Society, Los Alamitos, CA.
- Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.
- Sinno Jialin Pan, James T. Kwok, and Qiang Yang. Transfer learning via dimensionality reduction. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 677–682, 2008. AAAI Press, Menlo Park, CA.
- Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1187–1192, 2009. Morgan Kaufmann, San Francisco, CA.
- Ben Pfaff. Performance analysis of BSTs in system software. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):410–411, 2004.
- Ross J. Quinlan. Learning with continuous classes. In A. Adams and L. Sterling, editors, *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence (AI 1992)*, pages 343–348, 1992. World Scientific, Singapore.

- Alain Rakotomamonjy, Rémi Flamary, Gilles Gasso, and Stéphane Canu.  $\ell_p - \ell_q$  penalty for sparse linear and sparse multiple kernel multitask learning. *IEEE Transactions on Neural Networks*, 22(8):1307–1320, 2011.
- Mark Schmidt, Glenn Fung, and Rómer Rosales. Fast optimization methods for L1 regularization: A comparative study and two new approaches. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, volume 4701 of *LNCS*, pages 286–297, 2007. Springer, Berlin/Heidelberg, Germany.
- Martin Scholz and Ralf Klinkenberg. Boosting classifiers for drifting concepts. *Intelligent Data Analysis, Special Issue on Knowledge Discovery from Data Streams*, 11(1):3–28, 2007.
- Raquel Sebastião and João Gama. Change detection in learning histograms from data streams. In José Neves, Manuel Filipe Santos, and José Machado, editors, *Proceedings of the 14th Portuguese Conference on Artificial Intelligence (EPIA 2007)*, volume 4874 of *LNCS*, pages 112–123. Springer, 2007.
- Raimund Seidel and Cecilia Aragon. Randomized search trees. *Algorithmica*, 16(4):464–497, 1996.
- Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, pages 807–814, 2007. ACM, New York, NY.
- John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- Xipeng Shen, Yutao Zhong, and Chen Ding. Predicting locality phases for dynamic memory optimization. *Journal of Parallel and Distributed Computing*, 67(7):783–796, 2007.
- Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.
- Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25, 2009.

- 
- Alexander J. Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A Hilbert space embedding for distributions. In Marcus Hutter, Rocco A. Servedio, and Eiji Takimoto, editors, *Proceedings of the 18th International Conference on Algorithmic Learning Theory (ALT 2007)*, volume 4754 of *LNCS*, pages 13–31, 2007. Springer, Berlin/Heidelberg, Germany.
- Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2007)*, pages 667–676, 2007. ACM, New York, NY.
- Daniela Stojanova. Estimating forest properties from remotely sensed data by using machine learning. Master’s thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, 2009.
- Jan Struyf and Sašo Džeroski. Constraint based induction of multi-objective regression trees. In F. Bonchi and J. Boulicaut, editors, *Proceedings of the Fourth International Workshop on Knowledge Discovery in Inductive Databases (KDID 2005)*, volume 3933 of *LNCS*, pages 222–233, 2006. Springer, Berlin/Heidelberg, Germany.
- Ashok Subramanian. An explanation of splaying. *Journal of Algorithms*, 20(3): 512–525, 1996.
- Einoshin Suzuki, Masafumi Gotoh, and Yuta Choki. Bloomy decision tree for multi-objective classification. In Luc De Raedt and Arno Siebes, editors, *Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001)*, volume 2168 of *LNCS*, pages 436–447, 2001. Springer, Berlin/Heidelberg, Germany.
- Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.
- Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.
- Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, Department of Computer Science, Trinity College Dublin, Ireland, 2004.

- Vladimir N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.
- Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, Berlin/Heidelberg, Germany, second edition, 2000.
- Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.
- Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In Lise Getoor, Ted E. Senator, Pedro Domingos, and Christos Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*, pages 226–235, 2003. ACM, New York, NY.
- Yong Wang and Ian H. Witten. Inducing model trees for continuous classes. In Maarten van Someren and Gerhard Widmer, editors, *Poster Papers of the Ninth European Conference on Machine Learning (ECML 1997)*, pages 128–137, 1997. University of Economics, Faculty of Informatics and Statistics, Prague, Czech Republic.
- Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.
- Hugh E. Williams, Justin Zobel, and Steffen Heinz. Self-adjusting trees in practice for large text collections. *Software: Practice and Experience*, 31(10):925–939, 2001.
- S. Wold, A. Ruhe, H. Wold, and W. J. Dunn III. The collinearity problem in linear regression. the partial least squares (PLS) approach to generalized inverses. *SIAM Journal on Scientific and Statistical Computing*, 5(3):735–743, 1984.
- Rui Xu and Donald Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.
- S. Ben Yahia, T. Hamrouni, and E. Mephu Nguifo. Frequent closed itemset based algorithms: a thorough structural and analytical survey. *ACM SIGKDD Explorations Newsletter*, 8(1):93–104, 2006.
- Bernard Ženko. *Learning predictive clustering rules*. PhD thesis, University of Ljubljana, Faculty of computer and information science, Ljubljana, Slovenia, 2007.

- 
- Bernard Ženko and Sašo Džeroski. Learning classification rules for multiple target attributes. In Takashi Washio, Einoshin Suzuki, Kai Ming Ting, and Akihiro Inokuchi, editors, *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008)*, volume 5012 of *LNCS*, pages 454–465, 2008. Springer, Berlin/Heidelberg, Germany.
- Bernard Ženko, Sašo Džeroski, and Jan Struyf. Learning predictive clustering rules. In Francesco Bonchi and Jean-François Boulicaut, editors, *Revised selected and invited papers of the Fourth International Workshop on Knowledge Discovery in Inductive Databases (KDID 2005)*, volume 3933 of *LNCS*, pages 110–121, 2006. Springer, Berlin/Heidelberg, Germany.
- Ji Zhu, Saharon Rosset, Trevor Hastie, and Rob Tibshirani. 1-norm support vector machines. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS 2003)*, 2004. MIT Press, Cambridge, MA.

δευτε προς με παντες οι κοπιωντες και  
πεφορτισμενοι καγω αναπαυσω υμας