Efficient handling and analysis of experimental measurements is an essential part of research and development in a multitude of disciplines (e.g., engineering, chemistry, biology), since these contain information about the underlying processes. Researchers investigate processes by running experiments and gathering potentially a huge amount of data which is then to be evaluated. For environmental monitoring wireless sensor networks are used to collect data at spatially and temporally discrete positions. In mechanical engineering and related areas, potentially complex test-benches are set up and observations are recorded. Besides an efficient and effective way of exploring multiple results, researchers strive to discover correlations within the measured data. Moreover, model-based prediction of expected measurements can be highly beneficial for designing further experiments.

Typically, analytical functions or distributions are used to model the experimental data. Such models can offer a compact and intuitive representation of the underlying processes. Hence, predictions can be made at operating points for which no measurements were provided. One class of simple yet powerful functions suitable for such models are (piecewise) linear regression functions, which are often used in scientific databases for representing the data and performing prediction queries.

This thesis covers techniques for identifying piecewise linear models by building regression trees. New algorithmic solutions for building more compact and in the same time accurate models are developed and evaluated. Finally, with such models available in scientific databases, novel solutions are introduced, which enable a wide range of reverse engineered model-based predictions.
Learning Comprehensible Models for Analysis and Predictions in Scientific Databases

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades einer Doktorin der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatikerin
Anca M. Zimmer (geb. Ivanescu)
aus Brasov, Rumänien

Berichter: Universitätsprofessor Dr. rer. nat. Thomas Seidl
Universitätsprofessor Dr. Philippe Lenca

Tag der mündlichen Prüfung: 3.12.2013

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.
Anca Zimmer:

Learning Comprehensible Models for Analysis and Predictions in Scientific Databases

1. Auflage, 2014

Ergebnisse aus der Informatik
Band 2

Gedruckt auf holz- und säurefreiem Papier, 100% chlorfrei gebleicht.

Apprimus Verlag, Aachen, 2014
Wissenschaftsverlag des Instituts für Industriekommunikation und Fachmedien an der RWTH Aachen
Steinbachstr. 25, 52074 Aachen
Internet: www.apprimus-verlag.de, E-Mail: info@apprimus-verlag.de

Printed in Germany

ISBN 978-3-86359-205-9

D 82 (Diss. RWTH Aachen University, 2014)
To my family who supported me with love and encouragement.
## Contents

Abstract / Zusammenfassung 1

I  Introduction 5

1  The Need for Prediction Models 7
1.1  Use Cases ........................................ 12
1.2  Thesis Structure ................................... 16

2  Preliminaries 19
2.1  Regression Analysis ............................... 19
2.2  Partial Least Squares ............................... 23
2.3  Principal Hessian Directions ....................... 28

II  Model Trees 33

3  Regression Trees 35
3.1  Introduction ........................................ 36
3.2  Regression Trees with Axis-Parallel Splits ....... 39
3.3  Regression Trees with Oblique Splits ............. 43

4  Split Strategies using Clustering 53
4.1  Correlation Clustering ............................ 54
4.2  Regression Clustering ............................. 59
4.3  Model Clustering .................................. 61
4.4  Experiments ....................................... 65
4.5  Conclusion ....................................... 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Adaptive Model Tree on Data Streams</td>
<td>71</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>71</td>
</tr>
<tr>
<td>5.2</td>
<td>Adaptive Model Tree</td>
<td>77</td>
</tr>
<tr>
<td>5.3</td>
<td>Incremental Updates</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>Experiments</td>
<td>86</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusion</td>
<td>96</td>
</tr>
<tr>
<td>III</td>
<td>Hinging Hyperplane Models</td>
<td>97</td>
</tr>
<tr>
<td>6</td>
<td>Hinging Hyperplane Models</td>
<td>99</td>
</tr>
<tr>
<td>6.1</td>
<td>Hinge Functions</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>Hinge Finding Algorithm</td>
<td>103</td>
</tr>
<tr>
<td>6.3</td>
<td>Convergence with Line Search</td>
<td>105</td>
</tr>
<tr>
<td>6.4</td>
<td>Applications</td>
<td>108</td>
</tr>
<tr>
<td>7</td>
<td>Multivariate Hinging Hyperplane Models</td>
<td>111</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>112</td>
</tr>
<tr>
<td>7.2</td>
<td>Hinge Regression for Multiple Outputs</td>
<td>114</td>
</tr>
<tr>
<td>7.3</td>
<td>Finding the Consensus Separator (\hat{\Gamma})</td>
<td>116</td>
</tr>
<tr>
<td>7.4</td>
<td>Hinge Finding Algorithm for Multiple Outputs</td>
<td>121</td>
</tr>
<tr>
<td>7.5</td>
<td>Experiments</td>
<td>124</td>
</tr>
<tr>
<td>7.6</td>
<td>Conclusion</td>
<td>129</td>
</tr>
<tr>
<td>8</td>
<td>Efficient Initialization of the Hinge Finding Algorithm</td>
<td>131</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>131</td>
</tr>
<tr>
<td>8.2</td>
<td>Efficient Initialization of a Single Hinge</td>
<td>132</td>
</tr>
<tr>
<td>8.3</td>
<td>Fitting Several Hinges</td>
<td>134</td>
</tr>
<tr>
<td>8.4</td>
<td>Experiments</td>
<td>135</td>
</tr>
<tr>
<td>8.5</td>
<td>Conclusion</td>
<td>140</td>
</tr>
</tbody>
</table>
Abstract

Efficient handling and analysis of experimental measurements is an essential part of research and development in a multitude of disciplines (e.g., engineering, chemistry, biology), since these contain information about the underlying processes. Researchers investigate processes by running experiments and gathering potentially a huge amount of data which is then to be evaluated. For environmental monitoring wireless sensor networks are used to collect data at spatially and temporally discrete positions. In mechanical engineering and related areas, potentially complex test-benches are set up and observations are recorded. Besides an efficient and effective way of exploring multiple results, researchers strive to discover correlations within the measured data. Moreover, model-based prediction of expected measurements can be highly beneficial for designing further experiments.

Typically, analytical functions or distributions are used to model the experimental data. Such models can offer a compact and intuitive representation of the underlying processes. Hence, predictions can be made at operating points for which no measurements were provided. One class of simple yet powerful functions suitable for such models are (piecewise) linear regression functions, which are often used in scientific databases for representing the data and performing prediction queries.

This thesis covers techniques for identifying piecewise linear models by building regression trees. New algorithmic solutions for building more compact and in the same time accurate models are developed and evaluated. Finally, with such models available in scientific databases, novel solutions are introduced, which enable a wide range of reverse engineered model-based predictions.
Zusammenfassung


Typischerweise werden analytische Funktionen verwendet, um die experimentellen Daten zu modellieren. Solche Modelle bieten eine kompakte und intuitive Darstellung der zugrunde liegenden Prozessen. Daher können Vorhersagen für Betriebspunkte gemacht werden, für die keine Messungen bereitgestellt wurden. Eine Klasse solcher Modelle sind (stückweise) lineare Funktionen, die üblicherweise in wissenschaftlichen Datenbanken eingesetzt werden um die Daten darzustellen und Vorhersageanfragen zu beantworten.

Part I

Introduction
Chapter 1

The Need for Prediction Models

With the vast amount of data becoming available in the past years and the cheap access to computational power and data storage, the need for efficiently exploring and understanding the data is continuously growing. Data mining is concerned with discovering patterns and extracting knowledge from data, and emerged as a border area between data bases, machine learning, and statistics. An important data mining task is discovering frequent patterns, associations, and causal structures among sets of items in the transaction databases or data repositories. Another focus in data mining research is clustering, where the data is grouped such that objects in the same cluster are more similar to each other than objects from different clusters. Clusterings can provide a summarized view of the data, and can help discovering patterns and trends. A further research topic in data mining is data modeling for predictive analysis, which goes beyond analyzing trends to predicting their behavior. To understand the relationship between variables and to predict trends, a variety of statistical and analytical techniques are used to develop generalized models. A distinction is made between predictor variables and response variables, where the focus lies on describing the response variables based on the explaining predictors. While classification deals with class labels as response variables, prediction refers to continuous valued response variables. This thesis focuses on discussing procedures that can be seen as a form of regression analysis, concentrating on approaches
which describe the relationship between predictor and response variables.

Predictive analytics is popular in many application fields. In health care experts are using it to determine high risk patients, or modeling the diagnostic accuracy. Modeling customer purchasing patterns or forecasting stock market trends are prominent examples from the marketing and financial area. Another example is meteorology, where the interest lies in the weather forecast. The key aspect in these applications is to model a conditional distribution of the response variable, given a set of predictors.

Another broad area of application for predictive analytics is in scientific research. Scientists from various disciplines study processes by running several experimental measurements, and then describe these processes by drawing conclusions from these data. Analytic techniques help uncovering patterns and trends within scientific databases. The outcome may be used for understanding the underlying processes, for drawing conclusions about existing causalities, or for making decisions about further experiments. Predictive models in science are rigorous, delivering quantitative statements about what would happen under certain conditions. Predictor variables which can be controlled by the user and which turned out to have an impact on the response variable are used in a mathematical model to predict the response within the model’s boundaries. The focus in these applications is to identify causal relationships within the processes, to explain how they affect the response when the predictor has a certain value. Handling experimental measurements is an essential part of research and development in a multitude of disciplines. In the following some examples are mentioned.

- **Molecular biology**: The nowadays available experimental techniques allow the sequencing of the entire genome of organisms, the measurement of gene expressions and the dynamics of their regulatory interactions. The challenge in contemporary systems biology is to identify biological meaningful models of the regulatory interactions between the gene expressions, helping scientists to discover interactions in the gene regulatory network, to understand the cell functioning, and to design numerical simulations of biological systems (see [CFLM06,
PDdJFT08, FTS12]). This provides hypotheses, which can be then validated through laboratory experiments.

- **Chemistry:** The Structure-Activity Relationship (SAR) is a paradigm that assumes that the biological activity of molecules is dictated by their chemical structure. Quantitative SAR computes predictive models with chemical structures as predictors and the biological activity as responses. These models are of great interest in the pharmaceutical industry for drug discovery and toxicology, since they enable the prediction of activities of untested chemicals (see [NINAP10]), as well as in environmental science, where these models are employed for risk assessment (see [PFTW03, TRLL09] for details).

- **Sensor Networks:** Spatially distributed wireless sensors are often used to monitor physical phenomena, e.g., ecological habitat monitoring, water distribution, industrial sensing. Sensors are small devices with a low performance processing unit, a short range transmitter, and a short life battery, which collect data and send it forward to other special nodes with robust disk storage. An efficient way to reduce the communication costs without compromising data quality is to fit a prediction model through the collected data of a sensor and communicate this further, instead of transmitting all collected data (see [MBM10, LBSB07, DGM+04]).

- **Control Engineering:** Model predictive control (MPC) is a finite horizon optimization method with the goal to control and stabilize processes. A mathematical model of the process is built from measured data, which describes the behavior of the studied physical process. The process model plays an important role in MPC, being used for predicting the response of the process based on current measurements and future predictions. The empirical models have to be accurate enough to capture the system’s dynamic and simple enough for solving the optimization problem, and are subject of current research (see [BGPV05, FTMLM03, PJFTV07]).
In all these applications there is no a priori knowledge available about the nature of the underlying function. Therefore, a model is inferred from training data, generalizing it in a reasonable way to the unseen function. Hence, a better understanding of the underlying process is gained and predictions for new samples are possible.

An important issue regarding the quality of the prediction model is its generalization power. The data most probably contains noise, hence, the model does not need to describe it perfectly. If the model allows a larger error on the training data by not passing through all the samples, it is likely to result in a better generalization. On the other hand, if the model is not complex enough to capture the nonlinearity of the function, a systematic prediction error occurs. In this context, overfitting refers to a high prediction accuracy due a too complex model and underfitting refers to a low prediction accuracy due to a too simple model. A major issue in the computation of prediction models is to find the right trade-off between overfitting and underfitting, minimizing the prediction error.

These models should be easy to understand, require simple computations when making predictions and deliver accurate predictions. Selecting a modeling technique is an important design decision. A naive approach would be to identify for each query point the nearest neighbors in the training set, fit a linear regression model through them and use it to predict the response of the query. Not only that this approach does not build an explicit model of the data, but also, making predictions for incoming samples is computationally expensive. In the literature two major directions have been established: neural network methods, and regression analysis.

Neural networks are used for a variety of tasks, foremost among them being the predictive analysis. Designed as a highly interconnected group of nodes, neural networks model the relationship between predictors and response variables by applying weights to the nodes and sending the predictors through this network (more details are available in [Bis09]). Neural networks are accurate predictors, but the models are difficult to characterize and understand, and the interpretability of the model for the underlying process is often a requirement. To this end, explicit mathematical functions
are preferred, which can be obtained with regression analysis.

Again, the goal is to characterize an unknown function by estimating a good approximation based on the available samples, called estimator. Generally, an estimator can capture a wide range of functional forms. While least squares is well suited for linearly correlated data, nonlinearity is a natural characteristic of many real-world data. An easy solution is the linearization of the data, for example with a logarithmic or an exponential function, followed by a simple linear regression in the transformed space. Another solution is polynomial regression, which is solved as linear regression with the only difference that the different powers of the predictors are added as predictor variables. These approaches have closed form solutions for approximating nonlinear data, but have a limited use since they cannot express any arbitrary function. On the other hand, nonlinear least squares is able to approximate arbitrary nonlinear functions. However, it requires prior knowledge about the nonlinearity. Also, there is no closed form solution for this problem, and instead it must be solved iteratively whereat convergence is not guaranteed (more details are available in [DS98, HPS13]).

Another approach to approximate nonlinear functions is through piecewise linear regression, which is a good compromise between linear and nonlinear models, not requiring any prior knowledge about the type of nonlinearity. The main idea is to replace a global model, which might become too complex, with several local models fitted with linear least squares. Thus, the model becomes simple while still being able to accurately describe the data, and therefore this approach is preferred in many applications.

The increasing need of scientific databases to work with continuous values instead of discrete observations has been subject of recent research. Using statistical models for predicting continuous data has been discussed in [DGM+04, DGM+05]. By performing queries directly on the model instead of the raw data a more robust interpretation is provided, noise is removed, and missing values are filled up. In [TM08] the architecture of a data management system is introduced, which fundamentally integrates statistical models in form of regression trees into database systems. In addition, an algebraic query processor is proposed, which performs queries directly onto the piece-
wise linear model. Its performance depends on how compact the model representation is. A model with several pieces might be more accurate, but also slower for query processing. Hence, the segmentation has a direct influence on the results. However, finding an appropriate partitioning in the predictor space such that the model remains compact yet accurate is a nontrivial problem. While the optimal solution has been shown to be NP-hard (see [BGPV05, AM02] for more details), a well-established solution is to recursively break down the predictor space until the samples can be accurately described by a linear equation. The resulting construction is called regression tree, and is closely related to the decision tree, which is used for the prediction of categorical class labels.

In this thesis, approaches for constructing regression trees are studied, with the focus on partitioning the data such that compact and accurate models are obtained. Additionally, with such models available in scientific databases, novel solutions are introduced, which enable a wide range of reverse engineered model-based queries.

1.1 Use Cases

The approaches in this thesis found application in the mechanical engineering area, within several collaborations.

Control Engineering

Processes showing a highly nonlinear behavior and being sensitive to several influencing factors are intended to be controlled in order to reach stability. Insofar, process control has the task to manipulate the actuated variables (inputs to the system) in order to lead the outputs of the system to given reference values. While commonly linear models were considered, recent research is oriented towards extending the Model Predictive Control (MPC) framework to use nonlinear models. The basis of MPC controller is a model of the process, which correlates the input to the output signals. This model is used to predict the controlled outputs of the system, over a finite time hori-
1.1. Use Cases

Figure 1.1: Example measurements from control engineering: in the upper row and the lower row left the three outputs from the DIESEL data set are plotted, and in the lower right image the WIND TURBINE dataset is plotted.

Based on this prediction a given cost function is minimized, where the deviations of the controlled outputs to the reference values are considered. By solving this optimization problem an optimal control sequence for the input signals is determined. The optimization problem is solved at each sampling time: at the next time step a new instance of the optimization problem is solved over a shifted prediction horizon, realizing a feedback mechanism [Mac02]. Since these optimizations have to be made online, the linearity of the submodels is of crucial importance. Hence piecewise linear models are well suited for the description of the system. Since their number has a great impact on the runtime, and their accuracy influences directly the control process, the goal is to have few regions accurately describing the system.

One example from the control engineering area is the control of the low
temperature combustion (LTC) in a 1.9l four cylinder Diesel engine. LTC engines are a promising technology for reducing ecologically damaging exhaust gases [LSH+05]. With the introduction of this new combustion process several challenges arise, which make it a recent object of research. The LTC combustion process shows a highly nonlinear process behavior and is very sensitive to a high number of influencing factors [YZL09]. As a consequence a process control is necessary in order to operate the engine in this combustion mode. The input variables are the starting time of injection before top dead center ($U_{SOI}$), the amount of injected fuel mass ($U_{FMI}$), and the exhaust gas rate ($U_{EGR}$). The output variables are the position of the combustion average ($Y_{CA_{50}}$), which estimates the pollutant emissions, the indicated mean pressure ($Y_{IMEP}$), and the maximum cylinder pressure gradient ($Y_{dP_{max}}$), which correlates to the noise emissions. From this application field we use throughout this theses the DIESEL dataset, which contains 8020 such measurements. A subset of these measurements is illustrated in Figure 1.1, one plot for each output. Based on these measurements, the task is to build a model for each output for controlling the combustion in this engine.

Another example is the pitch angle control for a wind turbine, where the pitch angle is the angle at which the blade surface contacts the wind. The power coefficient ($C_p$) is the fraction of the power in the wind extracted by the wind turbine, which shall be maximized. $C_p$ is assumed to be a function of both the tip-speed ratio (TSR) and the pitch angle ($\lambda$). Any change of the wind or rotor speed affects the $TSR$, inducing a change of $C_p$. With help of a model, the pitch angle can be controlled to adjust the aerodynamic torque of the wind turbine and maximize $C_p$. Hence, the goal is to build a model for the MPC. Throughout this theses the WIND TURBINE dataset is used, which contains 40,680 measurements and is illustrated in the lower right image in Figure 1.1.

**Investigating the Fuel Injection Process**

The fuel injection process is responsible for mixture formation of air and fuel and sets the initial conditions for the subsequent combustion process. For this
Figure 1.2: The upper row illustrates images of the spray from the fuel injection process. In the lower row the corresponding spray features are emphasized.

Figure 1.3: Example measurements from the fuel injection process

purpose, the process is examined with the goal of optimizing it. To this end, experiments are carried out detached from the engine in a pressure chamber and the behavior of the fuel spray is captured with a high speed camera. The input settings of this experiment are the pressure in the chamber, the temperature in the chamber, the injection pressure and the injection time. The outputs are sequenced images of the spray. Three example images can be seen in the upper part of Figure 1.2. Relevant in these images is the form of the spray, quantified by penetration depth, the area of an approximating ellipse, and the spray width. Researchers aim at stabilizing this mixture formation process, by determining correlations between injection parameters and the spray form. To this end, a model of the process is required which
serves both for a better understanding and for planning future experiments. Throughout this thesis the SPRAY dataset is used, which contains 3200 experimental measurements with 4 dimensional inputs and 3 outputs, each in form of a time series of length 30. Figure 1.3 exemplary illustrates the behavior of the penetration depth and the spray area w.r.t. the time and injection time, which has to be modeled.

1.2 Thesis Structure

This thesis is structured in five parts: I Introduction, II Model Trees, III Hinging Hyperplane Models, IV Inverse Prediction, V Summary and Outlook, and finally the Bibliography, Acknowledgments and List of publications.

The introduction contains the motivation for predictive models, especially in the form of piecewise linear models, followed by a brief introduction to linear regression. In the context of under- and overfitting, some methods for dimensionality reduction are discussed, which are applied throughout the thesis.

In Part II regression trees are considered. The aim is to build compact, and hence faster to evaluate models with a good generalization performance. Correlation and regression clustering approaches are investigated for the identification of linearly correlated partitions, and a new subspace model clustering algorithm is proposed.

In many applications, however, the processes are not stationary, but the underlying models change over time. The data arrives continuously and cannot be stored; hence it must be processed online and is then discarded. This poses new challenges for the learning algorithms, which have to continuously adapt to local and global, abrupt and slow changes. Therefore, the learning of time-changing regression models is investigated, and an adaptive regression tree is proposed for online prediction of continuous values.

In Part III the focus lies on hinging hyperplane models. This is an approach for nonlinear function approximation, defined as a sum of basis functions. Their main advantage is that they deliver continuous piecewise linear
models. A novel extension of the hinge finding algorithm for the case of multiple output dimensions is proposed, as well as a new method for increasing the hinge finding efficiency by incorporating the geometrical information of the regression surface.

In Part IV the problem of inverse prediction is studied. While the most common use of the regression trees is to predict output values for a given input setting, the reverse problem is also of interest: specify the desired output and obtain recommendations for necessary input settings. This can contribute to the understanding of structural relationships within the process and also to planning the design for future experiments.

In conclusion, this work covers techniques for building regression trees, ranging from static data to dynamically flowing data, based on piecewise linear models. Part V summarizes the thesis and discusses future research directions.
Chapter 2

Preliminaries

Linear models are well suited to describe the functional relationships between predictor and response variables, as well as the effects caused by changes of the predictors. When the underlying process behaves non-linearly, piecewise linear models can be used to describe it. Compared to polynomial models of higher degree, these require less complex computations, are less prone to overfitting, and are easier to comprehend by the user. Therefore, linear and piecewise linear models are preferred as prediction models. This chapter presents a short overview over linear regression methods. Even though ordinary least squares is the best linear unbiased estimator, in some cases better results are achieved when the variance is reduced at the cost of giving up the unbiasedness. Hence, different shrinkage approaches are discussed: regularization, partial least squares, and the principal Hessian directions.

2.1 Regression Analysis

In the context of regression analysis there are two types of variables: predictor variables (also called inputs), and response variables (also called outputs). The goal is to explore the relationship between the predictor and the response variables. Least squares is the most simple and commonly applied form of regression. Given a dataset $D \subset \mathbb{R}^{d+1}$, with samples of the form
(x, y), where x = [x_1, ..., x_d] is the input vector and y ∈ ℝ the output, the relationship between the inputs and the outputs is modeled by fitting a function f : ℝ^d → ℝ such that the residual sum of squares is minimized:

$$\min \sum_{(x, y) \in D} (f(x) - y)^2.$$  \hspace{1cm} (2.1)

**Linear Regression**

In the case of linear regression, the fitted function has following form: $f_\beta(x) = \beta_0 + \beta_1 x_1 + ... + \beta_d x_d$, and the ordinary least squares provides a closed form solution for computing the coefficient vector $\beta$. For $|D| = n$, let $X \in \mathbb{R}^{n \times d}$ be a matrix having the sample inputs as rows, and $Y \in \mathbb{R}^{n}$ a vector containing the outputs. Assuming w.l.o.g. that the inputs are zero-mean variables, i.e. $\mu(x_i) = 0$, $\forall i \in \{1, ..., d\}$, then $\beta_0 = 0$ and $\beta = [\beta_1, ..., \beta_d]$. Equation 2.1 can be then rewritten as:

$$\min_\beta (X\beta - Y)^2 = \min_\beta (X\beta - Y)^\top (X\beta - Y)$$

$$= \min_\beta (\beta^\top X^\top X \beta - 2\beta^\top X^\top Y + Y^\top Y). \hspace{1cm} (2.2)$$

By differentiating Equation 2.2 and setting the result equal to zero, the following regression coefficients are obtained as solution to Equation 2.1:

$$\beta = (X^\top X)^{-1} X^\top Y$$  \hspace{1cm} (2.3)

This can be easily extended to multidimensional outputs $y \in \mathbb{R}^r$. Let $Y \in \mathbb{R}^{n \times r}$ be the matrix having the outputs as rows, then equation 2.3 becomes:

$$B = (X^\top X)^{-1} X^\top Y,$$  \hspace{1cm} (2.4)

where $B \in \mathbb{R}^{d \times r}$ has as columns the regression coefficients $\beta_j \in \mathbb{R}^d$ for each output $y_j$. More details are available in [DS98].

The Ordinary Least Squares (OLS) solution from Equation 2.3 is according to the Gauss Markov theorem ([Gau23]), the linear unbiased estimator...
2.1. Regression Analysis

with the lowest variance, under the assumption that the input dimensions are independent, the errors have expectation zero, are uncorrelated and have equal variances.

Polynomial Regression

Polynomial regression is a form of linear regression, in which the different powers of the predictors are also predictor variables. For example, for \( d = 1 \), the response can be estimated with a \( k \)-degree polynomial as follows:

\[ \hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \ldots + \beta_k x_1^k. \]

Still, these polynomial models are linear in terms of the unknown coefficients \( \beta \). Therefore, the solution can be computed as in the case of linear regression by using Equation 2.3. Though, as the degree of the polynomial increases, the model is able to capture increasingly complex behaviors.

Bias-Variance Trade-off

In the process of finding the model with the best generalization ability, the prediction error is an important quality measure. Assuming that by repeating the data generation process, several replicates of the data set are obtained containing a certain amount of randomness, and for each data set a model is trained. The error of a predictor due to bias is the difference between the average prediction and the average correct output. Bias measures how big is the systematic error of the prediction. The error of a predictor due to variance is the variability of the prediction of a given point, over the several models.

Considering the squared prediction error of an estimator:

\[ Err(x) = E \left[ (y - f(x))^2 \right] \]
it can be decomposed (following [FHT09]) in:

\[ Err(x) = E\left[ E[f(x)] - y \right]^2 + \left( E[f(x)] - y \right)^2 + \epsilon, \]

where the first term is the variance, the second one the bias, and the third one the error. Giving up the unbiasedness an estimator with a lower variance might be found.

Generally, it holds that complex estimators fit the training data better and are therefore less biased. On the other hand, under the assumption of several data set replicates, the different fitted models will differ a lot from each other, causing a high prediction variance. Conversely, less complex estimators have a low prediction variance, but in the same time a higher systematic error. In the regression context, the complexity of an estimator can be measured by the number of regression coefficients. Hence, the simplest estimator is the mean output. As a constant function it has no variance, but is biased, unless the underlying data is constant, and is the most extreme case of underfitting. In contrast, high order degree polynomials are very flexible, because of the high number of regression coefficients, and can easily lead to overfitting. In practice there exists a trade-off between bias and variance, and a reduced prediction error might be obtained by carefully balancing between them.

**Regularization**

Regularization is one way to control the trade-off between bias and variance, by trading the biasedness from the ordinary least squares against a lower variance. Considering Equation 2.3, one can observe that in the case that the underlying assumption of linear dependency among the inputs is not fulfilled, the inverse of the matrix \( X^\top X \) might be singular and cannot be inverted. Even if the matrix can be inverted, two inputs might still be almost perfectly correlated and thus the inversion is numerically unstable. Insofar, the least squares fit is closely related to the assumption that the inputs are independent, which might not be the case in many practical applications.

Ridge regression (RR), also known as Tikhonov regularization, intro-
Partial Least Squares

Ordinary least squares makes the assumption that the input variables are independent. But in many practical applications a large number of input dimensions is available, which are often correlated. Partial Least Squares (PLS) first introduced by H. Wold in [Wol66], assumes that the inputs are dependent among each other and, therefore, performs a dimensionality re-

duced separately and without knowledge of each other in [HK70] and [TA77], was designed for solving the above mentioned ill-posed problems. The proposed solution is to add a positive constant to the diagonal of the matrix, forcing the non-singularity. The coefficients \( \beta' \) are computed in this case as follows:

\[
\beta' = (X^T X + \lambda_R I)^{-1} X^T y,
\]

where \( I \in \mathbb{R}^{m \times m} \) is the identity matrix, and \( \lambda_R \in \mathbb{R} \) a regularization factor. The residuals' variance decreases for an increasing \( \lambda_R \), while the bias increases. With an appropriate value for \( \lambda_R \), robustness can be gained at the cost of little bias.

Besides enabling a numerically stable matrix inversion, RR can be also used to detect models hidden in subspaces. The regression coefficients \( \beta' \) from Equation 2.5 correspond to following minimization problem:

\[
\min_{\beta'} \left[ \sum_{(x,y) \in D} \left( f(x) - y \right)^2 + \lambda_R \sum_{j=1}^{d} \beta_j^2 \right].
\]

Since both the sum of squared residuals and the length of the coefficient vector are concurrently minimized, the unimportant coefficients shrink towards zero, leading to more generalized models.

There is an extensive literature dealing with different regularization techniques, e.g. LASSO (introduced in [Tib94]), Dantzig Selector (introduced in [CT07]), which goes far beyond the scope of this work. A good overview is provided in [FHT09] and [Ber08].
duction in the input space, such that most of the covariance between the inputs and outputs is kept. This technique was initially designed for the case in which the input dimensionality is much higher than the number of samples, in which case Equation 2.3 becomes underdetermined and ordinary least squares does not yield a solution.

PLS is designed for higher-dimensional response variables, but works similarly for univariate responses. Since PLS is not scale invariant, it assumes w.l.o.g. that both inputs and outputs have zero-mean variables and unit standard deviation, i.e. $\mu(x_j) = 0$, $\sigma(x_j) = 1$, $\forall j \in \{1, \ldots, d\}$, and $\mu(y_e) = 0$, $\sigma(y_l) = 1$, $\forall l \in \{1, \ldots, r\}$, which is easily obtained by z-score normalization. Considering the inputs and outputs in matrix form $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^{n \times r}$, PLS decomposes them as follows:

$$X = TP^T + E$$
$$Y = UQ^T + F,$$

where $P \in \mathbb{R}^{d \times \kappa}$ and $Q \in \mathbb{R}^{r \times \kappa}$ are projection matrices, also called loading matrices, and $T, U \subset \mathbb{R}^{n \times \kappa}$ are the projections of $X$ and $Y$ respectively onto the subspaces defined by $P$ and $Q$, also called scores. The two matrices $E$ and $F$ contain the residuals.

Computing a dimensionality reduction separately in the input space may lead to the loss of some valuable information about the causalities between the inputs and outputs. Therefore PLS was designed as an adaption of the principal component analysis, such that this information does not get lost. The computation of loading matrices $P$ and $Q$ are performed such that the covariance between $T$ and $U$ is maximized. After this decomposition is computed, the relationship between $X$ and $Y$ is modeled via standard linear regression between $T$ and $U$:

$$U = TB' \Rightarrow YQ = XPB' \Rightarrow Y = XPB'Q^T,$$

where $B' \in \mathbb{R}^{\kappa \times \kappa}$. Hence, the regression coefficients of $Y$ on $X$ are given by: $Y = XB$, with $B = PB'Q^T$. The first PLS technique, proposed by H. Wold,
is based on the NIPALS (nonlinear iterative partial least squares) algorithm, which is an efficient algorithm for computing the first few eigenvectors in a principal component analysis. The matrices $X$ and $Y$ are iteratively computed as sums of rank-1 matrices:

$$X = TP^T + E = \sum_{e=1}^{\kappa} t_e p_e^T + E$$

$$Y = UQ^T + F = \sum_{e=1}^{\kappa} u_e q_e^T + F,$$

where $t_e$, $p_e$, $u_e$, and $q_e$ are columns of the matrices $T$, $P$, $U$, and $Q$ respectively. In each iteration, one of each such column is computed, the outer products $t_e p_e^T$ and $u_e q_e^T$ are subtracted from $X$ and $Y$ respectively, and the next iteration is performed. While the bias-variance trade-off for ridge regression is controlled by the regularization factor $\lambda_R$, for PLS it is controlled by the number $\kappa$ of score vectors. Because individual scores are orthogonal, the unbiased OLS solution is obtained by using the full set of $d$ scores. Usually, score vectors corresponding to small eigenvalues are ignored, removing hereby some noise, hence $\kappa < d$ iterations are performed.

NIPALS for PLS is described in Algorithm 1, returning the score and loading matrices. First, the current score vector $u_e$ is randomly initialized with a column of matrix $Y$. Then, the current weight vector $w_e$ is computed as the regression of $X$ on $u_e$, and then normalized. The score vector $t_e$ is computed as the projection of $X$ onto $w_e$, and the loading $q_e$ is computed as the regression of $Y$ onto $t_e$. With the loading $q_e$, the score vector $u_e$ is obtained by projecting $Y$ onto it. This procedure is repeated until the convergence of $t_e$. Then, these vectors are stored as columns of the matrices $T$, $P$, $U$, and $Q$, and $X$ and $Y$ are replaced with their residuals.

Manne [Man87] and Hskuldsson [Hs88] have shown that the resulting vectors $w_e$ and $q_e$, from the $e$-th iteration, correspond to the first left and right singular vectors of $X^{(e)^T} Y^{(e)}$, which is the covariance matrix between
Algorithm 1 Partial Least Squares: NIPALS

**Input:** $X, Y, \kappa$

1: for $e \leftarrow 1 \ldots \kappa$ do
2: \hspace{1em} $t_e = t_{old} = [0, \ldots, 0]$
3: \hspace{1em} repeat
4: \hspace{2em} $u_e = Y_{l_e}$ for some random $l \in \{1, \ldots, r\}$
5: \hspace{2em} $w_e = X^\top u_e / u_e^\top u_e$
6: \hspace{2em} $t_{old} = t_e$
7: \hspace{2em} $t_e = Xw_e$
8: \hspace{2em} $q_e = Y^\top t_e / t_e^\top t_e$
9: \hspace{2em} $u_e = Yq_e / q_e^\top q_e$
10: \hspace{1em} until $\|t_e - t_{old}\| \approx 0$
11: \hspace{1em} $p_e = t_e^\top X / t_e^\top t_e$
12: \hspace{1em} $X = X - t_e p_e^\top$
13: \hspace{1em} $Y = Y - u_e q_e^\top$
14: end for

**Output:** $T, P, U, Q$

the inputs and outputs. Since

$$w_e X^{(e)} Y^{(e)} q_e = t_e^\top u_e = \frac{1}{n-1} \text{cov}(t_e, u_e),$$

it is guaranteed that in each iteration score vectors with maximal covariance are computed.

Hskuldsson also shows in [Hs88] some useful properties of PLS:

1. the vectors $w_e$ are orthogonal: $w_e^\top w_l = 0, \forall l \neq e \in \{1, \ldots, d\}$
2. the vectors $t_e$ are orthogonal: $t_e^\top t_l = 0, \forall l \neq e \in \{1, \ldots, d\}$
3. the vectors $w_e$ are orthogonal to $p_l$: $w_e^\top p_l = 0, \forall l < e \in \{1, \ldots, d\}$
4. the vectors $p_e$ are orthogonal in the kernel space of $X$:
   $p_e^\top (X^\top X)^{-1} p_l = 0, \forall l \neq e \in \{1, \ldots, d\}$
De Jong proposes in [dJ93] SIMPLS, a new PLS algorithm, described in Algorithm 2. For univariate outputs SIMPLS delivers the same result as NIPALS-PLS, and for multivariate outputs the result differs only slightly. Similarly to NIPALS, w and q are the first singular vectors of $X^\top Y$, but this time the scores are directly computed. The main difference is that $S = X^\top Y$ is deflated instead of X and Y, which leads to a more efficient computation and less memory requirements.

PLS models are suited to discover arbitrary oriented subspace models.

---

Algorithm 2: Partial Least Squares: SIMPLS

**Input:** $X, Y, \kappa$

1: $S = X^\top Y$

2: for $e \leftarrow 1 \ldots \kappa$ do

3: $w_e = \text{the first left singular vector of } S$

4: $q_e = \text{the first right singular vector of } S$

5: $t_e = Xw_e$

6: $\|t_e\| = t_e^\top t_e$

7: $t_e = t_e/\|t_e\|$

8: $w_e = w_e/\|t_e\|$

9: $q_e = q_e/\|t_e\|$

10: $p_e = X^\top t_e$

11: $u_e = Yq_e$

12: $v_e = p_e$

13: if $j > 1$ then

14: $v_e = v_e - V(V^\top p_e)$

15: $u_e = u_e - T(T^\top p_e)$

16: end if

17: $v_e = v_e/\|v_e\|$

18: $S = S - v_e(v_e^\top S)$

19: end for

**Output:** $T, P, U, Q$
in high dimensional data sets, and will be used in Chapters 4 and 5 for meaningful split computations in the construction of regression trees.

## 2.3 Principal Hessian Directions

With the purpose of visualizing high dimensional inputs and studying their relationship to the output, Li proposes in [Li92] a dimensionality reduction model of the form:

$$ y = f(p_1^\top x, ..., p_\kappa^\top x, \epsilon), $$

(2.6)

where the vectors $p_1, ..., p_\kappa$ describe the space with necessary information of $x \in \mathbb{R}^d$ about $y \in \mathbb{R}$ and the function $f$ remains unknown. Li refers to this set of $p_j$’s as the *effective dimension reduction* (edr) space. Since the Hessian matrix contains information about the shape of the regression surface, the main idea in [Li92] is that all directions orthogonal to it contain only noise and therefore can be neglected.

The Hessian matrix of a function $f : \mathcal{D} \rightarrow \mathbb{R}$, with the domain $\mathcal{D} \subseteq \mathbb{R}^d$, is a square matrix $H_f \in \mathbb{R}^{d \times d}$ of second order partial derivatives:

$$ H_f(x) = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2}
\end{bmatrix}, $$

which describes the local curvature of the function. Assuming a finite domain $\mathcal{D}$, the mean Hessian matrix is defined as the mean Hessian matrix, over all different locations $x \in \mathcal{D}$:

$$ \overline{H}_\mathcal{D} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} H_f(x) $$

The principal Hessian directions are the eigenvectors $p_1, ..., p_d$ of $\overline{H}_\mathcal{D}$, defining a new coordinate system with the property that the average curvatures
of \( f \) are successively the largest along the axes.

**Theorem 2.1.** If the Hessian matrix \( \overline{H}_D \) is multiplied from the right side with \( \Sigma_D \) before the eigenvalue decomposition, then the principal Hessian directions become invariant under affine transformation of the domain \( D \) of \( f \).

**Proof.** Consider an arbitrary transformation of the domain \( D \) of \( f \): \( \bar{x} = xA, \forall x \in D \), and

\[
\overline{H}_D = \frac{1}{|D|} \sum_{x \in D} H_f(xA)
\]

By applying the chain rule from the classical derivation calculus, the following relationship between their corresponding average Hessian matrices is obtained:

\[
H_{\bar{D}}(x) = A^{-1}H_D(x)A^{-1} \quad \Rightarrow \quad \overline{H}_D = (A^{-1})^T \overline{H}_D A^{-1}.
\]

By imposing that the new coordinate system is uncorrelated, i.e. \( \Sigma_{\bar{D}} = A \Sigma_D A^T = I \), it holds that \( A = O \Sigma_D^{-\frac{1}{2}} \) and \( A^{-1} = \Sigma_D^\frac{1}{2} O^{-1} = \Sigma_D^\frac{1}{2} O^T \), since:

\[
A \Sigma_D A^T = O \Sigma_D^{-\frac{1}{2}} \Sigma_D \left( O \Sigma_D^{-\frac{1}{2}} \right)^T = O O^T = I.
\]

The goal is to find the axes \( p_1, ..., p_d \) of a coordinate system in which the average curvature of \( f \) is the largest along the first axis, and the curvatures of the remaining axes are successively smaller:

\[
\max_{\Lambda \Sigma_D A^T} |e_1^T \overline{H}_D e_1| = \max_{\Lambda \Sigma_D A^T = I} |e_1^T (A^{-1})^T \overline{H}_D A^{-1} e_1| = \max_{\Lambda = O} |e_1^T \left( \Sigma_D^\frac{1}{2} O^T \right)^T \overline{H}_D \Sigma_D^\frac{1}{2} O^T e_1| = \max_{\Lambda = O} |(O^T e_1)^T \Sigma_D^\frac{1}{2} \overline{H}_D \Sigma_D^\frac{1}{2} O^T e_1|,
\]

with \( e_1 = [1, 0, ..., 0] \). Let \( a_1 = O^T e_1 \), then \( a_1 \) is the eigenvector corresponding to the largest eigenvalue \( \lambda_1 \) for the eigenvalue decomposition of \( \Sigma_D^\frac{1}{2} \overline{H}_D \Sigma_D^\frac{1}{2} \):

\[
\Sigma_D^\frac{1}{2} \overline{H}_D \Sigma_D^\frac{1}{2} a_1 = \lambda_1 a_1.
\]
The first eigenvector $p_1$ of $\mathbf{H}_D$ can be rewritten as:

$$p_1 = A^\top e_1 = (O\Sigma_D^{-\frac{1}{2}})^\top e_1 = \Sigma_D^{-\frac{1}{2}}O^\top e_1 = \Sigma_D^{-\frac{1}{2}}a_1,$$

and hence $a_1 = \Sigma_D^\frac{1}{2}p_1$. Equation 2.7 then becomes:

$$\Sigma_D^\frac{1}{2}\mathbf{H}_D\Sigma_D^\frac{1}{2}p_1 = \lambda_1 \Sigma_D^\frac{1}{2}p_1$$

$$\Sigma_D^{-\frac{1}{2}}\mathbf{H}_D\Sigma_Dp_1 = \lambda_1 \Sigma_D^\frac{1}{2}p_1$$

$$\Sigma_D^{-\frac{1}{2}}\Sigma_D^\frac{1}{2}\mathbf{H}_D\Sigma_Dp_1 = \lambda_1 \Sigma_D^{-\frac{1}{2}}\Sigma_D^\frac{1}{2}p_1$$

$$\mathbf{H}_D\Sigma_Dp_1 = \lambda_1 p_1.$$

After fixing the first axis $a_1$, the same procedure is repeated for maximizing $|e_j^\top \mathbf{H}_D e_j|, \forall j \in \{1, ..., d\}$. The affine invariant principal Hessian directions are hence the eigenvectors of the matrix $\mathbf{H}_D\Sigma_D$:

$$\mathbf{H}_D\Sigma_Dp_j = \lambda_j p_j, \quad \forall j \in \{1, ..., d\}$$

$$|\lambda_1| \geq ... \geq |\lambda_d|.$$  \hspace{1cm} (2.8)

The main idea is that if the average Hessian matrix $\mathbf{H}_D$ can be well estimated, then the most significant principal Hessian directions of $\mathbf{H}_D\Sigma_D$ define the edr space. Since in the most cases only samples of an unknown function are available, a method for estimating the average Hessian matrix from a set of sample points $\{(x_i, y_i)\}_{i=1}^n$ is necessary. Li proposed in [Li92] an estimation method, based on Stein’s Lemma (introduced in [Ste81]).

**Stein’s Lemma.** Let $z$ be a normally distributed random variable with mean $\mu_z$ and variance 1, and $g$ a differentiable function, then:

$$E[(z - \mu_z)g(z)] = E[g'(z)]$$

$$E[(z - \mu_z)^2g(z)] = E[g(z)] + E[g''(z)],$$

assuming that the involved derivations and expectations exist.
2.3. Principal Hessian Directions

Assuming $x$ and $y$ are normally distributed with mean $\mu_x$ and $\mu_y$ respectively, and $\tilde{x} = \Sigma_D^{-\frac{1}{2}}x$. Expressing the output $y$ as a function of the inputs: $y = g(x) = g(\Sigma_D^{\frac{1}{2}}\tilde{x})$, and using Stein’s Lemma leads to:

$$E[(\tilde{x} - \mu_{\tilde{x}})(\tilde{x} - \mu_{\tilde{x}})^\top (y - \mu_y)] = E[y - \mu_y] + E\left[\frac{\partial^2 y}{\partial \tilde{x}^2}\right],$$

where the first right hand term is zero and the second one corresponds to $\overline{H}_D$. Hence,

$$\overline{H}_D = E[(y - \mu_y)(\tilde{x} - \mu_{\tilde{x}})(\tilde{x} - \mu_{\tilde{x}})^\top]. \tag{2.9}$$

The relationship between $\overline{H}_D$ and $\overline{H}_D$ holds due the chain rule from the derivation calculus:

$$\overline{H}_D = \Sigma_D^{\frac{1}{2}}\overline{H}_D \Sigma_D^{\frac{1}{2}} \Rightarrow \overline{H}_D = \Sigma_D^{-\frac{1}{2}}\overline{H}_D \Sigma_D^{-\frac{1}{2}} \tag{2.10}$$

Setting Equation 2.9 into Equation 2.10, following is obtained:

$$\overline{H}_D = \Sigma_D^{-\frac{1}{2}}E[(y - \mu_y)(\tilde{x} - \mu_{\tilde{x}})(\tilde{x} - \mu_{\tilde{x}})^\top] \Sigma_D^{-\frac{1}{2}}$$

$$= \Sigma_D^{-\frac{1}{2}}\Sigma_D^{-\frac{1}{2}} E[(y - \mu_y)(x - \mu_x)(x - \mu_x)^\top] \left(\Sigma_D^{-\frac{1}{2}}\right)^\top \Sigma_D^{-\frac{1}{2}}$$

$$= \Sigma_D^{-\frac{1}{2}}\Sigma_D^{-\frac{1}{2}} E[(y - \mu_y)(x - \mu_x)(x - \mu_x)^\top] \Sigma_D^{-\frac{1}{2}} \Sigma_D^{-\frac{1}{2}}$$

$$= \Sigma_D^{-1} E[(y - \mu_y)(x - \mu_x)(x - \mu_x)^\top] \Sigma_D^{-1}$$

For $\Sigma_{yxx} = E[(y - \mu_y)(x - \mu_x)(x - \mu_x)^\top]$, following estimation for the average Hessian matrix is obtained:

$$\overline{H}_D = \Sigma_D^{-1}\Sigma_{yxx}\Sigma_D^{-1}. \tag{2.11}$$

Since the addition or subtraction of a linear function has no influence on the Hessian matrix, the output $y$ from Equation 2.11 can be replaced by the residuals $r$, obtaining

$$\overline{H}_D = \Sigma_D^{-1}\Sigma_{rxx}\Sigma_D^{-1}, \text{with } \Sigma_{rxx} = E[r(x - \mu_x)(x - \mu_x)^\top]. \tag{2.12}$$

Hence two dimensionality reduction methods are proposed: the first one
is called $y$-based, and computes $H_D \Sigma_D$ as $\Sigma_D \Sigma_{yx}^x$, and the second one is called $r$-based, and computes $H_D \Sigma_D$ as $\Sigma_D \Sigma_{rx}^x$. Li proposed in [Li92] a third estimation method, which fits through $y$ a quadratic polynomial of $x$. The Hessian matrix is evaluated for all samples points and the average of these matrices matrix computed. This method is referred to as $q$-based.

Li proves in [LLC00] that the introduced estimation methods work even in the case of general switching models, of the form:

$$y(x) = \begin{cases} 
    a_{11}x_1 + \ldots + a_{1d}x_d + a_{10}, & \text{if } h \cdot x \geq 0, \\
    a_{21}x_1 + \ldots + a_{2d}x_d + a_{20}, & \text{else},
\end{cases}$$

which do not poses a second order derivative, conferring the approach a more general character.

Although the approach proposed by Li was initially designed to support the visualization of high dimensional data, by choosing the right viewing angle, it can be also seen as a shrinkage method, since it computes a subset of regressors which best define the data. Its geometrical properties of the principal Hessian directions are used for the construction of a regression tree, as seen later in Chapters 3 and 8.
Part II
Model Trees
Chapter 3

Regression Trees

While linear regression computes a global model of the data, in real world data the inputs often interact in a complicated nonlinear way. Instead of a complex model, a piecewise linear model of the form:

\[
f(x) = \begin{cases} 
 x^\top \beta_1, & \text{if } x \in P_1 \\
 x^\top \beta_2, & \text{if } x \in P_2 \\
 \vdots \\
 x^\top \beta_m, & \text{if } x \in P_m 
\end{cases}
\]

can be used for approximating the underlying function, yielding a simple and intuitive description of the investigated data: for each partition \( P_j \), \( j \in \{1, \ldots, m\} \) in the input space a linear model applies. Any function can be arbitrarily good approximated with enough submodels, trading the simplicity against accuracy. An example is illustrated in Figure 3.1. If the partitions \( P_j \) are known, then linear regression can be performed independently in each of the partition. Generally, few partitions accurately described by linear models are preferred. Therefore, computing the partitions is closely connected to the computation of linear models, and is a challenging task. A solution is to recursively subdivide the input space into smaller regions and fit a linear model to each of the partitions. The tree used to represent this recursive partitioning is called regression tree, and has the inner nodes labeled with
Figure 3.1: Illustrate a piecewise linear function with four and eight partitions respectively, approximating the nonlinear behavior of the samples tests in form of inequalities in the input space, and predictive models in the leaves. Regression (or model) trees are a special type of decision trees with linear regression functions in the leaves, where continuous values are predicted, instead of class labels.

In the construction of regression trees, one deals with the trade-off between generalization and overfitting. A small tree yields a very general model, which is compact and easy to read, but might not be very accurate. Letting the tree grow bigger, increases the accuracy but it can also lead to overfitting, as well as high computation and prediction runtimes. Therefore, computing appropriate splits is an important issue, contributing to more compact and accurate models.

3.1 Introduction

For the task of learning a regression tree, a set observations of the form \((x, y)\), with input \(x = [x_1, \ldots, x_d] \in \mathbb{R}^d\) and output \(y \in \mathbb{R}\) is required, which is called training set and is denoted by \(T\), with \(|T| = n\). Typically, regression trees are binary, containing in each inner node a test in the input space, and in each leaf a linear model predicting the output. The training data is recursively partitioned, starting with all training data at the root. The split strategy determines how to partition a nodes dataset, and the stop criterion indicates when to terminate the recursive partitioning and build leaf models.
The overall goal is to minimize a chosen *impurity measure*. In the test phase, for each incoming sample the test is applied at each internal node, and the outcome decides whether to continue with the left or the right child node. When a leaf node is reached, the corresponding output is predicted by the leaf model. The classic procedure for building regression trees is presented in Algorithm 3.

**Algorithm 3 Regression Tree**

1: `function BUILDREGRESSIONTREE(T)`
2:    find the split with maximal impurity reduction
3:    split T in T₁ and T₂
4:    `if (stopping criterion(T₁) not reached) then`
5:        `BUILDREGRESSIONTREE(T₁)`
6:    `endif`
7:    `if (stopping criterion(T₂) not reached) then`
8:        `BUILDREGRESSIONTREE(T₂)`
9:    `endif`
10: `end function`

Returning to the example from Figure 3.1, the one dimensional input space is recursively divided building up the tree illustrated in Figure 3.2. Each cut in the tree represents a piecewise linear model of a different granularity and accuracy. The samples are most accurately described by linear models in the leaves.

In the following sections, different techniques for constructing regression trees found in the literature are discussed. The split strategy determines how the split is computed and is the main concern. However, the impurity assesses the goodness of a split and has a direct influence on the chosen split. Commonly used impurity measures for a model f are the mean absolute deviation:

\[
\text{mad}_f(T) = \frac{1}{|T|} \sum_{(x,y) \in T} |y - f(x)|, \tag{3.1}
\]
and the mean squared error:

$$\text{mse}_f(T) = \frac{1}{|T|} \sum_{(x,y) \in T} (y - f(x))^2,$$  \hspace{1cm} (3.2)

The choice between the mean average deviation and the mean squared error influences how the outliers are handled. While a model using $\text{mse}_f$ as impurity measure might better fit the data, it might also be more sensitive to outliers. On the other hand $\text{mad}_f$ is more robust against the outliers.

The overall goal in the construction of a regression tree is to minimize the impurity. Since computing the global minimum is infeasible, the algorithms for constructing regression trees are usually being greedy: the impurity is reduced as much as possible with each split, which corresponds to maximizing the impurity reduction in each split. Hence, the goal of a split of $T$ in $T_1$ and $T_2$ is to maximize following equation:

$$\Delta \text{Impurity}(\text{split}) = \text{Impurity}(T) - \left( \text{Impurity}(T_1) + \text{Impurity}(T_2) \right)$$
3.2 Regression Trees with Axis-Parallel Splits

The first regression trees proposed in the literature consider only axis parallel splits, compute all possible candidate splits and choose the one with the lowest impurity.

**CART**

Breiman et al. introduced in [BFSO84] the first regression tree, CART (Classification and Regression Tree), with the output means as piecewise constant models in the leaves. As impurity of a node, with a corresponding training set $T$, the mean squared prediction error is used:

$$\text{mse}_x(T) = \frac{1}{|T|} \sum_{(x,y) \in T} (f(x) - y)^2 = \frac{1}{|T|} \sum_{(x,y) \in T} = (x - y)^2 = \sigma_y^2(T), \quad (3.3)$$

whose root is the standard deviation in the output dimension. Hence, the impurity reduction of splitting $T$ in $T_1$ and $T_2$ is computed as:

$$\Delta \text{Impurity}(T) = \sigma_y(T) - \left( \sigma_y(T_1) + \sigma_y(T_2) \right) \quad (3.4)$$

The split strategy of CART considers each attribute separately and computes all possible splits. Among all these splits, the one with the maximal impurity reduction is chosen, according to Equation 3.4. However, this strategy is constrained to axis parallel splits. Rather than imposing restrictive stopping criteria, CART lets the tree grow very large. Finally, to avoid overfitting, the fully grown tree is pruned in order to improve the performance and accuracy of the decision tree on unseen data.

**M5**

M5, introduced by Quinlan in [Qui92], replaces the constant numerical values in the leaves with linear models, but continues using the impurity defined in Equation 3.3, used by CART. M5 computes the splits such that the residual sum of squares w.r.t. the output mean is minimized, but fits at the end a
Figure 3.3: (Left) Split based on minimizing the standard deviation of the output (Right) Split based on minimizing the mean squared error as in the classic the least squares model

function which minimizes the residual sum of squares w.r.t. a linear model, leading to poor results. This idea is illustrated in Figure 3.3, where, even if the standard deviation of the output in the right cluster is high, the data is well described by a linear function. The split computed by minimizing the error w.r.t. to the fitted linear model (right Figure) is obviously better than the one computed by minimizing the output standard deviation (left Figure).

In [Kar92] Karalic pointed out the accuracy benefits from using the mean squared error w.r.t. to the fitted least squares model as impurity measure, being thus consistent with the final least squares models in the leaves. While the computation of the standard deviation for all possible splits in one input dimension is efficient, computing the two least squares models and the corresponding residuals for all possible splits is more expensive. Therefore, heuristics are required in order to avoid an exhaustive search after the best split.

SUPPORT

Chaudhuri et al. introduced in [CcHLY94] the Smoothed and Unsmoothed Piecewise-POlynomial Regression Tree (SUPPORT). Instead of allowing the tree to grow extremely large and prune it in a post-processing step, SUPPORT has a look-ahead stopping rule, limiting the tree size. Also, by fitting
3.2. Regression Trees with Axis-Parallel Splits

Unlike CART and M5, SUPPORT does not exhaustively search for the best polynomial models instead of linear ones, the regression tree becomes much compact than the one computed by CART.

GUIDE

Loh introduces in [Loh02] GUIDE (Generalized, Unbiased Interaction Detection and Estimation), a new regression tree, which tries to overcome two deficiencies of SUPPORT: the inability to deal with categorical input dimensions and the inability to exhaustively search for the best splits, but computes for each input dimension \( j \) the best split point \( \alpha_j \). SUPPORT starts by fitting a linear model to the node's training data \( T \) and dividing the samples in two partitions \( T^+ \) and \( T^- \), one with positive residuals and one with negative residuals. Let \( x_i \in \mathbb{R}^d \) be the mean input values and \( z_i \in \mathbb{R}^d \) the input values shifted to the center, and \( \sigma_T x \in \mathbb{R}^d \) and \( \omega_T z \in \mathbb{R}^d \) their corresponding means. Then, for each input dimension \( j \in \{1, \ldots, d\} \), SUPPORT checks how different the two distributions are by using the following two Student's t-tests:

\[
t(1)_j = \frac{x_j^+ - x_j^-}{\sqrt{\frac{1}{|T^+|} + \frac{1}{|T^-|}}}
\]

\[
t(2)_j = \frac{z_j^+ - z_j^-}{\sqrt{\frac{1}{|T^+|} + \frac{1}{|T^-|}}}
\]

The intuition behind this is that if the fit is adequate, then the two distributions are not significantly different. The output of the two tests is the smallest of the two corresponding p-values from the table of values for the Student's t-distribution, denoted as \( \alpha_j \). The overall best split is in the input dimension \( j' = \arg\min_{1 \leq j \leq d} \alpha_j \), and the split point \( s_j' = \frac{x_{j'}^+ + x_{j'}^-}{2} \). To decide whether to continue splitting or not, a look-ahead cross validation stopping rule is proposed. For each fold, a nested sequence of trees is constructed from \( T \) and the corresponding mean squared error on the test set computed. If there is a tree with a substantially smaller error than just using the root, the frequency of nontrivial trees within the different folds is higher than a predefined threshold, then the split on \( T \) is performed.

To provide the samples in two partitions \( T^+ \) and \( T^- \), one with positive residuals and one with negative residuals. Let \( x_i \in \mathbb{R}^d \) be the mean input values and \( z_i \in \mathbb{R}^d \) the input values shifted to the center, and \( \sigma_T x \in \mathbb{R}^d \) and \( \omega_T z \in \mathbb{R}^d \) their corresponding means. Then, for each input dimension \( j \in \{1, \ldots, d\} \), SUPPORT checks how different the two distributions are by using the following two Student's t-tests:
sions, which support splits but do not contribute to linear models, and to detect pairwise interactions. Similarly to SUPPORT, in the search for the best split GUIDE fits a model through the node’s training set and separates the samples according to the sign of their residuum, replaces the Student’s test with the $\chi^2$ test for a better identification of curvatures in the data. The values of each predictor are additionally grouped in four categories, by dividing the range at the quartiles, yielding a $2 \times 4$ contingency table.

GUIDE comes in two variants, in the first one constant models are fitted in the leaves, which allows for an equal treatment of numerical and categorical input dimensions.

Besides considering each input dimension separately, GUIDE also considers them pairwise, uses the $\chi^2$ test to detect curvatures, and chooses the smallest $p$-value. If this value is from a single input dimension, it selects this input dimension to split the node. If it corresponds to an interaction between two input dimensions and both of them are numerical then the split which generates the smallest sum of errors is chosen, otherwise the one with a smaller $p$-value is chosen.

The second GUIDE variant computes linear models in the leaves, hence different roles need to be allocated to the predictors:

- numerical predictor used for splitting but not for fitting a linear model,
- numerical predictor used for fitting a linear model but not for splitting,
3.3 Regression Trees with Oblique Splits

- numerical predictor used for both splitting but and fitting a linear model,
- categorical predictor used for splitting but not for fitting a linear model.

The split strategy is similar to the first variant, with the difference that the split point in a selected input dimension is the sample median.

Loh observed that CART has a high variable selection bias, meaning that predictors which allow more splits are also more often preferred. He empirically showed that in the first GUIDE variant this is not the case, and proposed for the second variant a bootstrap calibration algorithm for bias correction.

Both SUPPORT and GUIDE are based on separating the data according to the residuals’ sign. While this might work in some cases (see left image in Figure 3.4), this strategy does not make sense in other, likely to occur, cases (see right image in Figure 3.4).

### 3.3 Regression Trees with Oblique Splits

The previous approaches have in common that they are restricted to axis-parallel splits. Considering all input dimensions simultaneously makes the detection of correlations between them possible, and facilitates the computation of oblique splits. Accurate models can be thus obtained with fewer splits and, therefore, more compact regression trees are built. In Figure 3.5 an example is shown, where, for separating two sample sets, three axis-parallel splits can be replaced by a simple oblique split.

With this additional degree of freedom though, the efficiency of building regression trees becomes an issue. Since splitting strategies play the main role in the construction of regression trees, efficient strategies for computing oblique splits are necessary. In the literature, the approaches consist of a partitioning strategy, followed eventually by a separation through a linear classifier. Since binary linear classifiers are more efficient and effective to compute than multi-class classifiers, usually binary regression trees are built.
Li proposed in [LLC00] to compute a regression tree by using the principal Hessian directions (PHDRT), the first regression tree with oblique splits. The main idea is to explore the geometric information of the regression surface, in order to find the direction in which the surface bends the most and use this to compute the best split. The variation of the gradient vectors at different locations describes the complexity of the regression surface. In the case of a linear function the gradient vectors are equal. For nonlinear functions though, the gradient vectors vary. The goal is to recursively split the input space, until the gradient vectors within each region become similar and the corresponding samples accurately described by a linear function. To estimate the shape of the multidimensional regression surface, the average Hessian matrix, as introduced in Chapter 2, is used.

The splitting strategy is divided in two phases: split direction selection and cut point selection. Based on the findings from [Li92], the average Hessian matrix is estimated with the $r$-based method. Its first eigenvector, the principal Hessian direction $p_1$, is used to determine the direction in which the regression surface bends the most, and choose it as the split direction. Once determined, all sample inputs $x \in T$ are projected onto $p_1$: $x' = x^\top p_1$, and in this reduced space with one input and one output a cut point is determined. The function describing a split computation is summarized in Algorithm 4.
3.3. Regression Trees with Oblique Splits

**Algorithm 4 PHD Regression Tree**

1. **function** PHDSPLIT(T)
2. set the cutpoint \( c = \mu_{x'} \),
3. split \( D \) in: \( T_+ = \{(x', y) \in T | x' \geq c \} \) and \( T_- = \{(x', y) \in T | x' < c \} \),
4. fit a linear regression \( l_+ \) in \( T_+ \), and \( l_- \) in \( T_- \),
5. compute \( \sigma_- \) and \( \sigma_+ \) as the residual standard deviations in \( T_+ \) and \( T_- \),
6. compute \( \sigma_w = \frac{|T_+| \cdot \sigma_+ + |T_-| \cdot \sigma_-}{|T|} \),
7. if \( \sigma_w \) is smaller than a threshold, stop computing and return \( c \),
8. if \( \sigma_+ \geq \sigma_- \) then
9. let \( \mu_{x'+} \) be the mean of \( x \in T_+ \),
10. let \( T_{++} = \{(x', y) \in T_+ | x' \geq \mu_{x'+} \} \),
11. fit a linear function \( l_{++} \) through \( T_{++} \),
12. set \( c \) as the cut point between \( l_- \) and \( l_{++} \),
13. else
14. let \( \mu_{x'-} \) be the mean of \( x \in T_- \),
15. let \( T_{--} = \{(x', y) \in T_- | x' \geq \mu_{x'-} \} \),
16. fit a linear function \( l_{--} \) through \( T_{--} \),
17. set \( c \) as the cut point between \( l_+ \) and \( l_{--} \),
18. end if
19. end function

The recursive splitting of PHDRT is stopped when either the sample size of a node is smaller than a user defined threshold, or the pHd is insignificant, meaning that its eigenvalue is smaller than another user defined threshold.

**SECRET**

Another regression tree with oblique splits was introduced by Dobra et al. in [DG02], called SECRET. The split strategy consists out of a clustering step followed by a classification step. In the clustering step the available training data is split in two clusters using the Expectation Maximization (EM) clus-
tering algorithm. Then, the samples are labeled according to the cluster with the highest membership probability and a classifier is trained in the input space to discriminate between these two clusters. While in the clustering step only numerical attributes can be involved, the training of a classifier also considers categorical attributes, allowing for a better separation of the data. Similarly to GUIDE, different roles are allocated to the predictors.

As efficient classifiers, Quadratic Discriminant Analysis (QDA) and Linear Discriminant Analysis (LDA) were used (see [FHT09]). QDA assumes that the samples from each class are normally distributed, and separates them by a quadratic surface. Let \( C = \{c_1, c_2\} \) be the cluster labels, \( T_{x_1} \) the set of samples projected onto the input space with cluster label \( c_1 \), described by \( N_1 = (\mu_{x_1}, \Sigma_{x_1}) \), and \( T_{x_2} \) the set of samples with cluster label \( c_2 \), described by \( N_2 = (\mu_{x_2}, \Sigma_{x_2}) \). The probability density of a sample \( x \) given the cluster \( c_j \) is:

\[
p(x|c_j) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_{x_j}|}} e^{-\frac{1}{2}(x-\mu_{x_j})^\top \Sigma_{x_j}^{-1}(x-\mu_{x_j})}
\]

The decision rule is:

\[
\delta(x) = \arg\min_{c_j \in C} P(c_j|x) \quad \text{Bayes rule}\ 
= \arg\min_{c_j \in C} \frac{p(x|c_j)P(c_j)}{P(x)} = \arg\min_{c_j \in C} p(x|c_j)P(c_j),
\]

where \( P(c_j|x) \) the posterior probability of label \( c_j \) given the sample \( x \), \( P(x) \) is the prior probability of \( x \), and \( P(c_j) \) the prior probability of the cluster \( c_j \). Computing the logarithm, the decision rule becomes:

\[
\delta(x) = \arg\min_{c_j \in C} \left\{ -\frac{1}{2} \log |\Sigma_{x_j}| - \frac{1}{2} (x-\mu_{x_j})^\top \Sigma_{x_j}^{-1}(x-\mu_{x_j}) + \log \frac{|T_{x_j}|}{|T_x|} \right\} \quad (3.5)
\]

The boundary between \( c_1 \) and \( c_2 \) is therefore described by the quadratic equation obtained by setting: \( \ln p(x|c_1) = \ln p(x|c_2) \).

LDA computes a linear boundary between the classes by assuming that the covariance matrices of the classes are equal. The pooled covariance
3.3. Regression Trees with Oblique Splits

Figure 3.6: Different views of a clustering: while the numerical input $x_c$ allows for a good clustering, the discrete input $x_d$ makes a meaningful split possible

The decision function from Equation 3.5 is simplified for LDA to:

$$\delta(x) = \arg\min_{c_j \in C} \left\{ x^T \Sigma^{-1}_{T_x} \mu_{x_j} - \frac{1}{2} \mu_{x_j} \Sigma^{-1}_{T_x} \mu_{x_j} + \log \frac{|T_{x_j}|}{|T_x|} \right\},$$

which corresponds to a linear hyperplane. This hyperplane is equivalent to the one minimizing Fisher’s separability criterion, which is described by the normal vector $n$:

$$n = \frac{\Sigma^{-1}_{T_x} (\mu_{x_1} - \mu_{x_2})}{\| \Sigma^{-1}_{T_x} (\mu_{x_1} - \mu_{x_2}) \|^2}.$$  

In order for SECRET to compute this oblique split, all numerical inputs are used. It might happen though, that although EM identifies two clusters, there is no suitable linear split in the continuous input dimensions. To improve the separability, SECRET integrates the discrete attributes in the split decision. An example from [DG02] is illustrated in Figure 3.6: the left figure shows the two identified clusters for which there exists no meaningful split in the numerical input space, $x_c$, but the additional discrete input dimension $x_d$, matrix is computed as:

$$\Sigma_{T_x} = \frac{1}{|T_x| - |C|} \sum_{c_j \in C} \sum_{x \in T_x} (x - \mu_{x_j})(x - \mu_{x_j})^T$$  

(3.6)
Figure 3.7: Identification of piecewise linear models with bounded error from the right figure, provides the necessary split.

In order to measure the quality of a split, SECRET uses following simplified Gini-gain for binary splits and two class labels:

$$
\Delta \text{gini}(T, T_1, T_2) = P(c_1|T)^2 \frac{\left( P(c_1|T_1) - P(T_1) \right)^2}{P(T_1)\left(1 - P(T_1)\right)},
$$

where $P(c_1|T)$ is the class prior probability of all observations in $T$, and $P(T_1)$ the fraction of data associated with $T_1$.

**A Bounded Error Approach**

Another approach investigated in the literature is to explicitly formulate the computation of piecewise linear models as an optimization problem. Considering the samples’ input matrix $X$ and output matrix $Y$, the system $X\beta = Y$ is usually not feasible, meaning that there exists no $\beta$ satisfying the equation. Therefore, the least squares approach fits a linear model through this samples which minimizes the mean squared error $\|X\beta - Y\|_2^2$. Still, in many real world data applications this might not deliver a satisfying accuracy, the predictions having a high error. Hence, the user searches for a minimal number of feasible subsystems (MIN PFS): $X_j\beta_j = Y_j, \forall j \in \{1, \ldots, k\}$. This solution
3.3. Regression Trees with Oblique Splits

delivers a partitioning into a minimal number of submodels, and the parameter sets for each partition. However, the real world measurements are often noisy and allowing a certain amount of error of the model is reasonable. Hence, a user searches for the minimum number of subsystems which simultaneously satisfy following conditions:

\[
X_1\beta_1 \leq Y_1 + \epsilon \quad \text{and} \quad X_1\beta_1 \geq Y_1 - \epsilon
\]

\[
X_1\beta_k \leq Y_k + \epsilon \quad \text{and} \quad X_k\beta_k \geq Y_k - \epsilon,
\]

The trade-off between accuracy and overfitting is controlled through the threshold \(\epsilon\). This idea is illustrated in Figure 3.7, where with a higher the threshold, more noise is allowed and less submodels are built. The complexity of MIN PFS is NP-hard, and therefore approximations have been proposed, e.g., by Amaldi et al. in [AM02], by Bemporad et al. in [BGPV05]. They proposed greedy algorithms, which start with all samples and search for the close to maximum feasible subsystem. After extracting it, this process is iteratively repeated. While the identification of the maximum feasible subsystem (MAX FS) is also NP-hard, relaxation methods have been extensively studied in the literature, and can be used to compute the close to maximum feasible system. An important feature of this algorithm is that points are grouped together if they lay on the same hyperplane, regardless of their distance to each other. While this is an advantage in the given context, it is in the same time a disadvantage, as discussed in the next section.

Separability in Input Space

Summing up, the greedy approaches for constructing a regression tree with oblique splits have two main steps in their splitting strategy. First, a clustering of the data is performed. Second, this clustering is used for labeling the data and computing a linear classifier which generalizes this clustering. Being the main step in the split computation, the clustering has a big influence on the impurity reduction of this split. Linearly correlated clusters enable
splits that are more meaningful compared to clusters based on Euclidean neighborhoods. The resulting regression tree is thereby both compact and accurate, insofar the data allows it. But the approaches which disregard completely the distances between the samples have as drawback the fact that they do not necessarily compute separable clusters in the input space.

**Definition 3.1.** Two sets of samples \( C_1, C_2 \in \mathbb{R}^d \) are called **strictly linearly separable**, if they are in disjoint half spaces, meaning that there exists a hyperplane defined by the normal \( n \in \mathbb{R}^d \) and an offset \( o \in \mathbb{R} \) such that:

\[
\forall x \in C_1 : \quad x^\top n + o \geq 0 \quad \text{AND} \quad \forall x \in C_2 : \quad x^\top n + o < 0.
\]

**Definition 3.2.** Two sets of samples \( C_1, C_2 \in \mathbb{R}^d \) are called **linearly separable up to a factor of** \( \delta \in [0, 1] \), if there exists a hyperplane defined by the normal \( n \in \mathbb{R}^d \) and an offset \( o \in \mathbb{R} \) such that:

\[
|C'_1| \geq \delta |C_1| \quad \text{AND} \quad |C'_2| \geq \delta |C_2|,
\]

where \( C'_1 \) and \( C'_2 \) are defined as:

\[
C'_1 = \{ x \in C_1 | x^\top n + o \geq 0 \} \quad \text{AND} \quad C'_2 = \{ x \in C_2 | x^\top n + o < 0 \}.
\]

Figure 3.8 illustrates two strictly linearly separable clusters on the left.
3.3. Regression Trees with Oblique Splits

Figure 3.9: Separability of correlation clusters in input space

(with $\delta = 1$), and two overlapping clusters on the right, which are linearly separable up to a factor of $\delta = 0.5$. Hence, in the best case, the clusters are separable up to a factor of $\delta = 1$, and in the worst case, where two clusters completely overlap, the clusters are separable up to a factor of around 0.5.

Returning to the construction of regression trees: trying to linearly separate classes which are not separable in the classification phase, leads to poor results. Since linear classifiers can handle a small $\epsilon$ such that:

$$\forall x \in C_1 : x^T n + o + \epsilon \geq 0 \quad \text{AND} \quad \forall x \in C_2 : x^T n + o - \epsilon < 0,$$

linearly separable clusters up to a factor $\delta$ close to 1 are desirable for the construction of regression trees.

An example with a single input dimension $x$ and a output dimension $y$, is illustrated in Figure 3.9. In the left image the grey and the black cluster are strongly linearly separable in the input space. In the right image the two clusters are separable up to a factor of only $\delta = 0.5$. Using this clustering to train a linear separator in the input space leads to a poor accuracy of the classification and to a poor accuracy of the fitted models. Although the two grey clusters lie distant from each other, they are well described by the same linear model. While in the left image all the grey points can be gathered in one cluster, in the right image they have to be separated in two clusters in order to be linearly separable up to a high factor from the black samples.
In conclusion, for the construction of regression trees it is important to cluster samples which are well described by a linear model disregarding their Euclidean distance to each other, as long as the clusters are linearly separable up to a high factor. Unfortunately, both SECRET and the bounded error approach fail to recognize these cases, always grouping samples which lie on the same hyperplane.
Chapter 4

Split Strategies using Clustering

As seen in the previous chapter, many existing regression trees have axis-parallel splits, which is a major restriction since the input dimensions usually have a complex dependency among each other. Considering the linear correlations among them, oblique splits deliver more compact and intelligible models. To compute oblique splits typically a two-stage splitting strategy is used: first a meaningful partitioning is found, and second this partitioning is used as input for a linear classifier which computes the linear split.

This chapter investigates different clustering strategies w.r.t. the computation of meaningful partitionings within the splitting strategy. While conventional clustering algorithms use the Euclidean neighborhood to group the data, specialized clustering algorithms are searched, which tend to group samples whose outputs are best described by linear functions. Kriegel et al. present in [KKZ09] a detailed survey over different types of clustering algorithms. Among them, correlation clustering algorithms take linear dependencies into account when partitioning the data. Even though input and output dimensions are not distinguished, the detected correlations can be used to reveal hidden causalities and build a prediction model. To this end, the applicability of correlation clustering and regression clustering approaches in the construction of regression trees is analyzed. Finally, a new clustering model is proposed which is better suited for the construction of a regression tree, as the experimental evaluation shows.
4.1 Correlation Clustering

Clustering algorithms have the task to group samples from a data set, such that the samples within the same group are similar to each other and samples from different groups are dissimilar. However, the "similarity" measure is not specified in the definition of clustering. Traditional clustering algorithms evaluate the similarity by means of the Euclidean distance. Correlation clustering assesses samples as similar if they lie on an arbitrary oriented hyperplane in the space. Intuitively this corresponds to a linear correlation between the dimensions, described by the hyperplane on which the samples lie. Hence, even if a set of samples has a low variance in the Euclidean space, they might not lie on a single hyperplane and are, therefore, not grouped together by a correlation clustering algorithm.

The local correlation analysis is in most approaches performed by the principal component analysis (PCA), which computes the hyperplane best fitting a given sample set. Let $X$ be the matrix containing the centered samples, then PCA first calculates the covariance matrix:

$$
\Sigma_X = \frac{1}{n} X^\top X,
$$

and then decomposes it into an eigenvector matrix $V$ and a diagonal matrix $\Lambda$ containing the eigenvalues: $\Sigma_X = V \Lambda V^\top$. The eigenvectors span a new coordinate system, in which the dimensions are not linearly correlated any more, and $\Lambda$ is the covariance matrix of the data when represented in this new coordinate system.

While ordinary least squares (OLS) concentrates on describing the causalities of the inputs onto the outputs, PCA treats all dimensions equally, and detects linear causalities among them. Both approaches compute a hyperplane which minimizes the squared errors. However, the difference lies in how this error is defined. In linear regression the vertical offsets, i.e. parallel to the output axis, from the hyperplane are minimized, while PCA minimizes the perpendicular offsets, i.e. perpendicular to the fitted plane. This idea is also illustrated in Figure 4.1, where for the samples OLS and PCA is consid-
4.1. Correlation Clustering

Figure 4.1: Model identification: (Left) with ordinary least squares (Right) with principal component analysis

...er separately. In the left image, one dimension is chosen to be the output and a linear regression plane is fitted. The magnified view in the circle shows that the error of a sample is the difference between its output value and the prediction value on the plane. In the right image the two eigenvectors with the highest values are spanning a plane, describing the data. The magnified view in the circle shows that error of a sample is its distance to the plane. Achtert et al. discuss in [ABK+06] the similarity between PCA and OLS, and how a quantitative regression model can be derived from correlation cluster identified by PCA.

However, PCA detects correlations which hold for a dataset as a whole. Often such linear correlations do not hold for the whole data set. Different local subsets may exhibit different correlations. The challenge is, thus, to identify clusters with a high correlation among the contained points.

Arbitrarily Oriented Projected Cluster Generation

Aggarwal et.al. introduced in [AY00] the first correlation clustering algorithm, ORCLUS (arbitrarily ORiented projected CLUSter generation), which combines k-Means with PCA.

By making use of the PCA, ORCLUS locally identifies hyperplanes in which the correlation between the attributes is minimized. The eigenvectors with a high corresponding eigenvalue define a hyperplane which best
describes the data points, since they describe the directions with the highest variance. The eigenvectors with a low corresponding eigenvalue describe the directions with lowest variance, in which clusters are identified. Let \( \mathcal{E} = \{ e_1, ..., e_\kappa \} \) be a set of \( \kappa \leq d \) orthonormal vectors spanning a \( \kappa \)-dimensional hyperplane. The projection of a sample \( x \in \mathbb{R}^d \) onto \( \mathcal{E} \) is \( x^\mathcal{E} = x^T \cdot [e_1 ... e_\kappa] \), where \([e_1 ... e_\kappa]\) is a matrix containing the vectors \( e_i \) as columns. The projected energy \( R(c_j, \mathcal{E}) \) of a cluster \( c_j \) onto the hyperplane \( \mathcal{E} \) is defined as the sum of the distances of all \( x \in c_j \) to the projected cluster centroid \( x^\mathcal{E}_j \):

\[
R(c_j, \mathcal{E}) = \frac{1}{|c_j|} \sum_{x \in c_j} \| x^\mathcal{E} - x^\mathcal{E}_j \|_2.
\]

The goal of ORCLUS is to discover clusters with a small projected energy in the identified subspaces. The necessary subspace \( \mathcal{E} \) with a low projected energy \( R(c_j, \mathcal{E}) \) for a set of points \( c_j \) is obtained by computing the PCA of the corresponding covariance matrix \( \Sigma_{c_j} \). The \( d - \kappa \) eigenvectors with the smallest corresponding eigenvalue are selected as \( \mathcal{E} \). Since along these directions the variance is least spread, they define the subspace with the lowest projected energy for \( c_j \).

ORCLUS is a hierarchical clustering algorithm, with the projected energy as distance measure, influencing by this the shape of the resulting clusters. ORCLUS has as input parameter the desired number of clusters \( k \). It starts with a high number \( k_0 \) of random samples as cluster representatives: \( C^{(1)} = \{ \{x_1\}, \{x_2\}, ..., \{x_{k_0}\} \} \), where \( C^{(t)} \) denotes the clustering in the \( t \)-th iteration. In order to obtain clusters which are best described by a linear equation, clusters are incrementally merged. In the \( t \)-th iteration \( C^{(t)} \) becomes \( C^{(t+1)} \) by merging the cluster pairs with the most similar least spread directions, i.e. the cluster pairs with minimal projected energy:

\[
(i, j) = \arg\min_{(i,j): c_i \cup c_j \in \mathcal{E}} R(c_i \cup c_j, \mathcal{E}_{c_i \cup c_j})
\]

In each iteration the number of clusters is reduced by a factor of \( \alpha \), specified by the user. After one iteration, each point \( x \) is assigned to cluster \( c_j \) for
4.1. Correlation Clustering

which the distance of its projection to the projected centroid is minimal:

$$\arg\min_{c_j \in C} (x_j^e - \bar{x}_j^e).$$

ORCLUS is also able to identify clusters in subspaces up to a dimensionality of \( l \), defined by the user. In the beginning, the clusters have the full dimensionality of the data space, and then the cluster dimensionality is slowly reduced with each iteration. The dimensionality reduction step \( \gamma \) for each iteration and the relative cluster reduction \( \alpha \) are related as follows:

$$\log_{\frac{1}{k_0}} \frac{k_0}{k} = \log_{\frac{1}{\gamma}} \frac{d}{l}. $$

ORCLUS focuses on the least spread principal axes in which the points form dense clusters. For building a linear model the remaining principal axes with a high variance are considered, since they describe the desired hyperplanes.

Clusters of Correlation Connected Objects

Another approach for correlation clustering is 4C, proposed in [BKKZ04]. This algorithm is a density based clustering which identifies correlations by adopting the quadratic form distance (QF) [FBF+94]. Since for 4C a cluster is defined as a dense region of data points, the number of clusters is automatically determined by the algorithm.

While the Euclidean distance weights the distance in all dimensions equally and does not consider the distances between different dimensions, the QF contains the weights for the distances in all pairwise dimensions in a similarity matrix \( A \in \mathbb{R}^{d \times d} \). Based on this similarity matrix, the QF between two samples \( x_1 \) and \( x_2 \) is defined as:

$$QF(x_1, x_2) = \sqrt{(x_1 - x_2)^T A (x_1 - x_2)}$$

The main idea of 4C is to compute a local PCA around each point \( x \) and use it to derive a similarity matrix \( A \) from the covariance matrix \( \Sigma: A = V \cdot \hat{\Lambda} \cdot V^T \), where \( \hat{\Lambda} \) is obtained from \( \Lambda \) by replacing the eigenvalues \( \lambda_j \) higher than
a threshold with its inverse $\frac{1}{\lambda_j}$, and the rest with zero. $\hat{\Lambda}$ has therefore low weights for the distances along the hyperplane describing the samples around $x$, and high weights for the orthonormal directions.

**Clustering in Arbitrary Subspaces Based on the Hough Transform**

Achtert et al. introduced CASH in [ABD⁺08], a correlation clustering based on the Hough transformation. The idea is to map each sample onto a trigonometric function in the parameter space, representing all possible subspaces in which this sample is contained. The intersection of the functions of two samples indicates a line through the samples in the original space. The aim is therefore to identify intersection points of many functions, representing a hyperplane on which many samples lie. In order to allow some degree of noise and for efficiency reasons, the parameter space is divided in grid cells and the algorithm identifies the cells through which many functions pass.

**Applicability of Correlation Clustering for the Split Computation**

None of the above mentioned clustering algorithms consider the problematic of cluster separability from Chapter 3, since these were not designed for data with input and output dimensions.

A problem of 4C is that it groups samples based on their density. Therefore, it might happen that samples roughly lying on the same hyperplane are not grouped together if they are far enough apart. The main problem of CASH is that it groups samples together which lie far apart from each other, and tends to generate many overlapping clusters. ORCLUS seems to be best suited for the construction of regression trees.

An example of the clustering results delivered by 4C and CASH is illustrated in Figure 4.2. Since the curvature of the underlying function is low and the measurements are dense in one input dimension and sparse in the other input dimension, 4C prefers to group all points only according to sparse dimension, even though a split along the dense input dimension would deliver two planes fitting the data much better. CASH deals mainly with the problem of cluster separability. Moreover, since 4C and CASH do not have a user de-
4.2 Regression Clustering

Späth introduced in [Spä79] a clustering algorithm specially designed for the regression case, which is based on the k-Means clustering algorithm. The task is to find $c_1, \ldots, c_k$ disjoint clusters, i.e. $c_j \cap c_e = \emptyset$ for $j \neq e$, with cor-

Figure 4.2: (Left) 4C clustering. (Right) CASH clustering

Figure 4.3: ORCLUS clusterings with four and six clusters

fined number of clusters, they are not directly applicable for the construction of regression trees. ORCLUS on the other hand, has delivered better results, as shown in Figure 4.3, and since the number of clusters is user defined, it can be straight-forwardly applied for the construction of regression trees.

4.2 Regression Clustering
responding linear equations defined by the regression coefficients $\beta_1, \ldots, \beta_k$, such that the sum of residual squares over the entire data set is minimized:

$$
\sum_{j=1}^{k} \sum_{(x_i, y_i) \in c_j} (y_i - x_i^T \beta_j)^2,
$$

Since the global optimum is expensive to compute, the exchange method applied in k-Means is employed, which is proved to converge in a local optimum. This clustering approach is summarized in Algorithm 5.

**Algorithm 5 Regression Clustering**

**Input:** $k, T$

1: compute a random partitioning of the samples: $c_1^{(0)}, \ldots, c_k^{(0)}$

2: compute the linear regression coefficients $\beta_1^{(0)}, \ldots, \beta_k^{(0)}$ for each cluster

3: $t = 0$

4: repeat

5: assign sample $(x_i, y_i) \in T$ to cluster $c_j^{(t+1)}$ if

$$
(y_i - x_i \beta_j^{(t)}) < (y_i - x_i \beta_e^{(t)}), \quad \forall j \neq e.
$$

6: compute for $c_1^{(t+1)}, \ldots, c_k^{(t+1)}$ the regression coefficients $\beta_1^{(t+1)}, \ldots, \beta_k^{(t+1)}$

7: $t = t + 1$

8: until no samples are reassigned

**Output:** $c_1^{(t)}, \ldots, c_k^{(t)}$ together with $\beta_1^{(t)}, \ldots, \beta_k^{(t)}$

The disadvantage of this algorithm is that it only considers the residual errors during partitioning, ignoring the distances in the original space and therefore also the separability of the clusters in the input space. The number of clusters is user defined, and this regression clustering (RC) can be thus easily used for the construction of regression trees.

Another approach which combines clustering and regression was proposed by Torgo in [Tor03]. In a first stage the data is clustered and in a
4.3. Model Clustering

In order for a clustering to be well suited for the construction of regression trees, it should compute linearly correlated clusters which are separable in the input space. Correlation and regression clustering, as well as bounded error approaches, are appropriate for the first requirement, no solution has been proposed for the second one. After the partitioning, the split is computed by means of a linear classifier. However, the quality of its output heavily depends on the separability of the clusters.

The novel MoClus (Model Clustering) is introduced here, which is a clustering designed to compute meaningful splits for regression trees. An important property of MoClus is that it has exchangeable data models, which are used during the clustering and as leaf models. In the following only linear models are considered: $M \in \{\text{OLS}, \text{RR}, \text{PLS}\}$, introduced in Chapter 2, but MoClus would also work with different kernels and polynomial functions. MoClus is an agglomerative clustering procedure, it starts with a high number of clusters and progressively merges them. Two main components are required when this course of action is followed: a measure for assessing the quality of a cluster, and a strategy for assigning points to clusters. As quality measure the mean squared error (from Equation 3.2) is used, and a assigning step is designed to ensure the cluster separability.

MoClus, summarized in Algorithm 6, takes the number of random seeds $k_0$ and final number of clusters $k$ as input parameters, together with $\alpha$, which indicates how many clusters are merged in each iteration. $T$ is the training
set, and $\mathcal{M}$ the type of model used for the clusters. In the initialization (lines 1-6) $k_0$ seeds are randomly chosen, each sample $o = (x, y) \in T$ is assigned to the closest seed, according to the $L_2$ norm in $\mathbb{R}^{d+1}$, and the mean for each cluster $C_i$ in $\mathbb{R}^{d+1}$ is used as a constant cluster model $f_i$.

After the initialization, several iterations are computed until the desired number of clusters is reached. In each iteration samples are assigned to clusters (cf. line 8-18), and then clusters are merged (cf. line 19). The assignment happens in two steps. First, each sample $(x, y) \in T$ is assigned to the cluster $C_i$ which best predicts the output: $\arg\min_{1 \leq i \leq k} c_i(f_i(x) - y)^2$. Then the projection $IC_i$ of each cluster $C_i$ onto the input space is computed, and a second assignment takes place in the input space. A linear separation of the clusters $IC_i$ is performed with LDA, and each object is assigned to a cluster according to the decision function $\delta_{LDA}$ (line 16). Finally, a model is learned for each newly built cluster (line 17). This ensures that there are no big overlaps between the clusters in the input space, and a linear separator can be better trained. In each iteration the number of clusters is reduced by a factor of $\alpha \in (0, 1)$.

Algorithm 7 describes how the number of clusters is reduced from $k$ to $k'$ in one iteration. All possible cluster merges $C_{ij}$ are computed, together with the corresponding prediction model $f_{ij}$. Among these possibilities the clusters merges with the lowest mse $(f_{ij}, C_{ij})$ are actually performed.

Note that the choice of $\mathcal{M}$ influences whether MoClus is a full space or a subspace clustering algorithm. If $\mathcal{M} \in \{PLS, RR\}$, then MoClus performs subspace clustering. Since ridge regression (RR) shrinks the regression coefficients of unimportant input dimensions towards zero, the identified subspace is an axis parallel one. On the other hand, partial least squares (PLS) projects the input dimensions onto an arbitrary oriented subspace. In this case, MoClus is related to the subspace correlation clustering algorithm ORCLUS. While ORCLUS describes the clusters by eigenvectors from the principal component analysis, and systematically reduces the cluster dimensionality in each iteration until a user defined value is reached, MoClus describes the clusters by a regression model and determines the intrinsic dimensionality of each cluster independently from the training data. For RR, the regu-
Algorithm 6 MoClus

**Input:** \( k_0 \in \mathbb{N}, k \in \mathbb{N}, \alpha \in [0, 1], \mathcal{T}, \mathcal{M} \)

1: select \( k_0 \) random seeds \( \{s = (x_i, y_i)\}_{i=0}^{k_0} \) from \( \mathcal{T} \)

2: for \( i \leftarrow 1 \ldots k_0 \) do

3: \( C_i = \{ o \in \mathcal{T} \mid L_2(o, s_j) > L_2(o, s_i), 1 \leq j \leq k_0 \} \)

4: \( f_i \leftarrow \frac{1}{|T|} \sum_{o \in C_i} y_l \)

5: end for

6: \( k_c = k_0 \)

7: repeat

8: assign the sample \( o_l \) to cluster \( C_i \): \( \arg\min_{1 \leq i \leq k_c} \left( \sum_{l \leq |T|} (f_i(x_l) - y_l)^2 \right) \), \( 1 \leq l \leq |T| \)

9: for \( i \leftarrow 1 \ldots k_c \) do

10: compute \( \mathcal{I}C_i \), the projection of \( C_i \) onto the input space

11: compute \( \mu_i \), centroid of \( \mathcal{I}C_i \)

12: end for

13: compute \( \Sigma \) (cf. Eq. 3.6)

14: compute the decision function \( \delta_{LDA} \) (cf. Eq. 3.7)

15: for \( i \leftarrow 1 \ldots k_c \) do

16: \( C_i = \{ o \in \mathcal{T} \mid \delta_{LDA}(o) = i, 1 \leq j \leq k_c \} \)

17: \( f_i \leftarrow \mathcal{M}(C_i) \)

18: end for

19: \( \text{MERGE}(C_1, \ldots, C_{k_c}, [\alpha k_c]) \)

20: \( k_c = \lfloor \alpha k_c \rfloor \)

21: until \( k_c = k \)

**Output:** \((C_1, f_1), \ldots, (C_k, f_k)\)

Regularization factor \( \lambda_R \) determines the amount of shrinkage, which, depending on the training data it can remove more or less dimensions. For PLS an appropriate \( \kappa < d \) is chosen such that at least a given fraction \( \varphi \in [0, 1] \) of the explainable output variance is covered:

\[
 r = \min_{1 \leq j \leq d} \frac{\sigma_0^2 - \sigma_j^2}{\sigma_0^2 - \sigma_d^2} \geq \varphi, \quad (4.1)
\]
Algorithm 7 MoClus: merge

1: function MERGE(C₁, ..., Cₖ, k′)
2:     compute all Cᵢⱼ = Cᵢ ∪ Cⱼ, 1 ≤ i ≠ j ≤ k
3:     repeat
4:         fᵢⱼ ← M(Cᵢⱼ), 1 ≤ i ≠ j ≤ k
5:         rᵢⱼ = mse(fᵢⱼ, Cᵢⱼ)
6:         perform best merge Cᵢᵢ′, Cⱼⱼ′ : (i′, j′) = min₁≤i,j≤k rᵢⱼ
7:         Cᵢ′ ← Cᵢᵢ′, Cⱼ′ ← ∅
8:     update all Cᵢⱼ and Cᵢ,j′, 1 ≤ i, j ≤ k
9:     k = k − 1
10:     until k = k′
11: return (C₁, f₁), ..., (Cₖ′, fₖ′)
12: end function

where σ₀ is the variance in the output dimension in the original data, and σᵣ is the variance in the output dimensions after the j-th iteration, i.e. the variance of the residuals after a linear model has been fitted in j dimensions. The final dimensionality of a cluster is therefore depending on the data.

Computational Complexity of MoClus

The complexity of MoClus depends on the initial number of clusters k₀ and the merging factor α. In a single merge step \(\frac{k_0(k_0-1)}{2}\) cluster pairs are computed and their models built. The complexity of building models is:

- **OLS**: The computation of the regression coefficients β from Equation 2.3 involves the inversion of matrix \(X^\top X\), with a complexity of \(O(d^3)\), and the matrix multiplications, with a complexity of \(O(nd^2)\). Hence, the total complexity for n samples is \(O(d^3 + nd^2)\).

- **RR**: Similar to OLS, the total complexity for n samples is \(O(d^3 + nd^2)\).

- **PLS**: In Algorithm 2 is the PLS computed in maximum d iterations, and each iteration has a complexity of \(O(nd)\) for \(Y \in \mathbb{R}^n\). The total complexity for n samples is \(O(nd^2)\).
4.4 Experiments

The complexity of building models is therefore $O(d^3 + nd^2)$. Since each sample belongs to a single cluster and is thus maximally involved in $k_c - 1$ pair computations, the cost for building the models for $\frac{k_c(k_c-1)}{2}$ pair clusters is $O(k_c(d^3 + nd^2))$, which is the complexity of Algorithm 7. MoClus starts with $k_0$ clusters and reduces them in each iteration with a fraction of $\alpha$ until $k$ clusters are left. Therefore, it holds that $k = \alpha^t k_0$, where $t$ is the number of merging iterations. The number of iterations performed by MoClus is, thus, $t = \lceil \log_\alpha \frac{k}{k_0} \rceil$. Hence, the total complexity is $O((d^3 + nd^2) \log^2 \frac{k}{k_0})$.

4.4 Experiments

This section investigates the applicability of clustering algorithms for the computation of splitting rules compared to MoClus, with the focus on the prediction accuracy. As a baseline $k$-Means [M+67] is used, which is a simple and effective clustering algorithm grouping the samples according to their Euclidean neighborhood. Other clusterings drawn for comparison are ORCLUS, which groups linearly correlated samples, and regression clustering, which groups the samples such that they are well described by regression models. For MoClus the linear models $\mathcal{M} \in \{\text{OLS}, \text{PLS}, \text{RR}\}$ were considered, but since the RR and the OLS models had very similar results, only the ones for OLS were plotted.

For this comparison, a binary regression is computed by iteratively dividing the data in two clusters and training a linear separator, namely linear discriminant analysis (LDA) and support vector machines (SVM) [CV95, Vap95]. In order to finish the recursive splitting, a stopping criterion is required. To decide whether a split of a training set $\mathcal{T}$ in $\mathcal{T}_1$ and $\mathcal{T}_2$ is meaningful or not, its relative error reduction is checked:

$$\frac{|\mathcal{T}_1| \text{mse}(f_1, C_1) + |\mathcal{T}_2| \text{mse}(f_2, C_2)}{\text{mse}(f, \mathcal{T})} \leq \tau_{\text{red}},$$

where $f = \mathcal{M}(\mathcal{T})$ and $f_i = \mathcal{M}(\mathcal{T}_i)$, $i \in \{1, 2\}$. In the following experiments $\tau_{\text{red}} = 1.2$ was used.
Finally, the results are compared to the ones obtained by existing regression trees with oblique splits SECRET and PHDRT.

The experiments were performed on real-world datasets from the UCI Machine Learning repository [BL13]. As artificial datasets the Fried dataset also used in [Fri91], the 2D Planes also used in [BFSO84], and the MV Delve dataset used in [IGD11] are considered. All used datasets and their characteristics are summarized in Table 4.1. W.l.o.g. the data is z-score normalized.

All evaluation experiments were conducted using a single-threaded Java-based implementation, and were run on Intel Core-2 Duo with 3 GHz and with 4G memory available to each process. The accuracy of a predictor is evaluated by means of the relative error:

\[
RE(T) = \frac{\text{mad}_{\text{regTree}}(T)}{\text{mad}_{\overline{y}}(T)} = \frac{\sum_{(x,y) \in T} |y - f(x)|}{\sum_{(x,y) \in T} |y - \overline{y}|},
\]

where mad is the mean absolute deviation of the predicted value from the
4.4. Experiments

real one. Hence, this measure is the relative improvement of the prediction accuracy obtained over just using the average in the output dimension. Hence, a RE of 1 means that there is no improvement over the naive approach, and a RE below 1 corresponds to an improvement.

The results for the artificial data, plotted in Figure 4.4, show that the accuracies of MoClus and PHDRT are very close to each other, and k-Means and ORCLUS have the poorest results. The results on the real world data sets are plotted in Figure 4.5. For the lower dimensional real data sets OLS models are best suited for MoClus. k-Means delivers separable clusters and
contributes to a prediction quality comparable to that of ORCLUS and RC, despite the fact that it does not consider linear correlations in the clustering process. However, MoClus performs similarly to the other clustering algorithms. For the higher dimensional data sets, MoClus with PLS models has a clear advantage over the other clustering approaches.

Regarding the linear classifier, both LDA and SVM were considered. For lower dimensional data sets there was no significant difference in the accuracies. Since the computation of LDA is far less complex, it was used for the lower dimensional data sets. For high dimensional data sets though the SVM contributed to better accuracies, and was preferred to LDA.

Applications

MoClus was applied for building models for the experimental measurement presented in Section 1.1. Some exemplary images of three dimensional subsets of measurements are plotted in Figures 4.6 - 4.7. For the SPRAY and DIESEL dataset, few models are sufficient to describe the regression surface. The WIND TURBINE dataset has a more complex surface, and more submodels were necessary for a good approximation. The plots show that neighboring samples which are linearly correlated are grouped together, creating good approximations of the regression surfaces, with only few submodels.
Figure 4.7: Application of MoClus on the SPRAY and DIESEL datasets
4.5 Conclusion

Many regression trees have axis parallel splits, which is a major restriction, since the input dimensions usually have a complex dependency among each other. However, they have an advantage in the higher dimensional spaces, since they can choose to ignore less important input dimensions. Considering linear correlations among the input dimensions and computing oblique splits more compact and intelligible models are obtained. However, in this case the dimensionality reduction becomes an issue. Different clustering approaches were considered in this chapter for the split computation of regression trees, which take into account linear correlations within the data. However, by only checking whether the samples lie on the same hyperplane or not and disregarding the positions in the input space, clusters emerge which are not separable in the input space. If used as input for linear classifiers, unsatisfactory results are obtained.

The novel MoClus is introduced which considers both aspects when clustering the samples. Moreover, it is formulated such that it works with arbitrary models. While in lower dim input spaces OLS models are sufficient, in higher dimensional cases the subspace PLS models have a clear advantage, being described by few arbitrarily oriented directions, which contain most of the information. Therefore, accurate and compact prediction models can be computed in high dimensional spaces. MoClus will be used in the next chapter for the regression tree AMT on streaming data.
Chapter 5

Adaptive Model Tree on Data Streams

With an ever-growing availability of data streams the interest in and need for efficient techniques dealing with such streams is increasing. A major challenge in this context is the accurate online prediction of continuous values in the presence of concept drift. In this chapter the novel Adaptive Model Tree (AMT) is introduced, designed to incrementally learn from the data stream, adapt to the changes, and to be able to perform real time accurate predictions at any time. For the partitioning within the split computation, the MoClus algorithm is used. For the previously discussed linear models, efficient and numerically stable online updates are discussed. Compared to state of the art streaming regression tree FIMT, AMT computes oblique splits, delivering a more compact and accurate model, and in some cases even more efficient.

5.1 Introduction

Recently, data streams have received a growing attention in a multitude of practical applications, e.g., scientific and engineering experiments, surveillance of systems and other dynamic environments. The data is represented as data sequences continuously arriving at high speed. Mining data streams
poses several challenges. While data streams are possibly endless, the available physical memory is limited. Hence algorithms can only scan the data once; the samples are processed in arriving order and then discarded. Only a limited amount of samples can be buffered and processed later. Another restriction is that the processing time of a single sample is limited. Since the data is continuously arriving, taking more time for one sample means dropping out other samples and losing information.

The main goal is to build a prediction model of the underlying process, which can be used to perform real time accurate predictions. However, the data arrives continuously and the underlying model often changes over time, concepts may shift or disappear, and new ones may appear. Hence, it is important to detect such concept drifts and maintain an up to date model of the data.

While several batch algorithms for constructing model trees have been proposed in the literature, the literature regarding incremental model trees is limited. In [PS05] a first incremental model tree was proposed, which is not suited for the streaming context. The state of the art model tree on streaming data is FIMT (introduced in [IGD11]), which allows only axis parallel splits and models in the full space.

In this chapter the new Adaptive Model Tree (AMT) for streaming data is introduced. The split strategy employs MoClus for the split decision, and trains a linear classifier to define the split boundary. Since AMT is not restricted to axis parallel splits, but allows oblique splits, it achieves good prediction accuracy with a more compact model. After a discussion of the related work in Section 5.1.1, the structure of the AMT is presented together with its efficient updates, followed by a detailed evaluation on both synthetic and real datasets.

5.1.1 Related Work

While a lot of research has been conducted towards building models for the prediction of categorical labels, little effort has been spent in the area of predicting continuous values. The first step towards an incremental con-
5.1. Introduction

The construction of regression trees was made by Potts et al. in [PS05]. In [FMN11] a windowing strategy for regression trees is proposed. Ikonomovska et al. introduced in [IGD11] a fast incremental model tree (FIMT) for learning from time-changing data streams, which is the state of the art regression tree for streaming data.

**Incremental Learning of Linear Model Trees**

In [PS05] two axis parallel batch splitting rules RD and RA are discussed and extended to the incremental case. The batch-RD splitting rule simply tests for each input dimension for constant number of potential splits the null hypothesis that the underlying function is linear, by checking whether the term:

$$ F_{RD} = \frac{(\text{mse} - \text{mse}_L - \text{mse}_R)(n - 2d)}{(\text{mse}_L + \text{mse}_R)d} \left(\frac{\text{mse}}{n - d} - \frac{\text{mse}_L + \text{mse}_R}{n - 2d}\right) $$

is distributed according to Fisher's $F$ distribution with $d$ and $n - 2d$ degrees of freedom. The split with the least probability of the null hypothesis is then chosen and performed. The RA splitting rule is the one used by SUPPORT and GUIDE (see Chapter 3.2), based on the sign of the residuals when a linear model is fitted through the nodes’ entire set of samples.

The tree construction algorithm stops splitting a leaf, when its contribution in the overall tree accuracy is below a threshold. This accuracy is measured as:

$$ \frac{1}{\sigma_y} \left(\frac{\text{mse}}{n - d} - \frac{\text{mse}_L + \text{mse}_R}{n - 2d}\right). $$

For the incremental update of the linear model in a leaf with an incoming sample, the recursive least squares (RLS) algorithm [Kal60] is used. Assuming that the vector $\beta_n \in \mathbb{R}^d$ contains the regression coefficients for the samples with input matrix $X \in \mathbb{R}^{n \times d}$ and output vector $Y \in \mathbb{R}^n$, and let $P_n = (X^TX)^{-1}$. For a new arriving sample $(x, y)$, the regression coefficients are updated as follows:

$$ \beta_{n+1} = \beta_n + \frac{P_n x(y - x^T \beta_n)}{1 + x^T P_n x}, $$
and
\[ P_{n+1} = P_n - \frac{P_n x x^T P_n}{1 + x^T P_n x}. \]
Hence, the RD splitting rule can be easily and loss-free adapted for an incremental computation. With each incoming sample, for each input dimension and each leaf node the corresponding linear model is updated and for each possible split, the two linear models are also updated, the best split computed and the significance of its improvement calculated.

The RA splitting rule is also adapted to the incremental computation. With each incoming sample, for each input dimension the linear model is updated, and the sign of the new sample determined according to it. Then, the mean and standard deviation values required for the Student-t test are incrementally updated and the tests performed.

Unfortunately, with a growing tree size the number of splits to be updated increases and thus the training time of new incoming sample also increases, making this approach infeasible for the streaming context, as shown in [IGD11].

Fast Incremental Model Tree

FIMT is an efficient incremental algorithm able to learn a prediction model from streaming data. For efficiency reasons it computes only axis parallel splits, which maximizes the reduction of the standard deviation in the output dimension:
\[
\text{red}_\sigma(h_a) = \sigma(T) - \frac{|T_1|}{|T|} \sigma(T_1) - \frac{|T_2|}{|T|} \sigma(T_2),
\]
where \( h_a \) is the split on the input dimension \( j \), partitioning \( T \) into \( T_1 \) and \( T_2 \), and \( \sigma(T) \) is the standard deviation of the output, which can be computed incrementally:
\[
\sigma(T) = \sqrt{\frac{1}{|T|} \sum_{i=1}^{|T|} (y_i - \bar{y})^2} = \sqrt{\frac{1}{|T|} \left[ \sum_{i=1}^{|T|} y_i^2 - \frac{1}{|T|} \sum_{i=1}^{|T|} y_i^2 \right]}. 
\]
A split is performed, if the ratio

\[ \hat{r} = \frac{\text{red}_\sigma(h_j)}{\text{red}_\sigma(h_j')} \]

of the best and second best estimated split quality is lower than 1. To this end the Hoeffding bound is employed [Hoe63], which states that the probability that the mean of the observed variables deviates from the real mean of the distribution by at most \( \epsilon \) with a confidence of \( 1 - \delta \). The range \( \epsilon \) is computed as:

\[ \epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}. \]

where \( R \) is the range of the observed random variables, i.e. ratios. If \( \hat{r} + \epsilon < 1 \), than the axis parallel split in the \( j \)-th dimension is implemented. To efficiently determine for each dimension the best split, a special data structure is proposed, namely the Extended Binary Search Tree (EBST), storing the attribute values for each sample. To conserve memory, FIMT periodically checks memory consumption. When a user-defined threshold is crossed, the leaves with the least error rate are removed and the corresponding EBST deleted.

FIMT uses perceptrons as linear models. Although initially introduced in [Ros58] for binary classification, perceptrons can be used as linear prediction function, expressing the output as a linear combination of the inputs:

\[ \hat{y} = w_0 + \sum_{i=1}^{d} w_j x_j. \]

The weights \( w_i \) are randomly initialized and then incrementally updated according to the Widrow-Hoff rule [WH+60]:

\[ w_i' = w_j + \eta (y_0 - \hat{y}) x_j, \quad \forall j \in \{1, ..., d\}, \]

where \( \eta \) is the learning rate. With a small \( \eta \) the learning process is slow, but the predictor becomes accurate. For higher learning rates, the learning process is faster but the predictor less stable.
Drift Detection

The underlying model in a data stream is changing in time, hence it is important to detect this changes and replace the obsolete models or submodels. Gamma et al. investigated in [GSR09] the continuous performance assessment of stream learning algorithms and drift detection. The authors defend the use of the prequential error with a forgetting (aging) mechanism for monitoring the prediction accuracy of models. At the arrival of the $i$-th sample, its prequential squared error is computed before including it in the model:

$$\text{pse}(x_i) = \left( f^{(i-1)}(x_i) - y_i \right)^2.$$  

The prequential error is then:

$$E_i = \text{pse}(x_i) + \gamma E_{i-1},$$  \hspace{1cm} (5.1)  

where $\gamma$ is a fading factor. The smaller $\gamma$ is, the sooner old errors fade.

After each new insertion it is checked if the underlying model has changed, causing a considerably increase of the error. This is done by using the Page-Hinckley test (PH) (first introduced in [Pag54]). The PH tracks the difference between the current accumulated error and minimum accumulated error:

$$PH_T = m_T - M_T,$$

with

$$m_T = \sum_{i=1}^{\lvert T \rvert} (\text{pse}(x_i) - \bar{\text{pse}}(x_i) - \epsilon_{PH}),$$

$$\bar{\text{pse}}(x_i) = \frac{1}{\lvert T \rvert} \sum_{i=1}^{\lvert T \rvert} \text{pse}(x_i),$$

and $M_T = \min_{i \in \{1, \ldots, |T|\}} \{m_i\}$. The parameter $\epsilon_{PH}$ specifies the minimal magnitude of changes to be considered. A change is detected when $PH_T$ is higher than a specified threshold $\tau_{PH}$. It has been shown in [SG09] that PH has both a small delay time until drift detection, and it is less time- and memory-
In every node, disregarding whether it is an inner node or a leaf, the pse of the subtree is computed by bottom-up back-propagating the error of the leaves. After each insertion at defined intervals it is checked whether a change took place. When a change has been detected in a node, an alternative model is grown in the background for the whole subtree. With the Q statistics the performance (in terms of the accumulated prequential error cf. Equation 5.1) of these two models is periodically compared:

$$Q_i^\gamma = \log \frac{E_i^b}{E_i},$$

where $E_i$ is the accumulated error at the arrival of the $i$-th training sample, and $E_i^b$ the one for the model grown in the background. As soon as the background model is more accurate ($Q_i^\gamma > 0$), it replaces the used model. Otherwise it is discarded.

This drift detection strategy has been used by FIMT and also in the following by AMT.

### 5.2 Adaptive Model Tree

This section describes the new adaptive model tree (AMT) and how it behaves with an incoming data stream. The AMT is built as a binary model tree: the training data is partitioned by MoClus in two clusters, and a linear split is trained. This procedure is recursively repeated until the stop criteria is fulfilled:

$$\frac{|T_1|}{|T|}\text{mse}(f_1, C_1) + \frac{|T_2|}{|T|}\text{mse}(f_2, C_2) = \text{mse}(f, T) \leq \tau_{\text{red}},$$

(5.2)

The adaption of the tree is two-sided: on the one hand, AMT adapts with the continuous arrival of samples generated by the same underlying function and increases its prediction accuracy, and on the other hand, it uses the earlier mentioned mechanism to detect drifts in the underlying function and to adapt itself to the change.
At the arrival of a new training sample \((x, y)\) the corresponding leaf model is searched, in order to be updated. If the decision about the path to be followed is based only on the input \(x\) available information is neglected, which might have been useful. Hence, in each inner nodes two splits are tracked: one in the input space, for classifying test samples, and one in the full space, for classifying training samples. With the arrival of each new training sample, not only the leaf model is updated, but also all splits in the inner nodes on the path. An example is illustrated in Figure 5.1: the two models, green and blue, are separated by \(\delta_x\) in the input space and by \(\delta_{xy}\) in the full space. The newly arrived red samples fit better to the green model, and according to \(\delta_{xy}\) are also assigned to it (see left image). With this new information the green model is updated, as well as both splits \(\delta_x\) and \(\delta_{xy}\) (right image).

In order to allow the model tree to grow online, a window of \(\tau_n\) training samples is stored in each leaf. As soon as the window is full, the samples are divided into train and test set, and a split is induced. If the split turns out to be meaningful (cf. Equation 5.2), then two leaf nodes are created and an additional classifier learned for the full space split. Otherwise, the collected samples are discarded and the collection starts again. Besides growing, pruning plays an important role. When the available memory is nearly exhausted, the leaves with the smallest prequential squared error are removed, like in FIMT. To this end, efficient incremental updates for both the linear models and the linear splits are required, which are subject of the next section.
5.3 Incremental Updates

Without loss of generality z-score normalized data is required. For a batch dataset, this is easily achieved, by computing the mean $\mu$ and standard deviation $\sigma$, and then each sample $s$ is normalized as: $s' = \frac{s - \mu}{\sigma}$. For a continuously arriving stream, the means and standard deviations for each variable must be updated. The mean of $n$ samples is updated at the arrival of the $(n + 1)^{th}$ sample $s$ as:

$$\mu^{(n+1)} = \frac{n}{n+1} \mu^{(n)} + \frac{1}{n+1} s,$$

and the standard deviation:

$$\sigma^{(n+1)} = \sqrt{\frac{1}{n+1} \left( \sum_{i=1}^{n+1} s_i^2 - \frac{1}{n+1} \left( \sum_{i=1}^{n+1} s_i \right)^2 \right)}.$$

Note that in order to perform online updates of the mean and standard deviation, only the number of seen samples ($n$), their sum ($\sum_{i=1}^{n} s_i$) and squared sum ($\sum_{i=1}^{n} s_i^2$) needs to be stored.

5.3.1 Incremental Updates of the Linear Models

Since the splits are computed with MoClus, this section considers the same linear models as presented in Chapter 3, and discusses how these can be incrementally updated.

Ordinary Least Squares

Considering the Cholesky decomposition of the covariance matrix $X^T X = LL^T$, the ordinary least squares from Equation 2.3 becomes: $LL^T \beta = X^T y$, which can be reformulated as

$$\alpha = L^T \beta$$

$$L \alpha = X^T y,$$
and solved efficiently via forward or backward substitution in $O(d^2)$. This formulation also allows for an efficient and numerically stable incremental update, since the Cholesky decomposition can be incrementally updated by using the rank-one update procedure from [GGMS74].

With the arrival of a new input vector $z$, the matrix $X \in \mathbb{R}^{n \times d}$ becomes \[
\begin{bmatrix}
X \\
z^T
\end{bmatrix} \in \mathbb{R}^{(n+1) \times d}.
\]
The matrix $X^TX$ then becomes:

\[
\begin{bmatrix}
X^Tz \\
z^T
\end{bmatrix} = X^TX + zz^T = LL^T + zz^T = \begin{bmatrix}
Lz^T \\
L^T
\end{bmatrix}
\]

Using a series of Givens rotations, the matrix $[z \ L]$ is transformed into $[0 \ L']$, and $L'$ is the new lower matrix in the Cholesky decomposition. A Givens rotation matrix $G(i, j) \in \mathbb{R}^{d \times d}$ has all elements equal zero, with following exceptions: $g_{kk} = 1$, for $k \neq i, j$, $g_{ii} = g_{jj} = c$, $g_{ij} = s$, and $g_{ji} = -s$, for $j > i$. When multiplied from the left to a matrix $A$, it transforms it such that $a_{ij} = 0$, and affects only the $i$-th and $j$-th rows of $A$. The values for $c$ and $s$ are computed as:

\[
c = \frac{a_{ii}}{\sqrt{a_{ii}^2 + a_{ji}^2}} \quad \text{and} \quad s = \frac{a_{ji}}{\sqrt{a_{ii}^2 + a_{ji}^2}}.
\]

The matrix transformation can be efficiently computed by simply updating the two rows as follows. Consider for every column $k$ the values $\tau_1 = a_{ik}$ and $\tau_2 = a_{jk}$. The new matrix elements are computed as $a_{ik} = c\tau_1 - s\tau_2$, and $a_{jk} = s\tau_1 + c\tau_2$.

The complexity of the Cholesky decomposition is $O(d^3)$, for a $(d \times d)$-matrix. The complexity of the incremental update of $L$ is $O(d^2)$, and hence more efficient than the recomputation. The components of the matrix $C = X^TX$ are computed as $c_{jk} = \sum_{i=1}^{n} x_{ij}x_{ik}$, for $\forall j, k \in \{1, ..., d\}$, and the components of the vector $v = X^Ty$ are computed as $v_j = \sum_{i=1}^{n} x_{ij}y_i$, for $\forall j \in \{1, ..., d\}$, and thus can be incrementally updated with each new arrived sample in $O(d^2)$. Hence, an OLS model can be incrementally updated in $O(d^2)$, without requiring to save all the samples. The same holds for RR.
Partial Least Squares

Considering the case of PLS (discussed in Section 2.2) with only one output dimension, the algorithm starts with \( X^{(1)} = X \in \mathbb{R}^{n \times d} \) and \( Y^{(1)} = Y \in \mathbb{R}^n \), and iteratively computes \( p_e \) (the columns of \( P \)) and the corresponding regression coefficient \( b_e \). In the \( e^{th} \) iteration, PLS performs the following steps:

1. compute the weight vector: \( w'_e \in \mathbb{R}^d \) as:

\[
w'_j e = \sum_{i=1}^{n} x_{ij} y_i, \quad 1 \leq j \leq d,
\]

and normalize it: \( w_e = w'_e / \|w'_e\| \).

2. compute the score vector \( t_e \in \mathbb{R}^n \):

\[
t_e = X^{(e)} w_e \Rightarrow t_{ie} = x_i^T w_e, \quad 1 \leq i \leq n,
\]

and its norm: \( \|t_e\| = \sum_{i=1}^{n} t_{ie}^2 \).

3. compute the loading vector \( p'_e \in \mathbb{R}^d \):

\[
p'_j e = X^{(e)\top} t_e \Rightarrow p'_j e = \sum_{i=1}^{n} x_{ij} t_{ie}, \quad 1 \leq j \leq d,
\]

and normalize its length to \( \|t_e\| \): \( p_e = p'_e / \|t_e\| \).

4. compute the corresponding regression coefficient \( b'_e \):

\[
b'_e = y^{(e)\top} t_e = \sum_{i=1}^{n} y_i t_{ie} \quad \text{and} \quad b_e = b'_e / \|t_e\|.
\]

5. compute the residual \( X^{(e+1)} \) and \( y^{(e+1)} \):

\[
X^{(e+1)} = X^{(e)} - t_e p_e \quad \text{and} \quad y^{(e+1)} = y^{(e)} - t_e b_e.
\]
As shown in [LWK87] and [Qin98], it is possible to update the regression coefficients computed by PLS, directly from an existing solution and a new sample, without the previous samples. An incremental update of PLS (iPLS), proposed in [VS00], iteratively updates $p_e$ and $b_e$, similar to PLS. Each of the mentioned steps is incrementally updated at the arrival of the $(n + 1)^{th}$ sample $(x^{(n+1)}, y^{(n+1)})$. In the $e^{th}$ iteration, iPLS performs following steps:

1. **update the weight vector** $w'_e$ from Equation 5.6:

   \[ w'_{je} = x^{(n+1)}_j y^{(n+1)}_n, \quad 1 \leq j \leq d, \]

   and $w_e = w'_e / \| w'_e \|$,

2. **update the score vector** from Equation 5.7 by adding a new element at the end:

   \[ t_{n+1,e} = x^{(n+1)\top} w_e, \]

   and update its norm $\| t_e \| += t_{n+1,e}^2$,

3. **update $p'_e$** from Equation 5.8:

   \[ p'_{je} = x^{(n+1)} t_{n+1,e}, \]

   and the loading: $p_e = p'_e / \| t_e \|$,

4. **update $b'_e$** from Equation 5.9:

   \[ b'_e = y^{(n+1)} t_{n+1,e} \quad \text{and} \quad b_e = b'_e / \| t_e \|, \]

5. **update the residuals**

   \[ x^{(n+1)} = t_{n+1,e} p_e \quad \text{and} \quad y^{(n+1)} = t_{n+1,e} b_e. \]

For iPLS all $d$ projections are computed, and only the first $r$ are used (cf. Equation 9.1) to generate $\beta$, since the intrinsic dimensionality of the model might change over time. Since each step contains a dot product computation
between vectors in $\mathbb{R}^d$ and $d$ iterations are performed for each update, the complexity of an update is $O(d^2)$. PLS can be directly built incremental, and the complexity is $O(nd^2)$ for $n$ samples.

### 5.3.2 Incremental Updates of the Linear Classifiers

There are two common approaches for defining a linear boundary between two classes: linear discriminant analysis (LDA) and support vector machines (SVM), which are consider for the proposed AMT. Although LDA is more efficient to compute, the SVM are more effective in high dimensional spaces.

#### Linear discriminant analysis

Already introduced in Chapter 3, LDA is a parametric generative approach which assumes that the classes are described by normal distributions with equal class covariance. LDA can be easily incrementally updated. The class means $\mu_i$ can be updated cf. Equation 5.3, and for the common covariance matrix from Equation 3.6 the Cholesky decomposition is performed and a rank-1 update is performed at the arrival of a new centered sample $z$. Hence, the incremental update of LDA has a complexity of $O(d^2)$.

#### Support vector machines

SVM’s linearly discriminate between two classes by training following decision function $H : \mathbb{R}^d \to \{-1, +1\}$:

$$H(x) = \text{sgn}(u(x)), \text{ with } u(x) = \sum_{i=1}^{\lfloor S \rfloor} \alpha_i c_i x^T x_i - b$$

where $S \subseteq T$ is the set of support vectors, $\alpha_i$ the Lagrange multiplier of $x_i \in S$, $c_i \in \{-1, +1\}$ the class of $x_i$, and $b$ the bias. In the linear case, the decision function can be transformed into a separating hyperplane, discarding the support vectors. Training an SVM consists of the minimization of the
following quadratic optimization problem (QP):

$$\min_{\alpha} \Psi(\alpha) = \min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{e=1}^{n} y_i y_e x_i^\top x_e \alpha_i \alpha_e - \sum_{i=1}^{n} \alpha_i,$$

subject to the constraints

$$\forall i : 0 \leq \alpha_i \leq C \quad (5.10)$$

where $C$ is a fixed maximal value for the Lagrange multipliers, and

$$\sum_{i=1}^{n} y_i \alpha_i = 0 \quad (\Leftrightarrow \sum_{y_i=1} \alpha_i = \sum_{y_i=-1} \alpha_i) \quad (5.11)$$

This is equivalent to

$$\sum_{y_i=1} \alpha_i = \sum_{y_i=-1} \alpha_i.$$

Since the size of the QP problem depends on the training set size, SVMs cannot efficiently handle large-scale training problems. The Sequential Minimal Optimization (SMO) algorithm proposed in [Pla99] efficiently finds the optimal support vectors, by decomposing this QP problem into a series of smallest possible optimization problems, involving the update of only two Lagrange multipliers. It uses the fact that the optimal solution of the QP problem is reached when the following Karush-Kuhn-Tucker (KKT) conditions are fulfilled:

$$\alpha_i = 0 \quad \Rightarrow \quad y_i u(x_i) \geq 1,$$
$$0 < \alpha_i < C \quad \Rightarrow \quad y_i u(x_i) = 1,$$
$$\alpha_i = C \quad \Rightarrow \quad y_i u(x_i) \leq 1.$$

Since the Lagrange multipliers $\alpha_i$ must obey a linear equality constraint, the smallest possible optimization problem involves two Lagrange multipliers $\alpha_i, \alpha_e$. The minimum along the direction of the linear equality constraint
from Equation 5.11 is computed and $\alpha_i$ is updated according to

$$
\alpha^{\text{new}}_e = \alpha_e + \frac{y_e(E_i - E_e)}{x_i^\top x_i + x_e^\top x_e - 2x_i^\top x_e}
$$

where $E_i = u(x_i) - y_i$ is the error of the $i$-th training example. Because of the bound constraint from Equation 5.10, $\alpha^{\text{new}}_e$ is clipped to the allowed interval. The value of $\alpha_i$ is then computed as $\alpha^{\text{new}}_i = \alpha_i + y_i y_e (\alpha_e - \alpha^{\text{new,clipped}}_e)$. To choose the order in which Lagrange multipliers $\alpha_i, \alpha_e$ are to be optimized, two heuristics were proposed in [Pla99]: one for $\alpha_i$ which defines the outer loop of the algorithm, and one for $\alpha_e$ which defines the inner loop of an iterative optimization process. The outer loop iterates through the training set. If an example violates the KKT conditions, it is eligible for the optimization. In the inner loop $\alpha_e$ is chosen such that the step width in the optimization is maximized, which is ensured by maximizing the absolute difference between the errors $|E_i - E_e|$.

In a streaming context, the optimal solution of SMO is a good starting point for the optimization at the arrival of a new sample, and therefore it is suited for incremental computations. In [BEWB05] the LASVM algorithm was introduced, as an adjustment of the SMO algorithm for streaming data. With the arrival of a new sample $z$, LASVM performs a process step, followed by a reprocess step, replacing the heuristic for choosing the next pair of Lagrange multipliers to be optimized. In the process step, a sample $x_i \in S$ is chosen as pair for $z$, such that $|E_i - E_z|$ is maximal, and $\alpha_i$ and $\alpha_z$ are updated in an SMO-optimization step. In the reprocess step a pair $x_i$ and $x_e$ is chosen from $S \cup \{z\}$, with maximal $|E_i - E_e|$, and optimized. Finally, all support vectors are removed, which have a very small probability to change.

While the result of both LDA and SVM consists of a linear splitting hyperplane in the input space, both approaches require to store more than a hyperplane in order to be able to perform incremental updates. LDA stores the covariance matrix $\Sigma$, and the means $\mu_1$ and $\mu_2$, and updates these. LASVM stores the support vectors together with their corresponding Lagrange multiplier, in order to perform incremental updates.
| Data Set     | Type       | $d$ | $|T|$    |
|-------------|------------|-----|---------|
| Power Cons. | real world | 6   | 2,075,260 |
| PSP         | (big)      | 60  | 2,338,122 |

Table 5.1: Data Sets

5.4 Experiments

This section investigates the prediction accuracy and efficiency of AMT on streaming data, in terms of runtime and memory, of the proposed AMT. The parameters and their robustness are discussed, and compared to the results obtained with the state of the art FIMT (with both constant and decaying learning rate). For the change detection of both AMT and FIMT $\epsilon_{\text{PH}} = 0.005$ and $\tau_{\text{PH}} = 8 \cdot \sigma_y$ were used (where $\sigma_y$ is the standard deviation in the output dimension before normalization), and a fading factor of $\gamma = 1 - 10^{-4}$ for the Q-statistics.

If nothing else is mentioned, prequential windows were used for the evaluation. This means that every arriving sample is first used for testing, then for training. The incoming data stream was z-score normalized, incrementally updating the mean and standard deviation (cf. Equations 5.3 and 5.4). The accuracy is evaluated by means of the relative error (cf. Equation 4.3).

For the evaluations the real-world datasets from Table 4.1 are used. Additionally, the big real world data sets from Table 5.1 are used, which are more appropriate for evaluating stream learning algorithms.

All evaluation experiments were conducted using a single-threaded Java-based implementation, and were run on Intel Core-2 Duo with 3 GHz and with 4G memory available to each process. The accuracy of a predictor is evaluated by means of the relative error (RE), defined in Equation 4.3, which measures the relative improvement of the prediction accuracy obtained over just using the average in the output dimension. Hence, a RE of 1 means that there is no improvement over the naive approach, and a RE below 1 corresponds to an improvement.
Table 5.2: Summary of the AMT parameters and the varied values

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameter</th>
<th>Varied values</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop criteria</td>
<td>$\tau_n \in \mathbb{N}$</td>
<td>$\tau_n \in {64, 75, 150, 250, 375, 500, 750, 1000, 1500, 2000}$</td>
</tr>
<tr>
<td></td>
<td>$0 \ll \tau_{red} \ll 10$</td>
<td>$\tau_{red} \in {0.5, 0.75, 1, 1.2, 1.5, 1.75, 2}$</td>
</tr>
<tr>
<td>MoClus</td>
<td>$2 \ll k_0 \ll 1000$</td>
<td>$k_0 \in {4, 8, 16, 32, 64}$</td>
</tr>
<tr>
<td></td>
<td>$\alpha \in [0, 1]$</td>
<td>$\alpha \in {0.1, 0.2, ..., 0.5, ..., 0.9}$</td>
</tr>
<tr>
<td></td>
<td>$\lambda_\Sigma$</td>
<td>$\lambda_\Sigma \in {10^{-6}, 10^{-4}, 10^{-2}, 10^{-1}}$</td>
</tr>
<tr>
<td>PLS</td>
<td>$\varphi \in [0, 1]$</td>
<td>$\varphi \in {0.5, 0.6, ..., 0.9, 0.95}$</td>
</tr>
<tr>
<td>RR</td>
<td>$\lambda_R \in \mathbb{R}^+$</td>
<td>$\lambda_R \in {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1}$</td>
</tr>
</tbody>
</table>

5.4. Experiments

The different models and linear separators of AMT come with parameters. To show that these are robust, several runs were performed in which all parameters were kept constant and only was one varied. Table 5.2 provides an overview of the parameters and the corresponding varied values. The bold parameter values are the ones used for the remaining experiments.

The results were averaged over the datasets from Tables 4.1 and 5.1. Figure 5.2 the corresponding standard deviation of the RE is plotted.

Even though a wide value range was tried for each parameter, the RE does not vary much. The highest registered standard deviation is of 0.068, while the most of them are around 0.02.

Comparing SVM and LDA

Both SVM and LDA were considered for the split computation, and their efficiency and the induced split quality compared. The results were averaged over several data sets divided in three categories: artificial data sets, real world data sets with a dimensionality lower or equal 16, and real world data sets with a dimensionality higher than 16, and plotted in Figure 5.3. SVM requires more storage than LDA, and has also less efficient updates. But for higher dimensional data sets, SVM induces more accurate splits.
AMT and FIMT on the artificial and medium data sets

The general observation for AMT is that for lower dimensional data, OLS models and linear splits computed with LDA are sufficient, whereas for high dimensional data PLS identifies best arbitrary oriented subspace models and SVM is better suited for the split computation. Therefore, the medium sized real world datasets from Table 4.1 are additionally divided in low dimensional (dimensionality ≤ 14), and high dimensional (dimensionality > 14). For the first category, OLS with LDA are used for AMT, and for the latter PLS with SVM.
To allow a comparison with the results presented in [IGD11], the evaluation method of 10-fold cross-validation is used. In Figure 5.4 AMT and
FIMT are compared on the artificial data sets. Since the 2D Planes dataset contains many noisy dimensions, the PLS models deliver better results. On the other hand, the Fried data set has independent input dimensions, hence OLS models perform best, as well as the axis parallel splits of FIMT. Regarding the tree size, AMT generates less leaf models as expected. In Figures 5.5 and 5.6 the results on the medium sized real-world data sets are plotted. For the most data sets AMT delivers more accurate models and/or more compact than FIMT.

The runtime of the above mentioned experiments are plotted in Table 5.3. AMT with LDA is much more efficient than FIMT, while AMT with SVM processes less samples each second but generates more accurate and compact models.

### Time-changing streaming data

Of high interest is the evaluation of AMT on streaming data. For the Power Consumption dataset the learning curves of AMT and FIMT are compared. Since the memory required for a node is approximatively equal for AMT and FIMT, the model size is evaluated in the number of nodes. For these experiments prequential windows of 25,000 samples are used, and a sliding step of 5,000 samples. The results in Figure 5.7 show that FIMT with a decaying learning rate has the poorest results, while AMT achieved the best accuracy, with a far smaller model.

To investigate the capability of AMT to adjust to the different types of drift (global abrupt drift, global gradual drift, local drift) the synthetic dataset Fried is used and modified as proposed in [IGD11], creating \(10^6\) samples. The Fried dataset is generated according to the function:

\[
f_F(x) = 10 \sin(\pi x_1x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon
\]
Figure 5.7: Learning curves and model size of AMT and FIMT on the Power Consumption dataset

For the global abrupt drift, this function is replaced after 25% of the stream with the following function:

\[ f_1(x) = 10 \sin(\pi x_4 x_5) + 20(x_2 - 0.5)^2 + 10x_1 + 5x_3 + \epsilon, \]

and the original function is restored for the last 25% of the stream. For the global gradual drift the function \( f_F \) is gradually replaced by \( f_1 \) starting with 25% until 35% of the stream. After 75% of the stream \( f_1 \) is gradually replaced by following function:

\[ f_2(x) = 10 \sin(\pi x_2 x_5) + 20(x_4 - 0.5)^2 + 10x_3 + 5x_1 + \epsilon. \]
For the local abrupt and expanding drift two alternative functions are defined:

\[ f_A(x) = 10x_1x_2 + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon, \]

\[ f_B(x) = 10 \cos(x_1x_2) + 20(x_3 - 0.5)^2 + e_4^2 + 5x_5 + \epsilon, \]

and two different regions \( R_1 \) and \( R_2 \):

\[
R_A : \begin{cases} 
  x_2 < 0.3 \\
  x_3 < 0.3 \\
  x_4 > 0.7 \\
  x_5 < 0.3 
\end{cases} 
\quad \text{and} \quad 
R_B : \begin{cases} 
  x_2 > 0.7 \\
  x_3 > 0.7 \\
  x_4 < 0.3 \\
  x_5 > 0.7 
\end{cases} .
\]

At the beginning \( f_F \) holds for the whole input space. After 25% of the stream \( f_A \) and \( f_B \) are introduced abruptly for the regions \( R_A \) and \( R_B \) respectively. After further 25% of the stream the two regions are expanded by removing the inequalities regarding \( x_5 \), and after 25% more of the stream, \( R_A \) and \( R_B \) are further expanded by additionally removing the inequalities regarding \( x_4 \).

For the local expanding drift another modification data set is generated, such that the drift regions are oblique instead of axis parallel. The function \( f_F \) is abruptly replaced by \( f_A \) starting with 25% but only in region \( R'_A \), defined as:

\[
\begin{cases} 
  x_2 < -x_3 + 0.3 \\
  x_4 > x_5 + 0.7 
\end{cases} ,
\]

After 50% of the stream, this region, in which \( f_A \) replaces \( f_F \), grows to:

\[
\begin{cases} 
  x_2 < -x_3 + 0.4 \\
  x_4 > x_5 + 0.6 
\end{cases} ,
\]

and after further 25% of the stream, it grows to:

\[ x_2 < -x_3 + 0.4. \]
5.4. Experiments

The remaining 5 noisy dimensions are left out in this experiment.

In Figures 5.8 and 5.9 the learning curves and number of nodes for AMT, with OLS models and LDA splits, and FIMT are plotted in the case of the different concept drifts. The advantage of the oblique splits is obvious, as FIMT builds thousands of nodes, while AMT less than 200. Even for the case of local expanding drift in axis-parallel regions, FIMT is continuously increasing.
Adaptive Model Tree on Data Streams

<table>
<thead>
<tr>
<th>Drift Type</th>
<th>FIMT</th>
<th>AMT / LDA</th>
<th>AMT / SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global abrupt drift</td>
<td>10,193</td>
<td>21,838</td>
<td>6,721</td>
</tr>
<tr>
<td>Global gradual drift</td>
<td>11,466</td>
<td>20,120</td>
<td>6,775</td>
</tr>
<tr>
<td>Local expanding drift - p</td>
<td>6,498</td>
<td>21,115</td>
<td>7,201</td>
</tr>
<tr>
<td>Local expanding drift - o</td>
<td>10,041</td>
<td>20,648</td>
<td>7,628</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>5,281</td>
<td>7,085</td>
<td>4,486</td>
</tr>
</tbody>
</table>

Table 5.4: Runtime comparison of AMT and FIMT on streaming data

its number of nodes, whereas AMT has a low constant number of nodes. For these experiments, the number of processed samples per second are summarized in Table 5.4. AMT with LDA splits is the most efficient among them, followed by FIMT. The SVM for computing the linear splits have a high runtime, but for some high dimensional data sets they are deliver much more accurate cluster splits than LDA.

Another interesting aspect is learning with limited computational power. In this context the relative errors of AMT and FIMT are compared in Table 5.5. With an available memory higher than 4 MB, AMT becomes more accurate than FIMT.

<table>
<thead>
<tr>
<th>Memory Size</th>
<th>2 MB</th>
<th>4 MB</th>
<th>8 MB</th>
<th>16 MB</th>
<th>32 MB</th>
<th>64 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMT</td>
<td>0.333</td>
<td>0.299</td>
<td>0.265</td>
<td>0.249</td>
<td>0.235</td>
<td>0.231</td>
</tr>
<tr>
<td>FIMT</td>
<td>0.305</td>
<td>0.287</td>
<td>0.277</td>
<td>0.276</td>
<td>0.274</td>
<td>0.276</td>
</tr>
</tbody>
</table>

Table 5.5: Accuracy for limited storage

Visualization of results.

To illustrate the resulting models of AMT and FIMT, the pixels of some sample images from the Corel Wang database [WLW01] were incrementally added to both model trees. The pixel coordinates were considered as inputs and the color values as outputs. Model trees of comparable size were build and their leaf models are visually illustrated in Figure 5.10. This visu-
alization makes clear how the oblique splits of AMT produce a more accurate reconstruction of the original image than the axis parallel splits of FIMT.

![Visualization of the obtained piecewise linear models](image)

Figure 5.10: Visualization of the obtained piecewise linear models

5.5 Conclusion

In this chapter a novel algorithm for learning model trees was proposed, called AMT. It computes oblique splits with the MoClus algorithm, introduced in Chapter 4, being able to handle different regression models. OLS, RR, and PLS models were investigated and it turned out that OLS models are suited for lower dimensional data, whereas PLS models are better suited for higher dimensional data, since they identify arbitrary oriented subspace models. To compute the split, based on the clustering, the two linear classifiers LDA and SVM were used, which are also incrementally updated with incoming data. LDA requires less storage, and computes and updates the splits more efficient than SVM. While for lower dimensional cases, LDA is sufficient, for higher dimensional data SVM contributes to more accurate models. Altogether, AMT is a more compact and accurate model tree than the state of the art FIMT, and if LDA is used, AMT is also more efficient.
Part III

Hinging Hyperplane Models
Chapter 6

Hinging Hyperplane Models

Researchers from various fields deal with experimental measurements in form of points. This data representation though is incomplete since it is only a discretization of the underlying process. For understanding this set of measurement points, researchers model these points by mathematical functions. These models should offer a compact and intuitive representation of the underlying process. Regression trees, discussed in Part II, are well suited for this task. However, the resulting piecewise affine functions are not guaranteed to be continuous. In some applications though, the processes represent continuous dynamics and require the continuity of the piecewise affine functions.

Hinging hyperplanes (HH) models, introduced by Breiman in [Bre93], offer a continuous approximation of nonlinear data. HH models are defined as a sum of basis functions, called hinge functions, each consisting of two continuously joined hyperplanes. A hinge function inherently contains a split in the input space, which can be used to transform a HH-model into a regression tree. This way a piecewise linear model of the data is built which offers a better understanding of the underlying process.
6.1 Hinge Functions

For a training set $T \in \mathbb{R}^{d+1}$ with samples of the form $(x, y)$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$, the regression hyperplane is defined as:

$$\hat{y} = \beta_0 + x_1 \beta_1 + ... + x_d \beta_d = x^\top \beta,$$

(6.1)

where $x = [1, x]$. A hinging hyperplane model approximates the unknown function $f$ by a sum of basis functions:

$$\hat{f}(x) = \sum_{k=1}^{K} h_k(x),$$

(6.2)

where $h_k : \mathbb{R}^d \to \mathbb{R}$ are hinge functions of the form:

$$h(x) = \begin{cases} x^\top \beta^+, & \text{if } x^\top \Delta \geq 0 \\ x^\top \beta^-, & \text{if } x^\top \Delta < 0 \end{cases},$$

(6.3)

The two hyperplanes described by $\beta^+$ and $\beta^-$ are joined together at their intersection $\{x : x^\top \Delta = 0\}$, where $\Delta = (\beta^+ - \beta^-)$ is called hinge. The hinge
6.1. Hinge Functions

$\Delta$ separates the samples in the input space in:

$$T^+ = \{x : x^T \Delta \geq 0\} \quad \text{and} \quad T^- = \{x : x^T \Delta < 0\}.$$

The explicit form of the hinge function $h(x)$ is either $h(x) = \min(x^T \beta^+, x^T \beta^-)$ in the case of concave functions, or $h(x) = \max(x^T \beta^+, x^T \beta^-)$ in the case of convex functions. Figure 6.1 shows an example for $h(x) = \max\{x^T \beta^+, x^T \beta^-\}$ for a set of samples with a single input.

Bounding the Approximation Error

Breiman proved in [Bre93] that if the underlying function $f$ generating the samples is sufficiently smooth, there is a constant $C(f)$:

$$\int \| \omega \|^2 |\tilde{f}(\omega)| d\omega = C < \infty,$$

where $\tilde{f}(\omega)$ is the Fourier transform of $f(x)$, such that

$$\left\| f - \sum_{k=1}^{K} h_k \right\|^2 \leq \frac{(2R)^4 C^2}{K},$$

where $R$ is the radius of the sphere in which the data set is contained. Hence, with the hinging hyperplane models nonlinear continuous functions can be approximated to an arbitrary precision, if the number of hinge functions $K$ is high enough. However, the number of fitted hinge functions is limited by the size of the data set.

Geometric Interpretation of the Hinges

Before describing the hinge finding method, the geometric interpretation of the hinges (cf. Figure 6.2) is discussed. Consider the regression hyperplane from Equation 6.1, another way to describe it is the Hessian normal form:

$$n_1 x_1 + \ldots + n_d x_d + n_{d+1} y - o = 0.$$  (6.4)
One representation can be easily converted into the other:

\[
(6.1) \Rightarrow (6.4) : \quad o = \beta_0, \quad n_j = \beta_j, j = 1 \ldots d, \quad n_{d+1} = -1,
\]

\[
(6.4) \Rightarrow (6.1) : \quad \beta_0 = \frac{o}{n_{d+1}}, \quad \beta_j = -\frac{n_j}{n_{d+1}}, j = 1 \ldots d.
\]

The hinge \( \Delta = (\delta_1, \ldots, \delta_{d+1}) \), obtained by setting \( x^\top \beta^+ = x^\top \beta^- \), describes a regression hyperplane in \( \mathbb{R}^{d+1} \). The intersection of \( \Delta \) with the input space represents the projection of the intersection between \( \beta^+ \) and \( \beta^- \) onto the input space. This intersection is denoted as \( \Gamma \) and referred to as the separator, since it separates the \( d \)-dimensional input space into two half spaces.

Let \( \Delta = (\delta_1, \ldots, \delta_{d+1}) \) be the parameters of the hinge for an output \( y \):

\[
y = x_1 \delta_1 + \ldots + x_d \delta_d + \delta_{d+1},
\]

By setting this equation equal to 0, the separator \( \Gamma \) in the input space is obtained in its Hessian form:

\[
x_1 \delta_1 + \ldots + x_d \delta_d + \delta_{d+1} = 0,
\]

with normal vector \( [\delta_1, \ldots, \delta_d]^\top \) and offset \( o = -\delta_{d+1} \).
6.2 Hinge Finding Algorithm

The hinge finding algorithm (HFA), introduced by Breiman in [Bre93], is described in Algorithm 8. The algorithm starts with an arbitrary initial hinge $\Delta^{(0)}$ and partitions the dataset into the two sets $\mathcal{T}^+$ and $\mathcal{T}^-$. Least squares regression is then used to fit a hyperplane $\beta^+$ to the observations in $\mathcal{T}^+$ and another hyperplane $\beta^-$ to the observations in $\mathcal{T}^-$. The new hinge becomes $\Delta^{(1)} = \beta^+ - \beta^-$, and the sets $S^+$ and $S^-$ are updated. These steps are repeated until the hinge function converges. Convergence means that the sum

$$\sum_{i=1}^{n} \text{sgn}(y_i - (\mathbf{x}_i^T \mathbf{w}) \Delta^{(e)})$$

is no longer decreasing.

**Algorithm 8** Hinge Finding Algorithm

1: function HFA($\mathcal{T}, \Delta^{(0)}$)
2: compute $\mathcal{T}^+ = \{x : x^T \Delta^{(0)} \geq 0\}$
3: compute $\mathcal{T}^- = \{x : x^T \Delta^{(0)} < 0\}$
4: set $e = 1$
5: repeat
6: compute the regression coefficients $\beta^+$ and $\beta^-$ of $\mathcal{T}^+$ and $\mathcal{T}^-$
7: compute $\Delta^{(e)} = \beta^+ - \beta^-$
8: compute $\mathcal{T}^+ = \{x : x^T \Delta^{(e)} \geq 0\}$
9: compute $\mathcal{T}^- = \{x : x^T \Delta^{(e)} < 0\}$
10: $e = e + 1$
11: until convergence
12: return hinge function $h$, defined by $\Delta^{(e)}, \beta^+, \beta^-$
13: end function
of the mean squared errors of $\beta^+$ and $\beta^-$ does not improve. A visualization of these steps is depicted in Figure 6.3.

Breiman proved in [Bre93] that if the underlying unknown function is a hinge function, then the HFA will converge exponentially fast towards this function. On arbitrary data sets though, this algorithm might not converge. It might happen that it gets into an endless loop, alternating between 2 hinges, or, alternatively, it might happen that the separator ends up outside the data support, delivering a non-valid result.

Fitting Several Hinges

For a better approximation of the observations in $\mathcal{T}$ several hinge functions $h_{[k]}(x)$ can be combined yielding $\hat{y} = \sum_{k=1}^{K} h_{[k]}(x)$. The first hinge $h_{[1]}(x)$ is computed as described in Algorithm 8. To compute the second hinge $h_{[2]}(x)$ a temporary data set $\mathcal{T}_{[2]}$ is generated that contains for each sample $(x, y) \in \mathcal{T}$ an observation $(x, y_{[2]})$, where $y_{[2]} = y - h_{[1]}(x)$ is the residuum that is not fitted by $h_{[1]}(x)$. After computing $h_{[2]}(x)$ the first hinge is refitted on a tempo-

Algorithm 9 Compute HH model

Input: $\mathcal{T}$

1: compute the first hinge function $h_{[1]}(x) = \text{HFA}(\mathcal{T}, \text{random } \Delta^{(0)})$
2: for $k \leftarrow 2 \ldots K$ do
3: compute $\mathcal{T}_{[k]} = (x, y_{[k]})$, with $y_{[k]}$ according to Equation 6.5
4: compute the $k$-th hinge function $h_{[k]}(x) = \text{HFA}(\mathcal{T}_{[k]}, \text{random } \Delta^{(0)})$
5: repeat
6: for $j \leftarrow 1 \ldots k$ do
7: compute $\mathcal{T}_{[j]} = (x, y_{[j]})$, with $y_{[j]}$ according to Equation 6.6
8: refit $h_{[j]}$ starting with the current hinge: $h_{[j]} = \text{HFA}(\mathcal{T}_{[j]}, \Delta^{(\text{ref})})$
9: end for
10: until no significant reduction in the mse of the resulting model
11: end for

Output: hinge functions $h_{[1]}, \ldots, h_{[K]}$
6.3 Convergence with Line Search

Pucar et al. showed in [PS98] that the HFA, presented in Algorithm 8, is a Newton algorithm applied to minimize the mean squared error of the resulting model. To this end, the objective function can be formulated as:

\[ V(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - h(x_i, \beta))^2, \]  

(6.7)

where \( \beta \) is a vector containing the coefficients of both regression hyperplanes:

\[ \beta = \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix}. \]

The goal is to find \( \beta \) that minimizes \( V(\beta) \). This can be achieved with Newton’s method, which starts with an initial guess \( \beta^{(0)} \) and iteratively makes steps towards the searched minimum:

\[ \beta^{(e+1)} = \beta^{(e)} - (\nabla^2 V)^{-1} \nabla V. \]  

(6.8)
Hence, the gradient \( \nabla V \) and the Hessian \( \nabla^2 V \) are required. The derivative of the hinge function with respect to \( \beta^+ \) is:

\[
\frac{dh(x, \beta)}{d\beta^+} = \begin{cases} x, & \text{if } x \in T^+ \\ 0, & \text{if } x \in T^- \end{cases}
\]

and analogously with respect to \( \beta^- \). The gradient of the function \( V \) is:

\[
\nabla V = \begin{pmatrix} \frac{\partial V}{\partial \beta^+} \\ \frac{\partial V}{\partial \beta^-} \end{pmatrix} = \begin{pmatrix} -\sum_{i=1}^{[T]} \frac{dh(x, \beta)}{d\beta^+} (y_i - h(x_i, \beta)) \\ -\sum_{i=1}^{[T]} \frac{dh(x, \beta)}{d\beta^-} (y_i - h(x_i, \beta)) \end{pmatrix} = \begin{pmatrix} -\sum_{x_i \in T^+} x_i (y_i - x_i^\top \beta^+) \\ -\sum_{x_i \in T^-} x_i (y_i - x_i^\top \beta^-) \end{pmatrix},
\]

and the Hessian is:

\[
\nabla^2 V = \begin{pmatrix} \frac{\partial}{\partial \beta^+} \left( -\sum_{x_i \in T^+} x_i (y_i - x_i^\top \beta^+) \right) & \frac{\partial}{\partial \beta^-} \left( -\sum_{x_i \in T^+} x_i (y_i - x_i^\top \beta^+) \right) \\ \frac{\partial}{\partial \beta^+} \left( -\sum_{x_i \in T^-} x_i (y_i - x_i^\top \beta^-) \right) & \frac{\partial}{\partial \beta^-} \left( -\sum_{x_i \in T^-} x_i (y_i - x_i^\top \beta^-) \right) \end{pmatrix}
= \begin{pmatrix} \sum_{x_i \in T^+} x_i^\top x_i & 0 \\ 0 & \sum_{x_i \in T^-} x_i^\top x_i \end{pmatrix}.
\]

Substituting these two matrices into Equation 6.8, following update rule for \( \beta \) is obtained:

\[
\beta^{(e+1)} = \beta^{(e)} + \left( \sum_{x_i \in T^+} x_i^\top x_i \right)^{-1} \left( \sum_{x_i \in T^+} x_i (y_i - x_i^\top \beta^+) \right) \cdot \left( \sum_{x_i \in T^-} x_i (y_i - x_i^\top \beta^-) \right)
\]
6.3. Convergence with Line Search

\[
\beta^{(e)} + \left( \begin{array}{c}
\sum_{x_i \in T^+} x_i^\top x_i^{-1} & \sum_{x_i \in T^+} x_i(y_i - x_i^\top \beta^+)

\sum_{x_i \in T^-} x_i^\top x_i^{-1} & \sum_{x_i \in T^-} x_i(y_i - x_i^\top \beta^-)
\end{array} \right)
\]

\[
= \beta^{(e)} + \left( \begin{array}{c}
\sum_{x_i \in T^+} x_i^\top x_i^{-1} & \sum_{x_i \in T^+} x_i y_i - \beta^+

\sum_{x_i \in T^-} x_i^\top x_i^{-1} & \sum_{x_i \in T^-} x_i y_i - \beta^-
\end{array} \right)
\]

It follows that:

\[
\beta^{(e+1)} = \beta^{(e)} + (\beta^{(e+1)} - \beta^{(e)}),
\]

concluding that the Newton’s minimization step corresponds to that proposed by Breiman in [Bre93]. This equivalence proves that the HFA is not guaranteed to converge, since Newton’s method is not converging either. However, the damped Newton’s method does converge, when applying following linear search between \(\beta^{(e)}\) and \(\beta^{(e+1)}\):

\[
\beta'^{(e+1)} = \beta^{(e)} + \lambda(\beta^{(e+1)} - \beta^{(e)}).
\]

This means that, if the new \(\beta'^{(e+1)}\) does not contribute to a reduction of the model’s mean squared error, then a binary search is started among the linear combinations of \(\beta^{(e)}\) and \(\beta^{(e+1)}\), by setting for example \(\lambda = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots\}\). Recall that:

\[
\beta^{(e)} = \begin{pmatrix} \beta^{(e)+} \\ \beta^{(e)-} \end{pmatrix} \quad \text{and} \quad \Delta^{(e)} = \beta^{(e)+} - \beta^{(e)-}.
\]

Hence, \(\beta^{(e)}\) contains the information about the hinge \(\Delta^{(e)}\) and implicitly about \(\Gamma^{(e)}\), which separates the input space in \(T^+\) and \(T^-\). For the HFA it means that, if the impurity of the model determined by \(\Delta^{(e+1)}\) has increased over \(\Delta^{(e)}\), then a binary search is performed among the linear combinations
of $\Delta^{(e)}$ and $\Delta^{(e+1)}$:

$$
\Delta^{(e+1)} = (1 - \lambda)\Delta^{(e)} + \lambda\Delta^{(e+1)},
$$

and the linear combination is chosen, which determines the smallest mean squared error. Figure 6.4 illustrates the geometrical interpretation of this idea: $\Gamma^e$ is the separator in the input space corresponding to $\Delta^{(e)}$, and $\Gamma^{e+1}$ the one corresponding to $\Delta^{(e+1)}$. Since the impurity of the model induced by $\Delta^{(e+1)}$ has increased, a search is initiated to find a linear combination between the two separators, which induces a model with a lower impurity than the one induced by $\Delta^{(e)}$. For $\lambda = 0.5$ the model induced by $\Delta^{(e+1)}$ has a lower impurity than the one induced by $\Delta^{(e)}$, therefore $\lambda$ is increased to 0.75 and $\Delta^{(e+1)}$ is again computed. This represents only a slight modification of the HFA and ensures in return that it converges in a local minimum.

### 6.4 Applications

Hinging hyperplane models were computed for the experimental measurement presented in Section 1.1. Some exemplary images are plotted in Figures 6.5 - 6.7. For the SPRAY and DIESEL dataset, few models are sufficient to describe the regression surface. The WIND TURBINE dataset has a more complex surface, which is approximated by four hinges.
6.4. Applications

Figure 6.5: Application of HFA on the SPRAY dataset

Figure 6.6: Application of HFA on the WIND TURBINE dataset
Figure 6.7: Application of HFA on the DIESEL dataset
Chapter 7

Multivariate Hinging Hyperplane Models

The hinging hyperplane approach is a nonlinear learning technique which computes a continuous piecewise function, consisting of linear functions over individual partitions in the predictor space. However, it is only designed for one predicted variable. In the case of \( r \) predicted variables, \( r \) hinging hyperplane models are computed independently from each other. To obtain a unique model of the data, the resulting \( r \) partitionings are overlapped. However, in this case the number of partitions grows quickly with \( r \) and the result is no longer being compact or interpretable.

This chapter extends the concept of hinging hyperplane model to the multivariate case. The main idea is to compute a common partitioning for the piecewise linear models of the different outputs. The novel mHFA algorithm is introduced, which simultaneously considers all response variables when building the multivariate model. It enforces common hinges, while at the same time restoring the continuity of the resulting functions. The model complexity no longer depends on the number of predicted variables, remaining compact and interpretable.
In many cases, real world data contains several input dimensions, which the user can control, and several output dimensions, which can only be observed. All these dimensions describe the same phenomenon or process and a model is required to describe it. This is either composed out of separate piecewise linear models for each output, or a multivariate piecewise linear model of the form:

$$\begin{align*}
\hat{y}_1 &= x^T \beta_{i,1} \\
&\vdots \\
\hat{y}_r &= x^T \beta_{i,r}
\end{align*}$$

$$\forall i \in \{1, \ldots, r\}.$$ 

The main difference to several piecewise linear models is that there is a common partitioning in the input space. Such multivariate models are especially required for the optimization problem within MPC controllers, described in Section 1.1, where the number of partitions has a great influence on the efficiency of the controller.

However, the hinge finding algorithm (HFA) is only designed to deal with one-dimensional output variables. The most straight-forward solution for the case of $r$ output dimensions is to build $r$ independent models, one for each output. Then, the resulting partitionings are overlapped in order to obtain a multivariate (or shared) hinging hyperplane model.

Note that in the case of overlapping the partitionings, the number of par-
partitions grows quickly with $K$, the number of hinges in the HH model, and with $r$, the number of output variables. Figure 7.1 illustrates the need for a multivariate model by means of an example with a 2-dimensional input space and a 3-dimensional output. The upper row illustrates the partitionings in the input space if a model is build independently for each output, together with the resulting overlapping partitioning. With such a high number of sub-models, the resulting multivariate model looses its property of being easily interpretable. The lower row shows that when the models for the different outputs are built simultaneously, with a common partitioning in the input space, the resulting partitioning of the multivariate model is much more compact.

Figure 7.2 shows an example with an one input, $x_1$, and two outputs, $y_1$ and $y_2$. When computing the two models independently, five partitions emerge. As part of the same measured phenomenon or process, the behaviors of the two outputs is correlated, and a common partitioning is possible. The right image shows a common partitioning for the two outputs, which can be obtained by simultaneously considering all output dimensions. In this case only three partitions in the input space are sufficient, instead of five.

This chapter introduces the mHFA, a multivariate hinging hyperplane
model which simultaneously considers all output dimensions in the construction phase and delivers a shared partitioning. Despite the common partitions, the continuity of the hinge functions is guaranteed for all predicted variables. Since additional constraints are built in, this shared model may be slightly less accurate than the independently built models. However, if some output dimensions are noisy this simultaneous consideration leads to a more robust model. The main advantage is, however, the compactness of the resulting model.

Similarly to how a simple hinging hyperplane model can be transformed into a regression tree, a multivariate hinging hyperplane model can be transformed into a multivariate regression tree. While the leaves of univariate regression trees contain a linear model, the leaves of multivariate regression trees contain several linear models, one for each output.

### 7.2 Hinge Regression for Multiple Outputs

For a given data set $\mathcal{T} \subset \mathbb{R}^{d+r}$, with $r > 1$, the goal is to find hinge functions for each output with identical separators, which simultaneously minimize the residual error in all output dimensions. The basic idea is to compute one hinge per output $y_j$ and to combine the corresponding separators $\Gamma_j$ to
7.3. Finding the Consensus Separator $\hat{\Gamma}$

Figure 7.4: Finding the best consensus separator $\hat{\Gamma}$.

a single consensus separator $\hat{\Gamma}$ (cf. Figure 7.3). Combining several separators in a meaningful manner in arbitrary dimensions is not straightforward. Moreover, by imposing a hinge on a hinge function, the continuity property of hinge functions gets lost. Hence, the regression models $\beta_j^+$ and $\beta_j^-$ for each output $y_j$ are forced to join continuously on $\hat{g}_j$ whose projection onto the input space is $\hat{\Gamma}$. The main steps of the procedure are

1. Generate separate data sets $T_j$ for each output $y_j$, i.e. $\forall (x, y) \in T : (x, y_j) \in T_j$

2. Compute a hinge $\Delta_j$ per output $y_j$ using $T_j$

3. Combine all separators $\Gamma_j$, $j = 1 \ldots r$ to a consensus separator $\hat{\Gamma}$

4. Force the $r$ regression models to join continuously at $\hat{g}_j$

Step 1 generates $r$ temporary data sets, and Step 2 finds a hinge for each output as described in Algorithm 8. Step 3 is described in the following Section, details on Step 4 are provided in Section 7.3. The entire hinge finding algorithm for multiple outputs is provided in Section 7.4.
7.3 Finding the Consensus Separator \( \hat{\Gamma} \)

A naive solution to combine the individual separators to a consensus separator \( \hat{\Gamma} = [\hat{\gamma}_1, \ldots, \hat{\gamma}_d] \) is to use a linear combination of the parameters as

\[
\hat{\gamma}_i = \frac{1}{r} \sum_{j=1}^{r} \gamma_{j,i}, \quad i = 1 \ldots d. \tag{7.1}
\]

This solution does not necessarily find the best consensus separator as depicted in Figure 7.4. Part a) illustrates an example in a 2-dimensional input space where two separators \( \Gamma_1 \) and \( \Gamma_2 \) are combined to consensus separator \( \hat{\Gamma} \) using Equation 7.1. \( \hat{\Gamma} \) is the bisector of the smaller angle between \( \Gamma_1 \) and \( \Gamma_2 \). Part b) shows that \( \hat{\Gamma} \) according to Equation 7.1 yields the bisector of the larger angle between \( \Gamma_1 \) and \( \Gamma_2 \), which does not represent the best consensus. The normal vectors of the two different solutions result from adding the individual normal vectors with different orientations, i.e. either \( n_1 + n_2 \) or \( n_1 - n_2 \). The two possible orientations of a normal vector \( n_j \) are denoted as \( \xi_j n_j \) with \( \xi_j \in \{-1, 1\}, j = 1 \ldots r \) and the normal vector of the consensus separator is defined as a sum of these oriented normals:

\[
\hat{n} = \frac{\sum_{j=1}^{r} \xi_j n_j}{\| \sum_{j=1}^{r} \xi_j n_j \|}. \tag{7.2}
\]

Insofar, the separator \( \hat{\Gamma} \) is searched, which is defined by its normal \( \hat{n} \) and its offset \( \hat{o} \), with minimal deviation from the \( r \) separators. The normal \( \hat{n} \) with minimal orientation deviation is found by minimizing the sum of angles \( \alpha_j \) between \( \hat{\Gamma} \) and each \( \Gamma_j \), where \( \alpha_j < \pi/2 \) is the smaller angle between the two separators. This is the same as maximizing the sum of all cosines of the \( \alpha_j \). The offset with minimal deviation is found by constraining \( \hat{\Gamma} \) to include the point with minimal distance to all \( \Gamma_j \). Figure 7.4 c) illustrates a consensus separator in a 2-dimensional input space.
7.3. Finding the Consensus Separator $\hat{\Gamma}$

**Definition 7.1. Consensus separator.** In an $d$-dimensional input space let $\Gamma_j, j = 1 \ldots r$, be separators defined by $x^\top n_j - o_j = 0$. The consensus separator $\hat{\Gamma}$ is defined by the following normal vector:

$$
\sum_{j=1}^{r} |\hat{n}^\top n_j| \rightarrow \max
$$

and contains the point $p$ with

$$
\sum_{j=1}^{r} (p^\top n_j - o_j)^2 \rightarrow \min
$$

**Lemma 7.1.** Let $\Xi = (\xi_1, \ldots, \xi_r)$ with $\xi_j \in \{-1, 1\}, j = 1 \ldots r$. The normal according to Equation 7.2 that maximizes Equation 7.3 is the one that maximizes the sum of cosines of all angles between oriented normals $\xi_i n_i$ and $\xi_j n_j$, defined as:

$$
\left( \sum_{i=1}^{r} \sum_{j=1}^{r} \xi_i \xi_j n_i^\top n_j \right)
$$

**Proof. (Lemma 7.1).** Let $\Xi = (\xi_1, \ldots, \xi_r)$ with $\xi_j \in \{-1, 1\}, j = 1, \ldots, r$ be the set of orientations for the normal vectors $n_1, \ldots, n_r$, such that Equation 7.5 is maximized. Next it is proven by contradiction that the normal $\hat{n}$ according to Equation 7.2, which uses the above orientations $\Xi$, is the one that maximizes the equation:

$$
\sum_{j=1}^{r} |\hat{n}^\top n_j|
$$

Hence, assume that Equation 7.5 is maximal for $\Xi$, but not Equation 7.6. This implies that $\exists e \in \{1, \ldots, r\}$ such that $|\hat{n}^\top n_e| \neq \xi_e \hat{n}^\top n_e$. Since $\xi_e \in \{-1, 1\}$, this further implies that $\hat{n}^\top n_e \xi_e < 0$. Substituting $\hat{n}$ from Equation 7.2 in $\hat{n}^\top n_e \xi_e < 0$ following is obtained:

$$
\left( \sum_{j=1}^{r} \xi_j n_j \right)^\top \xi_e n_e < 0
$$
This can be further transformed to:

\[ \sum_{j=1, j \neq e}^r \xi_j \xi_e n_j^\top n_e + \xi_e \xi_e n_e^\top n_e < 0 \]

\[ \sum_{j=1, j \neq e}^r \xi_j \xi_e n_j^\top n_e < -1. \]  

(7.8)

Next the set \( \Xi \setminus \{\xi_e\} \) is split into two sets \( K \) and \( L \) such that: \( \forall l \in L : \xi_l \xi_e n_l^\top n_e > 0 \), and \( \forall k \in K : \xi_k \xi_e n_k^\top n_e < 0 \). Equation 7.8 is then rewritten as:

\[ \sum_{l \in L} \xi_l \xi_e n_l^\top n_e + \sum_{k \in K} \xi_k \xi_e n_k^\top n_e < -1 \]

\[ \left| \sum_{l \in L} \xi_l \xi_e n_l^\top n_e \right| - \left| \sum_{k \in K} \xi_k \xi_e n_k^\top n_e \right| < -1, \]

concluding that:

\[ \left| \sum_{l \in L} \xi_l \xi_e n_l^\top n_e \right| < \left| \sum_{k \in K} \xi_k \xi_e n_k^\top n_e \right|. \]

Hence, by inverting \( n_e \), i.e. setting \( \xi'_e = (-1) \cdot \xi_e \), following is obtained:

\[ \left| \sum_{l \in L} \xi_l \xi'_e n_l^\top n_e \right| > \left| \sum_{k \in K} \xi_k \xi'_e n_k^\top n_e \right| \]

\[ \sum_{l \in L} \xi_l \xi'_e n_l^\top n_e + \sum_{k \in K} \xi_k \xi'_e n_k^\top n_e > 0. \]  

(7.9)

Since \( \xi_e \xi_e n_e^\top n_e = \xi'_e \xi'_e n_e^\top n_e = 1 \) it holds that:

\[ \sum_{j=1, j \neq e}^r \xi_j \xi'_e n_j^\top n_e + \xi'_e \xi_e n_e^\top n_e > 0. \]  

(7.10)

From Equation 7.7 and Equation 7.10 it follows that:

\[ \sum_{j=1}^r \xi_j \xi'_e n_j^\top n_e > \sum_{j=1}^r \xi_j \xi_e n_j^\top n_e. \]  

(7.11)

By switching \( \xi_e \) with \( \xi'_e \) in Equation 7.5 following is obtained according to
7.3. Finding the Consensus Separator \( \hat{\Gamma} \)

Equation 7.11:

\[
\sum_{j=1,j \neq e}^{r} \xi_j \xi_j^\top n_j n_i + \sum_{i=1}^{r} \xi_e \xi_e^\top n_e n_i > \sum_{j=1,j \neq e}^{r} \xi_j \xi_j^\top n_j n_i + \sum_{i=1}^{r} \xi_e \xi_e^\top n_e n_i
\]

(7.12)

Hence, a higher sum than before is obtained, which is a contradiction to the assumption that Equation 7.5 is maximal.

The consensus separator \( \hat{n} \) is therefore computed according to Lemma 7.1, \( \Xi \) being determined by using binary integer programming. To determine its offset, the point \( p \) that minimizes the sum of squared distances to all hyperplanes (cf. Equation 7.4) is required. Consider \( p = [p, 1] \), \( n_j = [n_j, d_j] \) and define the matrix \( N \in \mathbb{R}^{r \times (d+1)} \) as the matrix that contains as rows all \( n_j, j = 1 \ldots r \). Equation 7.4 can then be rewritten as

\[
\min_p ( (Np)^2 ) = \min_p ( (p^\top N^\top)(Np) ) = \min_p (p^\top A p)
\]

(7.13)

where \( A = N^\top N \) is an \((d+1) \times (d+1)\) matrix. A solution can be found by computing an eigenvalue decomposition of \( A \) or solving

\[
\frac{\partial}{\partial p} p^\top A p = 0 \iff \begin{bmatrix} A_1 \cdots A_d \end{bmatrix} p = \begin{bmatrix} - A_{d+1} \end{bmatrix}
\]

(7.14)

where \( A_i \) is the \( i \)-th column of matrix \( A \).

Forcing Continuous Joins

Recall that the two regression models \( \beta_j^+ \) and \( \beta_j^- \) for output \( y_j \) are continuously joined on a \((d-1)\)-dimensional hyperplane \( g_j \) whose projection onto the input space is the separator \( \Gamma_j \) (cf. Figure 6.2). To enforce the consensus separator \( \hat{\Gamma} \) it has to be ensured that the regression planes are continuously joined on an \((d-1)\)-dimensional hyperplane \( \hat{g}_j \) whose projection onto the input space is \( \hat{\Gamma} \). The idea is depicted in Figure 7.5 and consists of two parts:

(a) Compute a consensus hyperplane \( \hat{g}_j \), in which the two hyperplanes
shall be continuously joined

(b) Recompute $\beta^+_j$ and $\beta^-_j$ such that they both include $\hat{g}_j$ (cf. Figure 7.5 right)

Regarding (a), the consensus hyperplane $\hat{g}_j$ is found by computing a set of $d$ affinely independent points $\bar{p}_i$ that lie on $\hat{g}_j$. The hyperplane $\hat{g}_j$ itself lies in the $\hat{\Gamma}_\perp$-plane, which is the $d$-dimensional hyperplane that contains $\hat{\Gamma}$ and that is perpendicular to the input space (cf. Figure 7.5 left). Each point $\bar{p}_i$ is the average between two points $p^+_i$ and $p^-_i$: $p^+_i$ lies in the intersection of $\beta^+_j$ and the $\hat{\Gamma}_\perp$-plane, $p^-_i$ lies in the intersection of $\beta^-_j$ and the $\hat{\Gamma}_\perp$-plane, and it holds that $p^+_{i,l} = p^-_{i,l}$ for $l = 1 \ldots d$, i.e. they have the same input values.

To compute the points $p^+_i$, a set $Q \subset \mathbb{R}^d$ of points $q_i$ is chosen in the input space, $p^+_{i,l}$ is set equal to $q_{i,l}$, for $l = 1 \ldots d$ and the output values are determined as $p^+_{i,d+1} = [q_i, 1] \cdot \beta^+$. The $p^-_i$ are computed analogously and $\bar{p}_i = (p^+_i + p^-_i)^\frac{1}{2}$. Choosing $Q$ is done as follows: Let $x_{i,min} = \min_{(x,y) \in T} \{x_i\}$ and $x_{i,max} = \max_{(x,y) \in T} \{x_i\}$ be the minimal and maximal values in $T$ in dimension $i$. The input cube $\mathcal{I}^\square \subset \mathbb{R}^d$ is defined as the $d$-dimensional cube spanned by the minimum and maximum values

$$\mathcal{I}^\square = \left\{ (x_1, \ldots, x_d) \mid \exists l \ x_{l,min} \leq x_l \leq x_{l,max} \land \forall i \neq l \ x_i = x_{i,min} \lor x_i = x_{i,max} \right\}$$

The set $Q$ is chosen to be $d$ intersection points $q_i$ of $\hat{\Gamma}$ with the input cube $\mathcal{I}^\square$, which can easily be computed for arbitrary dimensions. Figure 7.5 (left) shows an example for $\mathcal{I}^\square$ and the two intersection points $q_1$ and $q_2$. 
(b) For the output $y_j$ let $\hat{S}_j^+$ and $\hat{S}_j^-$ be the two partitions of $T_j$ corresponding to the consensus separator $\hat{\Gamma}$. Next it is described how to recompute the regression plane $\beta_j^+$ using $\hat{S}_j^+$, $\beta_j^-$ is computed analogously. Let $X \in \mathbb{R}^{|\hat{S}_j^+| \times d}$ be the matrix that contains as rows all input vectors in $\hat{S}_j^+$ and $Y \in \mathbb{R}^{|\hat{S}_j^+| \times 1}$ the vector that contains all corresponding output values of the observations in $\hat{S}_j^+$. Similarly $Q \in \mathbb{R}^{|Q| \times d}$ is defined as the matrix that contains as rows all points $q_i \in Q$ and $Z \in \mathbb{R}^{|Q| \times 1}$ as the vector that contains all corresponding values $\bar{p}_{i,d+1}$. A regression plane that contains all points $\bar{p}_i$ can be approximated by assigning a high weight $w$ to $Q$ and $Z$ in the unconstrained least squares problem

$$ \min_{\beta} \left\| \begin{bmatrix} X & \beta \end{bmatrix} - \begin{bmatrix} Y & Z \end{bmatrix} \right\|_{\text{least squares}} $$

(7.15)

In the experiments from Section 7.5 $w = |T|$ was used, i.e. the weight for each point $\bar{p}_i$ is equal to the size of the data set.

### 7.4 Hinge Finding Algorithm for Multiple Outputs

The hinge finding algorithm for multiple outputs (mHFA) has a similar process flow as the HFA for a single output. It starts with a random hinge $\Delta^{(0)}$ that is iteratively improved until no significant error reduction occurs. Since the algorithm does not guarantee convergence to a global optimum, a number of $\nu$ random initializations are performed and the model yielding the smallest error is retained. Unlike the case with a single output, each observation has several errors, one for each output. To avoid an influence of different output ranges the errors per output $y_j$ are normalized by the corresponding range $\text{range}_j = y_{j,max} - y_{j,min}$. The approximation error for $T$ over all outputs is then:

$$ \text{Err}_T = \sum_{(x,y) \in T} \sum_{j=1}^r \left( y_j - \sum_{k=1}^K h_{j,k}(x) \left( \frac{1}{\text{range}_j} \right) \right)^2. $$

(7.16)
Algorithm 10 compute multivariate hinges

Input: $\mathcal{T}, K$

1: for $k = 1$ to $K$ do
2: create $\mathcal{T}[k]$;
3: compute $\Gamma_{\text{init}}$;
4: $\hat{\Gamma}[k] = \text{FINDCONSENSUSSEPARATOR}(\mathcal{T}[k], \Gamma_{\text{init}})$;
5: for $i = 1$ to $k$ do
6: create $\mathcal{T}[i]$;
7: $\hat{\Gamma}[i] = \text{FINDCONSENSUSSEPARATOR}(\mathcal{T}[i], \hat{\Gamma}[i])$;
8: end for
9: end for

When the error is no longer reduced from one iteration to the next, mHFA is stopped.

Algorithm 11 find consensus separator

1: function $\text{FINDCONSENSUSSEPARATOR}(\mathcal{T}, \Gamma_{\text{init}})$
2: create $\mathcal{T}_j, \forall j = 1, \ldots, r$;
3: $\text{forceJoin}(\Gamma_{\text{init}})$ for each output $j$
4: compute $newErr$;
5: $crtErr = \infty; \Gamma_{\text{new}} = \Gamma_{\text{init}}$
6: while $newErr < crtErr$ do
7: $crtErr = newErr$
8: call $HFA(\mathcal{T}_j, \Gamma_{\text{new}})$ for each output $j$
9: $\Gamma_{\text{new}} = \text{compute } \hat{\Gamma}$
10: $\text{forceJoin}(\Gamma_{\text{new}})$ for each output $j$
11: compute $newErr$;
12: end while
13: return $\Gamma_{\text{new}}$
14: end function
As in the case of a single output (cf. Chapter 6), for multiple outputs the approximation error can be reduced using \( K > 1 \) hinges and estimating output \( y_j \) as \( \hat{y}_j = \sum_{k=1}^{K} h_{j[k]}(x) \). To this end, for each hinge \( h_{j[k]}(x) \) a temporary data set is computed that contains as output values the residuals that are not yet fitted by all other hinge functions:

\[
\mathcal{T}_k = \left\{ (x_1, \ldots, x_m, y_1[k], \ldots, y_r[k]) \middle| (x_1, \ldots, x_m, y_1, \ldots, y_r) \in \mathcal{T} \right\},
\]

where \( y_{j[k]} = y_j - \sum_{i=1, i \neq k}^{K} h_{j[i]}(x) \) for \( j = 1 \ldots r \), and \( h_{j[i]}(x) \) is initially set to zero for all outputs and all hinges.

The main steps of the algorithm to fit \( K \) hinges to a given data set \( \mathcal{T} \) are summarized in Algorithm 10. The initial separator of the input space (cf. line 3) has a random normal vector, and contains the mean vector of all input vectors in \( \mathcal{T} \). The method \texttt{FindConsensusSeparator} is listed in Algorithm 11. Imposing the continuity, \texttt{forceJoin}(\( \Gamma \)) (cf. line 3) is done as described in Section 7.3, and \( HFA(\mathcal{T}_j, \Gamma_{\text{new}}) \) (cf. line 8) performs the hinge finding Algorithm 8 presented in Chapter 6 on \( \mathcal{T}_j \) using \( \Gamma_{\text{new}} \) as initialization.

**Complexity Analysis**

The main building block of mHFA is the HFA, which iterates until the error no longer decreases. Let \( \iota_1 \) and \( \iota_2 \) be the number of random initialization for HFA and mHFA, respectively, and \( \iota_{HFA} \) and \( \iota_{mHFA} \) the corresponding number of iterations until convergence. One such HFA iteration consists of fitting two regression hyperplanes and recomputing the two partitions \( \mathcal{T}^+ \) and \( \mathcal{T}^- \), and has a runtime complexity of \( O(|\mathcal{T}|^2 \cdot d + d^3) \). In total the runtime of HFA for one hinge is \( O(\iota_1 \cdot \iota_{HFA} \cdot (|\mathcal{T}|^2 \cdot d + d^3)) \). The mHFA algorithm first performs the HFA for each output independently, then finds a consensus separator and forces it to all outputs. Hence the runtime complexity is \( O(\iota_2 \cdot \iota_{mHFA} \cdot (r \cdot O(HFA) + 2^r + |\mathcal{T}|^2 \cdot d)) \). The exponential term is due to the binary integer program in the computation of the consensus hinge. Hence, the scalability of mHFA depends on three factors: dataset size, input
dimensionality, and output dimensionality. All of these aspects are investigated in Section 7.5. For the optimizations the high performance Gurobi solver [GO12] was employed for the experimental evaluations.

Trade-off between Quality, Compactness, and Runtime

There are two trade-offs which the user has to consider when intending to use HFA or mHFA. The first one is the trade-off between the quality and the compactness of a model. It generally holds, that the quality increases with more hinges at the cost of a poorer compactness. The second trade-off concerns the runtime complexity of building a model versus the quality of a model. This trade-off can be steered by the number of random initializations. The more noise or nonlinearity the data contains, the more probable it gets for the algorithms to get stuck in a local optimum. With more random initializations the probability of achieving a better result increases.

7.5 Experiments

In this section the mHFA algorithm is investigated in comparison with the HFA. First, it is showed that by imposing the constraint of a common partitioning in the input space, the quality of the resulting models is not considerably affected. Second, the scalability of mHFA is investigated w.r.t. different aspects.

For evaluating the model quality following two datasets are considered: the DIESEL dataset (introduced in Section 1.1) and the CONCRETE SLUMP dataset [Yeh07]. The CONCRETE SLUMP dataset contains 103 measured points, with 7 input dimensions and 3 output dimensions. The aim is to obtain from these measurements a material workability behavior model to predict the concrete slump, flow, and strength. To evaluate the quality of a constructed model the error as defined in Equation 7.16 is used. In the following experimental evaluation, \( \iota_1 = \iota_2 = 100 \) random initializations were performed and at the end the model with the lowest error was chosen. In all experiments 10-fold cross validation was computed.
Figure 7.6: Error and runtime measurements for the DIESEL dataset

Figure 7.6 illustrates the error of the different outputs for the DIESEL dataset, for the model constructed with mHFA compared to the models independently constructed with HFA. As expected, by forcing a common partitioning of the input space, mHFA obtains an increased error compared with HFA for every single output, since it has more constraints. This increase in error, though, turned out not to be very high. The lower right image illustrates the mean runtime of the two algorithms for one random initialization, ranging from 0.1 seconds for one hinge up to 3 seconds for three hinges (for mHFA).

Figure 7.7 illustrates the fitted models for 2 input dimensions (for $U_{FMI} = 10$). The model for each output is plotted separately, with two, three, and four hinges. These images show, that the partitioning of the input space is the same for each model, only the linear equations for each output differ. The more hinges are used, the better the model fits the data.

Figure 7.8 compares the errors of the computed models for the CONCRETE SLUMP dataset. For the slump flow the RMSE results that were reported in [Yeh07] for artificial neural networks and second order regression are
additionally plotted. For comparability, the root mean square error was used for the experiments:

\[ \text{RMSE} = \frac{1}{|\mathcal{T}|} \sqrt{\sum_{i=1}^{|\mathcal{T}|} (y_i - \hat{y}_i)^2}. \]

The artificial neural network has a smaller RMSE than both HFA and the mHFA. Although they have a higher prediction accuracy, they do not generate intelligible models but rather prediction functions which act as black
Figure 7.8: Error measurements for the SLUMP dataset

boxes. The model built with second order regression has a lower prediction accuracy than both hinge approaches, since it has less freedom degrees. Taking a closer look at the slump output and the flow output, one can see that in the test phase the RMSE is higher for the separately computed hinges. This is because of the generalization effect, which occurs when all outputs are simultaneously considered. Hence, overfitting is avoided as opposed to models for single noisy outputs.

Another interesting aspect is number $\kappa$ of required initializations. To empirically investigate the convergence of mHFA, in Figure 7.9 the resulting RMSE values are plotted for several random initializations. In the case of the DIESEL dataset, because of its nonlinearity, the more hinges are used, the faster mHFA converges. In the case of the CONCRETE SLUMP dataset, mHFA converges faster even with only one hinge. Choosing $\kappa$ generally depends on the linearity of the dataset, and on the number of desired hinges.

To evaluate the scalability of mHFA w.r.t. the dataset size, input and output dimensionality, synthetically generated data sets were used. The underlying functions are piecewise linear, based on 5 randomly generated hinge
functions. These continuous functions were sampled and a normally distributed noise $\mathcal{N}(0, 0.1)$ was added. To investigate the scalability of mHFA w.r.t. the dataset size, datasets with an output dimensionality of 3 and a varying input dimensionality ranging from 3 to 7 were generated. Figure 7.10 illustrates the runtime of mHFA averaged over 10 runs. The highest influence
on the runtime is the number of hinges fitted, because of the refitting performed after each fitted hinge. An increasing input dimensionality also has an influence on the runtime for larger data sets.

To evaluate the scalability of mHFA w.r.t. an increasing output dimensionality, datasets of size 10,000 were generated as described above, with an input dimensionality of 3, 4, 5, and 7, and an output dimensionality ranging from 2 to 30. The results are plotted in the lower right image in Figure 7.10. While the worst case runtime complexity is exponential with the output dimensionality, this is not observed in the experiments. One reason is the highly optimized Gurobi solver. Another reason is that correlated output dimensions cause the mHFA algorithm to converge faster.

7.6 Conclusion

Hinging hyperplanes yield accurate continuous piecewise linear models for datasets with a single output variable. This chapter introduced the mHFA approach that builds multivariate continuous piecewise linear models with a common partitioning of the input space for all output variables. In contrast to the naive approach, where the number of partitions quickly grows with the number of predicted variables, mHFA maintains the low model complexity. Common hinges are enforced and the continuity of the resulting functions is restored for each predicted variable. In summary mHFA yields compact and intelligible models with continuous functions and low approximation errors.
Chapter 8

Efficient Initialization of the Hinge Finding Algorithm

Hinging hyperplane models are well suited to represent continuous piece-wise linear models, but the hinge finding algorithm is guaranteed to converge only in local optima, and hence heavily depends on the initialization. In this chapter the problem of computing a meaningful initialization for the hinge finding algorithm is investigated, replacing the several random initializations proposed in the literature with a single initialization. To this end, the geometrical information of the regression surface is considered, more precise the direction with the highest curvature, and used to compute a meaningful initial hinge.

8.1 Introduction

In [LLC00] Li et al. proposed a regression tree, whose splitting strategy considers the geometric information of the regression surface. The idea is to project the data on the direction where the surface bends the most, defined by the principal Hessian direction, and perform the split there (see Section 3.3). This idea is illustrated in Figure 8.1. In the left image a 3-dimensional dataset is plotted. The bending direction of the underlying surface is well captured by the principal Hessian direction. In the right image the projec-
Figure 8.1: Example of a data set with the pHd describing the direction with the highest curvature and the projection of the samples along this direction shows that it is much easier to split the dataset along this dimension.

The drawback of the PHDRT is the split point selection, which aims at minimizing the difference between the standard deviations of the residuals in the two partitions. This is not always the best strategy, since it assumes a constant error over the entire dataset. In order to find the best split, the actual objective should be to minimize the sum of the two standard deviations. The training set is recursively partitioned, and this procedure is repeated. The continuity of the computed model is not imposed, although it is assumed for the unknown function when computing the Hessian matrix.

In this chapter an improvement of the hinge finding algorithm (cf. Algorithm 8) is proposed, by making use of this geometric information of the regression surface for finding a good initialization instead of several random initializations and efficiently compute a continuous model of the data.

8.2 Efficient Initialization of a Single Hinge

Instead of a random initialization for the HFA with a random hinge $\triangle^{(0)}$, the initial partitioning in $S^+$ and $S^-$ is computed according to a separating hyperplane in the input space, which is further called separator and denoted with $\delta^{(S)} \in \mathbb{R}^m$. To compute $\delta^{(S)}$, the average Hessian matrix $\bar{H}_x^{(S)}$ is approx-
8.2. Efficient Initialization of a Single Hinge

Figure 8.2: Computing a separator in the input space from a partitioning in the full space

imated for the data in \( S \), as presented in Section 2.3, and the corresponding pHd \( e_1^{(S)} \in \mathbb{R}^m \) computed.

The projected sample set onto the first pHd is \( S' \). On this dataset, the HFA is used to estimate a hinge function \( h^{(S')} \), whose hinge \( \Delta^{(S')} \) separates \( S' \) in two partitions: \( S'^+ \) and \( S'^- \). Hereby also a separation of \( S \) is obtained in \( S^+ \) and \( S^- \).

If only the input space of these two partitions is considered, \( S^+_x = \{ x_i : (x_i, y_i) \in S^+ \} \) and \( S^-_x = \{ x_i : (x_i, y_i) \in S^- \} \), a linear separator \( \delta^{(S)} \in \mathbb{R}^m \) can be computed in the input space by using the support vector machines (e.g. the efficient SMO proposed by Platt in [Pla99]) with a linear kernel. This separator is an approximative projection along the largest curvature in the input space, and thus a good initialization for the HFA, which is guaranteed to converge in a local optimum.

For the example in Figure 8.1, the split point in the projected space generates the partitioning in the full space as shown in Figure 8.2 (left). The projection of this partitioning onto the input space is illustrated in Figure 8.2 (right) together with the resulting separator.
8.3 Fitting Several Hinges

After the first hinge has been fitted, computing the residuals and repeating the procedure would be the straightforward procedure. But, by computing the residuals new curvatures in the data are obtained, which are not related with the original data and might influence the result. An example is shown in Figure 8.3. On the left side the original data is plotted, together with a fitted hinge. On the right side, the resulting residuals are plotted showing an additional bending pattern. When several hinges are fitted these added bending patterns influence the pHd computation, leading to poor results. Hence, throughout the computations only the output values are used, and not the residuals after fitting one or more hinges.

Another option is to recursively divide the data and compute the hinges on the partitions separately. But in this case the continuity property of the entire hinging hyperplane model is lost. Figure 8.4 shows a simple example for this case. Hence, when a new hinge is to be computed, all partitions $S_j \subset \mathcal{T}$ induced by the previously computed hinges are considered. For each non-empty partition $S_j$ the procedure described above is conducted, and a separator $\delta^{(S_j)}$ obtained. Even though computed as separators for $S_j \subset \mathcal{T}$, $\delta^{(S_j)}$ can serve as well as separators for $\mathcal{T}$, obtaining $\mathcal{T}^+$ as:

$$\mathcal{T}^+ = \{(x, y) \in \mathcal{T} : x^T \delta^{(S_j)} \geq 0\},$$
8.4. Experiments

Figure 8.4: Recursively splitting the data set leads to the loss of the continuity property

and $\mathcal{T}^-$ as:

$$\mathcal{T}^- = \{ (x, y) \in \mathcal{T} : x^T \delta(S_j) < 0 \},$$

where $x = [1, x]$. The separators $\delta(S_j)$ are used to initialize the HFA, and at the end the model with the highest accuracy is chosen. Note that the number of initializations is limited by the number of partitions, and hence it is never very large.

In Figure 8.5 an example is shown on a 3-dimensional subset of the DIESelo3 dataset. In the upper left corner the projection of the data onto the pHd is plotted, together with the fitted hinge which determines the separator $\delta(\mathcal{T})$. In the lower left image the first fitted hinge on $\mathcal{T}$ is plotted. The upper right image shows the projection of the points in the $S_1$ (grey/right) partition on their pHd, and the fitted hinge which determines $\delta(S_1)$. The lower right image shows the resulting model of the data with 3 hinges.

8.4 Experiments

This section evaluates this novel method for the initialization of HFA compared with the HFA introduced in [Bre93], showing that comparable or better models are obtained in a more efficient manner. The quality of a model is measured, by looking at the mean normalized error percentage:

$$error = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - f(x_i)|}{y_{max} - y_{min}} \cdot 100,$$
where \( f(x) \) is the linear regression of \( y \) against \( x \), \( y_{\min} = \min_{(x_i, y_i) \in T} \{ y_i \} \), and \( y_{\max} = \max_{(x_i, y_i) \in T} \{ y_i \} \).

In Figure 8.6 the results on the 3-dimensional WIND TURBINE dataset (introduced in Section 1.1) are presented. The model errors are plotted together with the corresponding runtime of the model computation. The lower plot shows the obtained errors of HFA for an increasing number of random initializations (runs). The bar chart on its right shows the error obtained with the improved initialization. The error of the latter is comparable with the error obtained by HFA after around 20 runs. In the upper part of the diagram the corresponding runtime of the model computation is plotted. In the case of random initializations, the runtime increases linearly with the number of initializations. The bar chart on the right shows the runtime required with the improved initialization. For 1 hinge the proposed method obtained...
8.4. Experiments

Figure 8.6: Error and runtime measurements for the Wind Turbine dataset

In around 2 seconds an error close to the one obtained by the HFA in 8.4 seconds. For 3 hinges the proposed method obtained in 8.4 seconds an error close to the one obtained by HFA in 67 seconds.

In Figure 8.7 the results on the Diesel dataset are illustrated. For this dataset, introduced in Section 1.1, the second and the third outputs are considered separately in DIESELO2 and DIESELO3. As for the Wind Turbine dataset, the model errors are plotted together with the corresponding runtime of the model computation. In the lower parts of the diagrams the error of the proposed method plotted, which is comparable with the error obtained by HFA after at around 10 iterations. The upper parts of the diagrams contain the runtime for the model computations. For 1 hinge the proposed method obtained on the DIESELO2 dataset in 0.24 seconds an error equal to the one obtained by HFA in 0.35 seconds, and for 3 hinges it obtained in 0.8 seconds.
Figure 8.7: Error and runtime measurements for the DIESELO2 and DIESELO3 datasets
8.4. Experiments

Figure 8.8: Hinging hyperplane model with the pHd-based initialization on the WIND TURBINE data set. An error equal to the one obtained by HFA in 2.7 seconds. Very similar results were obtained on the DIESEL03 dataset.

Not only that the computation becomes more efficient, but the quality of the models is improved. This is especially visible on the rather complex surface of the WIND TURBINE dataset. The resulting models with three and four hinges are illustrated in Figure 8.8. Compared to the hinging hyperplane
models from Figure 6.6 (on page 108), one can see that the splits are better adapted to the form of the surface. The proposed method has obviously a plus in efficiency compared to the HFA on complex surfaces. To obtain a more complex regression surface of a dataset, a higher number of random initializations for the HFA is required. This also implies a higher runtime. The \textsc{wind turbine} dataset is more complex, hence up to 30 runs are required to obtain a good model, and therefore the efficiency gain of our proposed method is higher.

8.5 Conclusion

This chapter investigated the use of the principal Hessian direction for improving the efficiency of the HFA, by using a suitable initialization. In the evaluations, a comparable accuracy is achieved in a more efficient manner. The improved method can be used, similarly as the HFA, for prediction tasks or for building regression trees. It can be also plugged into the mHFA algorithm, proposed in Chapter 7, for an increased efficiency in building multivariate regression trees.
Part IV

Prediction
Chapter 9

Reverse Engineering in Scientific Databases with Prediction Models

Using continuous models in scientific databases has received an increased attention in the last years. It allows for a more efficient and accurate querying, as well as predictions of the outputs even where no measurements were performed. The most common queries are on how the output looks like for a given input setting. This chapter investigates inverse model based queries on continuous models, where one specifies a desired output and searches for the appropriate input setting, which falls into the reverse engineering category. Two possible solutions are proposed: the first one is based on inverse regression (IR) and the second one is based on restraint optimization (RO). In the case of the inverse regression, imply switching the roles of input and output variables violates the assumption that the input variables are independent. Hence, the identification of an arbitrary oriented input subspace with independent dimensions is required. These two approaches are closely related, but the latter is more general. It facilitates the formulation of a wide range of queries, with specifications of fixed values and ranges in both input and output space, enabling the intuitive exploration of the experimental data and understanding the underlying process.
9.1 Introduction

As discussed in the previous chapters, prediction models are widely used in many areas. These models can be used for predicting output values for given input values, but also to explore the space of the process variables for a better understanding. The increasing need of scientific databases to work with continuous values instead of discrete observations, has been subject of recent research. Using statistical models for predicting continuous data has been discussed in [DGM⁺04, DGM⁺05]. By performing queries directly on the model instead of the raw data a more robust interpretation is provided, noise is removed, and missing values are filled up. However, the addressed queries are on how the output looks like for a given input setting. The reverse problem though, is also of interest: which input settings are required, in order to obtain a desired output. This reverse management problem has been discussed in [MGS11], and the so-called how-to queries defined. In [MS12] the authors present a how-to query engine for relational databases.

This chapter extends this type of queries on scientific databases with continuous models, which up to now only allow querying the output space by specifying input variables. Two solutions are proposed for these reverse model-based queries. The first one is based on inverse regression (IR), which is intuitive and efficient. The second one is based on restraint optimization (RO), and allows for more complex queries. Showing that the two approaches are closely related enforces the motivation for using the RO-based approach, trading efficiency against a higher flexibility. Finally, a series of model-based queries are described which are made possible.

Use Case: Research of the Fuel Injection Process

To illustrate the benefits of the inverse prediction queries, we consider a scientific application from the mechanical engineering field, namely the fuel injection process in the engine introduced in Section 1.1. A first impression of the process is gained by computing a regression model from this data, explaining the outputs (spray features) as functions of the inputs (experiment
settings). With this function one can predict how the spray features would look like for input settings which have not been measured, by simply using the computed model. However, for investigating the fuel injection process inverse queries of the type: which input parameters can yield a desired shape of the spray? are also of interest, e.g.:

- How should the chamber pressure and injection pressure be set, to obtain a penetration depth of 5:3?
- How should the chamber pressure and injection pressure be set, to obtain a penetration depth of 5:3, a spray width of 2:1, and an area of 23:7?

These inverse prediction queries allow for a better investigation of the experimental measurements and underlying process.

9.2 Related Work

In [CNS02] the authors introduce a scheme that allows for commercial database systems, which can either build or load data models for classification tasks, to operate on these models inside the database system. Since in many research areas a continuous process is investigated by acquiring a discrete set of measurements, the need for using statistical models for predicting continuous data has been discussed in [DGM+04, DGM+05]. Instead of directly performing queries on a sensor network, the authors propose to build a model from the readings, which is a more robust interpretation of the process, and use it to answer queries. With the help of a probabilistic model selection queries, range queries, and grouped aggregates over the values of sensor measurements are supported. Desphande et al. introduce in [DM06] the architecture of a data management system, which fundamentally integrates probabilistic and statistical models into database systems by defining the so-called model-based views. A declarative language for defining these views and declarative SQL queries over these views are supported. In [TM08] the FunctionDB database system is presented which deals with continuous
instead of discrete observations, and use an extended version of SQL that
allows querying this type of data. The algebraic query processor performs
queries directly onto mathematical functions, gaining both efficiency and
accuracy. Users can manage models like any other data, possessing the ad-
vantages of declarative queries and integration with existing database data.

Reverse queries, dealing with how the input should be in order to achieve
the desired output, are proposed in [MS12]. The rules and constraints are
translated into a mixed integer program and solved. The main difference to
the solutions presented in this chapter is that Tiresias is designed to work
with discrete tuples in relational databases.

Constraint databases on the other hand, deal with infinite relations and
algebraic query processing [Rev95, HF94], but search for closed form solu-
tions. In the presented solutions a single answer is searched for, which fits the
user-defined constraints best. The problem is reduced to a linear or quadratic
optimization problem which can be efficiently solved by an optimizer.

9.3 Problem Description

Throughout this chapter a data set \( \mathcal{D} \subset \mathbb{R}^{d+r} \) is considered, where an obser-
vation \((x, y) \in \mathcal{D} \) consists of an input vector \( x = [x_1, ..., x_d] \), and an output
vector \( y = [y_1, ..., y_r] \). Without loss of generality it is assumed that both inputs
and outputs are zero-mean variables and have unit standard deviation, i.e.
\( \mu(x_i) = 0, \sigma(x_i) = 1, \forall i \in \{1, ..., d\}, \) and \( \mu(y_j) = 0, \sigma(y_j) = 1, \forall j \in \{1, ..., r\} \).
According to the way the data emerges, it seems natural to build models
with the inputs as independent variables and outputs as dependent variables.
First, linear models are considered: each output dimension \( j \in \{1, ..., r\} \) be-
ing described by a linear equation: \( \sum_{i=1}^{d} a_{ij} x_i + a_{0j} = y_j \). Note that, since
the data is centered it holds that \( a_{0j} = 0, \forall j \in \{1, ..., r\} \). In Section 9.6 the
approach is extended to non-linear models.

These models can be used to predict an output for a given input, and to
perform diverse SELECT queries in the output space. But the interest lies in
making inverse predictions, i.e. to predict the input for a given output. First,
th computation of simple queries is presented, such as:

\[
\text{SELECT } m.x_1, m.x_2 \text{ FROM model AS m}
\]

\[
\text{WHERE } m.y_1 = c_1 \text{ AND } m.y_2 = c_2 \text{ AND } m.y_3 = c_3
\]

which are then extended in Section 9.6 to more complex ones, with range specifications and weights for each specification.

Since the above mentioned functions are generally not invertible, they cannot always directly be used for inverse prediction. The most straightforward solution is to build the models inversely, i.e. to set the outputs as independent variables and the inputs as dependent variables and perform linear regression. But the assumption, that the input variables (the measured outputs) are independent, is often not fulfilled in practical applications. A short example is shown in Figure 9.1, where for one input variable two output variables have been measured. In the right image the input and output dimensions are switched and the same points are plotted, where the linear dependency between the new input dimensions becomes clear. Therefore, in the inverse regression (IR) approach the linear dependencies must be first removed, which is the content of Section 9.4. Another alternative is to consider the above mentioned functions, with the inputs as independent variables and outputs as dependent variables, and use them for inverse regression. This solution is presented in Section 9.5.

### 9.4 IR-based Approach

For querying the input space according to some restraints in the output space, inverse regression (IR) is a straightforward solution, building linear regression models which explain the inputs in dependence of the outputs. As already mentioned in Section 9.3, the outputs might be linearly dependent. While linear regression assumes their independence, it fails fitting a \(d\)-dimensional regression hyperplane. In this case, the intrinsic linear independence in the output space has to be identified and removed.

To solve this problem, partial least squares (PLS), described in Section 2.2, can be employed, by first projecting the data onto a small number \(\kappa\) of
latent vectors, and then the performing regression. An appropriate \( \kappa < d \) is chosen such that at least a given fraction \( \varphi \in [0,1] \) of the explainable output variance is covered:

\[
    r = \min_{1 \leq i \leq d} \frac{\sigma_0^2 - \sigma_e^2}{\sigma_0^2 - \sigma_q^2} \geq \varphi,  \tag{9.1}
\]

where \( \sigma_0 \) is the variance in the output dimension in the original data, and \( \sigma_e \) is the variance in the output dimensions after the \( e \)th iteration, i.e. the variance of the residuals after a linear model has been fitted in \( e \) dimensions. The final number of latent vectors of a cluster is therefore depending on the data and a threshold \( \varphi \) specified by the user.

Let’s consider again the example from Figure 9.1, where \( y_1 \) and \( y_2 \) were linearly dependent. When running PLS with the outputs as independent variables, and the inputs as dependent variables, it identifies the latent vector \( q \) in the output space, and the 2-dimensional outputs are projected onto this vector, as shown in Figure 9.2 (left). Then, \( x \) is regressed on \( q \) as illustrated in Figure 9.2 (right). Hence, to perform inverse prediction for an arbitrary point in the \( (y_1, y_2) \)-space, this point is first projected onto the latent vector \( q \) and the projection is set into the linear equation.
9.5 RO-based Approach

Alternatively, inverse queries can be formulated as restraint optimization (RO) problems, based on the original linear regression models, which describe the outputs as linear combination of the inputs. After describing the RO-based approach, it is shown that in fact it is closely related to the IR-based approach, but at the same time allowing for the formulation of more complex queries.

Restraint Optimization

The RO-based approach uses the functions obtained by regressing $y$ on $x$ for inverse regression. For each output $y_j, j \in \{1, ..., r\}$, a function $f_j : \mathbb{R}^d \to \mathbb{R}$ defines a plane in $\mathbb{R}^{d+1}$. A restraint in the $j$-th output dimension is formally defined as a constant: $f_j(x) = c_j$, and obtain a $(d - 1)$-dimensional hyperplane $\gamma_j$ in $\mathbb{R}^d$, the input space. For a query the set of restraints in the output space is denoted as $C$. Since, as shown later, a restraint can be also formulated as a range or a bound, there is no need to specify more than one restraint for each output dimension. The query answer is the solution of the following equation system: $\sum_{i=1}^{d} a_{ij} x_i = c_j, \forall c_j \in C$. If $|C| = d$, the equa-
tion system has as many equations as unknown variables and there exists an exact solution. If $|C| < m$, the equation system is underdetermined, and all points lying on a $(d - |C|)$-dimensional hyperplane are valid query answers. If $|C| > d$, the equation system is overdetermined, and the existence of a solution is not guaranteed. This is the most interesting case, on which the rest of this chapter will focus.

The $|C|$ hyperplanes in the input space most probably will not intersect in a single point. Even if there exists no exact solution to the query, the setting in the input space is required, which satisfies all user defined restraints as well as possible in a least squares sense. Let $n_j \in \mathbb{R}^d$ be the normal of $\gamma_j$ and $o_j$ its offset, the distance of a point $p \in \mathbb{R}^d$ to $\gamma_j$ is computed as $d(p, \gamma_j) = n_j^T + o_j$.

The best possible solution in the input space is defined as the point with a minimal sum of squared distances to $\gamma_j$, $\forall c_j \in C$:

$$\min_{p \in \mathbb{R}^d} \sum_{c_j \in C} (n_j^T + o_j)^2$$  \hspace{1cm} (9.2)

Figure 9.3 (left) illustrates this idea for a 2-dimensional input space and 3 restraints $c_1$, $c_2$, and $c_3$. Each restraint is a line in the input space, obtained as the projection of the intersection of two planes in $\mathbb{R}^3$. The three lines $\gamma_1$, $\gamma_2$, and $\gamma_3$ are plotted in Figure 9.3 (right), and the optimal setting in the least squares sense is the point.

In the RO-based approach it is distinguished between the notions of restraint and constraint. While restraints refer to user preferred output values, which may be overdetermined and can only be achieved in a soft (i.e. least squares) sense, constraints refer to hard conditions specifying which solutions are possible.

**Equivalence between IR and RO**

Next it is shown that RO-based approach is equivalent to the PLS-based approach in the noise free case, for $d = 2$ and $|C| = 3$, i.e. $r = 3$. The idea of this proof can be extended for arbitrary $d$ and $|C| > d$. In Section 9.7 it is empirically shown that for higher $d$ and $r$ values the two approaches deliver,
9.5. RO-based Approach

Figure 9.3: (Left) Three user defined restraints $c_1$, $c_2$, and $c_3$ (Right) The corresponding optimization problem in the input space.

up to a very small numerical error, the same result.

Generally, the IR-based approach computes and uses following linear models:

$$
\sum_{j=1}^{r} y_j^T q_{ij} b_{ij} = x_i, \forall i \in \{1, ..., d\},
$$

(9.3)

where $y_T = [y_1, ..., y_r]$. The RO-based approach uses following linear models:

$$
\sum_{i=1}^{d} x_i a_{ji} = y_j, \forall j \in \{1, ..., r\}
$$

(9.4)

For $d = r$ there are in both Equation 9.3 and Equation 9.4 as many unknown variables as equations. The two types of linear models, from Equation 9.3 and 9.4, can be obviously transformed into each other, and therefore possess the same unique solution. Hence the RO-based approach and the PLS-based approach deliver the same solution in this case.

Let’s consider the case of $m = 2$ and $r = 3$, where the data is noise free and the three outputs thus linearly dependent. The task, as defined in Section 9.3, is to find an appropriate input setting such that the output values are as close as possible to the values $c_1$, $c_2$, and $c_3$.

First, the geometrical interpretation of the two approaches is intuitively
explained. PLS, as explained in Section 2.2, identifies the plane in the output space spanned by the two first latent vectors $q_1$ and $q_2$, on which all outputs in $D$ lie.

With a linear independent (2-dimensional) output space ordinary least squares can be performed. An arbitrary point in the output space $c^T = [c_1, c_2, c_3]$ is first projected onto this 2-dimensional plane: $c' = c^T[q_1, q_2] \in \mathbb{R}^2$, where an input is predicted via the regression plane. Figure 9.4 (left) illustrates this idea: there is a plane in the output space and the point $p_1$ which lies on it is a unique solution for Equation 9.3. The points $p_2$ and $p_3$ are first projected on this plane, and since their projection corresponds to $p_1$, the same input setting is returned as solution.

If the approach based on restraint optimization is considered, the Equation 9.4 provides one linear equation in output space by Gauss elimination, containing all unique solutions. Assuming normalized coefficients, following is obtained:

$$n_1y_1 + n_2y_2 + n_3y_3 - o = 0,$$

which is the Hessian form of the plane in output space on which all observations lie. Note that the same plane in output space is spanned by the two
latent vectors $q_1$ and $q_2$ computed by PLS. Let’s consider Equation 9.4:

$$G(x_1, x_2) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \\ f_3(x_1, x_2) \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{21}x_2 \\ a_{12}x_1 + a_{22}x_2 \\ a_{13}x_1 + a_{23}x_2 \end{pmatrix},$$

the Jacobian matrix $J_G$ describes the orientation of the tangent plane to this function:

$$J_G(x_1, x_2) = \begin{pmatrix} \frac{\partial G}{\partial x_1} \\ \frac{\partial G}{\partial x_2} \end{pmatrix}$$

Since $G(x_1, x_2)$ is a plane, $J_G \in \mathbb{R}^{3 \times 2}$ describes this plane. The two vectors lying in this plane are:

$$\begin{align*}
\frac{\partial G}{\partial x_1} &= \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix}, \\
\frac{\partial G}{\partial x_2} &= \begin{pmatrix} a_{21} \\ a_{22} \\ a_{23} \end{pmatrix}
\end{align*}$$

and the normal $n = [n_1, n_2, n_3]^T$ of the plane is their crossproduct:

$$n = \frac{1}{\| \frac{\partial G}{\partial x_1} \times \frac{\partial G}{\partial x_2} \|} \begin{pmatrix} a_{12}a_{23} - a_{13}a_{22} \\ a_{13}a_{21} - a_{11}a_{23} \\ a_{11}a_{22} - a_{12}a_{21} \end{pmatrix} \quad (9.6)$$

Both approaches obtain the same plane in output space, described by the normal $n$, and all outputs lying on this plane are images of both Equation 9.3 and Equation 9.4. Hence, if the user-defined output lies on this plane, then both approaches deliver the same input setting. In the case where the user-defined output does not lie on this plane and therefore no unique solution exists, the PLS-based approach projects an arbitrary point in the output space onto this plane of unique solutions, while with the RO approach this point is translated into a set of restraints.
Further it is shown that even for arbitrary points in the output space the two approaches deliver the same result (for $d = 2$ and $r = 3$), by showing that all points along a projection generate sets of restraints which have the same solution. Since the PLS-based approach projects this arbitrary point onto the plane of unique solutions, it remains to show that all points along a projection generate sets of restraints which have the same solution. This idea is illustrated in Figure 9.4: on the left the solution plane in the output space is plotted, together with three points $p_1$, $p_2$, and $p_3$. The point $p_1$ lies on the plane, and is the orthogonal projection of $p_2$ and $p_3$ onto the plane. In the right image, the restraints corresponding to these three points are plotted. The (green) restraints of $p_1$ intersect in one point, which is a unique solution for Equation 9.4. Next it is shown that the (red) restraints of $p_2$, and (blue) restraints of $p_3$ have as optimal solution the same point. To this end, following lemma is required, which will be used in the proof.

**Lemma 9.1.** Given the dataset $D \subset \mathbb{R}^{d+1}$, with inputs $x \in \mathbb{R}^d$ and outputs $y \in \mathbb{R}$, let $\alpha_0, \alpha_1, \ldots, \alpha_d$ be the regression coefficients obtained by least squares. For the z-score normalized data, i.e. centered and scaled to have unit standard deviation, for the corresponding standardized regression coefficients $a_0, \ldots, a_d$ it holds that:

$$a_0 = 0 \quad \text{and} \quad a_1^2 + a_2^2 + \ldots + a_d^2 = 1.$$

**Proof.** The standardized regression coefficients are (see [DS98] for details):

$$a_0 = 0 \quad \text{and} \quad a_j = \alpha_j \frac{\sigma_{x_j}}{\sigma_y}.$$

For the variance of the standardized output it holds that:

$$\text{Var}(Z(y)) = \text{Var}(a_0 + a_1 Z(x_1) + \ldots + a_d Z(x_d) + \epsilon) = 1,$$

hence:

$$a_1^2 \text{Var}(Z(x_1)) + \ldots + a_d^2 \text{Var}(Z(x_d)) + \text{Var}(\epsilon) = 1.$$

Since

$$\text{Var}(Z(x_j)) = 1, \forall j \in \{1, \ldots, d\},$$
and for $\epsilon = 0$, it holds that:

$$a_1^2 + ... + a_d^2 = 1.$$ 

\[\blacksquare\]

**Theorem 9.1.** Let $D = \{(x, y)\}_{i=1}^n$ with $x \in \mathbb{R}^2$, $y \in \mathbb{R}^3$ be a standardized data set, and let $p_1 \in \mathbb{R}^3$ be a point in output space for which there exists a unique solution $s^* \in \mathbb{R}^2$ of Equation 9.4. Then for each point $p_2 \in \mathbb{R}^3$, whose projection onto the plane of unique solutions is $p_1$, the RO-based approach delivers the same input setting $s = s^* \in \mathbb{R}^2$.

**Proof.** For an arbitrary point $p_2 = (c_1, c_2, c_3)$ the RO-based approach substitutes $c_j$ for $y_j$ in Equation 9.4, and following equations are obtained:

$$-c_j + a_{1j}x_1 + a_{2j}x_2 = 0, \ \forall j \in \{1, 2, 3\},$$

which correspond to lines in the input space.

Due to the data normalization $a_{1j}^2 + a_{2j}^2 = 1$ (cf. Lemma 9.1), these equations are in the Hesse normal form, and the distance of a point $s \in \mathbb{R}^2$ to these lines is:

$$d(s, \gamma_j) = -c_1 + a_{1j}s_1 + a_{2j}s_2, \ \forall j \in \{1, 2, 3\}.$$ 

According to the restraint optimization approach, the point in the input space is searched, which has a minimal sum of quadratic distances to these three lines:

$$\min_{s \in \mathbb{R}^2} f(s) = d(s, \gamma_1)^2 + d(s, \gamma_2)^2 + d(s, \gamma_3)^2$$

To optimize this objective function, the function $f$ is derived and the derivative is set to zero:

$$\begin{align*}
\frac{\partial f}{\partial s_1} &= \sum_{j=1}^{3} 2a_{1j}^2s_1 + 2a_{1j}a_{2j}s_2 - 2a_{1j}c_j = 0 \\
\frac{\partial f}{\partial s_2} &= \sum_{j=1}^{3} 2a_{2j}^2s_1 + 2a_{1j}a_{2j}s_2 - 2a_{2j}c_j = 0
\end{align*}$$
By rearranging terms the following equations are obtained:

\[
\begin{align*}
\frac{3}{s_1} \sum_{j=1}^{3} a_{1j}^2 + s_2 \sum_{j=1}^{3} a_{1j}a_{2j} - \sum_{j=1}^{3} a_{1j}c_j &= 0 \\
\frac{3}{s_1} \sum_{j=1}^{3} a_{2j}^2 + s_2 \sum_{j=1}^{3} a_{1j}a_{2j} - \sum_{j=1}^{3} a_{2j}c_j &= 0
\end{align*}
\]

Having two equations with two unknown variables, the following is obtained:

\[
s_1 = \frac{\sum_{j=1}^{3} a_{2j}^2 \sum_{j=1}^{3} a_{1j}c_j - \sum_{j=1}^{3} a_{1j}a_{2j} \sum_{j=1}^{3} a_{2j}c_j}{(\sum_{j=1}^{3} a_{1j}a_{2j})^2 - \sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{2j}^2}
\]

(9.7)

and

\[
s_2 = \frac{\sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{1j}a_{2j} - \sum_{j=1}^{3} a_{1j}a_{2j} \sum_{j=1}^{3} a_{1j}c_j}{(\sum_{j=1}^{3} a_{1j}a_{2j})^2 - \sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{2j}^2}
\]

(9.8)

Let's consider \( p_1 \in \mathbb{R}^3 \), the orthogonal projection of \( p_2 \) onto the plane of unique solutions, and let \( n = [n_1, n_2, n_3] \) be the normal of this plane then \( p_1 = (c_1 + \lambda n_1, c_2 + \lambda n_2, c_3 + \lambda n_3) \). The next step is to show that the result returned by the RO-based approach \( s \in \mathbb{R}^2 \) for \( p_2 \) is equal to the result \( s^* \in \mathbb{R}^2 \) returned for \( p_1 \). If the outputs are set equal to the new restraints, following equations are obtained:

\[-(c_j + \lambda n_j) + a_{1j}x_1 + a_{2j}x_2 = 0, \quad \forall j \in \{1, 2, 3\},\]

By repeating the steps as before \( s_1^* = s + \delta_1 \) and \( s_2^* = s_2 + \delta_2 \) are obtained, with:

\[
\delta_1 = \frac{\lambda \sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{1j}n_j - \sum_{j=1}^{3} a_{1j}a_{2j} \sum_{j=1}^{3} a_{2j}n_j}{(\sum_{j=1}^{3} a_{1j}a_{2j})^2 - \sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{2j}^2},
\]

and

\[
\delta_2 = \frac{\lambda \sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{1j}n_j - \sum_{j=1}^{3} a_{1j}a_{2j} \sum_{j=1}^{3} a_{1j}n_j}{(\sum_{j=1}^{3} a_{1j}a_{2j})^2 - \sum_{j=1}^{3} a_{1j}^2 \sum_{j=1}^{3} a_{2j}^2}.
\]

By showing that \( \delta_1 = 0 \) and \( \delta_2 = 0 \), it holds that \( s_1 = s_1^* \) and \( s_2 = s_2^* \), and hence the RO-based approach delivers the same result for \( p_1 \) and for \( p_2 \), just like the PLS-based approach.
9.5. RO-based Approach

Let's consider the numerator of $\delta_1$:

$$\sum_{j=1}^{3} a_{2j}^2 \sum_{j=1}^{3} a_{1j} n_j - \sum_{j=1}^{3} a_{1j} a_{2j} \sum_{j=1}^{3} a_{2j} n_j.$$ 

By rearranging the terms, the following is obtained:

$$n_1 (a_{11} a_{21}^2 + a_{11} a_{22}^2 + a_{11} a_{23}^2) - n_1 (a_{11} a_{21}^2 + a_{21} a_{12} a_{22} + a_{21} a_{13} a_{23}) + n_2 (a_{12} a_{22}^2 + a_{12} a_{21}^2 + a_{12} a_{23}^2) - n_2 (a_{12} a_{22}^2 + a_{22} a_{11} a_{21} + a_{22} a_{13} a_{23}) + n_3 (a_{13} a_{23}^2 + a_{13} a_{21}^2 + a_{13} a_{22}^2) - n_3 (a_{13} a_{23}^2 + a_{23} a_{12} a_{22} + a_{23} a_{12} a_{22})$$

and

$$n_1 (a_{22} (a_{11} a_{22} - a_{21} a_{12}) + a_{23} (a_{11} a_{23} - a_{21} a_{13})) + n_2 (a_{21} (a_{21} a_{12} - a_{22} a_{11}) + a_{23} (a_{21} a_{23} - a_{22} a_{13})) + n_3 (a_{21} (a_{21} a_{13} - a_{11} a_{23}) + a_{22} (a_{22} a_{13} - a_{21} a_{23})).$$

The terms in the parentheses are equal to the components of $n$ defined in Equation 9.6, and can be thus rewritten as:

$$n_1 (a_{22} n_3 - a_{23} n_2) + n_2 (a_{23} n_1 - a_{21} n_3) + n_3 (a_{21} n_2 - a_{22} n_1)$$

Note that the normalization factor of $n$ cancels out. By rearranging the terms, the numerator of $\delta_1$ is:

$$n_1 n_3 (a_{22} - a_{22}) + n_1 n_2 (a_{23} - a_{23}) + n_2 n_3 (a_{21} - a_{21}),$$

which is obviously zero. Hence, $\delta_1$ is also zero. Analogously it can be proven that $\delta_2 = 0$.

Summing up, both approaches have the same plane of unique solutions.
in the output space, and all outputs lying on this plane are images of both Equation 9.3 and Equation 9.4. Hence, if the user defined output lies on this plane, then both approaches deliver the same input setting. We have shown for \( m = 2 \) and \( r = 3 \), that for arbitrary points in the output space the two approaches deliver the same result by showing that all points along a projection generate sets of restraints which have the same solution.

### 9.6 Model-based Queries

While in the previous section it has been shown that the two approaches are closely related, this section introduces more complex queries, that can be formulated and solved with the RO-based approach. Instead of specifying a fixed value for each output, the user can specify a range, a lower bound or an upper bound. Such restraints and/or constraints can be also specified for the input dimensions. These new queries are defined as optimization problems, which can be efficiently solved.

**Range Restraints**

One or more restraints in the output space can be specified as a range:

```
SELECT m.x_1, m.x_2, m.x_3 FROM model AS m
WHERE m.y_1 \geq lb_1 \text{ AND } m.y_1 \leq ub_1
AND m.y_1 = c_1 \text{ AND } m.y_3 = c_3
```

If the projected restraints in the input space are considered, the problem is similar to the previous one, as shown in Figure 9.5 (left). Instead of one hyperplane in the input space \( \gamma_3 \), there are two parallel hyperplanes \( \gamma_{lb} \) and \( \gamma_{ub} \) for the output \( y_1 \). All points lying in between these two hyperplanes are considered to have a distance of zero to the range, like the point \( s_2 \) in Figure 9.5 (right). For the points outside the distance is defined as the shorter of the two distances to the lower and upper bound, as illustrated in Figure 9.5 (right) for the points \( s_1 \) and \( s_3 \). The distance of a point \( s \) to a range, described
Figure 9.5: (Left) The optimization problem in the input space with two fixed restraints and one range restraint defined by the user (Right) The distance of a point to a range restraint is zero, if the point lies within the range spanned by the projected restraints.

by two hyperplanes $\gamma_{lb}$ and $\gamma_{ub}$, is formally defined as:

$$d(s, [\gamma_{lb}, \gamma_{ub}])^2 = \frac{(|d(s, \gamma_{lb})| + |d(s, \gamma_{ub})| - w)^2}{4},$$

where $w$ is the distance between the two hyperplanes $\gamma_{lb}$ and $\gamma_{ub}$. To model the absolute value $|d(s, \gamma_{lb})|$ in the objective function a new continuous variable $\delta_{lb} = |d(s, \gamma_{lb})|$ is introduced, together with two constraints:

$$\delta_{lb} \geq d(s, \gamma_{lb}) \quad \text{and} \quad \delta_{lb} \geq -d(s, \gamma_{lb}),$$

and analogously $\delta_{ub}$ is introduced for $|d(s, \gamma_{ub})|$. Hence, Equation 9.9 becomes:

$$d(s, [\gamma_{lb}, \gamma_{ub}])^2 = \frac{(\delta_{lb} + \delta_{ub} - w)^2}{4},$$

The corresponding objective function is:

$$\min_{s \in \mathbb{R}^m} (d(s, \gamma_1)^2 + d(s, \gamma_2)^2 + d(s, [\gamma_{lb}, \gamma_{ub}])^2),$$
Figure 9.6: (Left) A range constraint in the input space forces the returned solution to have $s_2 \in [lb_2, ub_2]$ (Right) Example of the intersection of a hyper-plane $\gamma_j$ with the input space hyperbox.

with following linear inequality constrains:

$$\delta_{lb} \geq d(s, \gamma_{lb})$$
$$\delta_{ub} \geq -d(s, \gamma_{ub})$$

Bound restraints are a special case of range restraints, where the upper or lower bound are set to the observed minimum or maximum value in the respective dimension. Queries like:

```sql
SELECT m.x_1, m.x_2, m.x_3 FROM model AS m
WHERE m.y_1 \geq lb_1 AND m.y_2 = c_2 AND m.y_3 = c_3
```

are solved very similarly to the range queries by setting the other bound (lower or upper) to the minimum or maximum observed value in that output dimension.

**Constraints in the Input Space**

The user might be also interested in specifying fixed values or ranges for the input space. For example, in the following query a range for one of the input
dimensions is additionally specified:

\[
\text{SELECT } m.x_1, m.x_2, m.x_3 \text{ FROM model AS } m \\
\text{WHERE } m.y_1 \geq lb_1 \text{ AND } m.y_1 \leq ub_1 \\
\text{AND } m.y_2 = c_2 \text{ AND } m.y_3 = c_3 \\
\text{AND } m.x_2 \geq lb_{x2} \text{ AND } m.x_2 \leq ub_{x2}
\]

An important design decision is how these specifications are built into the optimization problem. One option is to add them as restraints into the objective function, in the same way as the specifications for the output dimensions. A second option is to add them as constraints for the objective function, forcing the result to completely fulfill them. In Figure 9.6 (left) an example is illustrated of a range constraint in the input space, which forces the optimal point to lie in the specified bound.

**Nonlinear Case**

To accurately capture real world data linear models typically do not suffice and piecewise (possibly higher order) models are used [BFSO84, Fri91, Kar92]. A regression tree is similarly build as a decision tree, having in each inner node a binary splitting criterion, e.g. a separating hyperplane, but in the leaves linear models instead of a class label. A specified restraint may be applicable for none, one, or several submodels. Hence, the restraint is applied to each linear submodel and afterwards it is checked whether it is valid or not. To do this, the restraint is intersected with each submodel’s hyperplane and this intersection is projected onto the input space. The restraint is valid if this projected hyperplane intersects the corresponding partition of the submodel.

As shown in Section 9.5, applying a restraint onto the \( j \)-th output yields a hyperplane \( \gamma_j \) in the input space \( \mathbb{R}^d \). Since in a regression tree each submodel has a corresponding partition in the input space, it is checked whether \( \gamma_j \) intersects this partition. If not, then the restraint is not valid for this submodel. To this end, for each input dimension a minimum and a maximum allowed value is considered. If these are not known, they can be set according to the values in the dataset, or very low and respectively very high. This
hypercube of allowed input values has $2^d$ vertices of the form $v = [v_1, ..., v_d]$, with $v_i \in \{x_{i,\text{min}}, x_{i,\text{max}}\}$ $\forall i \in \{1, ..., d\}$, and $d \cdot 2^{d-1}$ edges. An edge $e_{kl}$ connects two vertices $v_l$ and $v_k$ which differ in a single dimension $i$. Then, $\gamma_j$ is intersected with this hypercube by computing its intersection with each edge. The intersection point between $\gamma_j$ and $e_{kl}$ is $p = [v_1, ..., v^*_i, ..., v_d]$, with:

$$v^*_i = \left( n_1 v_1 + ... + n_{i-1} v_{i-1} + n_{i+1} v_{i+1} + n_d v_d + o_j \right) \frac{1}{n_i}.$$ 

An example of a 3-dimensional space intersected by a restraint $\gamma_j$ is shown in Figure 9.6 (right), where the intersection points with the edges are marked with red.

After determining these intersection points, it is checked whether these are contained in the submodel’s partition. This can be done by going down through the regression tree, applying the splitting rules, and check which partition has been reached. If at least one intersection point lies in the submodel’s partition, then the restraint is valid for it.

In the example from Figure 9.7 two submodels are illustrated, together with the corresponding partitions in the input space, and a restraint $c$ of the output. In Figure 9.7 (left) the restraint is applied to the red submodel, showing that the projected line intersects the corresponding partition and
is therefore valid. In Figure 9.7 (right) the restraint is applied to the blue model, showing that the projected line does not intersect to corresponding partition. Hence, only the restraint for the red submodel is considered for further computations.

In the next step, the restraint optimization is performed as in the linear case. Since such a linear submodel is valid only in a certain partition of the input space, the bounds of the corresponding submodel are added as additional constraints in the input space in the optimization problem. If more than one restraint is valid, then for each one a query result is computed separately. In this case several results to the query are available, which can be ranked according to the objective value. The smaller the objective value, the more interesting a suggested input setting might be.

**Weights**

In the RO-based approach weights can be applied for the restraints. The objective function from Equation 9.2 becomes then:

\[
\min_{s \in \mathbb{R}^d} \sum_{c_j \in C} w_j (n_j^T s + d_j)^2
\]

These weights can be user defined or problem specific. For example, weights corresponding to the slope of the regression hyperplane can be computed. In the optimization process the point is searched for, which lies as close as possible to all the hyperplanes in the input space. But such a distance in the input space implies different deviations from the specified restraint depending on the slope of the regression hyperplane. This idea is illustrated in Figure 9.8, where two hyperplanes with different slopes are plotted. The same distance \( \delta_x \) in the input space causes a different deviation \( \delta_y \) in the output space. The higher the slope of a hyperplane, the higher \( \delta_y \). The weight \( w_j \) is set directly proportional to the slope of the regression hyperplane of the \( j \)-th output. The slope of the regression hyperplane of the \( j \)-th output is assessed by considering the angle between its normal \( n_j \) and the vector defining the output direction \( \bar{y}_j = [0, \ldots, 0, 1] \), \( \bar{y}_j \in \mathbb{R}^{d+1} \). The higher the slope, the higher this
angle becomes, and therefore the higher its weight $w_j$ should be:

$$w_j = 1 - \cos(n_j, \vec{y}_j)$$

All these components presented in this section can be combined in several ways, offering the user wide range of options for mining experimental data.

**Underspecification**

The RO-based approach builds the model only once and uses it for all query types. If the user only wants to specify only few output restraints, only the corresponding models are considered. The PLS-based approach, on the other hand, requires a model with the specified output dimensions as input variables. Note that precomputing models with all possible subset combinations of output dimensions is only feasible for a very low output dimensionality.
9.7 Experiments

The described query methodologies are experimentally evaluated on both synthetic and real data sets. The equivalence between the two approaches is empirically quantified for the case of noise free and z-score standardized data, for varying input and output dimensionality, followed by the investigation of the behavior of the RO-based approach for varying input and output dimensionality. We then investigate the runtime of the two approaches, as well as their scaling properties for an increasing database size. Finally, the accuracy is evaluated on two real-world datasets: SPRAY and DIESEL.

9.7.1 Evaluation on Synthetic Data Sets

The synthetic data sets contain, if nothing else is specified, 10,000 z-score normalized objects generated by a random linear model and contain an uniformly distributed noise in the range of 0 – 7%. For each evaluation 1,000 queries were randomly chosen in the output space, and the results averaged over 10 iterations. A C++ prototype was constructed and the Gurobi optimizer [GO12] was used. The experiments were runed on a 2.67 GHz quad processor machine with 12 GB RAM.
Empirical Equivalence between RO-based and PLS-based Approach

In addition to the proof in Section 9.5, an empirical evidence is provided, that the two approaches deliver very similar results, for input dimensions higher than \( d = 2 \) and output dimensions higher than \( r = 3 \). Following experimental evaluation was performed: for each query both approaches were used to predict the input settings \( s \in \mathbb{R}^d \), and computed the relative difference between the two delivered results:

\[
\frac{\sum_{i=1}^{d} |s_i^{(RO)} - s_i^{(PLS)}|}{s_i^{(max)} - s_i^{(min)}} \cdot 100.
\]

The data sets have 2 to 30 input dimensions, and 3 to 40 output dimensions. The results are plotted in a matrix, in which the input dimensionality varies in the rows and output dimensionality varies in the columns. The cell color is formatted according to the contained value: the higher the value, the darker the color. This matrix is illustrated in Figure 9.9. The absolute difference between the results is always below 1\%, and the small errors and slightly growing with a growing number of input dimensions.

Runtime comparison

For the investigation of the runtime, three different output dimensionalities were chosen for the experiments: 60, 80, and 100; and varied the input dimensionality from 5 to 50. In Figure 9.10 the construction runtimes for the RO-based approach and for the for the IR-based approach are plotted. As expected the runtime of both approaches increases with an increasing input and output dimensionality, but the one for the IR-based approach is significantly lower. However, in the case of the RO-based approach the model is built only once, whereas for the IR-based approach it might be more often required to build a model, depending on the specified restraints. The IR-based approach requires a model with all the specified output dimensions as inputs, and if the output dimensionality is high it might not be feasible to compute and store models for all possible combinations of output dimensions.
9.7. Experiments

Figure 9.10: Runtime of the two approaches

Figure 9.11: Scalability of the two approaches with an increasing database size
The IR-based approach has a very efficient query processing, since it only needs to insert the restraint values into linear equations. So, if an IR model is available, then querying it is most efficient. The RO-based approach solves a quadratic optimization problem. Hence, the query runtime is investigated for the same settings as mentioned for the construction runtime comparison. The plotted results show that the query runtime is increasing with an increasing input and output dimensionality, but still is in the range of milliseconds and RO is more efficient than constructing a model with IR and afterwards predicting with it.

**Scaling of the construction runtime**

For analyzing how the construction runtime scales with an increasing database size, three settings were chosen: input dimensionality of 5 and output dimensionality of 25, input dimensionality of 25 and output dimensionality of 50, and input dimensionality of 50 and output dimensionality of 100; and the database size was varied from 10,000 to 100,000. The results are plotted in Figure 9.11. Both approaches scale linearly with an increasing database size, while the IR-based approach is more efficient. The query runtime is not influenced by the database size, since it operates on the available models.

**9.7.2 Evaluation on real world data sets**

The RO-based approach was evaluated on the SPRAY dataset and on the DIESEL dataset, both introduced in Section 1.1. For the SPRAY dataset the time was considered once as an input dimension and once as output dimension. In the first case, the dataset was decomposed into 4 data sets, each with 2 input dimensions: SPRAY#2 with time and chamber pressure as input, SPRAY#3 with time and gas temperature as input, SPRAY#4 with time and injection period as input, and SPRAY#5 with time and injection pressure as input. All these data sets have 3-dimensional output: penetration depth, spray width, area of the approximating ellipse. The model accuracies of these data sets is around 7%. The DIESEL dataset contains all input dimensions, the DIESEL#1 dataset contains only the first two input dimensions, and
the DIESEL #2 contains the only the first and the third input dimension. All of them contain all three output dimensions. The model accuracies are around 4%, 12%, and 7%. The models of these data sets are piecewise linear, computed with the hinging hyperplane algorithm from Chapter 6, and each have 2 linear models.

For each sample used as query, the outputs were considered as restraints for a query and performed inverse prediction with the RO-based approach without weights (simple RO), and with the weights depending on the slope of the regression hyperplanes (slope RO). In this experiment the IR-based approach was not considered, since it assumes only linear models. The results are plotted in Figure 9.12. The noise of the observations together with the
Reverse Engineering in Scientific Databases with Prediction Models

**Figure 9.14:** Prediction error on the Spray time series

**Figure 9.15:** Prediction error on the Spray time series

**Figure 9.16:** Prediction error on the Spray time series
slopes of the regression hyperplanes determine the prediction error. The results obtained with the slope weights are almost always better than without, and the improvements are better for higher slopes.

For the SPRAY dataset, the time can be also seen as an output dimension. Hence, the dataset contains 3-dimensional inputs and as outputs a time series for each spray feature, which were modeled with linear functions. For each observation the time series of different length were used (the first 5, 10, 15, 20, 25, and 30 time steps) as restraints. Each spray feature was considered separately, and all three together. In Figure 9.13 the results are plotted for the case where all three features are considered simultaneously. In Figure 9.14, Figure 9.15, and Figure 9.16 are the results illustrated for each spray feature separately. On these data sets, the PLS-based approach delivered better results, but especially in the case of higher dimensional outputs where the user specifies restraints only for a subset of outputs the models have to be build from the scratch for each query, which is more time consuming.

9.7.3 Conclusion

In this chapter the topic of inverse prediction on continuous models is discussed. In the case of experimental measurements mathematical functions are built, which describe the outputs as function of the inputs. Since these functions are seldom invertible, they cannot be directly used for inverse prediction. Two alternative ways to solve this problem are considered. In the IR-based approach the outputs are considered as independent variables and inputs as dependent variables. In this case partial least squares (PLS) is employed, in order to deal with the linear dependence in the output space. On the other hand, the RO-based approach uses the functions initially built and formulates an optimization problem. Additionally it is shown that, even though the two approaches seem to be different, they are related in fact.

In the experimental evaluation it is shown that the IR-based approach is more efficient to construct and to query, but depending on the selected outputs the model might be built from the scratch in order to answer a query. The main advantage of the RO-based approach is that it allows the formula-
of a wide range of queries. Not only output values can be specified by
the user, but also one or several ranges. Fixed or range constraints and/or
restraints can be also specified for the input dimensions. Weights can be
applied for each restraint, either user defined or problem dependent.
Part V

Summary and Outlook
In many applications and research areas, the task is to investigate processes or phenomena based on observations. Building a model of these observations serves the purposes of generalization, pointing out certain correlations and dependencies between the predictors, and their influence onto the behavior of the response. This generalization also allows to predict the response for predictors which have not been seen before. Piecewise linear models are well suited for this task, and employed in many practical applications, as motivated in Chapter 1.

This thesis handles efficient and effective methods for the construction of piecewise linear models. Since computing an optimal piecewise linear model from a set of samples is not feasible, a greedy divide and conquer approach is typically employed for this task. The result is materialized in so called regression trees, which contain a hierarchy of linear models, and each cut in the tree represents a piecewise linear model.

In Part II the focus lies on building piecewise linear models without the constrain of a continuous transition between the submodels. To this end, the split strategy consists out of a binary clustering followed by the computation of a linear hyperplane which separates them. For obtaining a compact model, the main challenge lies in the clustering. In Chapter 4 the new MoClus clustering is proposed. The clusters computed by this method are linearly correlated and also linearly separable in the input space, being an important requirement for the construction of regression trees. In Chapter 5 the new regression tree AMT for streaming data is introduced. By computing oblique splits and having as impurity measure the mean squared error w.r.t. the fitted linear model, it outperforms the state of the art regression tree for streaming data FIMT, building more compact and more accurate models. Also, in some cases it turn out to be also more efficient.

Part III concerns with hinging hyperplane models, which can be also written as continuous piecewise linear models. The hinging hyperplane models, presented in Chapter 6, are designed to deal with a single output dimension. In Chapter 7 the concept of multivariate regression tree is introduced, together with a method of constructing them. The main characteristic is that they have a common partitioning of the input space and different submodels
for each output. Like this, a compact model is constructed for the whole data set, instead of separate models for each output.

The hinge finding algorithm proposed in the literature is based on a random initialization, and converges only in a local optimum. This makes it necessary to perform several runs of the algorithm, in order to guarantee a good result. In Chapter 8 a initialization for the hinge finding algorithm is proposed, which considers the curvature of the regression surface and replaces the several runs, contributing to a great increase in efficiency for the hinge finding algorithm.

In Part IV the focus lies on using the computed models for making predictions. Using statistical models for predicting continuous data is straightforward, and is often applied in statistical databases. Typically, the addressed queries are on how the output looks like for a given input setting. Chapter 9 investigates the reverse problem: which input settings are required, in order to obtain a desired output. Two possible solutions are proposed: the first one is based on inverse regression (IR) and the second one is based on restraint optimization (RO). These two approaches are closely related, but the latter is more general, allowing the formulation of a wide range of queries.

Outlook

Future research questions can be mainly divided in three categories: improvements in the construction of regression trees, and extensions to cope with massive data sets and data streams.

Regarding the construction of regression trees presented in Chapter 4, the approach followed is greedy. In each step a split is performed, which aims at minimizing the sum of the mean squared errors of the emerging two submodels. Each split is performed permanently. An interesting future research objective is the design of an additional post processing step, after the tree construction is finished, improving the compactness of the resulting model. Merging neighboring submodels, updating the separating hyperplanes can help reducing its size, without loosing accuracy. This strategy comprises the reorganization of the tree structure, while traditional pruning strategies re-
move only existing nodes.

Another interesting open question is the design of outlier detection methods in the area of predictive analysis and regression analysis, for both batch and streaming algorithms. Furthermore, a thorough investigation of logistic regression as linear classifier within the split strategy is a future work objective.

Further research questions arise in the design of regression trees for streaming data, where not much work has been done in the literature. An important advantage of streaming regression trees is their ability to make anytime predictions. Depending on the available time for an incoming sample, a prediction can be made in any inner node while traversing the tree. Especially the computation of hinging hyperplane models, which are used in a variety of applications, is designed as a batch algorithm. An interesting topic for future research is to develop hinging hyperplane models for streaming data. Different drift detection methods, as well as windowing techniques are still to be investigated. While in the context of for streaming data, ensembles of classifiers have been successfully applied, ensembles of regression trees are still open for future research.

Another interesting approach for the future research is that the progress in scientific observational instruments and simulation software leads to huge amounts of data from heterogeneous sources, e.g. images, structured data, high dimensional data. Analyzing and building models from such data poses major challenges, and the solutions require techniques of large-scale parallelism in cluster or cloud environment.

In summary, this thesis provides novel techniques for effective and efficient construction of regression trees with substantial further research potential for streaming data and big data sets.
Bibliography


BIBLIOGRAPHY


Collaboration / Acknowledgments

This thesis would not have been possible without the collaboration in the group of Professor Seidl. Many of the presented ideas and techniques evolved from numerous fruitful discussions in the group. Professor Seidl is clearly to be credited with making numerous suggestions for improvements and additions. The high level of collaboration both in the group and with the students constitutes a key factor for the productive environment I found at Professor Seidls group. In order to clarify my contributions to the publications that resulted from the work presented in this thesis in order to give the outside reader a more complete picture than can be obtained by looking at the order of authors on the publications.

The topic of this thesis resulted from the SFB 686 project, about model-based control of homogenized low-temperature combustion, that I have been working on in the past three and a half years. In many fruitful discussions with researchers from the Institute of Heat and Mass Transfer (WSA) and the Institute of Automatic Control (IRT), at the RWTH Aachen University, I understood the data analysis challenges that researchers face.

The first steps toward the computation of piecewise linear models by employing correlation clustering algorithms were published in [IAAS11] and [ADH+11], in collaboration with Thivaharan Albin and Dirk Abel from IRT, RWTH Aachen. The new clustering approach MoClus from Chapter 4 and its employment for the streaming regression tree AMT, in Chapter 5, were evaluated in the diploma thesis of Michael Kurze.

The multivariate hinging hyperplane model and the mHFA algorithm presented in Chapter 7, are based on work done with Philipp Kranen. The contents of this chapter were published in [IKS12]. The initialization of the
HFA algorithm, from Chapter 8, was motivated by the fruitful discussions with Thivaharan Albin, and published in [IAAS12].

Credit to part of the inverse model-queries implementation regarding for the evaluations in Chapter 9 goes to Philip Driessen, whom I advised together with Philipp Kranen, during his master thesis on undetermined bidirectional prediction.
List of Publications


