Diss. ETH No. 16979

Extensions of L_2Boosting and an Application of LogitBoost

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Sciences

presented by
ROMAN WERNER LUTZ
Dipl. Math. ETH
born July 1, 1976
citizen of Zurich

accepted on the recommendation of
Prof. Dr. Peter Bühlmann, examiner
Prof. Dr. Sara van de Geer, co-examiner

2007

Acknowledgements

First, I would like to thank Prof. Peter Bühlmann for supervising this thesis. He gave me excellent help and support and suggested many interesting research directions. I was always welcome to discuss new ideas and he also gave me enough room to follow my own path.

I also thank Prof. Sara van de Geer for refereeing my thesis and for acting as co-examiner.

Further thanks go to my friends and colleagues at the Seminar für Statistik (SfS) for providing such a friendly atmosphere. This made the SfS not only a place to work but also a place to live.

I also thank my parents and my partner Denise for their ongoing support and their interest in my PhD work.

A special thank goes to the compute servers deb2 to deb8 at the SfS for their tireless work. Altogether they probably spent more time on this PhD thesis than I did.

Contents

Abstract

One goal of statistical learning is to establish a relationship between input variables and an output variable. The simplest and most widely used method is linear regression. Over the years a lot of sophisticated methods have been developed. One of them is boosting, which is an ensemble method. This means that it combines several outputs of a simple method to a strong "committee". This PhD thesis proposes various extensions of boosting.

A first extension uses a conjugate direction method instead of the gradient method to construct a new boosting algorithm (CDBoost). As a result, one obtains a fast forward stepwise variable selection algorithm. The conjugate direction of CDBoost is analogous to the constrained gradient in boosting. Using this analogy, CDBoost is generalized to: (i) include small step sizes (shrinkage) which often improves prediction accuracy; (ii) the non-parametric setting with fitting methods such as trees or splines, where least angle regression and the Lasso seem to be unfeasible. The step size in CDBoost has a tendency to govern the degree between L_0- and L_1-penalisation. This makes CDBoost surprisingly flexible. The different methods are compared on simulated and real datasets. CDBoost achieves the best predictions mainly in complicated settings with correlated covariates, where it is difficult to determine the contribution of a given covariate to the response. The gain of CDBoost over boosting is especially high in sparse cases with high signal-to-noise ratio and few effective covariates.

A second extension proposes multivariate L_2Boosting based on some squared error loss for multivariate data. It can be applied to multivariate linear regression with continuous responses and to vector autoregressive time series. It is proven, for i.i.d. data as well as for time series, that multivariate L_2Boosting can consistently recover sparse high-dimensional multivariate linear functions, even when the number of predictor variables $p = p_n$ and the dimension of the response $q = q_n$ grow almost exponentially with sample size n, i.e. $p_n = q_n = O(\exp(Cn^{1-\xi}))$ $(0 < \xi < 1, 0 < C < \infty)$, but the L_1-norm of the true underlying function is finite. This theory seems to be among the first to address the issue of large dimension of the response variable; the relevance of such settings is briefly outlined. Some cases where the multivariate L_2Boosting is better than multiple application of univariate methods to single response components are also identified, thus demonstrating that the multivariate approach can be very useful.

Five robustifications of L_2Boosting for linear regression with various robustness properties are considered in the third extension. The first two use the Huber loss as implementing loss function for boosting and the second two use robust simple linear regression for the fitting in L_2Boosting (i.e. robust base learners). Both concepts can be applied with or without down-weighting of leverage points. The last method uses robust correlation estimates and appears to be most robust. Crucial advantages of all methods are that they don't compute covariance matrices of all covariates and that they don't have to identify multivariate leverage points. When there are no outliers, the robust methods are only slightly worse than L_2Boosting. In the contaminated case though, the robust methods outperform L_2Boosting by a large margin. Some of the robustifications are also computationally highly efficient and therefore well suited for high dimensional problems.

Finally, I applied LogitBoost with a tree-based learner to the five performance prediction challenge datasets of the world congress on computational intelligence (WCCI) 2006. The number of iterations and the tree size were estimated by 10-fold cross-validation. A simple shrinkage strategy was added to make the algorithm more stable. The results are promising: I won the challenge.

Zusammenfassung

Ein Ziel von statistischem Lernen ist es, Zusammenhänge zwischen Input Variablen und einer Output Variablen zu finden. Die einfachste und am meisten verwendete Methode ist lineare Regression. Im Laufe der Zeit wurden viele raffinierte Methoden entwickelt. Eine davon ist Boosting, welche eine “Ensemble” Methode ist. Das heisst, sie vereint mehrere Outputs einer einfachen Methode zu einer starken Gesamtheit. Diese Dissertation beschreibt verschiedene Erweiterungen von Boosting.

Eine erste Erweiterung verwendet die Methode der “konjugierten Richtungen” anstatt der Gradientenmethode um einen neuen Boosting Algorithmus zu konstruieren (CDBoost). Als Resultat erhält man einen schnellen Algorithmus für Variablenwahl vorwärts. Die konjugierte Richtung von CDBoost ist analog zum eingeschränkten Gradienten beim Boosting. Unter Verwendung dieser Analogie wird CDBoost so verallgemeinert, dass man (i) kleine Schrittlängen (“Shrinkage”) verwenden kann, was sehr oft die Genauigkeit von Vorhersagen verbessert und dass man (ii) auch nicht-parametrische Methoden wie Bäume oder Splines verwenden kann, was bei “least angle regression” und dem Lasso nicht möglich zu sein scheint. Die Schrittlänge in CDBoost hat eine Tendenz, zwischen der L_0- und der L_1-Bestrafung zu regeln. Das macht CDBoost überraschend flexibel. Die verschiedenen Methoden werden anhand von simulierten und echten Datensätzen verglichen. CDBoost erreicht die besten Vorhersagen vorallem in komplizierten Situationen mit korrelierten Kovariablen, in denen es schwierig zu bestimmen ist, wieviel eine Kovariable zur Zielgrösse beiträgt. Der Gewinn von CDBoost über Boosting ist besonders gross in Fällen mit hohem Signal-zu-Rauschen Verhältnis und wenigen effektiven Kovariablen.

Die zweite Erweiterung schlägt multivariates L_2Boosting vor, welches auf einer quadratischen Verlustfunktion für multivariate Daten basiert. Es kann bei multivariater linearer regression mit kontinuierlichen Zielgrössen und für vektorielle autoregressive Zeitreihen verwendet werden. Wir beweisen für i.i.d. Daten als auch für Zeitreihen, dass multivariates L_2Boosting dünn besetzte, hoch-dimensionale, multivariate, lineare Funktionen konsistent schätzen kann und zwar sogar dann, wenn die Anzahl der erklärenden Variablen $p = p_n$ und die Dimension der Zielgrösse $q = q_n$ fast exponentiell wachsen mit der Anzahl Beobachtungen n, d.h. $p_n = q_n = O(\exp(Cn^{1-\xi}))$ $(0 < \xi < 1, 0 < C < \infty)$, aber die L_1-Norm der wahren zugrunde liegenden Funktion endlich ist. Diese Theorie scheint unter den ersten zu sein, die die Situation der hoch-dimensionalen Zielvariable behandeln. Die Relevanz solcher Situationen wird kurz umrissen. Es werden auch Fälle aufgezeigt, in denen das multivariate Boosting besser ist als mehrere Anwendungen von univariatem Boosting auf die einzelnen Komponenten der Zielgrösse, was darlegt, dass der multivariate Ansatz sehr nützlich sein kann.

In der dritten Erweiterung werden fünf Robustifizierungen von L_2-Boosting für lineare Regression mit verschiedenen Robustheitseigenschaften betrachtet. Die ersten beiden verwenden den Huber Verlust als implementierende Verlustfunktion für Boosting und die nächsten beiden verwenden robuste einfache lineare Regression für das anpassen beim L_2Boosting (d.h. robuste Basis Lerner). Beide Konzepte können mit oder ohne Heruntergewichten von Hebelpunkten angewendet werden. Die letzte Methode verwendet robuste Korrelationsschätzer und scheint die am meisten robuste zu sein. Entscheidende Vorteile aller Methoden sind, dass sie keine Kovarianzmatrizen aller Kovariablen berechnen und dass sie nicht multivariate Hebelpunkte identifizieren müssen. Falls es keine Ausreisser gibt, sind die robusten Methoden nur wenig schlechter als L_2Boosting. Bei kontaminierten Daten übertreffen die robusten Methoden L_2Boosting jedoch deutlich. Einige der Robustifizierungen können sehr effizient berechnet werden und sind deshalb gut geeignet für hoch-dimensionale Probleme.

Schliesslich habe ich LogitBoost mit einem auf Bäumen basierenden Lerner auf die fünf Datensätze der "Performance Prediction Challenge" des "World Congress on Computational Intelligence (WCCI) 2006" angewendet. Die Anzahl Boosting Iterationen und die Grösse der Bäume werden mit 10-facher Kreuzvalidierung bestimmt. Eine einfache “Shrinkage”-Strategie wurde hinzugefügt, um den Algorithmus stabiler zu machen. Die Resultate sind vielversprechend, habe ich doch den Wettbewerb gewonnen.

Chapter 1

Introduction

In statistical learning we have an outcome measurement Y that we want to predict by using a set of input variables $X_1, \ldots, X_p$. We refer to regression if the output is quantitative ($Y \in \mathbb{R}$) and to classification if the output is qualitative ($Y \in \{1, \ldots, K\}$). A statistical learning method takes a training set of input-output pairs and builds a prediction model, that can be used to predict the outcome for new unseen input objects. The goal is to construct a model that predicts as accurately as possible and that is at the same time as parsimonious as possible.

A popular learning idea is boosting, which was introduced by the machine learning community at the beginning of the 1990's (Schapire 1990, Freund 1995). It was originally designed for classification, but can also be used for regression. The idea was to combine many outputs of a "weak" classifier to form a powerful "committee". With "weak" we mean that the classifier is only slightly better than random guessing. In the following I will review some key steps in the development of boosting. Detailed introductions to boosting from a statistical point of view are given in Hastie, Tibshirani and Friedman (2001) or in Bühlmann and Hothorn (2006). A completely different introduction from the point of view of Machine Learning is given in Meir and Rätsch (2003).

1.1 AdaBoost

The first practical boosting algorithm is the AdaBoost algorithm for classification (Freund and Schapire 1996, 1997). Consider a two class problem with n observations $(x_1, y_1), \ldots, (x_n, y_n)$ with $y_i \in \{-1, 1\}$ and $x_i \in \mathbb{R}^p$. Take a simple classification method that can be applied to weighted data. The most popular choice is a classification tree. AdaBoost then works as follows: at the beginning, the classifier is applied to the original (unweighted) data. All the observations that are misclassified get more weight and the classifier is applied to a weighted version of the data. The weights of the misclassified observation of the second classifier are again increased and the classifier is once more applied. This process is repeated several times and the final classifier is a weighted majority vote among all the simple classifiers. In detail, AdaBoost works as follows:

AdaBoost algorithm

1. Initialize the observation weights $w_i = 1/n,\ i = 1, \ldots, n$, where n is the number of observations.
2. For $m = 1$ to M:
   (a) Fit a classifier $g^{(m)} : \mathbb{R}^p \to \{-1, 1\}$ to the data using weights w_i.
   (b) Compute
   $$\mathrm{err}^{(m)} = \frac{\sum_{i=1}^{n} w_i I(y_i \neq g^{(m)}(x_i))}{\sum_{i=1}^{n} w_i}.$$
   (c) Compute $\alpha^{(m)} = \log((1 - \mathrm{err}^{(m)})/\mathrm{err}^{(m)})$.
   (d) Set $w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot I(y_i \neq g^{(m)}(x_i))\right), i = 1, \ldots, n$.
3. Output $G(x) = \mathrm{sign}\left(\sum_{i=1}^{M} \alpha^{(m)} g^{(m)}(x)\right)$.

After the invention of boosting, there has been some mystery about it. It has been observed that boosting almost never over-fits no matter how many iterations are performed. The usual strategy was therefore to run boosting for a large number of iterations and to stop when the error curve (as a function of the number of iterations) levels off. Later it has been found that "it is clearly a myth that boosting methods do not over-fit" (Rätsch, Onoda and Müller 2001). It is mainly the nature of the zero-one loss used in classification which provides some resistance against over-fitting (Bühlmann and Yu 2000) and boosting was often applied to low noise problems, where it is harder to see clear over-fitting. It is therefore important to choose the number of iterations M carefully, especially in regression problems where we can't use the zero-one loss. The usual strategy to select M is cross-validation or to use a separate validation sample.

1.2 Gradient boosting

Another question was why boosting works so well. Some answers were provided by Breiman (1999), Friedman, Hastie and Tibshirani (2000) and Friedman (2001) as they showed that boosting can be viewed as a functional gradient descent algorithm. Friedman et al. (2000) pointed out that AdaBoost fits an additive model in a stagewise manner minimizing an exponential loss, and that it is probably not the reweighting that is responsible for the success of AdaBoost. They also developed a new boosting algorithm called LogitBoost which is based on the log-likelihood-loss and Newton optimization.

Friedman (2001) describes the very general gradient boosting algorithm. It is based on greedy function approximation by iterative approximate gradient descent and works for classification and regression. We need a differentiable loss function $L : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}_0^+$, with $\mathcal{Y} = \mathbb{R}$ for regression and $\mathcal{Y} = \{-1, 1\}$ for classification. Additionally, we need (instead of a base learner) a class of basis functions $\mathcal{F}$ with $f \in \mathcal{F} : \mathbb{R}^p \to \mathbb{R}$. The fitting is done iteratively by selecting in each iteration the basis function that is most parallel (most correlated) to the negative gradient of the loss function and adding them up successively. In detail, the algorithm works as follows:

Generic functional gradient descent (FGD) algorithm

1. Initialize $\hat{F}^{(0)} \equiv \underset{a \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} L(y_i, a)$.

2. For $m = 1$ to M:
   (a) Compute the negative gradient (also called pseudo-response)
   $$r_i = -\frac{\partial}{\partial F} L(y, F)|_{F=\hat{F}^{(m-1)}(x_i)}, \ i = 1, \ldots, n.$$
   (b) Fit a function $\hat{f}^{(m)}$ to the negative gradient by least squares
   $$\hat{f}^{(m)} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} (r_i - f(x_i))^2.$$
   (c) Do a line search
   $$\hat{k}^{(m)} = \arg\min_{k \in \mathbb{R}} \sum_{i=1}^{n} L(y_i, \hat{F}^{(m-1)} + k \cdot \hat{f}^{(m)}).$$
   (d) Update $\hat{F}^{(m)} = \hat{F}^{(m-1)} + \nu \cdot \hat{k}^{(m)} \cdot \hat{f}^{(m)}$, where $0 < \nu \leq 1$ is an additional step length factor.
3. Output $\hat{F}^{(M)}$ for regression and $\text{sign}(\hat{F}^{(M)})$ for classification.

The ν in step 2d is called shrinkage factor. The natural value is 1, but smaller values are often a better choice because they lead to less greedy and more stable procedures. The smaller we choose ν the more iterations we need. My experience is that $\nu = 0.1$ is usually small enough and can be taken as a good default.

1.3 L_2Boosting

The most widely used loss function for regression is the squared error loss $L(y, F) = (y - F)^2/2$. The corresponding boosting algorithm, L_2Boosting, is particularly simple: the negative gradient is just the current residual vector and the line search of step 2c of the FGD algorithm is trivial ($\hat{k}^{(m)}$ is always equal to 1). The algorithm amounts to iteratively fitting of residuals.

As basis functions we take the set of all linear functions that depend only on one component of the input: $\mathcal{F} = \{f : \mathbb{R}^p \to \mathbb{R} | f(x) = \alpha + \beta \cdot x_s\}$.

This means that in each iteration the residuals are fitted against the covariate that most reduces the residual sum of squares. The algorithm can be written as:

L_2Boosting with componentwise linear least squares

1. Initialize $\hat{F}^{(0)} = \bar{y}$.
2. For $m = 1$ to M:
   (a) Compute the current residuals
   $$r_i = y_i - \hat{F}^{(m-1)}(x_i),\ i = 1, \ldots, n.$$
   (b) Fit for each covariate $\mathbf{x}_j$ a simple linear least squares regression:
   $$\hat{\beta}_j = \frac{(\mathbf{x}_j - \bar{\mathbf{x}}_j)^T \mathbf{r}}{(\mathbf{x}_j - \bar{\mathbf{x}}_j)^T (\mathbf{x}_j - \bar{\mathbf{x}}_j)}, \quad \hat{\alpha}_j = -\hat{\beta}_j \bar{\mathbf{x}}_j, \quad j = 1, \ldots, p.$$
   Select the covariate which most reduces the L_2-loss:
   $$\hat{s} = \arg\min_j \frac{1}{2} \|\mathbf{r} - \hat{\alpha}_j - \hat{\beta}_j \mathbf{x}_j\|^2.$$
   (c) Update $\hat{F}^{(m)}(x) = \hat{F}^{(m-1)}(x) + \nu \cdot (\hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}})$.
3. Output $\hat{F}^{(M)}$.

The algorithm can be simplified when all covariates are centered (mean subtracted). Then the intercept is estimated in step 1 and in step 2b regressions through the origin are fitted.

L_2Boosting with componentwise linear least squares is not just a black box tool but generates sound linear models. It performs a simple kind of variable selection, because in each iteration only one coefficient is updated. This is usually not the way variable selection and fitting is done. The classical forward variable selection algorithm updates in every step all the coefficients of the variables included in the model. Additional strength of L_2Boosting with componentwise linear least squares is given by using a shrinkage factor ν smaller than 1. When we use for example $\nu = 0.1$, we only perform a small model update in each iteration and in the next iteration we can check whether there is now a better variable. If not, the same variable is chosen again and the model is carefully built. In high dimensions (many covariates), L_2Boosting is clearly superior to the classical forward variable selection.

When we use a "$\nu \to 0$", L_2Boosting yields solutions similar to the Lasso. In fact, L_2Boosting and the Lasso are for specific models special cases of least angle regression (see Efron, Hastie, Johnstone and Tibshirani 2004).

Typically we use cross validation to stop the boosting iteration prediction optimal. In this case, the fitted model usually contains too many variables. In fact, the smaller we take ν, the more variables are selected. From a practical point of view, this is not too bad, since it is easy to drop some variables out of the model in a second analysis. Also for prediction accuracy, some noise variables in the fitted model with small coefficients usually do not hurt.

1.4 Thesis overview

In chapters 2–5, this PhD thesis includes four papers. The first three are closely related to L_2Boosting and the forth is an application of LogitBoost.

Chapter 2 is the paper "Conjugate direction boosting" (Lutz and Bühlmann 2006c). Instead of the gradient method, a conjugate direction method is used to construct a new boosting algorithm (CDBoost).

Chapter 3 is the paper "Boosting for High-Multivariate Responses in High-Dimensional Linear Regression" (Lutz and Bühlmann 2006b). Parts of it are also contained in Bühlmann and Lutz (2006). A multivariate L_2Boosting algorithm based on some squared error loss for multivariate data is proposed. It can be applied to multivariate linear regression with continuous responses and to vector autoregressive time series.

Chapter 4 is a research report (Lutz and Bühlmann 2006a) that has been submitted. Five robustifications of L_2Boosting for linear regression with various robustness properties are considered. The first two use the Huber loss as implementing loss function for boosting and the second two use robust simple linear regression for the fitting in L_2Boosting (i.e. robust weak learners). Both concepts can be applied with or without down-weighting of leverage points. The last method uses robust correlation estimates.

Chapter 5 is the conference paper "LogitBoost with Trees Applied to the WCCI 2006 Performance Prediction Challenge Datasets" (Lutz 2006). I applied LogitBoost with a tree-based learner to the five WCCI 2006 performance prediction challenge datasets. The results are promising: I won the challenge.

Chapter 2

Conjugate direction boosting

Boosting in the context of linear regression has become more attractive with the invention of least angle regression (LARS), where the connection between the Lasso and forward stagewise fitting (boosting) has been established. Earlier it has been found that boosting is a functional gradient optimization. Instead of the gradient, we propose a conjugate direction method (CDBoost). As a result, we obtain a fast forward stepwise variable selection algorithm. The conjugate direction of CD-Boost is analogous to the constrained gradient in boosting. Using this analogy, we generalize CDBoost to: (i) include small step sizes (shrinkage) which often improves prediction accuracy; (ii) the non-parametric setting with fitting methods such as trees or splines, where least angle regression and the Lasso seem to be unfeasible. The step size in CDBoost has a tendency to govern the degree between L_0- and L_1-penalisation. This makes CDBoost surprisingly flexible. We compare the different methods on simulated and real datasets. CDBoost achieves the best predictions mainly in complicated settings with correlated covariates, where it is difficult to determine the contribution of a given covariate to the response. The gain of CDBoost over boosting is especially high in sparse cases with high signal to noise ratio and few effective covariates.

2.1 Introduction

The problem of subset selection in linear regression has been revived by the invention of least angle regression (LARS; Efron et al. 2004), where a connection between the Lasso (Tibshirani 1996) and forward stagewise linear regression has been established. A closely related idea of forward stagewise fitting originates from the machine learning community under the name of boosting (Schapire 1990, Freund 1995, Freund and Schapire 1996). As an ensemble scheme, a fitting method (called the weak or base learner) is repeatedly applied to reweighted data and its outputs are averaged to form the boosting estimator. Freund and Schapire's (1996) AdaBoost for binary classification is most popular, very often in conjunction with trees as base learner. An extensive overview over boosting is given for example in Meir and Rätsch (2003).

Further important work (Breiman 1998, 1999, Friedman 2001, Rätsch et al. 2001) has revealed that boosting is a functional gradient descent method. It can be applied in a variety of settings with different loss functions and many fitting methods. The use of the squared error loss for linear and non-parametric regression was proposed by Friedman (2001) under the name LS_Boosting and further extended and analyzed by Bühlmann and Yu (2003) under the name L_2Boosting. Other work on boosting for regression include Duffy and Helmbold (2000), Zemel and Pitassi (2001) and Rätsch, Demiriz and Bennett (2002).

In this paper we propose CDBoost, a method for linear regression with many covariates (or linear basis expansions with overcomplete dictionaries) which uses a conjugate direction method instead of the gradient method. As a result, we obtain a fast method for some type of forward stepwise variable selection in linear regression. It turns out that it produces the same solutions as the orthogonal greedy algorithm in function approximation (cf. Temlyakov 2000), although the algorithms are different. It is well known that forward stepwise variable selection algorithms can be unstable, resulting in poor predictive performance. In analogy to boosting using the concept of a conjugate direction instead of a constrained gradient, we can include small step sizes (shrinkage) in CDBoost when moving along conjugate directions. The algorithm decelerates, becomes more stable and the prediction accuracy can be greatly enhanced. The step size in CDBoost has a tendency to govern the degree between L_0-norm- and L_1-norm-penalisation of the regression coefficient vector. This is quite different from boosting where the step size cannot be chosen to yield a forward stepwise variable selection method (i.e. forward L_0-penalisation). Varying the step size thus makes CDBoost surprisingly flexible. Another advantage of conjugate directions is that we can use our method with any fitting method such as trees and smoothing splines. While this is analogous to boosting, it marks an essential difference to the Lasso or least angle regression (LARS).

2.2 Linear Regression

In univariate linear regression we assume a continuous response $\mathbf{y} \in \mathbb{R}^n$ and a set of d covariates $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d) \in \mathbb{R}^{n \times d}$. Here n denotes the sample size. The response $\mathbf{y}$ is modeled as a linear combination of the covariates plus a random error. In matrix notation:

$$\begin{aligned} &\mathbf{y} = \mathbf{X}\beta + \epsilon, \qquad \beta \in \mathbb{R}^d, \ \epsilon \in \mathbb{R}^n, \\ &\epsilon_1, \ldots, \epsilon_n \text{ i.i.d. with } \mathbf{E}\left[\epsilon_i\right] = 0, \ \operatorname{Var}\left(\epsilon_i\right) = \sigma^2. \end{aligned}$$

We assume w.l.o.g. that all covariates and the response are centered to have mean zero, so we need not worry about an intercept. We always assume that rank($\mathbf{X}$) = min($n-1, d$); note the reduced rank $n-1$ (if $d \geq n-1$) due to the centering of the covariates. Parameter estimation is most often done by least squares, minimizing the loss function

$$L(\beta) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\beta\|^2 = \frac{1}{2}\beta^T\mathbf{X}^T\mathbf{X}\beta - \mathbf{y}^T\mathbf{X}\beta + \frac{1}{2}\mathbf{y}^T\mathbf{y}.$$

Assuming $d \leq n-1$ and $\mathbf{X}$ of full rank d, we find the unique solution $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

2.2.1 Methods for variable selection

For $d > n-1$, the matrix $\mathbf{X}^T\mathbf{X}$ becomes singular and the least squares solution is not unique. For fairly large $d \leq n-1$, the least squares solution will often over-fit the data. In both cases we need a regularisation strategy. We focus here on methods that do variable selection.

Classic forward stepwise variable selection

We start with the empty model and sequentially add that covariate to the model which most improves the fit (most reduces the loss function L) when (re-) estimating regression coefficients by least squares corresponding to previously and currently selected covariates. Forward stepwise variable selection is a very greedy method. Covariates which are highly correlated with variables already included in the model have a small chance of being chosen.

L_2Boosting

We present here only the key ideas of L_2Boosting with componentwise linear least squares. Technical details are given in section 2.3, where we show that L_2Boosting is a constrained gradient optimization. More details can also be found in Friedman (2001) and Bühlmann and Yu (2003).

L_2Boosting is an iterative method which starts with the empty model (all β-coefficients equal to zero). In each iteration, the current residuals are fitted against the one covariate which gives the best least squares fit (using only one single covariate). The model is updated by adjusting only the coefficient corresponding to the chosen covariate; all other coefficients remain unchanged. This is a main difference to forward stepwise variable selection: one boosting update is computationally faster, but the coefficients in every iteration are no longer least squares estimates (in sub-models). But it is possible to choose a covariate again to adjust a coefficient which was badly estimated in an earlier step.

Conjugate direction boosting

Here, we give a brief sketch of our conjugate direction boosting algorithm (CDBoost). It turns out that our algorithm yields the same solutions as the orthogonal greedy algorithm which is known in nonlinear function approximation (cf. Temlyakov 2000). But our formulation with conjugate directions easily allows for extensions which include shrinkage (small step sizes) or which are applicable to the non-parametric settings.

CDBoost is a hybrid between forward stepwise variable selection and L_2Boosting. The selection of a new covariate is as in boosting: take the one covariate which fits the current residuals best. The coefficient update is like forward stepwise variable selection: compute the least squares solution in the model with the previously and currently selected covariates, by using a conjugate direction method. In other words: the conjugate direction boosting selects the covariate which would most improve the fit without adjusting the coefficients of the other variables. But after the selection, we adjust all the non-zero coefficients. Therefore, conjugate direction boosting (like forward stepwise variable selection but unlike L_2Boosting) finds the least squares solution (or a perfect fit) in $\min(n-1, d) = \operatorname{rank}(\mathbf{X})$ steps.

2.3 L_2Boosting

We denote by $F(\cdot) : \mathbb{R}^d \to \mathbb{R}$ the (linear) regression function of interest. The L_2Boosting algorithm with componentwise linear least squares as briefly outlined in section 2.2.1 can then be described as follows:

L_2Boosting algorithm for linear regression (function version):

Step 1: Center $\mathbf{x}_i$ and $\mathbf{y}$ to mean zero. Initialize $\hat{F}^{(0)}(x) \equiv 0$ and $m = 1$.

Step 2: Compute current residuals $r_i = y_i - \hat{F}^{(m-1)}(x_i)$ $(i = 1, \ldots, n)$. Compute the coefficients of all simple linear regressions of $\mathbf{r}$ $(\in \mathbb{R}^n)$ against each covariate alone: $\hat{\beta}_j = \mathbf{x}_j^T \mathbf{r} / \mathbf{x}_j^T \mathbf{x}_j$ $(j = 1, \ldots, d)$.
Select the covariate which most reduces the L_2-loss:
$\hat{k}^{(m)} = \arg\min_j \frac{1}{2} \|\mathbf{r} - \hat{\beta}_j \mathbf{x}_j\|^2$.
Update $\hat{F}^{(m)}(x) = \hat{F}^{(m-1)}(x) + \hat{\beta}_{\hat{k}^{(m)}} x_{\hat{k}^{(m)}}$.

Step 3: Increase iteration index m by one and go back to Step 2.

The number of iterations is usually estimated using a validation set or with cross validation.

In signal processing, this algorithm is known as matching pursuit (Mallat and Zhang 1993) and is used to decompose a signal into a linear expansion of waveforms.

The L_2Boosting algorithm can also be written as a constrained gradient method. This is not exactly the same as the functional gradient approach of Breiman (1998, 1999), Friedman (2001) and Rätsch et al. (2001), because we work entirely in the parameter space of β.

For a given β, the negative gradient of the loss function is

$$-\nabla L(\beta) = -(\mathbf{X}^T\mathbf{X})\beta + \mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta).$$

The idea is to optimize in the coordinate direction (thereby changing only one entry of β) which is most parallel to the negative gradient. This means we look for the $\hat{k}$ for which $| < -\nabla L, \mathbf{e}_{\hat{k}} > |$ becomes maximal, where $\mathbf{e}_{\hat{k}} \in \mathbb{R}^d$ is the unit vector with entry 1 at position $\hat{k}$. $\hat{k}$ is therefore merely the component of the negative gradient with the highest absolute value. We shall call $\mathbf{e}_{\hat{k}}$ constrained gradient.

Note that this alternative formulation is only equivalent to the above cited function version when all covariates are scaled to the same variance. We can then rewrite the L_2Boosting algorithm:

L_2Boosting algorithm for linear regression (gradient/coefficient version):

Step 1: Standardize $\mathbf{x}_i$ to zero mean and unit length ($\mathbf{x}_i^T\mathbf{x}_i = 1$). Standardize $\mathbf{y}$ to zero mean. Initialize $\hat{\beta}^{(0)} = \mathbf{0}$ and $m = 1$.

Step 2: Compute the negative gradient

$$-\nabla L^{(m)} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})$$

and determine the component with highest absolute value: $\hat{k}^{(m)} = \arg\max_j | - \nabla L_j^{(m)}|$.
Update $\hat{\beta}$: component $\hat{k}^{(m)}$ changes to

$$\hat{\beta}_{\hat{k}^{(m)}}^{(m)} = \hat{\beta}_{\hat{k}^{(m)}}^{(m-1)} + \mathbf{x}_{\hat{k}^{(m)}}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})$$

and all other components remain unchanged.

Step 3: Increase iteration index m by one and go back to Step 2.

2.4 Conjugate direction boosting

2.4.1 Conjugate direction and gradient optimization

Instead of the gradient method, we can use a *conjugate direction method* to minimize the quadratic function

$$L(\beta) = \frac{1}{2}\beta^T\mathbf{X}^T\mathbf{X}\beta - \mathbf{y}^T\mathbf{X}\beta + \frac{1}{2}\mathbf{y}^T\mathbf{y} =: \frac{1}{2}\beta^T\mathbf{A}\beta - \mathbf{b}^T\beta + c,$$
$$\mathbf{A} = \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d\times d} \text{ symmetric, positive definite}, \mathbf{b} = \mathbf{X}^T\mathbf{y} \in \mathbb{R}^d.$$

If $d > n-1$, the matrix $\mathbf{A}$ is not positive definite. This does not matter because we actually only use sub-matrices of $\mathbf{X}$ so that the corresponding $\mathbf{A}$ remains positive definite.

Conjugacy is a property similar to orthogonality. A set of nonzero vectors $\{\mathbf{p}_1, \dots, \mathbf{p}_d\}, \mathbf{p}_i \in \mathbb{R}^d$ is said to be conjugate with respect to the symmetric positive definite matrix $\mathbf{A}$ if

$$\mathbf{p}_i^T\mathbf{A}\mathbf{p}_j = 0 \quad \text{for all } i \neq j.$$

The importance in conjugacy lies in the fact that we can minimize L in d $(\leq n-1)$ steps by minimizing along the individual directions in a conjugate set. A conjugate direction method takes an arbitrary set of d conjugate directions and does individual minimization of L along these directions. In contrast to the gradient method, we reach the minimum after d $(\leq n-1)$ steps.

The question is how to find a set of conjugate directions. The canonical *conjugate gradient method* does the job very efficiently during the optimization process and not in advance. It takes the negative gradient as the first direction $\mathbf{p}_1$. For $k > 1$, $\mathbf{p}_k$ is a linear combination of the actual negative gradient and the previous $\mathbf{p}_{k-1}$ only:

$$\mathbf{p}_k = -\nabla L^{(k)} + \frac{\nabla L^{(k)T}\mathbf{A}\mathbf{p}_{k-1}}{\mathbf{p}_{k-1}^T\mathbf{A}\mathbf{p}_{k-1}}\mathbf{p}_{k-1}.$$

We do not need to store all the previous elements $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{k-2}$: $\mathbf{p}_k$ is automatically conjugate to these vectors. A proof of this remarkable property can be found for example in Nocedal and Wright (1999).

The *conjugate gradient method* is the same as *Partial Least Squares* (see for example Phatak and de Hoog 2002) and does not employ variable selection.

2.4.2 CDBoost: conjugate direction boosting for linear regression

We now describe our conjugate direction method which employs variable selection. In each step, only one component of $\hat{\beta}$ should change from zero to non-zero. We achieve this by choosing a special set of conjugate directions. Unfortunately, we loose the simplicity of the *conjugate gradient method*, because we have to store all the directions $\mathbf{p}_k$.

The first step is identical to L_2Boosting: we look for the component of the negative gradient with the highest absolute value, say component $\hat{k}^{(1)}$. The vector $\mathbf{e}_{\hat{k}(1)}$ is again called the constrained gradient. We then optimize along that first direction $\mathbf{p}_1 = \mathbf{e}_{\hat{k}(1)}$.

In the following steps we have to determine a direction that is conjugate to all previous directions. Again, we compute the negative gradient and its constrained version (unit-coordinate vector $\mathbf{e}_{\hat{k}(m)}$). The new direction $\mathbf{p}_m$ is a linear combination of the constrained gradient and all the previous directions. It is easy to compute the coefficients of the linear combination so that the new $\mathbf{p}_m$ is conjugate to all previous directions. In the m-th iteration we have

$$\mathbf{p}_m = \sum_{j=1}^{m-1} \lambda_j \mathbf{p}_j + \mathbf{e}_{\hat{k}(m)}.$$

The $\lambda's$ are determined by the $m-1$ equations:

$$\mathbf{p}_i^T \mathbf{A} \left(\sum_{j=1}^{m-1} \lambda_j \mathbf{p}_j + \mathbf{e}_{\hat{k}(m)} \right) = 0, \qquad i = 1, \ldots, m-1.$$

Using the property of conjugacy, we have

$$\lambda_i \mathbf{p}_i^T \mathbf{A} \mathbf{p}_i + \mathbf{p}_i^T \mathbf{A} \mathbf{e}_{\hat{k}(m)} = 0, \qquad i = 1, \ldots, m-1,$$

and

$$\lambda_i = -\frac{\mathbf{p}_i^T \mathbf{A} \mathbf{e}_{\hat{k}(m)}}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \qquad i = 1, \ldots, m-1.$$

Now we can formulate the

CDBoost algorithm with L_2-loss for linear regression:

Step 1: Standardize $\mathbf{x}_i$ to zero mean and unit length. Standardize $\mathbf{y}$ to zero mean. Initialize $\hat{\beta}^{(0)} = \mathbf{0}$ and $m = 1$.

Step 2: Compute the negative gradient

$$-\nabla L^{(m)} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)}) \tag{2.4.1}$$

and take the component with highest absolute value: $\hat{k}^{(m)} = \arg\max_j |-\nabla L_j^{(m)}|$. Define the constrained gradient by $\mathbf{e}_{\hat{k}(m)}$ (unit-coordinate vector).

For $i = 1, \ldots, m-1$ compute the coefficients for the linear combination (for $m = 1$ there are no $\lambda's$):

$$\lambda_i^{(m)} = -\frac{\mathbf{p}_i^T \mathbf{X}^T \mathbf{X} \mathbf{e}_{\hat{k}(m)}}{\mathbf{p}_i^T \mathbf{X}^T \mathbf{X} \mathbf{p}_i}.$$

Compute the new direction (for $m = 1$ the sum vanishes)

$$\mathbf{p}_m = \sum_{i=1}^{m-1} \lambda_i^{(m)} \mathbf{p}_i + \mathbf{e}_{\hat{k}(m)}. \tag{2.4.2}$$

Minimize along the direction of $\mathbf{p}_m$: i.e., update

$$\hat{\beta}^{(m)} = \hat{\beta}^{(m-1)} + \frac{(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})^T \mathbf{X} \mathbf{p}_m}{\mathbf{p}_m^T \mathbf{X}^T \mathbf{X} \mathbf{p}_m} \mathbf{p}_m. \tag{2.4.3}$$

Step 3: Increase iteration index m by one and go back to Step 2.

The number of iterations is again estimated using a validation set or cross validation.

We can take advantage of sparse vectors when implementing the CDBoost algorithm, in particular when $d \gg n$. The vectors $\mathbf{p}_m$ and $\hat{\beta}^{(m)}$ have m non-zero entries, and the unit-coordinate vectors $\mathbf{e}_k$ have only one non-zero entry. Thus, the terms $\mathbf{X}\mathbf{e}_{\hat{k}(m)}$, $\mathbf{X}\mathbf{p}_i$ $(i = 1, \ldots, m)$ and $\mathbf{X}\hat{\beta}^{(m-1)}$ can be computed efficiently in $O(nm)$ operations, and there is no need to compute $\mathbf{X}^T\mathbf{X}$ which is of particular interest if d is (very) large.

CDBoost is not as simple as the *conjugate gradient method*, because we have to store all the previous directions. On the other hand we have an algorithm which performs variable selection because only one coefficient of $\hat{\beta}^{(m)}$ in each step changes from zero to non-zero.

A fast forward stepwise variable selection algorithm

Here, we establish an equivalence of CDBoost to the orthogonal greedy algorithm (cf. Temlyakov 2000). The latter is similar to forward stepwise variable selection as described in section 2.2.1. It also employs least squares based on the selected covariates but a new covariate in the m-th selection step is chosen as the minimizer, with respect to j, of

$$|\mathbf{x}_j^T \mathbf{r}| / \mathbf{x}_j^T \mathbf{x}_j, \tag{2.4.4}$$

where $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)}$ are the current residuals. Note that forward stepwise variable selection chooses the covariate such that the loss $L(\beta)$ is minimized when fitting least squares on the previously and currently selected covariates.

Proposition 1 *The solutions of CDBoost and the orthogonal greedy algorithm coincide in every iteration $m \le \min(n-1, d) = rank(X)$. In particular, in every iteration CDBoost employs least squares fitting in the corresponding sub-model of selected covariates.*

Proof: (i) We first show that for every iteration m, $\hat{\beta}^{(m)}$ is the least squares solution using the selected m covariates. If we started our algorithm again with only these m covariates, we would reach the same model after m iterations and this is the least squares solution because we use a conjugate direction method. Thus, we have proved that CDBoost yields least squares solutions when fitting $\mathbf{y}$ to the selected covariates.

In particular, in every iteration CDBoost selects a covariate which has not been selected before and the saturated model is reached after $\min(n-1, d)$ iterations.

(ii) For the equivalence to the orthogonal greedy algorithm (OGA), we only need to show that CDBoost selects the same covariates as the OGA (since both algorithms employ least squares fitting on the selected covariates). But this holds by the equivalence of formula (2.4.1) for CDBoost and (2.4.4) for OGA (note that (2.4.1) is for standardized variables with $\mathbf{x}_j^T\mathbf{x}_j = 1$). □

It should be pointed out that CDBoost and the orthogonal greedy method are different algorithms. As a conjugate direction method, CD-Boost has natural generalizations to be used with small step sizes (see section 2.5) and to the non-parametric setting (see section 2.8).

Computations

We are interested in the computational complexity for computing the full path of solutions of a method: that is, all possible solutions from the empty model with no covariates to the fully saturated model from least squares (if $d \leq n-1$) or with zero residual sum of squares (if $d \geq n$).

Proposition 2 *The computational complexity of CDBoost for computing the full path of solutions is $O(nd \min(n, d))$.*

Proof: Consider the m-th iteration of CDBoost: the negative gradient can be computed at a cost of $O(nd)$, all the λ's at a cost of $O(nm)$ (reusing terms computed in earlier iterations), the new direction at a cost of $O(dm)$ and the update at a cost of $O(nd)$. Thus, the computational cost for the m-th iteration is

$$O(nd) + O(nm) + O(dm). \qquad (2.4.5)$$

Because the fully saturated model is fitted after $\min(n-1, d)$ iterations (which follows from Proposition 1), we get $m \leq \min(n-1, d)$. Hence by formula (2.4.5), the cost for computing the whole path of CDBoost solutions is $O(nd \min(n, d))$. □

The result from Proposition 2 can be compared with other methods. The computation of the whole path of the LARS solutions is of the same computational complexity $O(nd\min(n,d))$ (cf. Efron et al. 2004; in case of $d > n-1$, their statement of $O(n^3)$ is wrong and should be $O(n^2d) = O(nd\min(n,d))$). Thus, CDBoost and LARS are comparable in terms of computational cost: while the former does least squares on the selected variables, the latter employs shrinkage from L_1-penalisation.

The full path of classical forward stepwise variable selection costs $O(nd^2)$ (cf. Miller 2002), regardless whether d is smaller or greater than n. Thus, for $d < n$, forward stepwise variable selection is as fast as CDBoost or LARS; for $d \gg n$, the situation is markedly different and forward stepwise variable selection is no longer linear in the dimension d. The main point is that forward stepwise variable selection depends on the square of the number of covariates out of the model and CDBoost depends on the square of the number of covariates in the model.

The computational cost for the path of L_2Boosting until boosting iteration m is $O(ndm)$. In cases where the number of boosting iterations m is of the order of $\min(n,d)$, we have comparable computational speed to CDBoost or LARS. However, to compute the fully saturated model we would typically need infinitely many ($m = \infty$) iterations: the convergence speed depends in a complicated manner on the design matrix $\mathbf{X}$.

CDBoost and forward stepwise variable selection employ least squares fits based on the selected variables while LARS (Lasso) and L_2Boosting yield shrunken estimates. For cases where $d \gg n$, CDBoost can have a substantial computational advantage over forward stepwise variable selection.

2.5 Shrinkage or small step size

Noticed by Friedman (2001), the predictive accuracy of boosting (and forward stagewise fitting) can be improved by a simple shrinkage strategy: in each boosting step, only a small fraction ν (for example $\nu = 0.1$) of the optimal update is added. This can be interpreted as a small step size along a constrained gradient. Of course, there is a computational cost when using small step sizes, but it usually pays off as a clear improvement in accuracy of predictions.

We modify the update in *Step 2* of the L_2Boosting algorithm from section 2.3 to

$$\hat{F}^{(m)}(x) = \hat{F}^{(m-1)}(x) + \nu \cdot \hat{\beta}_{\hat{k}(m)} x_{\hat{k}(m)} \quad \text{(function version)}$$

or

$$\hat{\beta}^{(m)}_{\hat{k}(m)} = \hat{\beta}^{(m-1)}_{\hat{k}(m)} + \nu \cdot \mathbf{x}^T_{\hat{k}(m)}(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)}) \quad \text{(gradient version).}$$

The implementation of a similar strategy with small step sizes for forward stepwise variable selection or the orthogonal greedy algorithm (see section 2.4.2) is not straightforward. However, with CDBoost (which yields the same solutions as the orthogonal greedy algorithm, see Proposition 1) we have a natural concept of using small step sizes: in each iteration we only take a fraction of the optimal step along the conjugate direction.

2.5.1 Conjugate direction boosting with restart

A new problem arises when we use shrinkage for CDBoost. Without shrinkage, each covariate can only be chosen once. This is no longer true with shrinkage because the coefficients $\hat{\beta}^{(m)}$ are no longer least squares solutions (in sub-models). Thus, it may happen that a covariate already included in the model fits the current residuals best. In this case, we cannot find a *new* direction as in (2.4.2) which is conjugate to all previous directions, because there are only m conjugate directions in an m-dimensional space. A possible remedy is discussed next.

An easy and effective solution to the problem is to restart the algorithm when a variable is chosen the second time. We take the actual $\hat{\beta}^{(m-1)}$ as the new starting $\hat{\beta}$, delete all the conjugate directions $\mathbf{p}_i$ and start again. A justification for restarting is described below in Proposition 3. To keep track of the total number of iterations, denoted by m, we formulate our algorithm by marking the starting points which are denoted by an index s.

CDBoost algorithm with shrinkage and L_2-loss for linear regression:

Step 1: Standardize $\mathbf{x}_i$ to zero mean and unit length. Standardize $\mathbf{y}$ to zero mean. Initialize $\hat{\beta}^{(0)} = \mathbf{0}$, $m = 1$ and $s = 1$.

Step 2: Compute the negative gradient

$$-\nabla L^{(m)} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})$$

and take the component with highest absolute value: $\hat{k}^{(m)} = \arg\max_j |-\nabla L_j^{(m)}|$. Define the constrained gradient by $\mathbf{e}_{\hat{k}^{(m)}}$ (unit-coordinate vector).

If $\hat{k}^{(m)} \in \{\hat{k}^{(s)}, \dots, \hat{k}^{(m-1)}\}$ restart: Set $s = m$.

- If $m = s$:
$$\mathbf{p}_m = \mathbf{e}_{\hat{k}^{(m)}}.$$
- If $m > s$: For $i = s, \dots, m-1$ compute the coefficients for the linear combination
$$\lambda_i^{(m)} = -\frac{\mathbf{p}_i^T\mathbf{X}^T\mathbf{X}\mathbf{e}_{\hat{k}^{(m)}}}{\mathbf{p}_i^T\mathbf{X}^T\mathbf{X}\mathbf{p}_i},$$
and compute the new direction
$$\mathbf{p}_m = \sum_{i=s}^{m-1} \lambda_i^{(m)}\mathbf{p}_i + \mathbf{e}_{\hat{k}^{(m)}}.$$

Minimize along the direction of $\mathbf{p}_m$, i.e., update $\hat{\beta}$:

$$\hat{\beta}^{(m)} = \hat{\beta}^{(m-1)} + \nu\,\frac{(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})^T\mathbf{X}\mathbf{p}_m}{\mathbf{p}_m^T\mathbf{X}^T\mathbf{X}\mathbf{p}_m}\mathbf{p}_m.$$

Step 3: Increase iteration index m by one and go back to Step 2.

As already indicated, by the inclusion of shrinkage we no longer have a proper conjugate direction method and thus, we do not reach the least squares solution after $\min(n-1, d) = \text{rank}(\mathbf{X})$ iterations. With shrinkage, the goal is to generate a "good" path from the empty model to the least squares solution (which is obtained with possibly infinitely many iterations). The conjugate direction method is only used here to propose "good" directions rather than achieving fast convergence to a good submodel. In section 2.7 we show that shrinkage can dramatically improve the predictive performance.

We now give an illustrating example with 3 correlated covariates, $\beta = (3, 2, 1)^T$, low noise and $\nu = 0.005$. Covariate 1 has highest absolute gradient value and is therefore chosen first in CDBoost with shrinkage. After a tiny update in direction $\mathbf{p}_1 = (1, 0, 0)^T$, covariate 1 is still the best choice. This leads to a restart and we again take variable 1 followed by a step in direction $\mathbf{p}_1$. This scenario is repeated until $\hat{\beta}^{(43)} = (1.03, 0, 0)^T$, where covariate 2 gets higher absolute gradient value. Then, we perform an update in the direction $\mathbf{p}_2 = (-0.81, 1, 0)^T$ which is conjugate to $\mathbf{p}_1$. In the last two steps (along the directions $\mathbf{p}_1$ and $\mathbf{p}_2$) we moved the fraction ν from $\hat{\beta}^{(42)}$ to the least squares solution of covariate 1 and 2.

Prior to these two steps, covariate 1 had a higher absolute negative gradient value than covariate 2 and the same is true after these two steps (see the Proof of Proposition 3 below). Thus, we choose variable 1 which causes a restart and a step in direction $\mathbf{p}_1$ followed by the choice of variable 2 and a step in direction $\mathbf{p}_2$ followed by another restart with variable 1. This is repeated until $\hat{\beta}^{(139)} = (1.96, 0.93, 0)^T$, where variable 3 gets highest absolute negative gradient value. Then we perform an update in the direction $\mathbf{p}_3 = (-0.43, -0.48, 1)^T$ conjugate to $\mathbf{p}_1$ and $\mathbf{p}_2$. The next choice is covariate 1 which causes again a restart and we repeatedly find the sequence of variables 1, 2, 3, restart until we reach (theoretically after ∞ steps) the least squares solution $\hat{\beta} = (3.03, 1.94, 1.02)^T$.

In summary: the selected covariate indices are

$$\underbrace{1, 1, 1, 1, \ldots, 1}_{\text{42 times}}, \underbrace{1, 2, 1, 2, 1, 2, 1, 2, \ldots, 1, 2}_{\text{95 times the pair (1,2)}}, \underbrace{1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, \ldots}_{\infty \text{ times the triple (1,2,3)}}.$$

It is no coincidence that the restart was always caused by covariate 1, as described by the following result (the repeating pattern of selected pairs and triples is also a more general fact).

Proposition 3 *The CDBoost algorithm has the following property: It is always the first chosen covariate* $\mathbf{x}_{\hat{k}(1)}$ *which causes a restart.*

Proof: Suppose we have an arbitrary $\hat{\beta}^{(m)}$ and j conjugate directions involving j covariates. Now we restrict everything (L, ∇L, $\hat{\beta}$) to these j covariates (denoted with $\cdot|_j$). We can minimize $L|_j$ (find the least squares solution for these j covariates) by j individual minimizations along the j conjugate directions. For each of the j minimizations we take only ν of the optimal step length and therefore we only move the fraction ν from $\hat{\beta}^{(m)}$ to the least squares solution of the j covariates. The new negative gradient after these j steps is

$$\begin{aligned} -\nabla L^{(m+j)}|_j &= \mathbf{X}|_j^T\left(\mathbf{y} - \mathbf{X}|_j\left(\hat{\beta}^{(m)}|_j + \nu(\hat{\beta}^{(OLS)}|_j - \hat{\beta}^{(m)}|_j)\right)\right) \\ &= \mathbf{X}|_j^T\left(\nu\mathbf{y} - \nu\mathbf{X}|_j\hat{\beta}^{(OLS)}|_j + (1-\nu)\mathbf{y} - (1-\nu)\mathbf{X}|_j\hat{\beta}^{(m)}|_j\right) \\ &= (1-\nu)(-\nabla L^{(m)}|_j) \qquad (2.5.6) \end{aligned}$$

because the negative gradient at the least squares solution is $\mathbf{0}$. We see that the involved components of the negative gradient are scaled by the factor $1-\nu$ and therefore it is the same component which has highest absolute value before and after the j updates.

Suppose covariate $\hat{k}^{(1)}$ has highest absolute gradient value and CDBoost yields the following sequence of covariate indices at the beginning: $\hat{k}^{(1)}, \ldots, \hat{k}^{(j)}$, where all $\hat{k}^{(i)}, i = 1, \ldots, j$ are different. It follows from (2.5.6) that the variable with index $\hat{k}^{(1)}$ has higher absolute gradient value than $\hat{k}^{(2)}, \ldots, \hat{k}^{(j)}$. In the next iteration we therefore choose either $\hat{k}^{(1)}$ (restart) or a completely new covariate (no restart necessary). □

2.5.2 Sparseness

CDBoost shows an interesting behavior of sparseness as a function of the shrinkage factor ν: when using an estimated prediction-optimal stopping iteration (from a validation set), CDBoost with $\nu = 1$ has a tendency to yield a coefficient vector $\hat{\beta}$ with small L_0-norm (number of non-zero coefficients) while an infinitesimally small ν tends to yield small L_1-norm. Table 2.1 illustrates this for the model from section 2.7.1 with various signal-to-noise ratios.

We point out that the results from Table 2.1 are only on average (over 100 datasets), and the behavior of CDBoost on an individual

	stnr = 9		*stnr* = 4		*stnr* = 1	
ν	L_0-norm	L_1-norm	L_0-norm	L_1-norm	L_0-norm	L_1-norm
1	6.91	28.1	6.73	29.4	4.99	32.0
0.7	7.84	27.0	7.33	27.3	5.88	26.7
0.5	7.99	26.6	7.62	26.7	6.59	27.6
0.3	8.35	26.6	7.95	26.4	6.76	26.5
0.1	8.31	26.4	8.15	26.8	6.93	26.5
0.03	8.36	26.5	8.16	26.9	6.95	26.8
0.01	8.39	26.6	8.17	26.9	7.03	26.9

Table 2.1: *Average L_0- and L_1-norm of the coefficient vector $\hat{\beta}$ from CDBoost with various signal-to-noise ratios (stnr) and shrinkage factors ν for simulation model 1 (see section 2.7.1). The iteration is stopped with a validation set.*

dataset sometimes differs from the average tendency. Nevertheless, we may interpret the step size ν in CDBoost as a parameter governing the tendency for sparseness of solutions. For L_2Boosting, the sparseness of solutions is less sensitive to the choice of the shrinkage factor ν. For example, CDBoost with $\nu = 1$ yields least squares estimation on selected variables (Proposition 1) while in general, L_2Boosting with $\nu = 1$ produces shrunken least squares estimates on selected variables.

For the special case with an orthonormal design $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ and $d \leq n - 1$, a rigorous result holds. CDBoost with $\nu = 1$ yields L_0-penalized regression estimators while an infinitesimally small ν yields L_1-penalized solutions. For both cases, the iteration m in CDBoost corresponds to the penalty parameter for L_0- and L_1-penalisation, respectively. The first statement with $\nu = 1$ is a direct consequence of Proposition 1 and the orthonormality of the design; the case with infinitesimally small ν can be derived from the fact that CDBoost and L_2Boosting coincide due to the orthonormal design and therefore, the statement follows from Bühlmann and Yu (2005).

2.6 Connections to LARS

Efron et al. (2004) recently proposed least angle regression (LARS). A modification of LARS yields all Lasso solutions and another modification yields forward stagewise fitting (boosting with infinitesimal shrinkage). When we use boosting with a small shrinkage parameter ν, we almost reproduce the forward stagewise fitting path. The bigger we take ν, the more we depart from this path.

For the rest of this section we assume infinitesimal shrinkage for boosting and CDBoost. LARS and boosting yield the same path, when all the β-coefficients move monotonically away from zero as the LARS or boosting iterations increase. In this case CDBoost also leads to that path. For more general cases, turnarounds of coefficients (for example decreasing after increasing) are possible with LARS. Forward stagewise fitting behaves differently in such a case: it drops that variable from the active set of selected covariates and leaves its coefficient unchanged in the next step. The CDBoost algorithm works in between LARS and boosting/forward stagewise. Two examples with 3 covariates demonstrate this behavior. The following tables show the traces (endpoints of linear pieces) of the $\hat{\beta}$-coefficients from zero to the least squares solutions.

Example 1: LARS/CDBoost

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
2	0	0
5	3	0
2	10	6

Boosting/Stagewise

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
2	0	0
5	3	0
5	6	3
2	10	6

Example 2: LARS

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
7	0	0
13	5	0
19	3	5

Boosting/Stagewise/CDBoost

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
7	0	0
13	5	0
15	5	2
19	3	5

LARS always finds the least squares solution after 3 fairly large steps (in principle we can also evaluate intermediate models between two steps). Forward stagewise fitting drops covariate 1 in example 1 and covariate 2 in example 2 after step 2, the corresponding coefficient remains and it needs one more step to reach the least squares solution (note the turnaround of coefficient $\hat{\beta}_1$ in example 1 and $\hat{\beta}_2$ in example 2).

CDBoost behaves like LARS in the first example and like forward stagewise fitting in the second. Remember that CDBoost takes a lot of tiny steps to produce one LARS-step. At the beginning it always takes covariate 1 until the first LARS-step is reproduced, then covariate 1 and 2 in an alternating fashion. In example 1 it is the first chosen variable which turns around and CDBoost behaves like LARS. This is because covariate 1 is always chosen after a restart and its coefficient is adjusted in each step. In example 2 it is not the first chosen variable which turns around and CDBoost behaves like forward stagewise fitting. The selected variable indices with CDBoost in example 2 are: $1, 1, \ldots, 1,\ 1, 2, 1, 2, \ldots, 1, 2,\ 1, 3, 1, 3, \ldots, 1, 3,\ 1, 3, 2, 1, 3, 2, \ldots$.

2.7 Simulations with linear regression

Now, we compare CDBoost with boosting, forward stepwise variable selection and the three LARS methods (LARS, Lasso and forward stagewise fitting) for simulated linear regression models.

In our simulation study, the data is split into three parts: a training set, a validation set, both of equal size n and a test set of size 1000. The training set is used to fit the models and the validation set for stopping the iterations (the number of iterations is chosen to minimize the validation error). For CDBoost and boosting, the validation set is also used to chose the shrinkage factor ν. Finally we use the test set to measure the prediction error on the test set:

$$\frac{1}{|\text{testset}|} \sum_{x_i \in \text{testset}} (\hat{F}(x_i) - F(x_i))^2,$$

where $F(x) = x^T \beta$ is the true underlying regression function.

CDBoost (and boosting) has a second tuning parameter, the shrinkage factor ν which governs the tendency between L_0- and L_1-penalisation. To compare CDBoost to boosting we can fix ν and compare the methods (although the interpretation of a larger ν is not the same for the two methods). To compare CDBoost and boosting to forward stepwise variable selection and the three LARS methods, we use the validation set to choose among the shrinkage factors $0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03,\ 0.01$ (as well as to chose the number of iterations).

In the next two subsections we give some results of simulations where the model is always chosen to be linear with normally distributed errors.

2.7.1 Model 1: $n = 100,\ d = 10,\ d_{eff} = 5$

Setup for model 1

The size of the training set is $n = 100$ and the number of covariates is $d = 10$. The number of covariates with effective influence on the response is only $d_{\text{eff}} = 5$.

We draw the covariates from a multivariate normal distribution. This is done in two steps by first producing $\tilde{x}_i$ i.i.d. $\sim \mathcal{N}_d(0, \mathbf{I})$ and then multiplying $\tilde{\mathbf{X}}$ with the $d \times d$ matrix $\mathbf{B}$ to get the desired correlation structure ($\mathbf{X} = \tilde{\mathbf{X}}\mathbf{B}$). $\mathbf{B}$ is chosen to be (blank entries correspond to zero):

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & 1 & 1 & & \\ & & & & 1 & 1 & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 & 1 & 1 \\ & & & & & & 1 & 1 & 1 & 1 \\ & & & & & & & 1 & 1 & 1 \\ & & & & & & & & 1 & 1 \\ & & & & & & & & & 1 \end{pmatrix}.$$

This leads to a correlation matrix with approximately each third entry equal to zero. The other two thirds of the correlations range from 0.2 to 0.9. Finally we arbitrarily permute the columns of $\mathbf{X}$.

The next step is to choose the true β-vector: the first five components of β are $\beta_1, \ldots, \beta_5$ i.i.d. $\sim \mathcal{N}(5, 1)$, and the other five components are zero.

The last issue is to specify the variance for the normal distributed errors ϵ. We choose it via the more meaningful *signal-to-noise ratio* which we define as

$$stnr = \frac{\mathrm{Var}\,(F(x))}{\mathrm{Var}\,(\epsilon)}.$$

We select the desired signal-to-noise ratio and determine then the corresponding $\mathrm{Var}\,(\epsilon)$. A signal-to-noise ratio of 1 corresponds to a "population" R^2 equal to $\mathrm{Var}\,(F(x))\,/\mathrm{Var}\,(y) = 0.5$.

We simulate 100 datasets from each setting. Because the true β-coefficients are different for each simulation we cover a greater range of models. The overall performance is the mean over the 100 test errors. We also give the standard error of the mean divided by the mean (relative standard error/some kind of coefficient of variation). Paired sample Wilcoxon tests are used to quantify significance when comparing CDBoost to the other methods.

We report on the test error in a standardized form. We divide it by the test error of the best constant (location) model and multiply it by 100. Each method should be better than the best constant model and therefore achieve a value below 100. This standardized test error makes it possible to compare different settings. For convenience we still refer to it as test error.

Results for model 1

Table 2.2 shows the comparison of CDBoost and boosting. CDBoost performs better and the difference is significant for the higher *stnr*'s. Both methods do substantially better with shrinkage. While boosting needs small ν's for best performance, CDBoost gives good results for the whole range of shrinkage factors. Since there are five pure noise variables, the usage of some sort of L_0-penalisation seems to be adequate. This may explain why CDBoost performs best with larger ν's, see also section 2.5.2.

In table 2.3 and figure 2.1 we compare CDBoost to boosting, forward stepwise variable selection and the three LARS methods. For CDBoost and boosting, the validation set is used to choose the shrinkage factor ν for each simulation from the candidate values 0.7, 0.5, 0.3, 0.1, 0.03, 0.01. This gives better results compared to a fixed ν and shows that in some of the 100 simulations a solution close to forward stepwise variable selection is appropriate and in other cases a solution closer to LARS is

		stnr = 9			*stnr* = 4			*stnr* = 1		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	1.20			2.81			9.85		
	$\nu = 0.7$	1.10			2.34			7.47		
	$\nu = 0.5$	**1.08**			**2.22**			**7.32**		
	$\nu = 0.3$	1.08			2.24			7.39		
	$\nu = 0.1$	1.09			2.27			7.42		
	$\nu = 0.03$	1.09			2.28			7.53		
	$\nu = 0.01$	1.09			2.28			7.47		
Boo	$\nu = 1$	1.33	11.2	9e−05	2.91	3.7	1e−01	9.56	−3.0	5e−01
	$\nu = 0.7$	1.28	15.8	3e−05	2.71	15.7	2e−06	8.24	10.4	4e−04
	$\nu = 0.5$	1.20	11.8	4e−05	2.60	17.0	2e−05	7.71	5.3	2e−02
	$\nu = 0.3$	1.20	10.9	6e−03	**2.44**	8.6	2e−03	7.63	3.3	1e−01
	$\nu = 0.1$	1.20	10.5	2e−03	2.44	7.5	5e−04	7.50	1.0	8e−01
	$\nu = 0.03$	**1.20**	9.7	3e−02	2.44	7.4	9e−03	**7.46**	−0.9	6e−01
	$\nu = 0.01$	1.20	10.2	5e−03	2.44	7.1	5e−03	7.49	0.2	7e−01

Table 2.2: *Comparison of conjugate direction boosting (CDB) and boosting (Boo) for various signal-to-noise ratios (stnr) and shrinkage factor ν for simulation model 1. Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and each stnr is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests. The standard errors of the means divided by the means range from 4.9% to 6.2%.*

adequate. CDBoost (and also boosting) does a good job in choosing a right ν and is therefore very flexible.

CDBoost performs clearly and significantly best and forward stepwise variable selection leads to the worst models. The gain of CDBoost compared to boosting and forward stagewise increases with higher *stnr*'s and its gain compared to LARS and the Lasso is fairly constant. Forward stagewise fitting is always worse than boosting so infinitesimal shrinkage is not worthwhile in this example.

	stnr = 9			*stnr* = 4			*stnr* = 1		
	test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	1.00			2.19			7.16		
Boo	1.16	16.3	3e−04	2.40	9.5	1e−02	7.35	2.7	5e−02
FVS	1.18	17.6	4e−04	2.93	33.6	3e−07	9.83	37.3	3e−11
LAR	1.13	12.4	1e−04	2.45	11.7	5e−05	8.20	14.6	1e−08
Las	1.10	9.6	4e−04	2.38	8.5	9e−04	7.97	11.4	1e−07
For	1.21	20.6	7e−08	2.52	15.3	4e−07	7.93	10.7	9e−07

Table 2.3: *Comparison of conjugate direction boosting (CDB), boosting (Boo), forward stepwise variable selection (FVS) and the three LARS methods (LARS, Lasso, forward stagewise fitting) for simulation model 1. Given is the (standardized) mean of the test error (test e) over 100 simulations. The shrinkage factors ν for CDBoost and boosting are chosen using the validation set from the candidate values* 0.7, 0.5, 0.3, 0.1, 0.03, 0.01. *The loss compared to CDBoost is given in % and the p-values are from paired sample Wilcoxon tests. The standard errors of the means divided by the means range from 5.0% to 6.0%.*

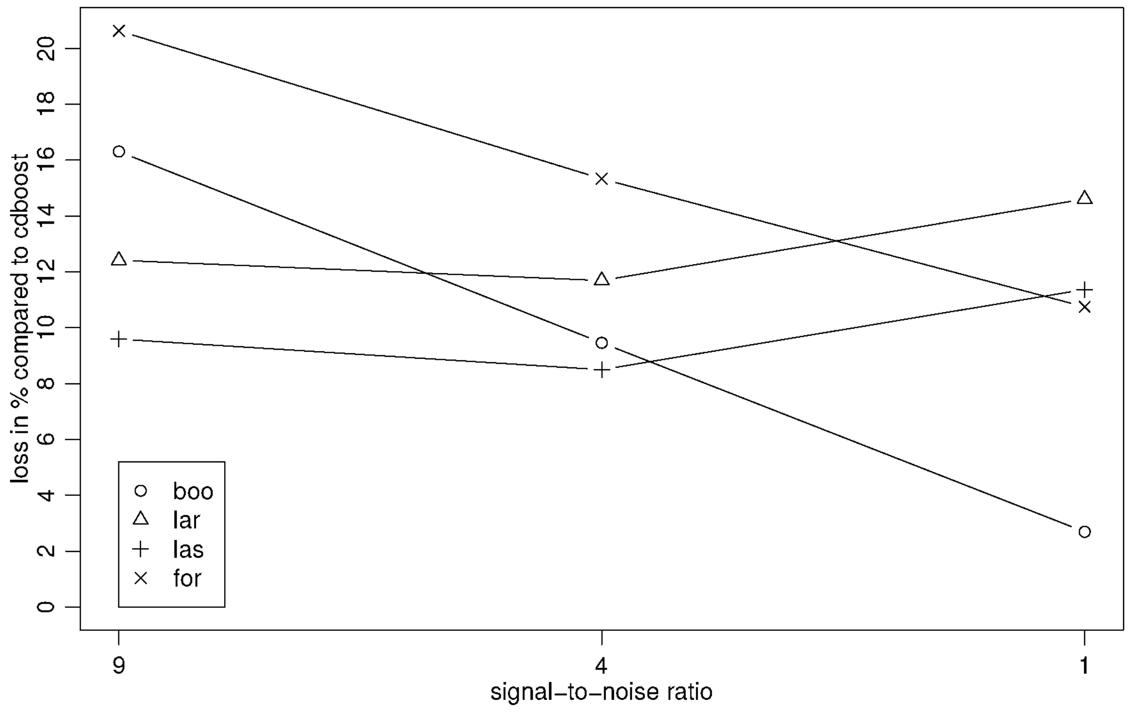

Figure 2.1: *Percentage loss of the different methods (without forward stepwise variable selection) compared to CDBoost for simulation model 1.*

2.7.2 Model 2: $n = 50,\ d = 2000,\ d_{eff} = 10$

Setup for model 2

We now reduce the size of the training set to $n = 50$ and increase the number of covariates to $d = 2000$, whereas $d_{\text{eff}} = 10$ variables have an effective influence on y. The covariates are constructed as in section 2.7.1 but with

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & & & & & & \\ 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 \\ & & & & & & 1 & 1 \end{pmatrix}$$

and without permuting the columns of $\mathbf{X}$. This leads to a correlation matrix with 2/3 in the secondary diagonals and 1/3 in the tertiary diagonals.

The ten non-zero coefficients are $\beta_{31}, \ldots, \beta_{35}, \beta_{66}, \ldots, \beta_{70}$ i.i.d. $\sim \mathcal{N}(5, 1)$. So there are two blocks of 5 correlated covariates with real influence on y. Because this setup is much harder than model 1 we use higher *stnr*'s.

Results for model 2

Table 2.4 shows the comparison of CDBoost and boosting. The results are similar to that of model 1. CDBoost performs better than boosting, especially for high *stnr*. In this model, ν should not be too small for CDBoost.

In table 2.5 and figure 2.2 we compare CDBoost to boosting, forward stepwise variable selection and the three LARS methods. For CDBoost and boosting, the validation set is used to choose the shrinkage factor ν for each simulation from the candidate values 0.7, 0.5, 0.3, 0.1, 0.03, 0.01.

		stnr = 16			*stnr* = 9			*stnr* = 4		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	23.7			26.4			36.5		
	$\nu = 0.7$	15.9			21.7			**33.0**		
	$\nu = 0.5$	15.3			**21.5**			34.2		
	$\nu = 0.3$	**15.3**			21.7			34.5		
	$\nu = 0.1$	15.8			22.4			36.2		
	$\nu = 0.03$	16.0			22.8			36.9		
	$\nu = 0.01$	16.1			23.0			37.2		
Boo	$\nu = 1$	28.0	17.8	2e−07	30.4	15.1	5e−07	38.5	5.7	5e−03
	$\nu = 0.7$	20.4	28.2	3e−08	25.8	18.9	2e−05	**34.7**	5.2	2e−01
	$\nu = 0.5$	19.0	24.6	4e−11	24.1	11.9	3e−05	36.3	6.0	6e−03
	$\nu = 0.3$	18.0	18.2	2e−15	24.5	13.1	1e−12	37.2	7.8	1e−08
	$\nu = 0.1$	16.6	5.0	1e−10	**22.9**	2.3	3e−06	36.6	1.1	3e−02
	$\nu = 0.03$	16.4	2.5	8e−07	23.0	0.7	2e−02	37.0	0.2	2e−01
	$\nu = 0.01$	**16.4**	1.8	9e−05	22.9	−0.0	1e−00	37.1	−0.1	4e−01

Table 2.4: *Comparison of conjugate direction boosting (CDB) and boosting (Boo) for various signal-to-noise ratios (stnr) and shrinkage factor ν for simulation model 2. Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and each stnr is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests. The standard errors of the means divided by the means range from 3.2% to 5.9%.*

Again, results are similar to model 1 with CDBoost performing significantly best.

2.8 Generalization to arbitrary learners

We now generalize our approach to any base learner, for example trees or componentwise smoothing splines (Bühlmann and Yu 2003, see also section 2.9) for non-parametric regression. In principle, we could use linear expansions in tree-type or B-spline basis functions, resulting in a huge design matrix. Instead, we develop an iterative, computationally more efficient approach. The idea is to construct a matrix $\tilde{\mathbf{X}}$ during the iteration process, rather than using a fixed design matrix $\mathbf{X}$ given in

	$stnr = 16$			$stnr = 9$			$stnr = 4$		
	test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	13.7			19.6			30.7		
Boo	15.1	10.1	1e−06	20.7	5.5	2e−02	32.1	4.6	7e−03
FVS	20.1	46.3	2e−07	24.1	22.7	7e−05	34.9	13.7	2e−02
LAR	15.8	14.6	3e−08	22.9	16.3	1e−11	37.6	22.4	1e−13
Las	15.7	14.0	1e−07	22.7	15.4	7e−11	37.2	21.2	1e−13
For	16.4	19.6	4e−13	23.0	17.1	3e−13	37.3	21.5	4e−14

Table 2.5: *Comparison of conjugate direction boosting (CDB), boosting (Boo), forward stepwise variable selection (FVS) and the three LARS methods (LARS, Lasso, forward stagewise fitting) for simulation model 2. Given is the mean of the (standardized) test error (test e) over 100 simulations. The shrinkage factors ν for CDBoost and boosting are chosen using the validation set from the candidate values* 0.7, 0.5, 0.3, 0.1, 0.03, 0.01. *The loss compared to CDBoost is given in % and the p-values are from paired sample Wilcoxon tests. The standard errors of the means divided by the means range from 3.2% to 5.6%.*

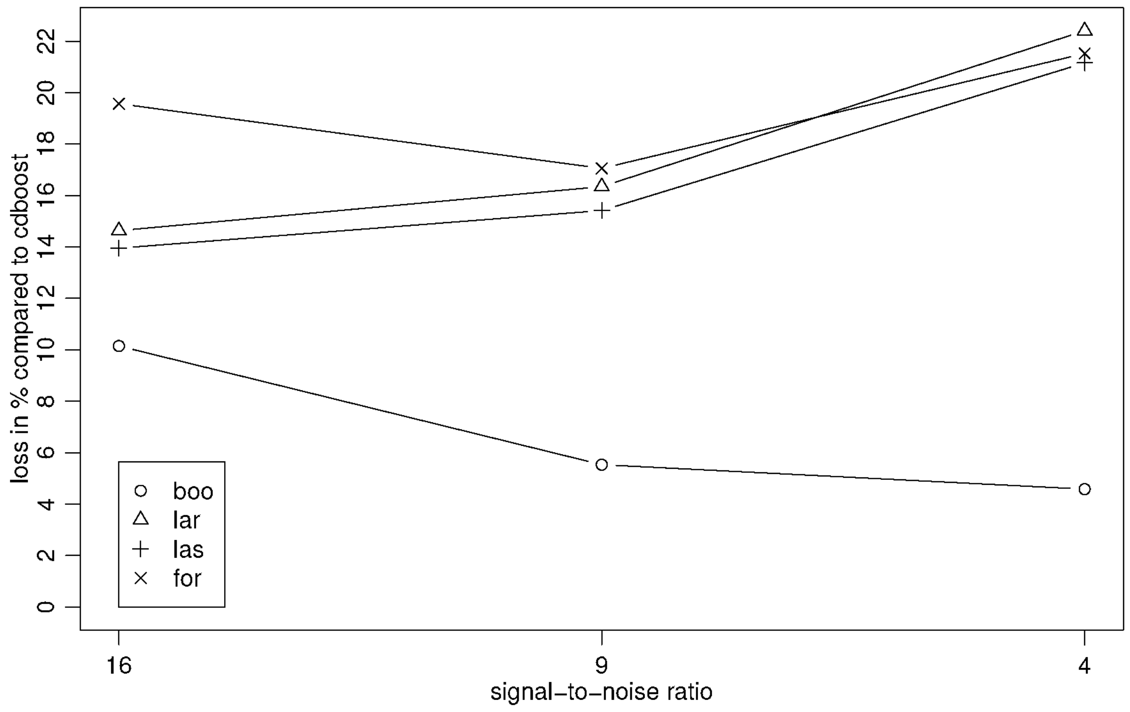

Figure 2.2: *Percentage loss of the different methods (without forward stepwise variable selection) compared to CDBoost for simulation model 2.*

advance. In each iteration the learner produces a vector of fitted values for the training observations (which can be viewed as an estimated basis function) and that vector is taken as a new column of $\tilde{\mathbf{X}}$. Thus, $\tilde{\mathbf{X}}$ grows in each step by one column, and this matrix $\tilde{\mathbf{X}}$ is then used in our CDBoost method. The following algorithm makes the idea precise:

General CDBoost algorithm with L_2-loss:

Step 1: Standardize $\mathbf{y}$ to zero mean. The standardization of the $\mathbf{x}_i$ is not necessary. Initialize $\tilde{\mathbf{X}} = \text{null}$, $\hat{\beta}^{(0)} = \text{null}$ and $m = 1$.

Step 2: Compute the current residuals $\mathbf{r} = \mathbf{y} - \tilde{\mathbf{X}}\hat{\beta}^{(m-1)}$ and fit the base learner to them using the predictor matrix $\mathbf{X}$. The fit is denoted by $\hat{f}^{(m)}(x)$ and the vector of fitted values by $\tilde{\mathbf{x}}_m \in \mathbb{R}^n$.

Increase $\tilde{\mathbf{X}}$ by one column by adding $\tilde{\mathbf{x}}_m$. Now $\tilde{\mathbf{X}}$ is of dimension $n \times m$. Extend $\hat{\beta}^{(m-1)}$ and all direction vectors $\mathbf{p}_i, i < m$, by a zero so that they all have length m.

For $i = 1, \ldots, m-1$ compute the coefficients for the linear combination of $\mathbf{p}_m \in \mathbb{R}^m$ (for $m = 1$ there are no $\lambda's$):

$$\lambda_i^{(m)} = -\frac{\mathbf{p}_i^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{e}_m}{\mathbf{p}_i^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{p}_i}.$$

Compute the new direction (for $m = 1$ the sum vanishes)

$$\mathbf{p}_m = \sum_{i=1}^{m-1} \lambda_i^{(m)} \mathbf{p}_i + \mathbf{e}_m.$$

Minimize in the direction of $\mathbf{p}_m$, that means update $\hat{\beta}$:

$$\hat{\beta}^{(m)} = \hat{\beta}^{(m-1)} + \frac{(\mathbf{y} - \tilde{\mathbf{X}}\hat{\beta}^{(m-1)})^T \tilde{\mathbf{X}} \mathbf{p}_m}{\mathbf{p}_m^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{p}_m} \mathbf{p}_m.$$

Step 3: Increase iteration index m by one and go back to Step 2.

The resulting function estimator in iteration m is then

$$\hat{F}^{(m)}(x) = \bar{y} + \sum_{i=1}^{m} \hat{\beta}_i^{(m)} \hat{f}^{(i)}(x)$$

and can be evaluated at any x.

The main difference to the linear CDBoost is that we do not compute the negative gradient, but fit the learner to the current residuals. The predicted values give $\tilde{\mathbf{x}}_m$. Loosely speaking, the "constrained gradient" corresponding to (2.4.1) is $\mathbf{e}_m \in \mathbb{R}^m$, because $\tilde{\mathbf{x}}_m$ is already a constrained functional gradient (fitting the residuals best); note the correspondence with the function and gradient version of L_2Boosting in section 2.3. Having a constrained gradient, we continue as in the case of linear regression: we compute $\mathbf{p}_m \in \mathbb{R}^m$ and optimize in that direction. This means that we adjust the coefficients of the fitted learners already included in the model.

The addition of shrinkage is implemented as follows: we recall that in the linear regression case we restart when a variable is chosen a second time. For arbitrary learners this would happen when obtaining the identical fitted values as in an earlier step. It can be difficult to determine numerically whether we have the same fitted values as before. We therefore propose restarting when the absolute correlation between $\tilde{\mathbf{x}}_m$ and a $\tilde{\mathbf{x}}_i$, $i < m$ exceeds a certain threshold. This threshold should not be too low, because we only want to restart when it is really necessary. We found that 0.999 works fine for sample sizes between 50 and 500.

2.9 Simulations with trees and splines

Using a simulated model we compare the (general) CDBoost with boosting. We choose $n = 100$, $d = 10$, $d_{\text{eff}} = 5$ and the covariates as in section 2.7.1. The true function F is

$$F(x) = c_1 x_1 + c_2 x_2 + 9 c_3\, \varphi(x_3) + 1.4 c_4 \sin(2x_4) + 2.7 c_5/(1 + \exp(-3x_5))$$

where φ is the density of the standard normal distribution and $c_1, \ldots, c_5 \sim$ log-Uniform$[-0.35, 0.35]$. The model for the response is $y = F(x) + \epsilon$ with $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$.

		stnr = 9			*stnr* = 4			*stnr* = 1		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	7.38			11.15			26.5		
	$\nu = 0.7$	5.74			9.05			21.5		
	$\nu = 0.5$	5.57			8.73			20.5		
	$\nu = 0.3$	**5.28**			8.27			20.1		
	$\nu = 0.1$	5.36			**8.22**			**19.1**		
	$\nu = 0.03$	5.59			8.60			19.9		
	ν selec.	5.01			7.93			19.0		
Boo	$\nu = 1$	6.21	−15.9	7e−09	9.90	−11.3	5e−04	23.5	−11.5	3e−05
	$\nu = 0.7$	5.90	2.7	4e−01	9.23	2.0	7e−01	21.7	1.0	1e−00
	$\nu = 0.5$	**5.61**	0.6	7e−01	8.74	0.1	6e−01	20.5	−0.1	9e−01
	$\nu = 0.3$	5.62	6.4	2e−02	**8.59**	3.9	2e−02	20.1	0.2	9e−01
	$\nu = 0.1$	5.64	5.3	1e−04	8.68	5.6	4e−06	20.0	4.4	1e−02
	$\nu = 0.03$	5.65	1.2	1e−02	8.71	1.2	2e−04	**20.0**	0.2	7e−02
	ν selec.	5.49	9.5	2e−04	8.48	7.0	2e−04	20.2	6.3	2e−02

Table 2.6: *Comparison of CDBoost (CDB) and boosting (Boo) with componentwise cubic smoothing splines for various signal-to-noise ratios (stnr) and shrinkage factor ν (and ν also selected by the validation set from the candidate values $\nu = 0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03$). Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and stnr is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests. The standard errors of the means divided by the means range from 3.0% to 4.6%.*

Table 2.6 shows the results using componentwise cubic smoothing splines with 3 degrees of freedom as base learner. This means that in each iteration we fit a smoothing spline with 3 degrees of freedom for each predictor individually, and then select the one which reduces the residual sum of squares most. CDBoost and Boosting both perform better with shrinkage. It is especially CDBoost which benefits from the selection of ν by the validation set, and it is then clearly better than boosting.

Table 2.7 shows the results using stumps (trees with two terminal nodes) as base learner. Boosting, once more, needs small ν for best performance and also CDBoost gives better results for smaller ν. Both methods perform almost equally well and the selection of ν by the val-

		stnr = 9			*stnr* = 4			*stnr* = 1		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	17.7			25.5			49.0		
	$\nu = 0.7$	11.6			15.8			30.9		
	$\nu = 0.5$	10.7			14.5			28.0		
	$\nu = 0.3$	**10.2**			14.2			26.5		
	$\nu = 0.1$	10.2			**14.1**			26.3		
	$\nu = 0.03$	10.3			14.1			**26.3**		
	$\nu = 0.01$	10.3			14.1			26.4		
	ν selec.	10.3			14.2			26.9		
Boo	$\nu = 1$	20.0	13.3	5e−09	27.6	8.3	1e−04	53.2	8.6	1e−04
	$\nu = 0.7$	15.8	36.1	0e+00	21.7	36.8	0e+00	39.4	27.5	8e−13
	$\nu = 0.5$	13.4	25.8	2e−16	17.6	21.5	2e−16	31.5	12.7	6e−08
	$\nu = 0.3$	11.3	10.1	2e−11	14.8	4.3	7e−05	27.1	2.2	5e−02
	$\nu = 0.1$	10.5	2.2	2e−02	14.0	−0.6	7e−01	26.4	0.5	6e−01
	$\nu = 0.03$	10.4	0.7	7e−01	**14.0**	−1.1	5e−01	**26.4**	0.4	6e−02
	$\nu = 0.01$	**10.4**	0.4	1e−00	14.0	−0.9	9e−01	26.4	0.0	3e−01
	ν selec.	10.6	2.9	3e−02	14.4	1.4	3e−01	27.7	2.8	2e−01

Table 2.7: *Comparison of conjugate direction boosting (CDB) and boosting (Boo) with stumps for various signal-to-noise ratios (stnr) and shrinkage factor ν (and ν also selected by the validation set from the candidate values $\nu = 0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03,\ 0.01$). Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and stnr is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests. The standard errors of the means divided by the means range from 2.0% to 2.9%.*

idation set does not pay off, because the shrinkage factors 0.7 and 0.5 lead to bad models.

Because the true underlying model is continuous, it is no surprise that both methods are clearly worse with stumps than with splines as base learner.

2.10 Real data examples

Finally, we compare CDBoost and boosting using two real datasets.

2.10.1 Los Angeles ozone data

The Los Angeles ozone dataset (Breiman and Friedman 1985) consists of $n = 330$ complete observations of $d = 8$ explanatory variables. We randomly split it into a training set of size 220 and a test set of 110. We use five-fold cross-validation to choose the number of iterations/steps. The splitting is repeated 100 times and the test errors are averaged.

We only fit main effects models: linear models with all covariates ($d = 8$) and linear models with all covariates and the squared covariates ($d = 16$). In addition, we apply componentwise smoothing splines with 3 degrees of freedom and stumps as learners. The results are contained in table 2.8 (abbreviations of methods as in table 2.3).

There are differences between the learners but almost none between the methods. In contrast to the simulated models, CDBoost does not deteriorate when ν becomes very small and the selection of ν by the validation set does not pay off. This is because we work with only one dataset and therefore there should be a single ν which is adequate for all random splits. In addition, the selection of ν by the validation set would give better results when we exclude the shrinkage factor 0.7 from the candidate values.

2.10.2 Leukemia data

The Leukemia dataset (Golub, Slonim, Tamayo, Huard, Gassenbeek, Mesirov, Coller, Loh, Downing, Caligiuri, Bloomfield, and Lander 1999) is from a microarray experiment with 72 samples and 3571 genes. Although it is a binary classification problem, we treat it as a regression problem with outcome 0 and 1 and L_2-loss (see also Zou and Hastie 2005 in their analysis of microarray data). We use the fitted values to classify the samples with cut-point 1/2 and compute the misclassification rate. Again, we average again over 100 random splits with 50 training cases

	Shrinkage factor ν							
	1	0.7	0.5	0.3	0.1	0.03	0.01	selected
CDB linear	21.1	21.0	**21.0**	21.0	21.0	21.0	21.0	21.0
Boo linear	21.1	21.0	**20.9**	20.9	21.0	21.0	21.0	21.0
FVS linear	21.1							
LAR linear	21.0							
Las linear	21.0							
For linear	21.1							
CDB quadratic	18.9	18.6	**18.5**	18.5	18.5	18.5	18.5	18.5
Boo quadratic	18.9	18.5	**18.4**	18.4	18.5	18.5	18.5	18.5
FVS quadratic	19.0							
LAR quadratic	18.6							
Las quadratic	18.6							
For quadratic	18.5							
CDB splines	18.7	18.6	18.1	18.0	17.9	**17.9**	17.9	18.2
Boo splines	18.3	18.1	17.9	17.9	**17.9**	17.9	17.9	17.9
CDB stumps	22.6	19.9	19.2	18.9	18.9	**18.8**	18.8	19.2
Boo stumps	22.5	21.6	20.5	19.3	18.9	**18.9**	18.9	19.1

Table 2.8: *Comparison of different methods for the ozone dataset. Given is the mean of the test error over 100 random splits. CDBoost and boosting are applied with different shrinkage factors ν (the best values are in bold face). Additionally, the validation set is also used to select the shrinkage factor ν from the candidate values* 0.7, 0.5, 0.3, 0.1, 0.03, 0.01.

and 22 test cases. The number of iterations is estimated with five-fold cross-validation. Table 2.9 shows the results.

In this example, CDBoost performs better than the other methods. The selection of ν by the validation set does again not pay off (the results would again be better when we exclude the shrinkage factors 0.7 and 0.5 from the candidate values).

2.11 Discussion

We propose a conjugate direction boosting method (CDBoost) for linear and more general regression with potentially very many covariates.

	Shrinkage factor ν								
	1	0.7	0.5	0.3	0.1	0.03	0.01	0.003	selected
CDB	8.91	5.05	4.41	4.64	4.32	4.27	**4.14**	4.32	4.55
Boo	9.23	7.82	6.59	5.23	5.45	5.27	**5.18**	5.23	5.73
FVS	8.50								
LAR	5.32								
Las	5.23								
For	5.23								

Table 2.9: *Comparison of different methods for the leukemia dataset. Given is the mean of the test error (misclassification rate in %) over 100 random splits. CDBoost and boosting are applied with different shrinkage factors ν (the best values are in bold face). Additionally, the validation set is also used to select the shrinkage factor ν from the candidate values* 0.7, 0.5, 0.3, 0.1, 0.03, 0.01, 0.003.

For linear regression, our CDBoost is a special version of forward stepwise variable selection (see Proposition 1): in high-dimensional settings with very many covariates, it is computationally faster than traditional forward stepwise variable selection (see Proposition 2).

The concept of conjugate directions allows the inclusion of shrinkage (small step sizes), analogously to boosting. Shrinkage can yield substantial or even dramatic improvements of predictive performance; the same is true for boosting. But the role of weak or no shrinkage has a more pronounced interpretation than in boosting, as discussed in section 2.5.2. We also discuss in section 2.6 that CDBoost with infinitesimally small shrinkage yields solutions which are very similar to boosting, Lasso and other versions of least angle regression (LARS). Thus, varying the step size makes CDBoost surprisingly flexible ranging from the Lasso, LARS (L_1-penalty methods) and boosting to forward stepwise variable selection (some sort of L_0-penalisation). In some cases, e.g. in our simulation model 2 in section 2.7.2, the shrinkage factor ν should not be too small for CDBoost, indicating that we should move a little more towards least squares fitting in sub-models.

Our CDBoost method has the potential to outperform boosting, LARS and the Lasso. It works especially well in complicated settings with correlated covariates where it is not obvious how much a covariate contributes to the response. The gain of using CDBoost is also more often pronounced in sparse cases with high signal to noise ratio and few effective covariates (because CDBoost is flexible enough to cover the whole range from LARS and boosting to forward stepwise variable selection; and for cases with large signal to noise ratios and few effective covariates, a solution closer to forward stepwise variable selection is expected to be good).

CDBoost, boosting and the three LARS methods perform almost equally well for real datasets with only a few covariates. It can, however, be difficult to see clear differences in real data examples because the irreducible error ϵ is also contained in the test error. For the high-dimensional leukemia data though, CDBoost performs better than the other methods.

Finally, the concept of conjugate directions in CDBoost allows for a direct generalization to non-parametric fitting methods. Such an extension to the non-parametric setting is also possible with boosting but not with LARS. One drawback is that CDBoost is often computationally more expensive and less generic than boosting. For example, it is unclear how to modify CDBoost for classification using reweighted least squares since we would need to combine reweighting with conjugacy.

Chapter 3

Boosting for High-Multivariate Responses in High-Dimensional Linear Regression

We propose a boosting method, multivariate L_2Boosting, for multivariate linear regression based on some squared error loss for multivariate data. It can be applied to multivariate linear regression with continuous responses and for vector autoregressive time series. We prove, for i.i.d. data as well as for time series, that multivariate L_2Boosting can consistently recover sparse high-dimensional multivariate linear functions, even when the number of predictor variables $p = p_n$ and the dimension of the response $q = q_n$ grow almost exponentially with sample size n, i.e. $p_n = q_n = O(\exp(Cn^{1-\xi}))$ $(0 < \xi < 1, 0 < C < \infty)$, but the ℓ_1-norm of the true underlying function is finite. Our theory seems to be among the first to address the issue of large dimension of the response variable; the relevance of such settings is briefly outlined. We also identify empirically some cases where our multivariate L_2Boosting is better than

multiple application of univariate methods to single response components, thus demonstrating that the multivariate approach can be very useful.

3.1 Introduction

Boosting, originally proposed as an ensemble scheme for classification, i.e. AdaBoost (Freund and Schapire 1996), has attracted a lot of attention both in the machine learning and statistics literature, mainly because of its success as an excellent prediction method in numerous examples. The pioneering work by Breiman (1998, 1999) demonstrated that the AdaBoost ensemble method can be represented as a gradient descent approximation in function space, see also Friedman et al. (2000). This has opened new possibilities for better understanding and new versions of boosting. In particular, such gradient descent methods can be applied to different loss functions, each yielding another boosting algorithm. L_2Boosting which uses the squared error loss (L_2-loss) has been demonstrated to be a powerful method for univariate regression (Friedman 2001, Bühlmann and Yu 2003, Bühlmann 2006).

We propose here a boosting method with some squared error loss (Gaussian negative log-likelihood) for multivariate data, called multivariate L_2Boosting. We restrict ourselves to linear models (linear basis expansions). They can be very high-dimensional in terms of the response or predictor dimension, and we allow for seemingly unrelated regressions (SUR; Zellner 1962, 1963) where each response may have another design matrix (other predictor variables). The SUR model is more general than the multivariate setting where each covariate has an influence on all response variables. Our multivariate L_2Boosting takes potential correlations among the components of the multivariate error-noise into consideration: that is, we account for the fact that the responses are still exhibiting conditional dependence given all the predictor variables. We prove that our boosting method is able to consistently recover sparse high-dimensional multivariate functions, even when the number of predictor variables $p = p_n$ and the dimension of the response $q = q_n$ grow almost exponentially with sample size n, i.e. $p_n = q_n = O(\exp(Cn^{1-\xi}))$ $(0 < \xi < 1,\ 0 < C < \infty)$. The mathematical arguments are extending the analysis for boosting for high-dimensional univariate regression (Bühlmann 2006). Our theory seems to be among the first for the setting of large dimension of the response (for its practical relevance, see the paragraph after next).

We also demonstrate the use of our multivariate L_2Boosting for multivariate, q_n-dimensional time series $\{\mathbf{x}_{(t)}\}_{t\in\{1,\ldots,n\}}$, where q_n can grow as fast as any polynomial in the sample size n. We prove a consistency result for stationary, linear processes which are representable as a sparse vector autoregressive model of order ∞.

From a theoretical perspective it is interesting how far we can go with dimensionality when the true underlying structure is sparse. From a practical point of view, there are many applications nowadays with large predictor dimension p, notably a broad variety of data mining problems belong to this setting. There are also some applications where q is very large. We mention multi-category classification with a huge number of categories: in Kriegel, Kroger, Pryakhin and Schubert (2004), the categories are subsets of functions from gene ontology (see also Remark 1 in section 3.4). Another application is briefly outlined in section 3.4.1. In the context of time series, some of the graphical modelling for many stochastic processes fall into our setting of q-dimensional linear time series, e.g the partial correlation graph (cf. Dahlhaus and Eichler 2003).

Besides presenting some theory, we also identify empirically some cases where our multivariate L_2Boosting is better than methods based on individual estimation: we compare with individual univariate L_2Boosting and with another L_2Boosting method in a multivariate regression model where every predictor variable either influences all or none of the response components. Some real data sets are analyzed as well.

3.2 Multivariate Linear Regression

We consider the multivariate linear regression model with n observations of a q-dimensional response and a p-dimensional predictor (for more detailed information, see for example Seber (1984) or Timm (2002)). In matrix notation:

$$\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{E}, \tag{3.2.1}$$

with $\mathbf{Y} \in \mathbb{R}^{n\times q}$, $\mathbf{X} \in \mathbb{R}^{n\times p}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$ and $\mathbf{E} \in \mathbb{R}^{n\times q}$. We denote with $\mathbf{y}_{(i)}$ the response of the i-th sample point (row-vector of $\mathbf{Y}$)

and with $\mathbf{y}_k$ the k-th response-variable for all sample points (column-vector of $\mathbf{Y}$). For each $\mathbf{y}_k$ $(k = 1, \ldots, q)$ we have a univariate regression model with the predictor matrix $\mathbf{X}$ and the coefficient vector $\mathbf{b}_k$. For the row-vectors $\mathbf{e}_{(i)}$ $(i = 1, \ldots, n)$ of the error matrix, we assume $\mathbf{e}_{(i)}$ i.i.d., $\mathbb{E}[\mathbf{e}_{(i)}] = \mathbf{0}$ and $\mathrm{Cov}(\mathbf{e}_{(i)}) = \mathbf{\Sigma}$. Additionally, we assume w.l.o.g. that all covariates and responses are centered to have mean zero, so we need not worry about intercepts.

The ordinary least squares estimator (OLS) of $\mathbf{B}$ is given by (assuming $\mathbf{X}$ is of full rank p)

$$\hat{\mathbf{B}}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{3.2.2}$$

and is nothing else than the OLS's of the q univariate regressions. In particular, it is independent of $\mathbf{\Sigma}$.

To test whether a covariate has a significant influence on the multivariate response we can use Wilk's Λ, which is derived from the likelihood ratio test. For an overall test with null-hypothesis $H_0 : \mathbf{B} = \mathbf{0}$ we compare the empirical covariance matrix of the residuals to the one from the responses:

$$\Lambda = \frac{|(\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}}_{OLS})^T(\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}}_{OLS})|}{|\mathbf{Y}^T\mathbf{Y}|},$$

where $|\cdot|$ denotes the determinant of a matrix. We reject the null hypothesis H_0 if Λ is smaller than a critical value.

3.2.1 Forward stepwise variable selection

As for univariate regression, we can define a multivariate forward stepwise variable selection algorithm in a straightforward manner: start with the empty model and add in each step the most significant covariate according to Wilk's Λ. Notice that in each step the entries of a whole row $\mathbf{b}_{(j)}$ of $\mathbf{B}$ are changed from zero to non-zero by using OLS on the reduced space of all included covariates. Therefore, this approach is not suited for the SUR model where a covariate may only have an effect on some but not all components of the response.

3.3 L_2Boosting for multivariate linear regression

For constructing a boosting algorithm, we define a loss function and a base procedure (simple fitting method). The latter is usually called "weak learner" in the machine learning community: it is an estimator which is repeatedly used in boosting.

3.3.1 The loss function

Regarding the loss-function, we use the negative Gaussian log-likelihood as a starting point:

$$-l(\mathbf{B},\mathbf{\Sigma}) = -\log((2\Pi)^{nq/2}|\mathbf{\Sigma}|^{n/2})+\frac{1}{2}\sum_{i=1}^{n}(\mathbf{y}_{(i)}^T-\mathbf{x}_{(i)}^T\mathbf{B})\mathbf{\Sigma}^{-1}(\mathbf{y}_{(i)}^T-\mathbf{x}_{(i)}^T\mathbf{B})^T.$$

As before, $|\cdot|$ denotes the determinant of a matrix. The maximum likelihood estimator of $\mathbf{B}$ coincides with the OLS solution in (3.2.2) and is therefore independent of $\mathbf{\Sigma}$. The covariance matrix $\mathbf{\Sigma}$ becomes only relevant in the seemingly unrelated regressions (SUR) model when there are covariates which influence only a few components of the response.

Because $\mathbf{\Sigma}$ is usually unknown, we use the following loss function

$$L(\mathbf{B}) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{y}_{(i)}^T - \mathbf{x}_{(i)}^T\mathbf{B})\mathbf{\Gamma}^{-1}(\mathbf{y}_{(i)}^T - \mathbf{x}_{(i)}^T\mathbf{B})^T, \tag{3.3.1}$$

where $\mathbf{\Gamma}$ is the implementing covariance matrix. We may use for it an estimate of $\mathbf{\Sigma}$ (e.g. from another model-fit such as univariate boosting for each response separately) or we can choose something simpler, e.g. $\mathbf{\Gamma} = \mathbf{I}$ (in particular if q is large, see also Remark 2). The choice for $\mathbf{\Gamma}$ will show up again in our Theorem 1 in section 3.4 (and Theorem 2 in section 3.5): there it becomes clear that also $\mathbf{\Gamma} = \mathbf{I}$ can be a very reasonable choice.

3.3.2 The componentwise linear least squares base procedure

Now we specify the base procedure which will be repeatedly used in boosting. Given is the design matrix $\mathbf{X}$ and a pseudo-response matrix $\mathbf{R} \in \mathbb{R}^{n\times q}$ (which is not necessarily equal to $\mathbf{Y}$).

We focus here exclusively on what we call the componentwise linear least squares base learner. It fits the linear least squares regression with one selected covariate (column of $\mathbf{X}$) and one selected pseudo-response (column of $\mathbf{R}$) so that the loss function in (3.3.1), with $\mathbf{R}$ instead of $\mathbf{Y}$, is reduced most. Thus, the base procedure fits one selected matrix element of $\mathbf{B}$:

$$\begin{aligned}
(\hat{s}\hat{t}) &= \underset{1\le j\le p, 1\le k\le q}{\arg\min} \; \{L(\mathbf{B}); B_{jk} = \hat{\beta}_{jk}, B_{uv} = 0 \; (uv \neq jk)\} \\
&= \underset{1\le j\le p, 1\le k\le q}{\arg\max} \; \frac{\left(\sum_{v=1}^{q} \mathbf{r}_v^T \mathbf{x}_j \Gamma_{vk}^{-1}\right)^2}{\mathbf{x}_j^T \mathbf{x}_j \Gamma_{kk}^{-1}}, \\
\hat{\beta}_{jk} &= \frac{\sum_{v=1}^{q} \mathbf{r}_v^T \mathbf{x}_j \Gamma_{vk}^{-1}}{\mathbf{x}_j^T \mathbf{x}_j \Gamma_{kk}^{-1}}, \\
\hat{B}_{\hat{s}\hat{t}} &= \hat{\beta}_{\hat{s}\hat{t}}, \;\; \hat{B}_{jk} = 0, \;\; (jk) \neq (\hat{s}\hat{t}). \qquad (3.3.2)
\end{aligned}$$

Corresponding to the parameter estimate, there is a function estimate $\hat{\mathbf{g}}(\cdot) : \mathbb{R}^p \to \mathbb{R}^q$ defined as follows: for $\mathbf{x} = (x_1, \ldots, x_p)$,

$$(\hat{\mathbf{g}})_\ell(\mathbf{x}) = \begin{cases} \hat{\beta}_{\hat{s}\hat{t}} x_{\hat{s}} & \text{if } \ell = \hat{t}, \\ 0 & \text{if } \ell \neq \hat{t}, \end{cases} \quad \ell = 1, \ldots, q.$$

From (3.3.2) we see that the coefficient $\hat{\beta}_{jk}$ is not only influenced by the k-th response but also by other response-components, depending on the partial correlations of the errors (via $\mathbf{\Gamma}^{-1}$ if $\mathbf{\Gamma}$ is a reasonable estimate of $\mathbf{\Sigma}$) and by the correlations of the other response-components with the j-th covariate (i.e. $\mathbf{r}_v^T \mathbf{x}_j$).

3.3.3 The boosting algorithm

The base learner is fitted many times to different pseudo-responses $\mathbf{R}$ and the function estimates are added up as described by the algorithm below. We build the multivariate regression function $\hat{\mathbf{f}} : \mathbb{R}^p \to \mathbb{R}^q$ step by step, where $\hat{\mathbf{f}}(\mathbf{x}) = \hat{\mathbf{B}}^T\mathbf{x}$.

Multivariate L_2Boosting with componentwise linear least squares

Step 1 (initialization): $\hat{f}_k^{(0)}(.) \equiv 0,\ k = 1, \dots, q$. Set $m = 1$.

Step 2: Compute the current residuals $\mathbf{r}_{(i)}^{(m)} = \mathbf{y}_{(i)} - \hat{\mathbf{f}}^{(m-1)}(\mathbf{x}_{(i)})$ ($i = 1, \dots, n$) and fit the base learner to them as in (3.3.2). The fit is denoted by $\hat{\mathbf{g}}^{(m)}(.)$.

Update $\hat{\mathbf{f}}^{(m)}(\cdot) = \hat{\mathbf{f}}^{(m-1)}(\cdot) + \hat{\mathbf{g}}^{(m)}(\cdot)$.

Step 3 (iteration): Increase the iteration index m by one and go back to Step 2 until a stopping iteration m_{stop} is met.

Multivariate L_2Boosting is thus iteratively fitting of residuals where in each step we change only one entry of $\mathbf{B}$. Also, every iteration m corresponds to an estimate $\hat{\mathbf{B}}^{(m)}$ with $\hat{\mathbf{f}}^{(m)}(\mathbf{x}) = (\hat{\mathbf{B}}^{(m)})^T\mathbf{x}$. The estimate $\hat{\mathbf{f}}^{(m_{stop})}(.)$ is an estimator of the multivariate regression function $\mathbb{E}[\mathbf{y}|\mathbf{x} = \cdot]$.

It is often better to use some shrinkage in *Step 2*: this has been first recognized by Friedman (2001), and there are also some supporting theoretical arguments for it (Efron et al. 2004, Bühlmann and Yu 2005). We modify *Step 2* to:

$$\hat{\mathbf{f}}^{(m)}(\cdot) = \hat{\mathbf{f}}^{(m-1)}(\cdot) + \nu \cdot \hat{\mathbf{g}}^{(m)}(\cdot),$$

with $\nu < 1$, for example $\nu = 0.1$. We then need more iterations but often achieve better out-of-sample predictions. The boosting algorithm does depend on ν, but its choice is surprisingly insensitive as long as it is taken to be "small". On the other hand, the number of boosting iterations m_{stop} is a much more crucial tuning parameter.

The computational complexity of the multivariate L_2Boosting algorithm for m iterations is $O(npqm)$ if $\mathbf{\Gamma}$ is diagonal and $O(npq^2m)$ for arbitrary $\mathbf{\Gamma}$.

3.3.4 Stopping the boosting iterations with the corrected AIC

The number of iterations m_{stop} can be estimated by cross validation, a separate validation set or by an internal AIC criterion. We pursue the latter because of its computational attractiveness.

First we recall the definition of the AIC for the multivariate linear regression model. For $d \leq p$ covariates in a sub-model M_d

$$AIC(M_d) = \log(|\hat{\mathbf{\Sigma}}(M_d)|) + \frac{2qd}{n},$$

where $\hat{\mathbf{\Sigma}}(M_d)$ is the MLE of the error covariance-matrix. Note that we have a total of $q \cdot d$ parameters. In small samples, the corrected AIC (Hurvich and Tsai 1989 and Bedrick and Tsai 1994) is often a better model selection tool:

$$AIC_c(M_d) = \log(|\hat{\mathbf{\Sigma}}(M_d)|) + \frac{q(n+d)}{n-d-q-1}.$$

To apply the AIC or the AIC_c for boosting we have to determine the number of parameters or degrees of freedom of boosting as a function of the number of iterations. Clearly, the degrees of freedom of boosting increase as the number of iterations grow, but this increase is heavily sub-linear (Bühlmann and Yu 2003).

We first consider the hat-operator of the base learner in (3.3.2), mapping $\mathbf{Y}$ to $\hat{\mathbf{Y}} = \mathbf{X}\hat{\mathbf{B}}$. After having selected the j-th predictor and k-th component of the response, the fitting is a linear operation which can be represented by a hat-matrix. In the multivariate case, we stack the q responses $\mathbf{y}_1, \ldots, \mathbf{y}_q$ end-to-end in a vector of length nq (written as $vec(\mathbf{Y})$). The hat-matrix is then of dimension $nq \times nq$ and, with the j-th predictor and the k-th response selected in the base learner, it is of the form

$$\mathbf{H}^{(jk)} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{H}^j \frac{\Gamma^{-1}_{k1}}{\Gamma^{-1}_{kk}} & \mathbf{H}^j \frac{\Gamma^{-1}_{k2}}{\Gamma^{-1}_{kk}} & \dots & \mathbf{H}^j \frac{\Gamma^{-1}_{kq}}{\Gamma^{-1}_{kk}} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \end{pmatrix} \begin{array}{l} \\ \\ \\ \longleftarrow k\text{-th row,} \\ \\ \\ \\ \end{array}$$

where each entry is a $n \times n$ matrix and the non-zero matrix-entries are at row k and $\mathbf{H}^j = \mathbf{x}_j \mathbf{x}_j^T / \mathbf{x}_j^T \mathbf{x}_j$ is the hat-matrix of the univariate componentwise linear learner using the j-th predictor variable.

Due to the nature of iterative fitting of residuals, the hat-matrix of multivariate L_2Boosting after m iterations is then (c.f. Bühlmann and Yu 2003, Bühlmann 2006)

$$\mathbf{K}_m = \mathbf{I} - (\mathbf{I} - \nu \mathbf{H}^{(\hat{s}_m \hat{t}_m)})(\mathbf{I} - \nu \mathbf{H}^{(\hat{s}_{m-1} \hat{t}_{m-1})}) \dots (\mathbf{I} - \nu \mathbf{H}^{(\hat{s}_1 \hat{t}_1)}).$$

Here, $(\hat{s}_m \hat{t}_m)$ denote the selected covariate and response-component from the base learner in (3.3.2) in boosting iteration m. The computation of the hat-matrix has a complexity of $O(n^2 p + n^3 q^2 m)$ and is not feasible if n (and/or q) is large.

The trace of $\mathbf{K}_m$ gives the number of degrees of freedom. For the AIC_c we need the degrees of freedom (number of equivalent parameters) per response variable: thus, we divide the total number of degrees of freedom by q to get the average number of degrees of freedom per response. The AIC and the AIC_c for multivariate L_2Boosting as functions of the number of iterations m then become:

$$AIC(m) = \log(|\hat{\mathbf{\Sigma}}(m)|) + \frac{2 \cdot \text{trace}(\mathbf{K}_m)}{n},$$

$$AIC_c(m) = \log(|\hat{\mathbf{\Sigma}}(m)|) + \frac{q(n + \text{trace}(\mathbf{K}_m)/q)}{n - \text{trace}(\mathbf{K}_m)/q - q - 1},$$

where $\hat{\mathbf{\Sigma}}(m) = n^{-1} \sum_{i=1}^{n} (\mathbf{y}_{(i)} - \hat{\mathbf{f}}^{(m)}(\mathbf{x}_{(i)}))(\mathbf{y}_{(i)} - \hat{\mathbf{f}}^{(m)}(\mathbf{x}_{(i)}))^T$. The number of boosting iterations is chosen to minimize the AIC or AIC_c, respectively:

$$\hat{m}_{stop} = \underset{0 \le m < M}{\arg\min}\, AIC_c(m),$$

where M is a pre-specified large, upper bound for the candidate number of boosting iterations.

3.3.5 L_2Boosting with whole rows of B

Multivariate L_2Boosting changes in each step only one entry of $\mathbf{B}$. This might be suboptimal if we believe that a covariate has either some influence on all response-components or no influence at all. It may then be better to update in each step a whole row of $\mathbf{B}$. This can also be done with a L_2Boosting type algorithm, which we call "row-boosting": we select in each step the covariate which gives the best multivariate fit to the current residuals (according to Wilk's Λ) and add it to the multivariate function estimate. This algorithm is more closely related to multivariate forward variable selection, see section 3.2.1, with the difference that we don't adjust the coefficients of the covariates already included in the model.

3.4 Consistency of multivariate L_2Boosting

We present here a consistency result for multivariate L_2Boosting in linear regression where the number of predictors and the dimension of the response are allowed to grow very fast as sample size n increases. Consider the model

$$\begin{aligned}
&\mathbf{y}_{(i)} = \mathbf{f}(\mathbf{x}_{(i)}) + \mathbf{e}_{(i)},\ i = 1,\ldots,n,\ \ \mathbf{y}_{(i)}, \mathbf{e}_{(i)} \in \mathbb{R}^{q_n},\ \ \mathbf{x}_{(i)} \in \mathbb{R}^{p_n},\\
&\mathbf{f}(\mathbf{x}) = \mathbf{B}^T\mathbf{x},\ \ \mathbf{B} \in \mathbb{R}^{p_n \times q_n}, \qquad (3.4.1)\\
&\mathbf{x}_{(i)} \text{ i.i.d. and } \mathbf{e}_{(i)} \text{ i.i.d., independent of } \{\mathbf{x}_{(i)}; 1 \le i \le n\}\\
&\text{with } \mathbb{E}[\mathbf{e}_{(i)}] = \mathbf{0} \text{ and } \mathrm{Cov}(\mathbf{e}_{(i)}) = \mathbf{\Sigma}.
\end{aligned}$$

Because p_n and q_n are allowed to grow with n, also the predictors and the responses depend on n. We ignore this notationally most of the time. To identify the magnitude of B_{jk} we assume $\mathbb{E}|x_{(1)j}|^2 = 1,\ j = 1,\ldots,p_n$.

We make the following assumptions:

(A1) The dimension of the predictor and the response in model (3.4.1) satisfies $p_n = O(\exp(Cn^{1-\xi}))$, $q_n = O(\exp(Cn^{1-\xi}))$ $(n \to \infty)$, for some $0 < \xi < 1, 0 < C < \infty$.

(A2) $\sup_{n\in\mathbb{N}} \sum_{j=1}^{p_n} \sum_{k=1}^{q} |B_{jk,n}| < \infty$.

(A3) For the implementing $\mathbf{\Gamma}$ in 3.3.1:
$\sup_{n\in\mathbb{N},1\le k\le q_n} \sum_{\ell=1}^{q_n} |\Gamma^{-1}_{k\ell,n}| < \infty, \quad \inf_{n\in\mathbb{N},1\le k\le q_n} \Gamma^{-1}_{kk,n} > 0$.

(A4) $\sup_{1\le j\le p_n} \|x_{(1)j}\|_\infty < \infty$, where $\|x\|_\infty = \sup_{\omega\in\Omega} |x(\omega)|$ (Ω denotes the underlying probability space).

(A5) $\sup_{1\le k\le q_n} \mathbb{E}|e_{(1)k}|^s < \infty$ for some $s > 2/\xi$ with ξ from (A1).

Assumption (A1) allows for very large predictor and response dimensions relative to the sample size n. Assumption (A2) is a l_1-norm sparseness condition for the underlying multivariate regression function $\mathbf{f}(\cdot)$. If q_n grows with sample size, it seems quite restrictive. But we describe a potential application in section 3.4.1 (second example), where (A2) could be reasonable even if q_n grows. Assumption (A3) is a sparseness condition on $\mathbf{\Gamma}^{-1}$ which holds when choosing $\mathbf{\Gamma} = \mathbf{I}$ (or other reasonable diagonal matrices). Assumption (A4) and (A5) are the same as in Bühlmann (2006); (A4) can be relaxed at the price of a polynomial growth $O(n^\delta)$ $(0 < \delta < \infty)$ in (A1) and assuming sufficiently high-order moments, cf. section 3.5.

Theorem 1 *Consider the model (3.4.1) satisfying (A1)-(A5). Then, the multivariate L_2Boosting estimate $\hat{\mathbf{f}}^{(m_n)}$ with the componentwise linear learner from (3.3.2) satisfies: for some sequence $(m_n)_{n\in\mathbb{N}}$ with $m_n \to \infty$ $(n \to \infty)$ sufficiently slowly,*

$$\mathbb{E}_{\mathbf{x}}\left[\left(\hat{\mathbf{f}}^{(m_n)}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\right)^T \mathbf{\Gamma}^{-1} \left(\hat{\mathbf{f}}^{(m_n)}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\right)\right] = o_p(1) \ (n \to \infty),$$

where $\mathbf{x}$ denotes a new observation, independent of and with the same distribution as the $\mathbf{x}_{(i)}$, $i = 1, \dots, n$.

A proof is given in section 3.9. Theorem 1 says that multivariate L_2Boosting recovers the true sparse regression function even if the dimensions of the predictor and response grow almost exponentially with sample size n.

Remark 1 *We can also use the multivariate L_2Boosting for multi-category classification with q categories labelled by $1, \dots, q$. This can be encoded with a multivariate q-dimensional response $\mathbf{y} = (y_1, \dots, y_q)$, where*

$$y_j = \begin{cases} 1 & \textit{if the category-label } = j, \\ 0 & \textit{if the category-label } \neq j. \end{cases}$$

Assuming that the data $(\mathbf{x}_{(1)}, \mathbf{y}_{(1)}), \dots, (\mathbf{x}_{(n)}, \mathbf{y}_{(n)})$ are independent and identically distributed, the conditional probabilities $\pi_j(\mathbf{x}) = \mathbb{P}[y_j = 1|\mathbf{x}]$ are linear in $\mathbf{x}$ and if (A1)-(A4) hold, then multivariate L_2Boosting is consistent: e.g. with $\mathbf{\Gamma} = \mathbf{I}$, $\sum_{j=1}^{q} \mathbb{E}_{\mathbf{x}}[(\hat{\pi}_j^{(m_n)}(\mathbf{x}) - \pi_j(\mathbf{x}))^2] = o_P(1)$.

The proof of Remark 1 is a consequence of Theorem 1. Note that for binary classification, we typically encode the problem by a univariate response. Multi-category problems could also be represented with a $q-1$-dimensional response. But this would require to tag a particular label as the complement of all others; we typically want to avoid such arbitrariness.

3.4.1 Two potential applications with large response dimension q

One problem is classification (see Remark 1) of biological objects such as genes or proteins into *subsets* of various functional categories, e.g. in Gene Ontology (GO) (cf. Kriegel et al. 2004). Because many biological objects belong to many functional categories, the labels for classification are subsets of functional categories, resulting in a large value of q (and p is large here as well).

Another application occurs when screening for associations of q candidate random variables $\mathbf{y}_1, \dots, \mathbf{y}_q$ with a system of p target variables $\mathbf{x}_1, \dots, \mathbf{x}_p$. This occurs in Wille et al. (2004) when screening expressions of $q = 795$ genes which exhibit some potential associations to the expressions of $p = 39$ genes from two biosynthesis pathways in Arabidopsis thaliana. We would like to know whether the partial correlation $\text{Parcor}(\mathbf{y}_k, \mathbf{x}_j | \{\mathbf{x}_u; u \in \{1, \dots, p\} \setminus j)$ is zero or not, for all $1 \leq k \leq q,\ 1 \leq j \leq p$. This is equivalent to check in linear regressions

$$\mathbf{y}_k = B_{jk}\mathbf{x}_j + \sum_{1 \leq u \leq p, u \neq j} B_{uk}\mathbf{x}_u + \mathbf{e}_k,$$

whether $B_{jk} = 0$ or not. We could imagine that only a few of the q candidate variables $\mathbf{y}_1, \dots, \mathbf{y}_q$ have something to do with the p target variables $\mathbf{x}_1, \dots, \mathbf{x}_p$ (i.e. there are many k's where $B_{jk} \equiv 0$ for all j) and that existing relations between the candidate and target variables are sparse in terms of the corresponding regression coefficients, i.e. (A2) could be a reasonable assumption.

Remark 2 *Using an estimate of $\mathbf{\Sigma}$ for $\mathbf{\Gamma}$ may result in a poor fit when q is large relative to n. In this case we may choose something simpler, e.g. diag($\hat{\mathbf{\Sigma}}$) or $\mathbf{I}$ (only reasonable when the responses are standardized) or a convex-combination $\gamma\hat{\mathbf{\Sigma}} + (1-\gamma)diag(\hat{\mathbf{\Sigma}})$ with $0 < \gamma < 1$ small. If $\mathbf{\Gamma}$ is diagonal, multivariate L_2Boosting fits q independent univariate linear regressions. For each response it produces the same sequence of selected covariates as univariate L_2Boosting. The only difference of the multivariate method is that it mixes the individual sequences of selected covariates for the different responses and uses only one stopping iteration. From a theoretical point of view, the multivariate method (even for $\mathbf{\Gamma} = \mathbf{I}$) allows to derive consistency for growing q.*

3.5 Multivariate L_2Boosting for vector AR processes

Obviously, the boosting method from section 3.3 can be used for vector autoregressive processes (VAR, see for example Reinsel (1993) or Lütkepohl (1993))

$$\mathbf{x}_{(t)} = \sum_{j=1}^{p} \mathbf{A_j}\mathbf{x}_{(t-j)} + \mathbf{e}_{(t)}, \quad t \in \mathbb{Z}, \tag{3.5.1}$$

where $\mathbf{x}_{(t)} \in \mathbb{R}^q$ is the q-dimensional observation at time t, $\mathbf{A_j} \in \mathbb{R}^{q\times q}$ and $\mathbf{e}_{(t)} \in \mathbb{R}^q$ i.i.d. with $\mathbb{E}[\mathbf{e}_{(t)}] = \mathbf{0}$ and $\mathrm{Cov}(\mathbf{e}_{(t)}) = \mathbf{\Sigma}$. The model is stationary and causal if all roots of $\det(\mathbf{I} - \sum_{j=1}^{p} \mathbf{A_j} z^j)$ ($z \in \mathbb{C}$) are greater than one in absolute value.

For observations $\mathbf{x}_{(t)}$ ($t = 1, \dots, n$), the equation in (3.5.1) can be written as a multivariate regression model as in (3.2.1) with $\mathbf{Y} = [\mathbf{x}_{(p+1)}, \dots, \mathbf{x}_{(n)}]^T \in \mathbb{R}^{(n-p)\times q}$, $\mathbf{B} = [\mathbf{A_1}, \dots, \mathbf{A_p}]^T \in \mathbb{R}^{qp\times q}$ and $\mathbf{X} \in \mathbb{R}^{(n-p)\times qp}$ the corresponding design matrix.

The consistency result from Theorem 1 carries over to the time series case. We assume that the data is generated from the following $q = q_n$-dimensional VAR(∞) model:

$$\mathbf{x}_{(t)} = \sum_{j=1}^{\infty} \mathbf{A_j} \mathbf{x}_{(t-j)} + \mathbf{e}_{(t)}, \quad t \in \mathbb{Z}, \tag{3.5.2}$$

with $\mathbf{e}_{(t)} \in \mathbb{R}^{q_n}$ i.i.d. with $\mathbb{E}[\mathbf{e}_{(t)}] = \mathbf{0}$, $\mathrm{Cov}(\mathbf{e}_{(t)}) = \mathbf{\Sigma}$ and $\mathbf{e}_{(t)}$ independent of $\{\mathbf{x}_{(s)};\ s < t\}$. Again, we ignore notationally that the model and its terms depend on n due to the growing dimension q_n. Assume that:

(B1) $\{\mathbf{x}_{(t)}\}_{t \in \mathbb{Z}}$ in (3.5.2) is strictly stationary and α-mixing with mixing coefficients $\alpha_n(\cdot)$.

(B2) The dimension satisfies: $q = q_n = O(n^{\delta})$ for some $0 < \delta < \infty$.

(B3) $\sup_{n \in \mathbb{N}} \sum_{j=1}^{\infty} \sum_{k,v=1}^{q_n} |A_{kv;j,n}| < \infty, \quad A_{kv;j,n} = (\mathbf{A}_{\mathbf{j},n})_{kv}$.

(B4) The mixing coefficients and moments satisfy: for some $s \in \mathbb{N}$ with $s > 2(1+\delta) - 2$ (δ as in (B2)) and $\gamma > 0$

$$\sum_{k=1}^{\infty} (k+1)^{s-1} \alpha_n(k)^{\gamma/(2s+\gamma)} < \infty,$$

$$\sup_{1 \le k \le q_n, n \in \mathbb{N}} \mathbb{E}|x_{(t)k}|^{4s+2\gamma} < \infty, \qquad \sup_{1 \le k \le q_n, n \in \mathbb{N}} \mathbb{E}|e_{(t)k}|^{2s+\gamma} < \infty.$$

Theorem 2 *Assume the model (3.5.2), satisfying the assumptions (B1)-(B4) and require that (A3) holds. Consider multivariate L_2Boosting with componentwise linear least squares (as in section 3.3) using $p = p_n$ lagged variables (as in model (3.5.1)) with $p_n \to \infty$, $p_n = O(n^{1-\kappa})$ $(n \to \infty)$, where $2(1+\delta)/(s+2) < \kappa < 1$. Then, the assertion from Theorem 1 holds with $\mathbf{f}(\mathbf{x}) = \sum_{j=1}^{\infty} \mathbf{A_j} \mathbf{x}_{(t-j)}$, $\hat{\mathbf{f}}^{(m_n)}(\mathbf{x}) = \sum_{j=1}^{p_n} \hat{\mathbf{A}}_{\mathbf{j}}^{(m_n)} \mathbf{x}_{(t-j)}$ and $\mathbf{x}$ a new realization from (3.5.2), independent from the training data.*

A proof is given in section 3.9. Note that if in (B4) the mixing coefficients decay exponentially and all moments exist, i.e. for a suitably regular Gaussian VAR(p) of finite order, Theorem 2 holds for arbitrarily large δ in (B2) and arbitrarily small $\kappa > 0$, implying $p_n = O(n^{1-\kappa})$ is allowed to grow almost as fast as n.

3.6 Simulation study

In this section we compare multivariate L_2Boosting (MB) to individual L_2Boosting (IB, univariate L_2Boosting for each response alone; cf. Bühlmann (2006)), row-boosting (RB, see section 3.3.5) and multivariate forward stepwise variable selection (MFS, see section 3.2.1) on simulated data sets.

3.6.1 Design

The sample size is always $n = 50$ and the number of responses is $q = 5$. We take two numbers of covariates ($p = 10$ and $p = 30$) and two proportions of non-zero entries of $\mathbf{B}$ ($p_{eff} = 0.2$ and $p_{eff} = 0.5$, where $p_{eff} = 0.2$ means that 20% of the entries of $\mathbf{B}$ are non-zero).

The covariates are generated according to a multivariate normal distribution with covariance matrix $\mathbf{V}$,

$$\mathbf{x}_{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{V}), \text{ with } V_{kv} = 0.9^{|k-v|}.$$

The value 0.9 seems to be pretty high, but when having $p = 30$ covariates, the average correlation between the covariates is 0.42 only. Smaller values lead to similar results among the boosting methods, only MFS performs then a bit better.

For the true coefficient-matrix $\mathbf{B}$ we take two different types, characterized by the non-zero entries: for the first type, we arbitrarily choose the $q \cdot p \cdot p_{eff}$ non-zero entries of $\mathbf{B}$ with the only constraint that each response must depend on at least one covariate. We will call this type "**B** arbitrary" (this is the case of seemingly unrelated regressions). For the other type, we randomly choose $p \cdot p_{eff}$ rows of $\mathbf{B}$ and set the entries of the whole rows unequal to zero ("**B** row-complete"). The non-zero entries of $\mathbf{B}$ are for both types i.i.d. $\sim \mathcal{N}(0, 1)$.

The errors are again generated according to a multivariate normal distribution with covariance-matrix $\mathbf{\Sigma}$,

$$\mathbf{e}_{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}).$$

The diagonal elements of $\mathbf{\Sigma}$ are constructed to give individual signal-to-noise ratios of 0.71, 0.84, 1.00, 1.19, 1.41. The off-diagonal elements of $\mathbf{\Sigma}$ are chosen to give the following correlations between the errors:

$$\mathrm{Cor}(\mathbf{e}_k, \mathbf{e}_v) = \rho^{|k-v|},$$

with ρ taking the values 0, 0.6, and 0.9.

All responses are standardized to unit variance to make them comparable.

The design of this simulation comprises two types of $\mathbf{B}$-matrices, three values for the correlations between the errors, two values for the number of predictors and two values for the number of effective predictors. A complete factorial design over all these levels gives rise to 24 settings. Each setting is replicated 100 times and the different methods are applied.

To select the number of boosting iterations or the number of steps in MFS we use either a validation set of size 50 or the AIC_c. For all boosting methods we choose the shrinkage factor $\nu = 0.1$.

For the implementing covariance-matrix $\mathbf{\Gamma}$ in MB we use the empirical covariance-matrix of the residuals $\mathbf{r}_{(i)}^{IB}$ of the IB:

$$\mathbf{\Gamma} = \hat{\mathbf{\Sigma}} = n^{-1} \sum_{i=1}^{n} \mathbf{r}_{(i)}^{IB} (\mathbf{r}_{(i)}^{IB})^T.$$

3.6.2 Performance measure

In simulations we can measure how close the prediction for an additional observation comes to the true value. For the k-th response, the mean squared prediction error is given by

$$MSPE_k = \int \left(\mathbf{x}^T (\mathbf{b}_k - \hat{\mathbf{b}}_k) \right)^2 dP(\mathbf{x}) = (\mathbf{b}_k - \hat{\mathbf{b}}_k)^T \mathbf{V} (\mathbf{b}_k - \hat{\mathbf{b}}_k).$$

Our performance measure is the mean of the individual $MSPE$'s

$$MSPE = q^{-1} \sum_{k=1}^{q} MSPE_k.$$

This is a reasonable measure, because we have standardized the responses.

3.6.3 Results

The results are summarized in table 3.1 and figure 3.1. We give the mean of the $MSPE$ of the 100 replicates (multiplied by 1000) for each method and setting. Additionally, paired sample Wilcoxon tests are performed which compare for each setting the best method to the other three methods. A p-value below $1e-9$ is set to zero. The iterations are stopped with a validation set.

For $\rho = 0$, multivariate L_2Boosting is a few percent worse than individual L_2Boosting. But for $\rho = 0.6$ and $\rho = 0.9$ MB performs significantly better than IB and the gain can be up to a factor of 1.5 (for less correlated predictors the gain is even bigger). Thus, MB is able to exploit the additional information of the multivariate response.

As expected, MB and IB perform well when **B** is arbitrary and RB performs well when **B** is row-complete. MFS gives only good results in the easier settings, especially with **B** row-complete, $p = 10$ and $p_{eff} = 0.2$. It is interesting to see that MB performs best in the case when **B** is row-complete, $\rho = 0.9$ and $p_{eff} = 0.5$ even tough the setting favors methods which work with whole rows of **B**.

The given results come about with stopping by a validation set. Stopping methods which only use the training data (like the AIC_c) lead on average to worse results because they use much less information. Therefore we can use the validation set stopping as a benchmark to assess the performance of the AIC_c stopping: MB is 6.3% worse (median over all 24 settings) when we use the AIC_c instead of the validation set. RB is 3.5% worse, MFS 10.2% worse and IB 25.0% worse. The AIC_c stopping works relatively better for the multivariate methods (MB, RB and also MFS) than for IB. A possible explanation is that MB and RB have to be stopped only once and not q times. This gives less variability in the final boosting estimate and makes it easier to stop at a good point. Note that for IB, it is desirable or even essential to allow for individual stopping iterations because we often need varying complexities for modelling the different response variables.

				MSPE				Wilcoxon p-value			
B	ρ	p	p_{eff}	MFS	RB	IB	MB	MFS	RB	IB	MB
arbitr.	0.0	10	0.2	84	63	**50**	51	0	0		1e−1
arbitr.	0.0	10	0.5	96	71	**66**	67	0	8e−5		9e−2
arbitr.	0.0	30	0.2	176	125	**112**	116	0	1e−9		5e−3
arbitr.	0.0	30	0.5	216	132	**130**	135	0	1e−1		1e−3
arbitr.	0.6	10	0.2	73	60	50	**44**	0	0	2e−6	
arbitr.	0.6	10	0.5	93	71	67	**62**	0	8e−8	3e−4	
arbitr.	0.6	30	0.2	164	116	109	**100**	0	0	4e−6	
arbitr.	0.6	30	0.5	203	126	127	**117**	0	6e−6	5e−7	
arbitr.	0.9	10	0.2	62	53	49	**33**	0	0	0	
arbitr.	0.9	10	0.5	93	71	68	**51**	0	0	0	
arbitr.	0.9	30	0.2	149	107	110	**72**	0	0	0	
arbitr.	0.9	30	0.5	183	115	126	**85**	0	0	0	
row-c.	0.0	10	0.2	**26**	41	48	50		0	0	0
row-c.	0.0	10	0.5	70	**66**	67	71	6e−2		2e−1	4e−6
row-c.	0.0	30	0.2	123	**105**	118	121	1e−4		1e−9	0
row-c.	0.0	30	0.5	203	**132**	136	139	0		7e−2	1e−5
row-c.	0.6	10	0.2	**25**	38	49	50		1e−9	0	0
row-c.	0.6	10	0.5	64	**60**	64	63	2e−2		6e−4	7e−2
row-c.	0.6	30	0.2	109	**101**	120	110	3e−2		0	3e−6
row-c.	0.6	30	0.5	186	**128**	137	129	0		1e−5	8e−1
row-c.	0.9	10	0.2	**31**	33	50	45		6e−2	6e−9	2e−5
row-c.	0.9	10	0.5	62	54	63	**48**	5e−5	2e−1	1e−9	
row-c.	0.9	30	0.2	**88**	88	120	89		7e−1	0	5e−1
row-c.	0.9	30	0.5	179	120	137	**102**	0	4e−7	0	

Table 3.1: *Mean squared prediction error MSPE, multiplied by 1000, of multivariate forward stepwise variable selection (MFS), row-boosting (RB), individual L_2Boosting (IB) and multivariate L_2Boosting (MB) averaged over the 100 replicates. The best method for each setting is in bold face. P-values of the paired sample Wilcoxon tests, which compare for each setting the best method to the other three methods, are also given.*

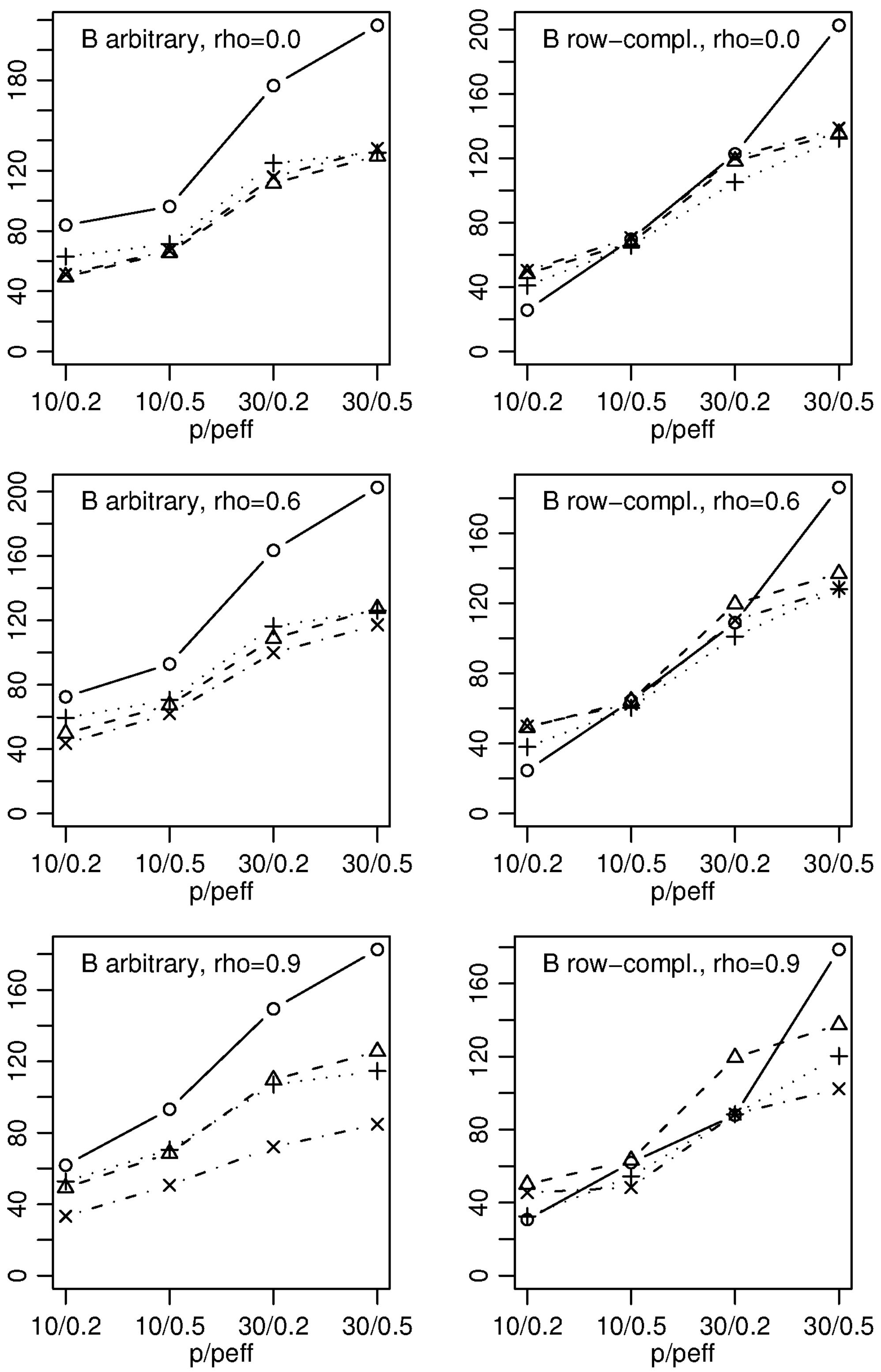

Figure 3.1: *Mean squared prediction error MSPE, multiplied by 1000, of multivariate forward stepwise variable selection (○), row-boosting (+), individual L_2Boosting (△) and multivariate L_2Boosting (×).*

3.7 Real data

We have analyzed the following data sets:

Chemical reaction data (Box and Youle 1955; Rencher 2002): This is a planned experiment involving a chemical reaction with 3 input (predictor) variables (temperature, concentration, time) and 3 output (response) variables (percentage of unchanged starting material, percentage converted to the desired product, percentage of unwanted by-product). We fit a quadratic model including the first order interactions (product of the predictor variables). This gives a total of 9 covariates.

Macroeconomic data (Klein, Ball, Hazlewood and Vandome 1961; Reinsel and Velu 1998): This is a 10 dimensional time series from the United Kingdom from 1948 - 1956 with quarterly measurements. 5 terms are taken as predictor variables (total labor force, weekly wage rates, price index of imports, price index of exports, price index of consumption) and 5 terms are taken as response variables (industrial production, consumption, unemployment, total imports, total exports). We ignore the time-dependency of the observations and fit again a quadratic model with first order interactions.

Chemometrics data (Skagerberg, MacGregor and Kiparissides 1992; Breiman and Friedman 1997): This is a simulation of a low density tubular polyethylene reactor. There are 22 predictor variables (20 reactor temperatures, wall temperature of the reactor, feed rate of the reactor) and 6 responses (number-average molecular weight, weight-average molecular weight, frequency of long chain branching, frequency of short chain branching, content of vinyl groups, content of vinylidene groups). Because the responses are skewed, they are all log-transformed.

Arabidopsis thaliana data (Wille et al. 2004): This is a microarray experiment. There are 795 genes (the responses) which may show some association to 39 genes (the predictors) from two biosynthesis pathways in A. thaliana. All variables are log transformed.

All responses are standardized to unit variance to make them comparable. The predictive accuracy of each method is estimated by leave-one-out cross-validation (for the A. thaliana data set we used 5-fold cross-validation):

$$MSPE_{CV} = q^{-1}\sum_{k=1}^{q} n^{-1}\sum_{i=1}^{n}(y_{(i)k} - \hat{f}_k^{-i}(\mathbf{x}_{(i)}))^2.$$

Note that we compare the prediction with the observation, the latter being an unbiased rough estimate for the true unknown function $\mathbf{f}$. Therefore the prediction accuracy contains also the error variances which makes it harder to see clear differences between the methods.

The datasets are summarized in table 3.2 and the results are given in table 3.3. We use 5-fold cross validation and the AIC_c to stop the iteration. The implementing $\mathbf{\Gamma}$ in MB is the empirical covariance-matrix $\hat{\mathbf{\Sigma}}$ of the residuals from IB (for the A. thaliana data set we used the diagonal matrix diag($\hat{\mathbf{\Sigma}}$)).

Data set	n	p	q	aac
Chemical reaction	19	9	3	0.56
Macroeconomic	36	20	5	0.71
Chemometrics	56	22	6	0.48
A. thaliana	118	39	795	0.21

Table 3.2: *Summary of the analyzed data sets: sample size (n), number of predictors (p), number of responses (q) and average absolute empirical correlation between the responses (aac).*

Data set	OLS	MFS		RB		IB		MB	
		CV	AIC_c	CV	AIC_c	CV	AIC_c	CV	AIC_c
Chemical	1.343	1.261	0.616	0.532	0.500	0.744	0.527	0.488	0.479
Macroe.	0.499	0.209	0.224	0.193	0.197	0.194	0.195	0.202	0.204
Chemom.	0.411	0.360	0.386	0.253	0.262	0.260	0.208	0.259	0.263
A. thaliana	0.753					0.559	0.556	0.551	

Table 3.3: *Leave-one-out (5-fold for A. thaliana) cross-validated mean squared prediction error $MSPE_{CV}$ for four data sets. Iteration stopped either by 5-fold cross validation or AIC_c. The AIC_c-stopping for A. thaliana is not easily feasible (see section 3.3.4) and the computation of Wilk's Λ (MFS and RB) is only possible if $n > q$.*

MFS performs worst, but there is no overall best boosting method. As mentioned already in section 3.6.3, it seems easier to stop the iteration for MB and RB than for IB. Therefore, the cross-validation stopping and the AIC_c stopping differ only slightly for MB and RB.

For IB, stopping by AIC_c works much better than using cross validation in two examples. The mean squared prediction error of 0.208 for the chemometrics data is quite good compared to the numbers published in Breiman and Friedman (1997). We remark here that we only have rounded data (taken from Skagerberg et al. (1992)) and therefore we get slightly different prediction errors (e.g. for OLS: 0.411 instead of 0.431 in Breiman and Friedman (1997)).

3.8 Conclusions

We propose a multivariate L_2Boosting method for multivariate linear models. The multivariate L_2Boosting inherits the good properties from its univariate counterpart: it does variable selection and shrinkage. Our multivariate L_2Boosting method is suitable for a variety of different situations: (i) multivariate linear regression, with or without seemingly unrelated regressions (SUR), and with covariates which can be arbitrarily correlated; (ii) for multivariate vector autoregressive time series. The method is particularly powerful if the predictor dimension p or the dimension of the response q are large relative to sample size n.

Our multivariate L_2Boosting takes potential correlations among the components of the multivariate error-noise into account. It is therefore very different from OLS and other methods which work on individual responses only. Correlation among the errors can arise from various sources: for example via an unobservable covariate which influences the responses in the same way.

We prove here, for i.i.d. data as well as for time series, that multivariate L_2Boosting can consistently recover sparse, very high-multivariate and very high-dimensional linear functions. When having high-multivariateness, a non-trivial element arises how to control the estimation error over all multivariate components simultaneously: our theory seems to be among the first which actually addresses such questions.

An important question in multivariate regression is whether "jointness" pays off: is the multivariate method better than q estimates from a univariate method? Our simulation study shows that multivariate L_2Boosting outperforms individual univariate L_2Boosting by a substantial amount when the errors are correlated and is almost as good when the errors are independent. On real data, we were not able to see a clear difference (which may be masked by substantial noise variance): this has already been found in other work, e.g. Brooks and Stone (1994).

3.9 Proofs

3.9.1 Proof of Theorem 1

The proof of Theorem 1 is similar as in Bühlmann (2006), where the univariate case is discussed. We define an appropriate Hilbert space and dictionary of basis functions; then, it is sufficient to prove Lemma 1 from Bühlmann (2006) for the setting of multivariate L_2Boosting.

A population version

The L_2Boosting algorithm has a population version which is known as "matching pursuit" (Mallat and Zhang 1993) or "weak greedy algorithm" (Temlyakov 2000).

Consider the Hilbert space $L_2(P) = \{\mathbf{f} : \mathbb{R}^{p_n} \to \mathbb{R}^{q_n}; \|\mathbf{f}\|^2 = \langle \mathbf{f}, \mathbf{f} \rangle < \infty\}$ with inner product $\langle \mathbf{f}, \mathbf{g} \rangle = \int \mathbf{f}(\mathbf{x})^T \mathbf{\Gamma}^{-1} \mathbf{g}(\mathbf{x}) dP(\mathbf{x})$. Here, the probability measure P is generating the predictor $\mathbf{x}$ in model (3.4.1). To be precise, the probability measure $P = P_n$ and the function $\mathbf{f} = \mathbf{f}_n$ depend on n, but we often ignore this notationally (a uniform bound in (3.9.4) will be a key result to deal with sequences of Hilbert spaces).

Denote the components of $\mathbf{x} = (x_1, \ldots, x_{p_n})$ viewed as a scalar or a 1-dimensional function from $\mathbb{R}^{p_n} \to \mathbb{R}$ by

$$g_j(\mathbf{x}) = x_j$$

and denote the components of $\mathbf{x} = (x_1, \ldots, x_{p_n})$ viewed as a q_n-dimensional vector or a function from $\mathbb{R}^{p_n} \to \mathbb{R}^{q_n}$ with only component k different from zero by

$$(\mathbf{g}_{(j,k)})_l(\mathbf{x}) = \begin{cases} x_j & \text{if } l = k, \\ 0 & \text{if } l \neq k. \end{cases}$$

For notational simplicity, we assume that $\|\mathbf{g}_{(j,k)}\| = \int x_j^2 \Gamma_{kk}^{-1} dP(x_j) = \Gamma_{kk}^{-1} = 1$ for all k (it simplifies e.g. the formula (3.9.2)); the proof for non-equal Γ_{kk}^{-1} would work analogously using the second assumption in (A3).

Define the following sequence of remainder functions, called matching pursuit or weak greedy algorithm:

$$\begin{aligned} &R^0\mathbf{f} = \mathbf{f}, \\ &R^m\mathbf{f} = R^{m-1}\mathbf{f} - \left\langle R^{m-1}\mathbf{f}, \mathbf{g}_{(s_m,t_m)} \right\rangle \mathbf{g}_{(s_m,t_m)},\ m = 1, 2, . \end{aligned} \tag{3.9.1}$$

where (s_m, t_m) would be ideally chosen as

$$(s_m, t_m) = \underset{1 \le j \le p_n; 1 \le k \le q_n}{\arg\max} \left| \left\langle R^{m-1}\mathbf{f}, \mathbf{g}_{(j,k)} \right\rangle \right|.$$

The choice functions (s_m, t_m) are often infeasible to realize in practice, because we have finite samples. A weaker criterion is: for every m (under consideration), choose any (s_m, t_m), which satisfies for some $0 < d \le 1$

$$\left| \left\langle R^{m-1}\mathbf{f}, \mathbf{g}_{(s_m,t_m)} \right\rangle \right| \ge d \cdot \sup_{1 \le j \le p_n; 1 \le k \le q_n} \left| \left\langle R^{m-1}\mathbf{f}, \mathbf{g}_{(j,k)} \right\rangle \right|. \tag{3.9.2}$$

Of course, the sequence $R^m\mathbf{f} = R^{m,s,t}\mathbf{f}$ depends on $(s_1, t_1), (s_2, t_2), \ldots, (s_m, t_m)$ how we actually make the choice in (3.9.2). Again, we will ignore this notationally.

It easily follows that

$$\mathbf{f} = \sum_{j=0}^{m-1} \left\langle R^j\mathbf{f}, \mathbf{g}_{(s_{j+1},t_{j+1})} \right\rangle \mathbf{g}_{(s_{j+1},t_{j+1})} + R^m\mathbf{f}.$$

Temlyakov (2000) gives a uniform bound for the algorithm in (3.9.1) with (3.9.2).

If the function $\mathbf{f}$ is representable as

$$\mathbf{f}(\mathbf{x}) = \sum_{j,k} B_{jk}\mathbf{g}_{(j,k)}(\mathbf{x}),\ \sum_{j,k} |B_{jk}| \le D < \infty, \tag{3.9.3}$$

which is true by our assumption (A2), then

$$\|R^m\mathbf{f}\| \le D(1 + md^2)^{-d/(2(2+d))},\ 0 < d \le 1 \text{ as in (3.9.2).} \tag{3.9.4}$$

To make the point clear, this bound holds also for sequences $R^m\mathbf{f} = R^{m,s,t,n}\mathbf{f}$ which depend on the choice function (s,t) in (3.9.2) and on the sample size n (since $\mathbf{x} \sim P$ depends on n and also the function of interest $\mathbf{f}$): all we have to assume is the condition (3.9.3).

A sample version

The multivariate L_2boosting algorithm can be represented analogously to (3.9.1). We introduce the following notation:

$$\langle \mathbf{f}, \mathbf{g} \rangle_{(n)} = n^{-1} \sum_{i=1}^{n} \mathbf{f}^T(\mathbf{x}_{(i)}) \mathbf{\Gamma}^{-1} \mathbf{g}(\mathbf{x}_{(i)}) \text{ and } \|\mathbf{f}\|^2_{(n)} = \langle \mathbf{f}, \mathbf{f} \rangle_{(n)}$$

for functions $\mathbf{f}, \mathbf{g} : \mathbb{R}^{p_n} \to \mathbb{R}^{q_n}$. As before, we denote by $\mathbf{Y} = (\mathbf{y}_{(1)}, \ldots, \mathbf{y}_{(n)})^T$ the matrix of response variables.

Define

$$\hat{R}^1_n \mathbf{f} = \mathbf{f} - \left\langle \mathbf{Y}, \mathbf{g}_{(\hat{s}_1, \hat{t}_1)} \right\rangle_{(n)} \mathbf{g}_{(\hat{s}_1, \hat{t}_1)},$$
$$\hat{R}^m_n \mathbf{f} = \hat{R}^{m-1}_n \mathbf{f} - \left\langle \hat{R}^{m-1}_n \mathbf{f}, \mathbf{g}_{(\hat{s}_m, \hat{t}_m)} \right\rangle_{(n)} \mathbf{g}_{(\hat{s}_m, \hat{t}_m)}, \; m = 2, 3, \ldots,$$

where

$$(\hat{s}_1, \hat{t}_1) = \operatorname*{arg\,max}_{1 \le j \le p_n; 1 \le k \le q_n} |\left\langle \mathbf{Y}, \mathbf{g}_{(j,k)} \right\rangle_{(n)}|,$$
$$(\hat{s}_m, \hat{t}_m) = \operatorname*{arg\,max}_{1 \le j \le p_n; 1 \le k \le q_n} |\left\langle \hat{R}^{m-1}_n \mathbf{f}, \mathbf{g}_{(j,k)} \right\rangle_{(n)}|, \; m = 2, 3, \ldots.$$

With some abuse of notation, we denote by $\hat{R}^{m-1}_n \mathbf{f}$ and $\mathbf{g}_{(\hat{s}_m, \hat{t}_m)}$ either functions from $\mathbb{R}^{p_n} \to \mathbb{R}^{q_n}$ or $n \times q_n$ matrices evaluated at the observed predictors. We emphasize here the dependence of $\hat{R}^m_n$ on n since finite-sample estimates $\left\langle \hat{R}^{m-1}_n \mathbf{f}, \mathbf{g}_{(j,k)} \right\rangle_{(n)}$ are involved. We also assume without loss of generality (but simplifying the notation) that $\|\mathbf{g}_{(j,k)}\|_{(n)} \equiv 1$ for all j, k and n (note that we have already assumed w.l.o.g. before that $\|\mathbf{g}_{(j,k)}\| \equiv 1$ for all j, k): then, the formulae above are the same as in (3.3.2) (because $\|\mathbf{g}_{(j,k)}\|_{(n)} = \mathbf{x}_j^T \mathbf{x}_j \Gamma^{-1}_{kk}$). Hence, $\hat{R}^m_n \mathbf{f} = \mathbf{f} - \hat{\mathbf{f}}^{(m)}$.

For analyzing $\|\hat{R}^m_n \mathbf{f}\| = \mathbb{E}_{\mathbf{x}} |(\hat{\mathbf{f}}^{(m_n)}(\mathbf{x}) - \mathbf{f}(\mathbf{x}))^T \mathbf{\Gamma}^{-1} (\hat{\mathbf{f}}^{(m_n)}(\mathbf{x}) - \mathbf{f}(\mathbf{x}))|$, which is the quantity in the assertion of Theorem 1, we need some uniform laws of large numbers, as discussed below.

Uniform laws of large numbers

Lemma 1 *Under the assumptions (A1)-(A5), with $0 < \xi < 1$ as in (A1),*

(i) $\sup_{1 \le j,u \le p_n; 1 \le k,v \le q_n} \left| \left\langle \mathbf{g}_{(j,k)}, \mathbf{g}_{(u,v)} \right\rangle_{(n)} - \left\langle \mathbf{g}_{(j,k)}, \mathbf{g}_{(u,v)} \right\rangle \right| = \zeta_{n,1} = O_P(n^{-\xi/2})$,

(ii) $\sup_{1 \le j \le p_n; 1 \le k \le q_n} \left| \left\langle \mathbf{g}_{(j,k)}, \mathbf{E} \right\rangle_{(n)} \right| = \zeta_{n,2} = O_P(n^{-\xi/2})$,

(iii) $\sup_{1 \le j \le p_n; 1 \le k \le q_n} \left| \left\langle \mathbf{g}_{(j,k)}, \mathbf{f} \right\rangle_{(n)} - \left\langle \mathbf{g}_{(j,k)}, \mathbf{f} \right\rangle \right| = \zeta_{n,3} = O_P(n^{-\xi/2})$,

(iv) $\sup_{1 \le j \le p_n; 1 \le k \le q_n} \left| \left\langle \mathbf{g}_{(j,k)}, \mathbf{Y} \right\rangle_{(n)} - \left\langle \mathbf{g}_{(j,k)}, \mathbf{Y} \right\rangle \right| = \zeta_{n,4} = O_P(n^{-\xi/2})$.

Proof: Assertion (i):

$$\begin{aligned}
&\sup_{j,u,k,v} \left| \left\langle \mathbf{g}_{(j,k)}, \mathbf{g}_{(u,v)} \right\rangle_{(n)} - \left\langle \mathbf{g}_{(j,k)}, \mathbf{g}_{(u,v)} \right\rangle \right| = \\
&= \sup_{j,u,k,v} \left| n^{-1} \sum_{i=1}^{n} \mathbf{g}_{(j,k)}^T(\mathbf{x}_{(i)}) \boldsymbol{\Gamma}^{-1} \mathbf{g}_{(u,v)}(\mathbf{x}_{(i)}) - \mathbb{E}\left[\mathbf{g}_{(j,k)}^T(\mathbf{x}_{(i)}) \boldsymbol{\Gamma}^{-1} \mathbf{g}_{(u,v)}(\mathbf{x}_{(i)}) \right] \right| = \\
&= \sup_{j,u,k,v} \left| n^{-1} \sum_{i=1}^{n} x_{(i)j} \Gamma_{kv}^{-1} x_{(i)u} - \mathbb{E}\left[x_{(1)j} \Gamma_{kv}^{-1} x_{(1)u} \right] \right| = \\
&= \sup_{j,u,k,v} |\Gamma_{kv}^{-1}| \left| n^{-1} \sum_{i=1}^{n} g_j(\mathbf{x}_{(i)}) g_u(\mathbf{x}_{(i)}) - \mathbb{E}\left[g_j(\mathbf{x}_{(1)}) g_u(\mathbf{x}_{(1)}) \right] \right| = \\
&\le \sup_{k,v} |\Gamma_{kv}^{-1}| \cdot O_P(n^{-\xi/2}) = O_P(n^{-\xi/2}).
\end{aligned}$$

We have used here that $\sup_{j,u} |n^{-1} \sum_{i=1}^{n} g_j(\mathbf{x}_{(i)}) g_u(\mathbf{x}_{(i)}) - \mathbb{E}\left[g_j(\mathbf{x}_{(1)}) g_u(\mathbf{x}_{(1)}) \right] | = O_P(n^{-\xi/2})$ (Bühlmann 2006), and also the first assumption in (A3).

Assertion (ii): We write

$$\left\langle \mathbf{g}_{(j,k)}, \mathbf{E} \right\rangle_{(n)} = \sum_{v=1}^{q_n} n^{-1} \sum_{i=1}^{n} x_{(i)j} \Gamma_{kv}^{-1} e_{(i)v} = n^{-1} \sum_{i=1}^{n} g_j(\mathbf{x}_{(i)}) Q_i(k), \tag{3.9.5}$$

where $Q_i(k) = \sum_{v=1}^{q_n} \Gamma_{kv}^{-1} e_{(i)v}$.

Note that $Q_i(k)$ is independent from $\mathbf{X}, \mathbb{E}[Q_i(k)] = 0$ for all i, k and

$$\sup_k \mathbb{E}|Q_i(k)|^s \leq \sup_k (\sum_{v=1}^{q_n} |\Gamma_{kv}^{-1}|(\mathbb{E}|e_{(1)v}|^s)^{1/s})^s < \infty, \quad (3.9.6)$$

using assumptions (A3) and (A5). The form in (3.9.5) with the moment property in (3.9.6) is the same as in Lemma 1 (ii) from Bühlmann (2006).

Assertions (iii): Note that

$$\sup_{j,k} | \left\langle \mathbf{g}_{(j,k)}, \mathbf{f} \right\rangle_{(n)} - \left\langle \mathbf{g}_{(j,k)}, \mathbf{f} \right\rangle | \leq \sum_{u,v} |B_{uv,n}| \cdot \zeta_{n,1} = O_P(n^{-\xi/2})$$

using assumption (A2) and the bound from assertion (i).

Assertion (iv): This follows immediately from assertions (ii) and (iii). □

The rest of the proof is the same as in Bühlmann (2006). We only have to replace the basis functions g_j by our double indexed basis functions $\mathbf{g}_{(j,k)}$. □

3.9.2 Proof of Theorem 2

As we have seen from the proof of Theorem 1, a substantial part of the analysis can be borrowed from Bühlmann (2006): we only need to reconsider uniform laws of large numbers, as in Lemma 1, but for dependent data. This can be done by invoking the following result.

Lemma 2 *Consider sequences $\{Z_{t,n}\}_{t\in\mathbb{Z}}$, $n \in \mathbb{N}$, which are strictly stationary and α-mixing with mixing coefficients $\alpha_{Z,n}(\cdot)$. Assume that $\mathbb{E}[Z_{t,n}] = 0$ for all $n \in \mathbb{N}$, $\sup_{n\in\mathbb{N}} \mathbb{E}|Z_{t,n}|^{2s+\gamma} < \infty$ for some $s \in \mathbb{N}$, $\gamma > 0$, and the mixing coefficients satisfy for some constants $0 < C_1, C_2 < \infty$:*

$$\sum_{k=0}^{\infty} (k+1)^{s-1} \alpha_{Z,n}(k)^{\gamma/(4s+\gamma)} < C_1 p_n^s + C_2,$$

where $s \in \mathbb{N}$ is linked to the moments of $Z_{t,n}$ as above. Then,

$$\mathbb{E}|n^{-1}\sum_{t=1}^{n} Z_{t,n}|^{2s} = O(p_n^s n^{-s}) \ (n \to \infty).$$

Proof: The reasoning can be done analogously to the proof of Theorem 1 in Yokoyama (1980). □

The only part of the proof of Theorem 1 which needs to be changed is Lemma 1. A version of Lemma 1 also holds for stationary VAR(∞) processes; the predictor variables at time t are the p_n lagged q_n-dimensional variables $\mathbf{x}_{(t-1)}, \ldots, \mathbf{x}_{(t-p)}$ and the response variable is the current $\mathbf{x}_{(t)}$.

Instead of exponential inequalities we first invoke Markov's inequality and then Lemma 2. For example, for the analogue of Lemma 1 (i) we bound

$$\begin{aligned}&\mathbb{P}[|(n-p_n)^{-1}\sum_{t=p_n+1}^{n} x_{(t-j)k}x_{(t-u)v} - \mathbb{E}[x_{(t-j)k}x_{(t-u)v}]| > \varepsilon]\\ &\le \varepsilon^{-2s}\,\mathbb{E}|(n-p_n)^{-1}\sum_{t=p_n+1}^{n} x_{(t-j)k}x_{(t-u)v} - \mathbb{E}[x_{(t-j)k}x_{(t-u)v}]|^{2s}.\end{aligned} \tag{3.9.7}$$

We now observe that $Z_{t,n} = x_{(t-j)k}x_{(t-u)v} - \mathbb{E}[x_{(t-j)k}x_{(t-u)v}]$ is still stationary and α-mixing whose coefficients satisfy the requirement from Lemma 2. Due to different lags j and u, the mixing coefficients of $Z_{t,n}$ usually don't decay for the first $|j-u|$ lags (therefore the special construction with $C_1 p_n^s + C_2$ in Lemma 2). Invoking Lemma 2 for the right hand side of (3.9.7) we get

$$\begin{aligned}&\mathbb{P}[|(n-p_n)^{-1}\sum_{t=p_n+1}^{n} x_{(t-j)k}x_{(t-u)v} - \mathbb{E}[x_{(t-j)k}x_{(t-u)v}]| > \varepsilon]\\ &\le \varepsilon^{-2s}O(p_n^s n^{-s}) = O(n^{-s\kappa})\end{aligned}$$

since $p_n = O(n^{1-\kappa})$ by assumption. For the supremum over the different lags and components we then get

$$\begin{aligned}&\mathbb{P}[\sup_{1\le j,u\le p_n, 1\le k,v\le q_n}|(n-p_n)^{-1}\sum_{t=p_n+1}^{n} x_{(t-j)k}x_{(t-u)v} - \mathbb{E}[x_{(t-j)k}x_{(t-u)v}]| > \varepsilon]\\ &\qquad = O(p_n^2 q_n^2 n^{-s\kappa}) = O(n^{2(1+\delta)-(s+2)\kappa}).\end{aligned}$$

Hence, since $\kappa > 2(1+\delta)/(s+2)$, we have proved that there exists a $c > 0$ such that

$$\sup_{1\le j,u\le p_n, 1\le k,v\le q_n} |(n-p_n)^{-1} \sum_{t=p_n+1}^{n} x_{(t-j)k}x_{(t-u)v} - \mathbb{E}[x_{(t-j)k}x_{(t-u)v}]|$$
$$= O_P(n^{-c}).$$

The version of Lemma 1 (ii) follows analogously; and the versions of Lemma 1 (iii) and (iv) can be proved exactly as in Lemma 1. □

Chapter 4

Robustified L_2Boosting

We consider five robustifications of L_2Boosting for linear regression with various robustness properties. The first two use the Huber loss as implementing loss function for boosting and the second two use robust simple linear regression for the fitting in L_2Boosting (i.e. robust base learners). Both concepts can be applied with or without down-weighting of leverage points. Our last method uses robust correlation estimates and appears to be most robust. Crucial advantages of all methods are that they don't compute covariance matrices of all covariates and that they don't have to identify multivariate leverage points. When there are no outliers, the robust methods are only slightly worse than L_2Boosting. In the contaminated case though, the robust methods outperform L_2Boosting by a large margin. Some of the robustifications are also computationally highly efficient and therefore well suited for truly high dimensional problems.

4.1 Introduction

Freund and Schapire's AdaBoost algorithm for classification (Freund and Schapire 1996) has attracted much attention in the machine learning community and related fields, mainly because of its good empirical performance. Some boosting algorithms for regression were also proposed, but the first practical algorithm was not possible until Breiman (1999) showed, that boosting can be viewed as a functional gradient descent algorithm. Friedman (2001) then proposed LS_Boost (least squares boosting, we will call it L_2Boosting) and also more robust boosting methods in conjunction with regression trees.

Boosting with the L_2-loss (L_2Boosting) and componentwise linear fitting was worked out in detail in Bühlmann (2006). It is essentially the same as Mallat and Zhang's (1993) matching pursuit algorithm in signal processing and very similar to stagewise linear model fitting (see for example Efron et al. (2004)). Boosting is then not just a black box tool, but fits sound linear models. It does variable selection and coefficient shrinkage and for high dimensional problems, it is clearly superior to the classical model selection methods.

The usage of the L_2-loss is dangerous when there are outliers. Friedman (2001) discussed some robust boosting algorithms with regression trees. In this paper we develop some robust boosting algorithms for linear models by using either robust implementing loss functions in boosting or robust estimators as base (weak) learners. They all do variable selection and estimation of regression coefficients. Some of our methods are also well suited for very high dimensional problems with many covariates and/or large sample size. Besides more classical work in robust fitting and variable selection for linear models (Ronchetti and Staudte 1994, Ronchetti, Field and Blanchard 1997, Morgenthaler, Welsch and Zenide 2003), our approaches are closest to "robust LARS" (Van Aelst, Khan and Zamar 2004). However, the concepts of robust loss functions and robust base learners are much more general.

4.2 L_2Boosting with componentwise linear least squares

We consider the linear model $\mathbf{y} = \mathbf{X}\beta + \boldsymbol{\epsilon}$ with $\mathbf{y} \in \mathbb{R}^n$ and $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p) \in \mathbb{R}^{n \times p}$ and will use boosting methods for fitting it. For a boosting algorithm we need a loss function $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+$, that measures how close a fitted value $\hat{F}(x_i)$ comes to the observation y_i and a base learner (simple fitting method), that yields a function estimate $\hat{f} : \mathbb{R}^p \to \mathbb{R}$. L_2Boosting uses the L_2-loss $L(y, F) = (y - F)^2/2$ and as base learner we take componentwise linear least squares, which works as follows: a response $\mathbf{r} \in \mathbb{R}^n$ with $\bar{\mathbf{r}} = 0$ is fitted against $\mathbf{x}_1, \ldots, \mathbf{x}_p$:

Componentwise linear least squares learner

$$\begin{aligned}
\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \ x \in \mathbb{R}^p \\
\hat{\beta}_j &= \frac{(\mathbf{x}_j - \bar{\mathbf{x}}_j)^T \mathbf{r}}{(\mathbf{x}_j - \bar{\mathbf{x}}_j)^T (\mathbf{x}_j - \bar{\mathbf{x}}_j)}, \quad \hat{\alpha}_j = -\hat{\beta}_j \bar{\mathbf{x}}_j, \quad 1 \le j \le p, \\
\hat{s} &= \arg\min_{1 \le j \le p} ||\mathbf{r} - \hat{\alpha}_j - \hat{\beta}_j \mathbf{x}_j||^2 = \arg\max_{1 \le j \le p} |\hat{\beta}_j| \cdot \mathrm{sd}(\mathbf{x}_j) = \\
&= \arg\max_{1 \le j \le p} |\operatorname{corr}(\mathbf{r}, \mathbf{x}_j)|.
\end{aligned}$$

In words: we fit a simple linear regression with one selected covariate. The selected covariate is the one which gives the smallest residual sum of squares. This is equivalent to the variable that gives the "largest contribution to the fit" or has the highest absolute correlation with the response $\mathbf{r}$. The requirement $\bar{\mathbf{r}} = 0$ is without loss of generality for boosting since we always center the response variable before, see the algorithm below. The learner can be simplified if all covariates are centered (mean subtracted). Then we can fit simple linear regressions through the origin.

A boosting algorithm constructs iteratively a function $\hat{F} : \mathbb{R}^p \to \mathbb{R}$ by considering the empirical risk $n^{-1} \sum_{i=1}^n L(y_i, F(x_i))$, $x_i \in \mathbb{R}^p$ and pursuing iterative approximate steepest descent in function space. This means that in each iteration, the negative gradient of the loss function is fitted by the base learner. L_2Boosting is especially simple, because the negative gradient becomes the current residual vector and the algorithm amounts to iteratively fitting of residuals:

L_2Boosting with componentwise linear least squares

1. Initialize $\hat{F}^{(0)} \equiv \arg\min_{a \in \mathbb{R}} \sum_{i=1}^n L(y_i, a) \equiv \bar{\mathbf{y}}$. Set $m = 0$.

2. Increase m by 1. Compute the negative gradient (also called pseudo response), which is the current residual vector

$$r_i = -\frac{\partial}{\partial F} L(y, F)|_{F = \hat{F}^{(m-1)}(x_i)} = y_i - \hat{F}^{(m-1)}(x_i), \ i = 1, \ldots, n.$$

3. Fit the residual vector $(r_1, \ldots, r_n)$ to $\mathbf{x}_1, \ldots, \mathbf{x}_p$ by the componentwise linear least squares base procedure
$$(x_i, r_i)_{i=1}^n \longrightarrow \hat{f}^{(m)}(\cdot).$$
4. Update $\hat{F}^{(m)}(\cdot) = \hat{F}^{(m-1)}(\cdot) + \nu \cdot \hat{f}^{(m)}(\cdot)$, where $0 < \nu \leq 1$ is a step length factor.
5. Iterate steps 2 to 4 until $m = m_{stop}$ for some stopping iteration m_{stop}.

The number of iterations $m = m_{stop}$ is usually estimated using a validation set or with cross validation. The step-length factor ν is also called shrinkage factor and is typically less crucial than m_{stop}. The natural value is 1, but smaller values have empirically proven to be a better choice. We will always use $\nu = 0.3$. Since the base learner yields a linear model fit in one covariate and because of the linear up-date in step 4, L_2Boosting with componentwise linear least squares yields a linear model fit (with estimated coefficient vector $\hat{\beta}^{(m)}$). Since least squares fitting is used, the method is not robust to outliers.

4.3 Robustifications

There are several ways to robustify L_2Boosting with componentwise linear least squares as described next. Whenever we need a robust location estimate we will use the Huber estimator with MAD scale (see Huber 1964, Huber 1981 and Hampel, Ronchetti, Rousseeuw and Stahel 1986). The Huber Ψ-function is given by
$$\Psi_c(x) = \min\{c, \max\{x, -c\}\} = x \cdot \min\{1, \frac{c}{|x|}\}.$$
As robust scale estimator we use the Q_n estimator of Rousseeuw and Croux (1993). It is defined as
$$Q_n(x_1, \ldots, x_n) = 2.2219 \cdot \{|x_i - x_j|; i < j\}_{(k)},$$
where $k = \binom{\lfloor n/2 \rfloor + 1}{2}$. That is, we take the k-th order statistic of the $\binom{n}{2}$ inter-point distances. The Q_n estimator has a breakdown point of 50% and an efficiency of 82% at the Gaussian distribution (Rousseeuw and Croux 1993).

4.3.1 Boosting with a robust implementing loss function

The easiest robustification is to use a robust loss function, e.g. the Huber loss function (the derivation yields the Huber Ψ-function):

$$L_c(y, F) = \begin{cases} (y - F)^2/2, & |y - F| \leq c, \\ c * (|y - F| - c/2), & |y - F| > c. \end{cases}$$

The parameter c should be chosen in dependence of the scale of $y - F$. We choose it adaptively in each iteration as $c = 1.345 \cdot \text{MAD}(\{y_i - \hat{F}^{(m-1)}(x_i), i = 1, \ldots, n\})$ as proposed in Friedman (2001). The negative gradient in step 2 of the boosting algorithm then becomes the huberized residual vector. As learner we can take componentwise linear least squares as described above. This means we look in each iteration for the covariate that best fits the huberized residuals (the criterion is the huberized residual sum of squares). We found that it is better to also estimate an intercept in each iteration than robust centering the covariates and estimate the intercept only at the beginning. We shall call this version *RobLoss* boosting.

Remark 3 *If we want to exactly apply the Gradient Boost algorithm of Friedman (2001) we have to do an additional line search between step 3 and 4. This means each* $\hat{f}^{(m)}$ *must be multiplied by a constant to minimize the* L_c*-loss. This would yield factors slightly greater than 1. Since we are going to use shrinkage* $\nu = 0.3$ *it is not important to know the optimal (greedy) step size exactly and we can omit the additional line search.*

It is obvious that RobLoss boosting is only robust in “Y-direction” but not in “X-direction”. But it is not possible to incorporate “X-direction-robustness” in the loss function and therefore, we do it in the base procedure when fitting the negative gradient in step 3. The idea is to down-weight the leverage points and to use weighted least squares for fitting (a Mallows type estimator). Since every covariate is fitted alone, the weight of an observation is solely determined by the value of the one covariate. Therefore, the same observation can have different weights for the p candidate fits with the p covariates in one iteration, according to its outlyingness of the corresponding coordinate. For the weights we use ($w_{ij}^{position}$ is the weight of observation i when fitting the j-th covariate)

$$w_{ij}^{position} = \min\{1, \frac{1.345}{|(x_{ij} - \text{Huber}(\mathbf{x}_j))/\,\text{MAD}(\mathbf{x}_j)|}\}.$$

We can go one step further and down-weight only the leverage points that have also a large residual (a Schweppe type estimator). The weights we use in iteration m are

$$w_{ij} = \frac{\Psi_{1.345 \cdot w_{ij}^{position}}\left(\frac{y_i - \hat{F}^{(m-1)}(x_i)}{\text{MAD}(\{y_i - \hat{F}^{(m-1)}(x_i),\ i=1,\ldots,n\})}\right)}{\Psi_{1.345}\left(\frac{y_i - \hat{F}^{(m-1)}(x_i)}{\text{MAD}(\{y_i - \hat{F}^{(m-1)}(x_i),\ i=1,\ldots,n\})}\right)}.$$

So far, we only discussed how to fit the pseudo response to a covariate, but the selection of the "best" covariate is equally important. It seems quite natural to select the covariate that gives the smallest weighted residual sum of squares. But since the p simple linear fits in each iteration use different weights for the same observation, this can lead to bad choices. It is better to use the estimated $\hat{\beta}_j$'s and to select the variable that has highest $|\hat{\beta}_j| \cdot Q_n(\mathbf{x}_j)$: note that this is of the form as used in the classical componentwise linear least squares base procedure in section 4.2. Roughly speaking we choose the covariate that contributes the most to the fit. We shall call this version *RobLossW* boosting. Here is the formal description of the learner (w_{ij} as described above with r_i instead of $y_i - \hat{F}^{(m-1)}(x_i)$):

Componentwise linear weighted least squares learner

$$\begin{aligned}
\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \\
(\hat{\alpha}_j, \hat{\beta}_j) &= \arg\min_{\alpha,\beta} \sum_{i=1}^{n} w_{ij}(r_i - \hat{\alpha}_j - \hat{\beta}_j x_{ij})^2, \\
\hat{s} &= \arg\max_{1\le j\le p} |\hat{\beta}_j| \cdot Q_n(\mathbf{x}_j).
\end{aligned}$$

4.3.2 Boosting with robust regression learner

Here we use the idea of iteratively fitting of residuals. Instead of fitting the ordinary residuals by least squares, we use a robust linear regression:

Componentwise robust linear regression learner

$$\begin{aligned}\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}},\\ (\hat{\alpha}_j, \hat{\beta}_j) &= \text{robust linear fit of } \mathbf{r} \text{ against } \mathbf{x}_j,\\ \hat{s} &= \underset{1\le j\le p}{\arg\max} |\hat{\beta}_j| \cdot \mathit{scale}(\mathbf{x}_j).\end{aligned}$$

In each iteration of the boosting algorithm we calculate a robust linear regression with each covariate alone (and an intercept). This needs more computation than the RobLoss methods, since the robust fits are usually calculated iteratively itself (the RobLoss methods do in some sense only the first iteration of the iteration). As criterion for the variable selection we use again $\arg\max_j |\hat{\beta}_j| \cdot \mathrm{scale}(\mathbf{x}_j)$, where $\mathrm{scale}(\mathbf{x}_j)$ is a scale estimate that will be specified below. This is much better than using a robust estimation of residual standard error.

For the robust linear fit of the base procedure we use two different types: M-regression with Huber's Ψ-function (and rescaled MAD of the residuals) and a Schweppe type bounded influence (BI) regression (see for example Hampel et al. (1986)) with Huber's Ψ-function and position weights as described in section 4.3.1. We chose these types of robust regression to have a direct comparison to the RobLoss methods. One could even use MM-regression, but this would be computationally very expensive. The proposed algorithms will be called *RobRegM* boosting and *RobRegBI* boosting. For the former method, which is robust in "Y-direction" but not in "X-direction", we use the standard deviation as scale estimate for the variable selection and for the latter, which is robust in "Y- and X-direction", we use the Q_n estimator.

We expect that RobLoss and RobRegM boosting perform similar and likewise for RobLossW and RobRegBI boosting. The former methods first huberize the residuals and then use (weighted) least squares and the latter methods use robust methods with the same huberization and weighting. As already mentioned, the RobLoss methods do in some sense the first iteration of the robust fitting of the RobReg methods. The great advantage of the former methods is that they are much faster. Thus, the latter methods must achieve better performance to be worthwhile.

4.3.3 Boosting with robust correlation learner

It is also possible to use robust correlation estimators to construct a base procedure. The idea is the following: in each iteration, the covariate with the highest robust correlation with the residuals is chosen, see also the selection in classical componentwise least squares described in section 4.2. We shall call this version *RobCor* boosting:

Robust correlation learner

$$\begin{aligned}
\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \\
\hat{\beta}_j &= \text{RobCor}(\mathbf{x}_j, \mathbf{r}) \cdot Q_n(\mathbf{r})/Q_n(\mathbf{x}_j), \\
\hat{\alpha}_j &= \text{Huber}(\mathbf{r}) - \hat{\beta}_j \cdot \text{Huber}(\mathbf{x}_j), \\
\hat{s} &= \arg\max_{1\le j\le p} |\,\text{RobCor}(\mathbf{x}_j, \mathbf{r})|.
\end{aligned}$$

Recall that it is not important to have very accurate $\hat{\beta}_j$'s because we use shrinkage $\nu = 0.3$. As robust correlation estimate we use a proposal from Huber (1981) with the Q_n estimator as module:

$$\text{RobCor}(\mathbf{x}, \mathbf{y}) = \frac{Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} + \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2 - Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} - \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2}{Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} + \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2 + Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} - \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2}.$$

4.3.4 Stopping the boosting iteration

To stop the boosting iteration, we propose to use cross validation or a separate validation set. After each iteration we use the actual model to predict on the validation sample and we measure the quality of the fit. For L_2Boosting we use the mean squared prediction error on the validation set and for the robust methods, we use a robust measure of prediction error. Ronchetti et al. (1997) propose to use the Huber loss of the errors of the validation set. We found that using the Q_n-estimator of the errors on the validation set gives better results and therefore, we tune all robust boosting methods with the Q_n-estimator.

4.3.5 Properties of the robust boosting methods

An unaesthetic property of RobCor boosting with $\nu = 1$ is that the same covariate can be chosen consecutively. This is because the robust correlation between the residuals and a covariate is usually not equal to zero after fitting the covariate in the iteration before. This is in contrast to L_2Boosting and the RobReg methods, where, with $\nu = 1$, we cannot improve the fit by selecting and fitting the same covariate as in the iteration before. For the RobLoss methods it can also happen that they choose the same covariate consecutively (even with $\nu = 1$).

RobCor boosting empirically shows the following behavior: after running for a large number of iterations, it always selects the same variable and gets stuck. Then the estimated coefficients are all of approximately equal size and successive coefficients are of opposite sign. This means that RobCor boosting estimates the coefficient of the selected variable to high and in the next iteration it undoes the previous step. The good thing is that this doesn't happen until over-fitting occurs, so we would stop the iteration before anyway. We can also delay the getting stuck by choosing a smaller ν.

Regarding the break down point we can state the following simple proposition:

Proposition 4 *The boosting method inherits the breakdown point of the base procedure.*

Since we are performing only a finite number of iterations, the whole boosting algorithm can only break down when the base learner breaks down in one iteration. RobCor boosting has therefore a breakdown point of 0.5.

4.4 Simulation study

In this section we compare the different boosting methods on simulated data sets from a linear model $\mathbf{y} = \mathbf{X}\beta + \epsilon$ as described at the beginning of section 4.2.

4.4.1 Design

The sample size of the training set (and also the validation set, used to stop the iteration) is $n = 100$ and the number of covariates is $p = 10$ in the first example and $p = 100$ in the second example. The true coefficient vector β is an arbitrary permutation of $(8, 7, 6, 5, 4, 0, 0, 0, 0, 0)^T$ for $p = 10$ and $(18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 0, \ldots, 0)^T$ for $p = 100$. Thus, we have 5 effective and 5 noise variables for $p = 10$ or 10 and 90 for $p = 100$, respectively. We use two design matrices for the covariates and two error distributions. For the first design (normal design), the covariates are generated according to a multivariate normal distribution with mean zero and $\mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) = \Sigma_{ij} = 0.5^{|i-j|}$. The second design (leverage design) is the same, except that 10% of the data-points are multiplied by 5. This is done after the true y-values have been determined. Therefore we have bad leverage points. The error distributions are standard normal (N) and 90% standard normal and 10% from $\mathcal{N}\left(0, 5^2\right)$ (10%N5). The errors are multiplied by a constant to give a signal-to-noise ratio of 4 in the first example and 9 in the second example for normal errors. Each setting is replicated 100 times.

4.4.2 Performance measure

Our main performance measure is the mean squared prediction error, which can be calculated as

$$\widehat{intercept}^{\,2} + (\hat{\beta} - \beta)^T \mathbf{\Sigma} (\hat{\beta} - \beta).$$

We use paired sample Wilcoxon tests to examine whether a method is significantly better than another. Additionally, as a measure of variable selection accuracy, we count how many variables have been chosen until the 5 (respectively 10) effective covariates have been selected (optimal would be 5 or 10, respectively).

4.4.3 Results for $p = 10$

	Normal design				Leverage design			
	N		10%N5		N		10%N5	
L2	7.7	(4.1)	25.5	(13.9)	138.4	(24.5)	148.4	(27.9)
RobLoss	8.9	(5.0)	11.4	(5.8)	87.2	(41.3)	102.8	(47.5)
RobRegM	8.8	(5.3)	11.4	(5.9)	86.1	(41.0)	99.9	(44.4)
RobLossW	9.1	(5.2)	11.5	(6.0)	19.7	(8.8)	25.4	(11.8)
RobRegBI	9.1	(4.8)	11.8	(6.0)	19.3	(8.8)	25.4	(12.2)
RobCor	11.3	(6.8)	13.8	(7.7)	15.5	(9.5)	18.8	(9.4)

Table 4.1: *Mean squared prediction error of the different boosting methods when stopping with a validation set, averaged over 100 replicates for $p = 10$. The standard deviations are given in parentheses.*

Table 4.1 gives the average of the mean squared prediction error when stopping with the validation set. The results are as expected. For the normal design with normal errors, L_2Boosting performs significantly best. The robust methods are only slightly worse, except perhaps RobCor boosting which is significantly worse than the other robust methods. For the normal design with error 10%N5, L_2Boosting performs much (and significantly) worse than the robust methods, and RobCor boosting is still significantly worse than the other robust methods.

The leverage design shows the biggest differences. L_2Boosting performs really bad and RobLoss and RobRegM boosting are not much better. RobCor boosting performs best and all differences are highly significant (except the pairs RobLoss-RobRegM and RobLossW-RobRegBI).

In table 4.2, we examine whether we can stop the boosting iteration at a good point. Given is the percentage loss when using a validation set (of same size as the training set) to stop the iteration compared to optimal stopping (stopping at the iteration which gives smallest mean squared prediction error, also called "oracle" stopping). The stopping works quite satisfyingly for L_2Boosting for the normal design, but not so well for the robust methods. They lose about 20% in performance compared to optimal (oracle) stopping.

In table 4.3 we state how many variables have been chosen (on average) until all 5 effective covariates have been selected. The outcomes

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	13	10	48	45
RobLoss	25	22	14	19
RobRegM	25	21	13	16
RobLossW	24	21	19	16
RobRegBI	23	24	19	19
RobCor	27	26	33	25

Table 4.2: *Percentage loss of stopping with a validation set compared to optimal stopping for* $p = 10$.

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	5.2	5.7	6.7	7.1
RobLoss	5.2	5.3	6.6	6.7
RobRegM	5.1	5.2	6.6	6.9
RobLossW	5.2	5.2	5.9	6.1
RobRegBI	5.2	5.2	5.4	5.7
RobCor	5.3	5.4	5.4	5.6

Table 4.3: *Average number of covariates that have been chosen until all 5 effective variables have been selected for* $p = 10$.

confirm more or less the results of table 4.1, in the sense that the methods with bad prediction performance usually also select too many variables. The big difference is that RobRegBI boosting selects the variables better than RobLossW boosting, although they have comparable predictive power. This indicates that some noise variables in the fitted model with small coefficients don't hurt for prediction.

4.4.4 Results for $p = 100$

The results for $p = 100$ are contained in tables 4.4, 4.5 and 4.6. This setting is much harder. The methods not only have to cope with outliers but also with high dimensional observations. The results are qualitatively more or less the same as for $p = 10$. The main differences are: for

	Normal design				Leverage design			
	N		10%N5		N		10%N5	
L2	112	(41)	371	(166)	1060	(231)	1203	(220)
RobLoss	124	(45)	186	(81)	538	(210)	668	(252)
RobRegM	125	(45)	189	(77)	560	(242)	665	(245)
RobLossW	138	(49)	196	(87)	417	(170)	569	(221)
RobRegBI	134	(48)	200	(84)	389	(148)	525	(196)
RobCor	187	(77)	268	(129)	326	(149)	472	(231)

Table 4.4: *Mean squared prediction error of the different boosting methods when stopping with a validation set, averaged over 100 replicates for* $p = 100$*. The standard deviations are given in parentheses.*

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	4	6	126	68
RobLoss	9	8	8	13
RobRegM	10	8	12	11
RobLossW	9	9	6	9
RobRegBI	8	9	8	10
RobCor	8	7	8	13

Table 4.5: *Percentage loss of stopping with a validation set compared to optimal stopping for* $p = 100$*.*

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	10.9	16.8	26.3	48.5
RobLoss	11.0	12.2	27.3	33.1
RobRegM	10.9	12.0	27.7	34.3
RobLossW	11.4	12.4	25.5	34.8
RobRegBI	11.1	12.2	20.9	26.9
RobCor	12.1	13.9	16.9	20.3

Table 4.6: *Average number of covariates that have been chosen until all 10 effective variables have been selected for* $p = 100$*.*

the normal design, RobCor performs much worse than the other robust methods. For the leverage design, the differences between the methods are less pronounced but there is now a significant difference between RobLossW and RobRegBI boosting.

The stopping of the boosting iteration is relatively easier in this high dimensional setting. The robust methods lose only about 10% compared to optimal stopping.

4.5 Real data

As a real data set we analyze the measurements of Maguna, Núñez, Okulik and Castro (2003) (see also Maronna, Martin and Yohai 2006). There are 38 observations (17 monocarboxylic, 9 dicarboxylic and 12 unsaturated carboxylic acids) and the goal is to predict the logarithm of the aquatic toxicity (y) from nine molecular descriptors ($x_1, \ldots, x_9$). The scatterplot matrix of the data (not included) shows a quite good linear dependence between y and x_1 except for some outliers. The other covariates have no clear "univariate" influence on y.

We applied the boosting methods with shrinkage factor $\nu = 0.3$ and used 5-fold cross-validation to stop the boosting iterations. The results are as follows: L_2-, RobLoss and RobRegM boosting select several times x_1 and x_3 at the beginning and then also some other covariates. The residual plots (not included) show no outliers. RobLossW, RobRegBI and RobCor boosting select only several times x_1 and then stop. The residual plots (not included) indicate some clear outliers, which are leverage points.

A closer look at the data shows that all the clear outliers are unsaturated carboxylic acids. RobLossW, RobRegBI and RobCor boosting lead to the insight that there is no linear model which fits all the data well. L_2-, RobLoss and RobRegM boosting find a second variable (x_3) that seems to explain also the unsaturated carboxylic acids, but this is doubtful.

4.6 Conclusions

We compared several robust boosting methods to L_2Boosting. For the ideal normal case, the robust methods are only slightly worse than L_2Boosting. In the contaminated case though, the robust methods outperform L_2Boosting by a large margin. An advantage of the boosting methods (for example over robust LARS) is that they don't have to compute covariance matrices of the covariates or to identify multivariate leverage points.

RobLoss, RobLossW and RobCor boosting are computationally efficient and hence well suited also for truly high dimensional problems. In the high dimensional setting, the differences between the methods are less pronounced, because the methods not only have to cope with outliers but also with high dimensional observations.

The additional computation of RobRegM boosting does not pay off. It has no advantages over RobLoss boosting. The case is different for RobRegBI boosting. It is better than RobLossW boosting in variable selection and the predictions are more accurate in high dimensions. The prediction and variable selection performance of RobCor boosting are surprisingly good in the contaminated case.

It seems quite natural that it is harder to stop the robust boosting methods than L_2Boosting in the normal case, since they use a robust measure of prediction error. In the leverage case though, the stopping works quite well for the robust methods and disastrously for L_2Boosting. The stopping works also well in the high dimensional setting, for which the boosting methods are mainly designed.

In practice, it is always a good advice to employ more than one method and to compare the results. Our robustified versions of L_2Boosting offer additional possibilities for good, advanced data analysis.

Chapter 5

LogitBoost with Trees Applied to the WCCI 2006 Performance Prediction Challenge Datasets

We apply LogitBoost with a tree-based learner to the five WCCI 2006 performance prediction challenge datasets. The number of iterations and the tree size is estimated by 10-fold cross-validation. We add a simple shrinkage strategy to make the algorithm more stable. The results are very promising since we won the challenge.

5.1 Introduction

In recent years a lot of new methods for classification have been proposed and therefore there is a need for a fair comparison. The WCCI 2006 performance prediction challenge (http://www.modelselect.inf.ethz.ch) offers a platform for that. The challenge consists of five binary classification problems. Each problem consists of a training set (input-output-pairs) to fit the model and a test set (inputs only) to predict on new data and evaluate the performance. The datasets are summarized in Table 5.1.

Dataset	Variables/ Features	Training Samples	Test Samples	Proportion of class +1
Ada	48	4562	41471	0.248
Gina	970	3468	31532	0.492
Hiva	1617	4229	38449	0.035
Nova	16969	1929	17537	0.285
Sylva	216	14394	130858	0.062

Table 5.1: *The five challenge datasets.*

The performance measure is the balanced error rate (BER) which is the average of the errors on each class on the test set. An additional task on the prediction challenge is to predict the BER (make a BER guess) which will be realized on the test set (generalisation BER). The final test score combines the BER and the guess error.

We decided to use LogitBoost (Friedman et al. 2000) in conjunction with trees. LogitBoost is a "statistical" version of Freund and Schapire's well known and successful AdaBoost (Freund and Schapire 1997) because it minimizes the negative binomial log-likelihood instead of the exponential loss. LogitBoost has already been applied successfully to high dimensional microarray data by Dettling and Bühlmann (Dettling and Bühlmann 2003).

The most popular choice for the base learner are trees. They can easily model different degrees of interaction (model complexities) and no variable transformations are needed.

5.2 Methods

The terminology we are going to use is $x_i \in \mathbb{R}^p$ for the input of the i-th observation/sample and $y_i \in \{-1, +1\}$ for the label/output of the i-th observation ($i = 1, \ldots, n$; n is the sample size). We recode the y_i to $y_i^* = (y_i + 1)/2 \in \{0, 1\}$. The LogitBoost algorithm works in the logistic framework. This means we have a predictor function $F : \mathbb{R}^p \to \mathbb{R}$ and conditional probabilities $p(x) = P[Y^* = 1|X = x]$, which are linked by

$$p(x) = \frac{\exp(F(x))}{1 + \exp(F(x))} \quad \text{and} \quad F(x) = \log\left(\frac{p(x)}{1 - p(x)}\right).$$

Since the BER is used as performance measure, the misclassification of a sample belonging to the smaller class (always +1) is punished harder than the misclassification of a sample belonging to the bigger class (−1). Therefore, it is a good advice to classify the doubtful observations as +1. The best cut off is the proportion of class +1 in the data. For Ada for example, the sample i is classified as +1, if $p(x_i) > 0.248$.

5.2.1 LogitBoost

The LogitBoost algorithm uses Newton steps for fitting a logistic model by maximum binomial likelihood. The algorithm works as follows:

1. Start with $F^{(0)}(x_i) = 0$ and $p(x_i) = \frac{1}{2}$, $i = 1, \ldots, n$.
2. Repeat for $m = 1, \ldots, M$:
   (a) Compute the weights and working response
   $$\begin{aligned} w_i &= p(x_i)(1 - p(x_i)), \\ z_i &= \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))}. \end{aligned}$$
   (b) Fit the function $f^{(m)}(x)$, using the tree-based learner, by a weighted least-squares regression of z_i to x_i using weights w_i.
   (c) Update $F^{(m)}(x_i) = F^{(m-1)}(x_i) + \nu f^{(m)}(x_i)$ and $p(x_i) = \exp(F^{(m)}(x_i))/(1 + \exp(F^{(m)}(x_i)))$.

An implementation protection is necessary: enforce thresholds on the weights and working responses:

$$\begin{aligned} w &= \max(w, 10^{-15}), \\ z &= \min(\max(z, -z_{max}), z_{max}). \end{aligned}$$

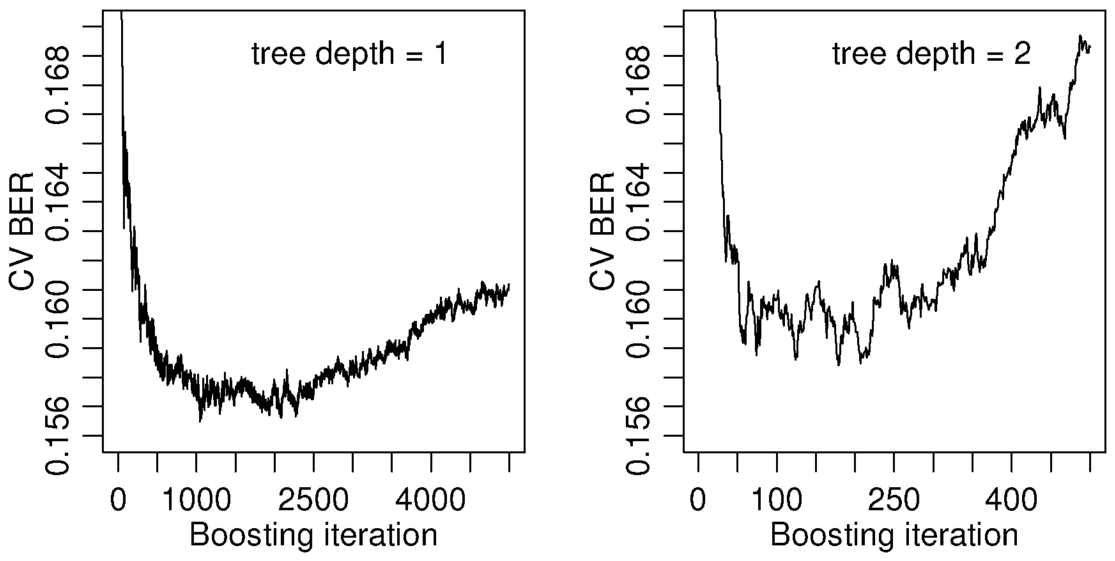

Figure 5.1: *CV BER as a function of the boosting iteration for Ada for $\nu = 0.3$ with trees of depth one (left) and depth two (right). Both curves show the usual behavior of first going down and then going up again because of over-fitting. Trees of depth one are the better choice because we reach a smaller CV BER.*

Friedman, Hastie and Tibshirani (Friedman et al. 2000) suggest to use values between 2 and 4 for z_{max}, but our experience is, that this is too small, especially for unbalanced data. So we use $z_{max} = 30$ for Hiva and $z_{max} = 10$ for the other datasets.

The number of iterations M is estimated by 10-fold cross-validation (CV). This means that the data is split into ten parts. One part is set aside and the algorithm is run for a large number of iterations on the other nine parts. At each iteration, we predict on the excluded part and calculate the BER. The procedure is repeated for each part and the ten resulting BER curves (as functions of the boosting iteration) are averaged to give the CV BER curve (see Figure 5.1). The stopping iteration M is the iteration which minimizes the CV BER. Actually, we did the cross-validation three times with different folds and averaged the results.

The learner (fitting method) in step 2b) is a regression tree of prefixed depth. The depth controls the degree of interaction of the fitted model. At each iteration the tree is grown without pruning to the prefixed depth and each split is allowed (even when only one observation remains in a node). We run the whole boosting algorithm five times. In each run we use another tree depth: in the first run, the learner is always a tree of depth one (stump); in the second run, the learner is always a tree of depth two (tree with four terminal nodes); and so on until depth five. The tree depth which gives the best CV BER curve is finally chosen. When different tree depths lead to approximately to the same CV BER, we take the smaller one to keep the model simple. Most of the time a tree depth of one or two is enough (see also Figure 5.1).

The ν in step 2c) is the so-called shrinkage factor. The natural value is one, but smaller values are often a better choice. This makes the algorithm slower, since more iterations are needed, but more stable, since the steps taken are smaller. Additionally, this often leads to some improvement of predictive power and only rarely to a deterioration. We suggest to start with $\nu = 1$ and look at the CV BER curve as a function of the boosting iteration (see Figure 5.2). If the curve is very rough around its minimum, or if the minimum occurs already after a few iterations, we reduce ν by a factor of approximately three and rerun the algorithm until the CV BER curve is smooth enough. We don't have a strict mathematical criterion for stopping and we choose ν by visual inspection. Most of the time a value of 0.3 or 0.1 is reasonable.

Our BER guess for the test set (generalisation BER) is the CV BER at the stopping iteration. On the one hand, this is a little bit too pessimistic, since the estimation of generalisation BER by CV is biased upward because only nine tenth of the data is used to fit the model. On the other hand, it is slightly too optimistic, since the stopping iteration is explicitly chosen to minimize the CV BER on the given training data. This effect however is most of the time small, since the CV BER curve is usually smooth (ensured by a small ν). Our hope is that the two effects cancel each other out.

5.2.2 Special Treatment of the Nova Dataset

Because the Nova dataset has a large number of variables and is sparse (each variable is binary and most of the values are zero), some preprocessing seems to be adequate. We drop the variables with only one or two entries equal to one (training and test set) and calculate the principal components with the remaining 16881 variables (with centered and

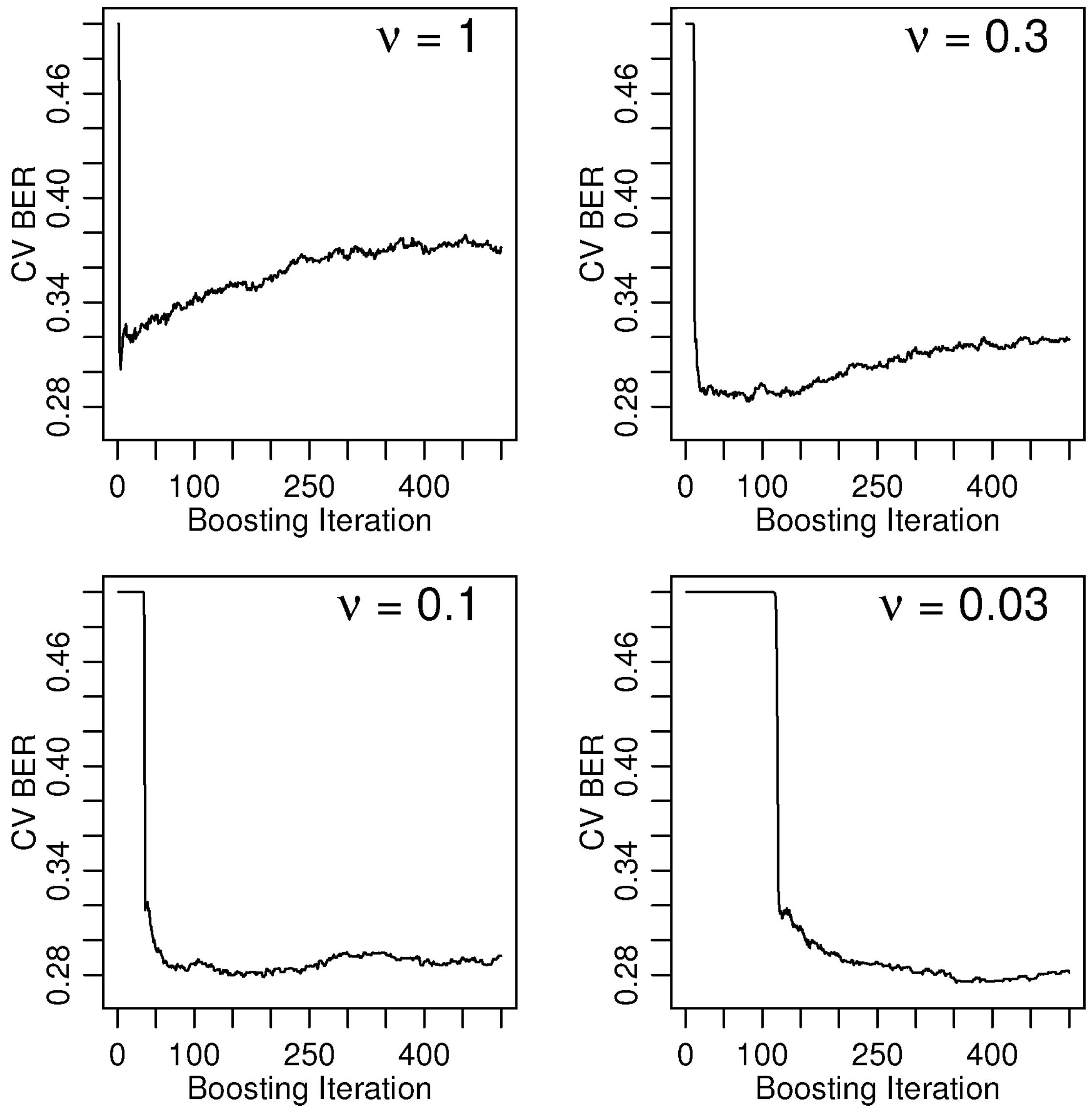

Figure 5.2: *CV BER as a function of the boosting iteration for Hiva with trees of depth two for $\nu = 1, 0.3, 0.1, 0.03$. For $\nu = 1$ the minimum occurs already after four iterations and the CV BER curve goes up again quickly. Smaller ν lead not only to smoother curves but also to smaller minima.*

scaled variables). It is important to use also the observations from the test set for the calculation. We take the first 400 principal components for LogitBoost.

5.2.3 Variable Pre-selection

To reduce the high dimensional datasets one could do a variable pre-selection with simple univariate tests. It is not clear whether this improves or worsens the performance of LogitBoost. At least it reduces the computation time a lot. We try both and apply LogitBoost with and without variable pre-selection. For the pre-selection we use the Wilcoxon test for continuous variables (the two groups are given by the labels y) and the Fisher exact test for the binary variables (with the 2×2 contingency table constructed with the x-variable and the labels y). Each variable with a p-value above 0.1 is dropped. This threshold is chosen ad hoc and reduces the number of variables (principal components for Nova) for Ada from 48 to 38, for Gina from 970 to 482, for Hiva from 1617 to 686, for Nova from 400 to 237 and for Sylva from 216 to 81.

5.3 Results

5.3.1 LogitBoost without and with Variable Pre-selection

Table 5.2 contains the results without and Table 5.3 with variable pre-selection. The outcomes are very similar. In the challenge, these two submissions ranked fourth and second. For Ada and Sylva, a main effects model performed best. For Hiva and Nova, first order interactions were needed and for Gina, quite complicated trees of depth five were fitted. This is no surprise since Gina was originally a ten class classification problem. The two classes for the challenge were constructed by combining five original classes at a time. For Ada, Gina and Sylva, a shrinkage factor ν of 0.3 seems to be small enough, while for the very unbalanced Hiva dataset a smaller value is needed.

Our BER guess method worked quite satisfactorily. On Ada, Hiva and Sylva we were too optimistic, while on Gina and Nova we were too pessimistic.

Dataset	Tree depth	ν	No. of iterations	CV BER = BER guess	BER on test set
Ada	1	0.3	1043	0.1565	0.1712
Gina	5	0.3	741	0.0415	0.0385
Hiva	2	0.03	353	0.2756	0.2888
Nova	2	0.1	294	0.0506	0.0491
Sylva	1	0.3	273	0.0058	0.0064
Average				0.1060	0.1108

Table 5.2: *Results without variable pre-selection. CV BER is the cross-validated BER on the training set at the stopping iteration and also our BER guess.*

Dataset	Tree depth	ν	No. of iterations	CV BER = BER guess	BER on test set
Ada	1	0.3	979	0.1550	0.1708
Gina	5	0.3	1308	0.0388	0.0357
Hiva	2	0.03	249	0.2795	0.2946
Nova	2	0.1	365	0.0503	0.0469
Sylva	1	0.3	229	0.0062	0.0066
Average				0.1060	0.1109

Table 5.3: *Results with variable pre-selection. CV BER is the cross–validated BER on the training set at the stopping iteration and also our BER guess.*

5.3.2 A Mix: Predicted Probabilities Averaged

We made a third submission for which we averaged the predicted probabilities of LogitBoost with and without variable pre-selection (with acronym “LB tree mix”). Our BER guess for each data set is the minimum of the two individual BER guesses. The results are contained in Table 5.4. For Ada and Nova, the BER on the test set of the mix was better than the BER of the two individual methods. For the other data sets, the BER of the mix was always nearer to the BER of the better individual method. All in all, this gave a small improvement. In the challenge, this submission was fifth.

Dataset	BER guess	BER on test set	Guess error	Test score	Rank
Ada	0.1550	0.1705	0.0155	0.1859	7
Gina	0.0388	0.0361	0.0027	0.0386	5
Hiva	0.2756	0.2904	0.0148	0.3035	9
Nova	0.0503	0.0467	0.0036	0.0498	5
Sylva	0.0058	0.0064	0.0006	0.0070	15
Average	0.1051	0.1100	0.0074	0.1170	8.2

Table 5.4: *Results for the mix (averaged probabilities of LogitBoost with and without variable pre-selection).*

5.3.3 Intercept Adaptation

For our forth and challenge winning submission we made a last small adjustment by adapting the intercept on the logit scale (the acronym “LB tree mix cut adapted” is kind of misleading). We added the same constant to all $F(x_i)$ of the mixed submission so that the average of the resulting probabilities is exactly the proportion of class +1 in the data. This is a kind of bias correction and improved the BER of four data sets and had no effect for Gina. The results are given in Table 5.5. For the BER guess we took the same values as for the mixed submission (except for Hiva where we hoped to be better and reduced the BER guess from 0.2756 to 0.27).

Dataset	BER guess	BER on test set	Guess error	Test score	Rank
Ada	0.1550	0.1696	0.0146	0.1843	3
Gina	0.0388	0.0361	0.0027	0.0386	5
Hiva	0.2700	0.2871	0.0171	0.3029	8
Nova	0.0503	0.0458	0.0045	0.0499	8
Sylva	0.0058	0.0063	0.0005	0.0067	7
Average	0.1040	0.1090	0.0079	0.1165	6.2

Table 5.5: *Results with “intercept adaptation”.*

5.4 Conclusions

We have shown by winning the challenge that LogitBoost with trees can perform very well on high-dimensional classification problems.

There are three tuning parameters, which have to be chosen: the shrinkage factor ν, the tree depth and the number of iterations M.

Our choice of the shrinkage factor ν is somewhat arbitrary. We chose it by visual inspection of the CV BER curve. A simple alternative would be to take always $\nu = 0.1$ as a good default. Smaller values are seldom needed and $\nu = 1$ should not be used. With a small ν we can ensure that enough iterations are performed and that all the relevant variables have been chosen during the iteration. The selection of the number of iterations M then becomes easier, too.

The tree depth and the number of iterations M can simply be chosen by CV.

Bibliography

Bedrick, E. J. and Tsai, C.-L. (1994). Model selection for multivariate regression in small samples, *Biometrics* **50**: 226–231.

Box, G. and Youle, P. (1955). The exploration of response surfaces: an example of the link between the fitted surface and the basic mechanism of the system, *Biometrics* **11**: 287–323.

Breiman, L. (1998). Arcing classifier (Pkg: p801-849), *The Annals of Statistics* **26**(3): 801–824.

Breiman, L. (1999). Prediction games and arcing algorithms, *Neural Computation* **11**: 1493–1517.

Breiman, L. and Friedman, J. H. (1985). Estimating optimal transformations for multiple regression and correlation (C/R: p598-619), *Journal of the American Statistical Association* **80**: 580–598.

Breiman, L. and Friedman, J. H. (1997). Predicting multivariate responses in multiple linear regression (Disc: p37-54), *Journal of the Royal Statistical Society, Series B, Methodological* **59**: 3–37.

Brooks, R. and Stone, M. (1994). Joint continuum regression for multiple predictands, *Journal of the American Statistical Association* **89**: 1374–1377.

Bühlmann, P. (2006). Boosting for high-dimensional linear models, *Annals of Statistics* **34**: 559–583.

Bühlmann, P. and Hothorn, T. (2006). Boosting: a statistical perspective, *Technical Report 136*, Seminar für Statistik ETH Zürich.

Bühlmann, P. and Lutz, R. W. (2006). *Frontiers in Statistics*, Imperial College Press, chapter Boosting Algorithms: with an Application to Bootstrapping Multivariate Time Series.

Bühlmann, P. and Yu, B. (2000). Comment on "Additive logistic regression: A statistical view of boosting" (Pkg: p337-407), *The Annals of Statistics* **28**(2): 377–386.

Bühlmann, P. and Yu, B. (2003). Boosting with the L2-loss: Regression and classification, *Journal of the American Statistical Association* **98**: 324–339.

Bühlmann, P. and Yu, B. (2005). Boosting, model selection, lasso and nonnegative garrote, *Technical Report 127*, Seminar für Statistik ETH Zürich.

Dahlhaus, R. and Eichler, M. (2003). Causality and graphical models for time series, *in* P. Green, N. Hjort and S. Richardson (eds), Highly structured stochastic systems, Oxford University Press.

Dettling, M. and Bühlmann, P. (2003). Boosting for tumor classification with gene expression data, *Bioinformatics* **19**: 1061–1069.

Duffy, N. and Helmbold, D. (2000). Leaveraging for regression, *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory.*

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. (2004). Least angle regression, *Annals of Statistics* **32(2)**: 407–451.

Freund, Y. (1995). Boosting a weak learning algorithm by majority, *Information and Computation* **121**: 256–285.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of online learning and an application to boosting, *J. Comput. System Sciences* **55**: 119–139.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm, *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine., *The Annals of Statistics* **29**(5): 1189–1232.

Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (Pkg: p337-407), *The Annals of Statistics* **28**(2): 337–373.

Golub, T., Slonim, D., Tamayo, P., Huard, C., Gassenbeek, M., Mesirov, J., Coller, H., Loh, M., Downing, J., Caligiuri, M., Bloomfield, C., and Lander, E. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science* **286**: 531–537.

Hampel, F., Ronchetti, E., Rousseeuw, P. and Stahel, W. (1986). *Robust Statistics: The Approach Based on Influence Functions*, Wiley Series in Probability and Math. Statistics, Wiley, New York.

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics, Springer-Verlag, New York.

Huber, P. J. (1964). Robust estimation of a location parameter, *Annals of mathematical statistics* **35**: 73–101.

Huber, P. J. (1981). *Robust Statistics*, Wiley, N. Y.

Hurvich, C. M. and Tsai, C.-L. (1989). Regression and time series model selection in small samples, *Biometrika* **76**: 297–307.

Klein, L., Ball, R., Hazlewood, A. and Vandome, P. (1961). *An economic model of the United Kingdom*, Oxford: Blackwell.

Kriegel, H.-P., Kroger, P., Pryakhin, A. and Schubert, M. (2004). Using support vector machines for classifying large sets of multi-represented objects, *Proc. 4th SIAM Int. Conf. on Data Mining*, pp. 102–114.

Lütkepohl, H. (1993). *Introduction to Multiple Time Series Analysis*, 2nd edn, Springer-Verlag.

Lutz, R. and Bühlmann, P. (2006a). Robustified L_2Boosting, *Technical Report 139*, Seminar für Statistik ETH Zürich.

Lutz, R. W. (2006). Logitboost with trees applied to the WCCI 2006 performance prediction challenge datasets, *Proceedings of the IJCNN 2006*.

Lutz, R. W. and Bühlmann, P. (2006b). Boosting for high-multivariate responses in high-dimensional linear regression, *Statistica Sinica* **16(2)**: 471–494.

Lutz, R. W. and Bühlmann, P. (2006c). Conjugate direction boosting, *Journal of Computational and Graphical Statistics* **15(2)**: 287–311.

Maguna, F. P., Núñez, M. B., Okulik, N. B. and Castro, E. A. (2003). Improved QSAR analysis of the toxicity of aliphatic carboxylic acids, *Russian Journal of General Chemistry* **73**: 1792–1798.

Mallat, S. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries, *IEEE Transactions on Signal Processing* **41(12)**: 3397–3415.

Maronna, R. A., Martin, R. D. and Yohai, V. J. (2006). *Robust Statistics, Theory and Methods*, Wiley Series in Probility and Statistics, John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England.

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging, *in* S. Mendelson and A. Smola (eds), *Advanced Lectures on Machine Learning*, Springer, pp. 119–184.

Miller, A. (2002). *Subset selection in regression*, Chapman & Hall.

Morgenthaler, S., Welsch, R. E. and Zenide, A. (2003). *Theory and Applications of Recent Robust Methods*, Birkhäuser, chapter Algorithms for Robust Model Selection in Linear Regression.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*, Springer Series in Operations Research, Springer-Verlag, NY.

Phatak, A. and de Hoog, F. (2002). Exploiting the connection between pls, lanczos methods and conjugate gradients: alternative proofs of some properties of pls, *Journal of Chemometrics* **16**: 361–367.

Rätsch, G., Demiriz, A. and Bennett, K. (2002). Sparse regression ensembles in infinite and finite hypothesis spaces, *Machine Learning* **48**: 193–221.

Rätsch, G., Onoda, T. and Müller, K.-R. (2001). Soft margins for AdaBoost, *Machine Learning* **42**: 287–320.

Reinsel, G. C. (1993). *Elements of Multivariate Time Series Analysis*, Springer-Verlag, N. Y.

Reinsel, G. C. and Velu, R. P. (1998). *Multivariate reduced-rank regression: theory and applications*, Springer-Verlag Inc.

Rencher, A. C. (2002). *Methods of multivariate analysis*, John Wiley & Sons.

Ronchetti, E. and Staudte, R. G. (1994). A robust version of Mallow's c_p, *Journal of the American Statistical Association* **89**: 550–559.

Ronchetti, E., Field, C. and Blanchard, W. (1997). Robust linear model selection by cross-validation, *Journal of the American Statistical Association* **92**: 1017–1023.

Rousseeuw, P. J. and Croux, C. (1993). Alternatives to the median absolute deviation, *Journal of the American Statistical Association* **88**(424): 1273–1283.

Schapire, R. E. (1990). The strength of weak learnability, *Machine Learning* **5**: 197–227.

Seber, G. A. F. (1984). *Multivariate Observations*, Wiley Series in Probability and Mathematical Statistics, Wiley, New York.

Skagerberg, B., MacGregor, J. and Kiparissides, C. (1992). Multivariate data analysis applied to low-density polyethylene reactors, *Chemometr. Intell. Lab. Syst.* **14**: 341–356.

Temlyakov, V. (2000). Weak greedy algorithms, *Advances in Computational Mathematics* **12**: 213–227.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society, Series B, Methodological* **58**: 267–288.

Timm, N. H. (2002). *Applied Multivariate Analysis*, Springer-Verlag, N. Y.

Van Aelst, S., Khan, J. A. and Zamar, R. H. (2004). Robust linear model selection based on least angle regression. *http://hajek.stat.ubc.ca/~ruben/website/cv/RobLARS.pdf*

Wille, A., Zimmermann, P., Vranova, E., Fürholz, A., Laule, O., Bleuler, S., Hennig, L., Prelic, A., von Rohr, P., Thiele, L., Zitzler, E., Gruissem, W. and Bühlmann, P. (2004). Sparse graphical gaussian modeling of the isoprenoid gene network in arabidopsis thaliana, *Genome Biology* **5(11) R92**: 1–13.

Yokoyama, R. (1980). Moment bounds for stationary mixing sequences, *Z. Wahrscheinlichkeitstheorie verw. Gebiete* **52**: 45–75.

Zellner, A. (1962). An efficient method of estimating seemingly unrelated regressions and tests for aggregation bias, *Journal of the American Statistical Association* **57**: 348–368.

Zellner, A. (1963). Estimators for seemingly unrelated regression equations: Some exact finite sample results (corr: V67 p255), *Journal of the American Statistical Association* **58**: 977–992.

Zemel, R. and Pitassi, T. (2001). A gradient-based boosting algorithm for regression problems, *NIPS-13: Advances in Neural Information Processing Systems, 13*, MIT Press, Cambridge, MA.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net, *Journal of the Royal Statistical Society, Series B, Statistical Methodology* **67**: 301–320.

Curriculum Vitae

I was born on July 1, 1976 in Zurich. After attending primary school from 1983 to 1989 in Ebmatingen (ZH) and Wallisellen (ZH), I spent five years at the Kantonsschule Zurich Oerlikon until 1994. Then I moved to Germany and obtained the Abitur at the Siebold Gymnasium in Würzburg in 1996.

I started my studies in Mathematics at ETH Zurich in October 1996 and graduated in March 2001 with a diploma thesis on "Vertrauensintervalle bei Dosis-Wirkungs-Modellen".

In June 2001, I joined the statistical consulting group at the Seminar für Statistik (SfS), ETH Zurich. In October 2002 I enrolled as a PhD student and worked since as a teaching assistant at the SfS.