Advanced Methods for Regression and Classification

Lecture Notes

Prof. Dr. Peter Filzmoser

Peter.Filzmoser@tuwien.ac.at

Institute of Statistics and Mathematical Methods in Economics

TU Wien, Austria

Vienna, October 2022

Preface

The field of statistics has drastically changed since the introduction of the computer. Computational statistics is nowadays a very popular field with many new developments of statistical methods and algorithms, and many interesting applications. One challenging problem is the increasing size and complexity of data sets. Not only for saving and filtering such data, but also for analyzing huge data sets new technologies and methods had to be developed.

This manuscript is concerned with linear and nonlinear methods for regression and classification. In the first chapters "classical" methods like least squares regression and discriminant analysis are treated. More advanced methods like "generalized additive models", tree-based methods and nearest neighbor methods follow.

Each chapter introducing a new method is followed by a chapter with examples from practice and solutions with R. The results of different methods are compared in order to get an idea of the performance of the methods.

The manuscript is essentially based on the book "The Elements of Statistical Learning", Hastie et al. 2001.

 The R-logo is used throughout the course notes, and it refers to demonstrations and examples in R and gives the section and page where they can be found.

Contents

Part I

Fundamentals

Chapter 1

The linear regression model

The aim of a regression model is to describe the output variable y in terms of one or more input variables $x_1, x_2, \ldots, x_p$. The output variable is often called "response" or "dependent variable", and the inputs are called "predictors" or "independent variables".

In general, a regression model has the form

$$y = f(x_1, x_2, \ldots, x_p) + \varepsilon,$$

where f represents a linear or even non-linear function of the inputs. The error term ε reveals that in general it will be impossible to find an exact relationship between $f(x_1, x_2, \ldots, x_p)$ and y, but only an approximate one, thus

$$y \approx f(x_1, x_2, \ldots, x_p).$$

A very special and important case is the linear regression model, where we consider linear combinations of the input variables of the form

$$f(x_1, x_2, \ldots, x_p) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j.$$

"Linear combination" means that the input variables get first weighed with constants and are then summarized. The result should explain the output variable as good as possible, which leads to the **linear regression model**

$$y = \beta_0 + x_1\beta_1 + \ldots x_p\beta_p + \varepsilon. \tag{1.1}$$

The terms β_j are unknown parameters or coefficients, which will be estimated from given data. The variables x_j can come from different sources:

- quantitative inputs, for example the height of different people
- transformations of quantitative inputs, such as log, square-root, or square
- basis-expansions, such as $x_2 = x_1^2,\ x_3 = x_1^3$, leading to a polynomial representation
- numeric or "dummy" coding of the levels of qualitative inputs
- interactions between variables, for example $x_3 = x_1 \cdot x_2$

No matter the source of the x_j, the model is linear in the parameters.

Typically we estimate the parameters β_j from a set of training data of the following form

$$\begin{pmatrix} y_1 & x_{11} & x_{12} & \cdots & x_{1p} \\ y_2 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & & & & \\ y_i & x_{i1} & x_{i2} & \cdots & x_{ip} \\ \vdots & & & & \\ y_n & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} = \begin{pmatrix} y_1, & \boldsymbol{x}_1 \\ y_2, & \boldsymbol{x}_2 \\ & \vdots \\ y_i, & \boldsymbol{x}_i \\ & \vdots \\ y_n, & \boldsymbol{x}_n \end{pmatrix} \tag{1.2}$$

Each $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is a feature measurement for the i-th case and y_i is the value of the i-th observation. The estimated parameters are denoted by $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ and by inserting those values in the linear model (1.1) for each observation one gets:

$$\hat{y}_i = \hat{f}(x_{i1}, x_{i2}, \ldots, x_{ip}) = \hat{\beta}_0 + \sum_{j=1}^{p} x_{ij}\hat{\beta}_j$$

The fitted values $\hat{y}_i$ should be as close as possible to the real measured values y_i. The differences $y_i - \hat{y}_i$ are called **residuals**. The definition of "as close as possible" as well as an estimation of how good the model actually is, will be given in the following.

1.1 Least Squares (LS) regression

With the notation above we can formulate the linear regression model for every observation $i = 1, \ldots, n$, which is

$$y_i = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j + \varepsilon_i$$

with the i-th error term ε_i. A request often made is the independence of the error terms from each other and normal distribution with expectation 0 and constant variance σ^2, thus $\varepsilon_i \sim N(0, \sigma^2)$. Note that in this case the ε_i and thus y_i are treated as random variables, whereas the x_{ij} are seen as fixed values. The assumption on the error term is important once we want to do "inference", i.e. construct confidence intervals for the regression parameters and perform statistical hypothesis tests. For the purpose of estimating the regression parameters we do not rely on distributional assumptions.

1.1.1 Parameter estimation

The multiple linear regression model can conveniently be written in matrix notation,

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

with the n values $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$ of the response, the "design matrix"

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & & & & \\ 1 & x_{i1} & x_{i2} & \cdots & x_{ip} \\ \vdots & & & & \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} = \begin{pmatrix} 1, & \boldsymbol{x}_1 \\ 1, & \boldsymbol{x}_2 \\ & \vdots \\ 1, & \boldsymbol{x}_i \\ & \vdots \\ 1, & \boldsymbol{x}_n \end{pmatrix}$$

and the error terms $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)^\top$.

Our aim is to estimate the regression coefficients $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\top$. The most popular estimation method is least squares, in which we choose the coefficients which minimize the residual sum of squares (RSS)

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 .$$

This approach makes no assumptions about the validity of the model, it simply finds the best linear fit to the data. The least squares estimator turns out to be

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{1.3}$$

Proof:
Let $\boldsymbol{X}$ denote a $(n \times (p+1))$-matrix which each row being an input vector (with a 1 in the first position). Similarly, let $\boldsymbol{y}$ be the n-vector of outputs in the training data set. Then we can write the residual sum of squares as

$$\begin{aligned} \mathrm{RSS}(\boldsymbol{\beta}) &= (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \\ &= \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 \end{aligned}$$

with the Euclidean norm $\|\cdot\|$. This is a quadratic function in the $p+1$ parameters. Since we are looking for the smallest possible value of RSS, we have a classical minimization problem. Differentiating with respect to $\boldsymbol{\beta}$ yields

$$\begin{aligned} \frac{\partial \mathrm{RSS}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \begin{pmatrix} \frac{\partial}{\partial \beta_0}\mathrm{RSS}(\boldsymbol{\beta}) \\ \frac{\partial}{\partial \beta_1}\mathrm{RSS}(\boldsymbol{\beta}) \\ \vdots \\ \frac{\partial}{\partial \beta_p}\mathrm{RSS}(\boldsymbol{\beta}) \end{pmatrix} \\ &= -2\boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \\ \frac{\partial^2 \mathrm{RSS}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} &= 2\boldsymbol{X}^\top \boldsymbol{X} \end{aligned}$$

Assuming (for the moment) that $\boldsymbol{X}$ is nonsingular (thus there are at least as many observations as parameters and the observations do not lie in a subspace of lower dimension), and hence $\boldsymbol{X}^\top \boldsymbol{X}$ is positive definite (thus invertible) the solution is a minimum. By setting the first derivative to zero we get

$$\boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = \boldsymbol{0} \tag{1.4}$$

and then obtain the normal equations

$$(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta}) = \boldsymbol{X}^\top \boldsymbol{y},$$

and their unique solution $\hat{\boldsymbol{\beta}}$

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}.$$

□

Now the estimated regression parameters $\hat{\boldsymbol{\beta}}$ that provide the best fit in the sense of minimizing RSS can be used for the prediction. If we use the available data $\boldsymbol{x}_i$ as an input (an not new x-information), we talk about the fitted or estimated value, which is given by

$$\hat{y}_i = \hat{f}(\boldsymbol{x}_i) = (1, \boldsymbol{x}_i)\hat{\boldsymbol{\beta}},$$

and it can be compared to y_i. This can be done for each of the n observations which leads to

$$\begin{aligned} \hat{\boldsymbol{y}} &= \boldsymbol{X}\hat{\boldsymbol{\beta}} \\ &= \underbrace{\boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top}_{\text{Hat matrix } H} \boldsymbol{y} \end{aligned}$$

The matrix $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top$ is called the "hat matrix" because it puts the hat on $\boldsymbol{y}$.

It might happen that the columns of $\boldsymbol{X}$ are not linearly independent (for example, if two input variables perfectly correlate), so that $\boldsymbol{X}$ is not of full rank. Then $\boldsymbol{X}^\top\boldsymbol{X}$ is singular and the least squares coefficients $\hat{\boldsymbol{\beta}}$ are not uniquely defined. This case appears most often when one or more qualitative inputs are coded in a redundant fashion (recoding mostly eliminates the correlation) or in signal and image analysis, where the number of inputs p can exceed the number of training cases n (the features are typically reduced by filtering or regularization). If this situation happens, we have to use different methods for estimating the regression parameters, and these will be discuss in the following chapters.

Section 4.1.1, page 27

1.1.2 Tests and confidence intervals

This now refers to "statistical inference", which we now develop for the LS estimator. For this purpose we need to see the error terms $\boldsymbol{\varepsilon}$ (and thus also the responses $\boldsymbol{y}$) as random variables.

Theorem 1.1.2.1 (Gauss-Markov Theorem) *Under the model assumption*

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim N_n(\boldsymbol{0}, \sigma^2\boldsymbol{I})$$

and the condition that the input variables are fixed, the LS estimator $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$ is the best, linear, unbiased estimator (BLUE). Unbiased means that $I\!E(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$. "Best" means that among all linear unbiased estimators, the LS estimator has the smallest sampling variance, thus it is the most accurate one in this class of estimators.

Proof: see, e.g., Schönfeld 1969

Based on the above assumptions one can also derive the distribution of the LS estimator as $\hat{\boldsymbol{\beta}} \sim N_{p+1}(\boldsymbol{\beta}, \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1})$. We have furthermore:

- $(n-p-1)\hat{\sigma}^2 \sim \sigma^2\chi^2_{n-p-1}$, where $\hat{\sigma}^2 = \frac{1}{n-p-1}(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}})^\top(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}})$ is an unbiased estimator for σ^2;
- $\hat{\boldsymbol{\beta}}$ and $\hat{\sigma}^2$ are statistically independent.

Proof see e.g. Schönfeld 1969

Based on these distributional properties we can form tests and confidence intervals for the parameters β_j.
To test the hypothesis $H_0 : \beta_j = 0$, $H_1 : \beta_j \neq 0$, we form

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{d_j}},$$

where d_j is the jth diagonal element of $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$. Under the null hypothesis, z_j is $z_j \sim t_{n-p-1}$, and hence a large absolute value of z_j will lead to the rejection of the hypothesis.

If σ would be known, z_j would follow a standard normal distribution:

$$\begin{aligned} z_j &= \frac{\hat{\beta}_j}{\sigma\sqrt{d_j}} \\ z_j &\sim N(0,1) \end{aligned}$$

The difference between the tail quantities of the t-distribution and the standard normal becomes negligible as the sample size increases and so typically the normal quantiles are used. It can also be shown that the z_j match the F-statistic described in Chapter 2 for the comparison of models if only one parameter is tested to be zero. For large n the quantiles of $F_{p_1-p_0,n-p_1-1}$ approach those of the $\chi^2_{p_1-p_0}$.

The test for $\beta_j = 0$ can be used to obtain a confidence interval i.e. for a test at level $\alpha = 0.05$ the critical values are $z_{\alpha/2}, z_{1-\alpha/2} = 1.96$ (if σ is known). The $1-\alpha$ confidence interval for β_j is:

$$[\hat{\beta}_j - z_{1-\alpha/2} \underbrace{\sqrt{d_j}\hat{\sigma}}_{\mathrm{SD}(\hat{\beta}_j)}, \hat{\beta}_j + z_{1-\alpha/2}\sqrt{d_j}\hat{\sigma}]$$

An approximative 95%-confidence interval is:

$$\hat{\beta}_j \pm 2 \cdot \mathrm{SD}(\hat{\beta}_j)$$

Even if the Gaussian error assumption does not hold, this interval will be approximately correct, with its coverage approaching $(1-\alpha)$ for $n \to \infty$.

Section 4.1.2, page 28

1.1.3 Decomposition of the variance of y

Idea: The deviation of the observed value around the mean is decomposed in an explained (captured by the regression) component $(\hat{y}_i - \overline{y})$ and an unexplained component $(y_i - \hat{y}_i)$. We have: TSS = RegSS + RSS with

- Total Sum of Squares: $\mathrm{TSS} = \sum_{i=1}^{n}(y_i - \overline{y})^2$
- Residual Sum of Squares: $\mathrm{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Regression Sum of Squares: $\mathrm{RegSS} = \sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2$

Proof:
With $y_i - \overline{y} = (y_i - \hat{y}_i) + (\hat{y}_i - \overline{y})$, the total variation (TSS) of $\boldsymbol{y}$ can be decomposed as follows:

$$\begin{aligned}
\mathrm{TSS} &= \sum_{i=1}^{n}(y_i - \overline{y})^2 \\
&= \sum_{i=1}^{n}((y_i - \hat{y}_i) + (\hat{y}_i - \overline{y}))^2 \\
&= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2 + 2\sum_{i=1}^{n}(y_i - \hat{y}_i)(\hat{y}_i - \overline{y})
\end{aligned}$$

where

$$\begin{aligned}
\sum_{i=1}^{n}(y_i - \hat{y}_i)(\hat{y}_i - \overline{y}) &= \sum_{i=1}^{n}(y_i\hat{y}_i - y_i\overline{y} - \hat{y}_i\hat{y}_i + \hat{y}_i\overline{y}) \\
&= \sum_{i=1}^{n} y_i\hat{y}_i - \overline{y}\underbrace{\sum_{i=1}^{n} y_i}_{n\overline{y}} - \sum_{i=1}^{n}\hat{y}_i^2 + \overline{y}\underbrace{\sum_{i=1}^{n}\hat{y}_i}_{n\overline{y}} \\
&= \sum_{i=1}^{n} y_i\hat{y}_i - \sum_{i=1}^{n}\hat{y}_i^2 \\
&= \sum_{i=1}^{n}\hat{y}_i(y_i - \hat{y}_i) \\
&= \sum_{i=1}^{n}\hat{y}_i\hat{\varepsilon}_i = 0
\end{aligned}$$

□

With this decomposition, we get the F-statistic of Equation (2.1). The "Analysis of Variance (ANOVA)", which is used to compare nested models, is based on the F-statistic.

In doing so, the hypothesis $H_0 : \beta_1 = \cdots = \beta_p = 0$ is tested against $H_1 : \beta_j \neq 0$ for any $j = 1, \ldots, p$. Under H_0, the regression would only give noise. The ANOVA table then is:

	DF	MeanSS
RegSS	p	RegSS/p
RSS	$n-p-1$	$\text{RSS}/(n-p-1)$

with

$$F_0 = \frac{\text{RegSS}/p}{\text{RSS}/(n-p-1)} = \frac{n-p-1}{p}\frac{R^2}{1-R^2} \sim F_{p,n-p-1},$$

where the above distributional assumptions of the independent residuals have to be met.


Section 4.2.1, page 29

Chapter 2

Comparison of models and model selection

One could be interested in reducing the number of input variables used to explain the output variable since this could simplify the model, making it easier to understand. In addition, the measuring variables is often expensive, a smaller model would therefore be cheaper. This means that we need a tool to compare different models.

2.1 Test for several coefficients to be zero

Let us assume that we have two models of different size

$$\begin{aligned} M_0 &: y = \beta_0 x_0 + \beta_1 x_1 + \cdots + \beta_{p_0} x_{p_0} + \varepsilon \\ M_1 &: y = \beta_0 x_0 + \beta_1 x_1 + \cdots\cdots + \beta_{p_1} x_{p_1} + \varepsilon \end{aligned}$$

with $p_0 < p_1$. Here, p_1 could be equal to p, which means that model M_1 is the full model. By simultaneously testing several coefficients to be zero we want to find out if the additional variables $x_{p_0+1}, \ldots, x_{p_1}$ in model M_1 provide a significant explanation gain to the smaller model M_0. If, for instance, we would like to find out if a categorical variable with k levels can be excluded from the model, one has to test if all dummy-variables used to represent those k levels can be set to zero.
Rephrasing we get: H_0 : "the small model is true", meaning that the model M_0 is acceptable. Basis for this test is the residual sum of squares

$$\mathrm{RSS}_0 = \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \sum_{j=1}^{p_0} \hat{\beta}_j x_{ij} \right)^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

for model M_0 and respectively

$$\mathrm{RSS}_1 = \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \sum_{j=1}^{p_1} \hat{\beta}_j x_{ij} \right)^2$$

for model M_1. This leads to the following test statistic

$$F = \frac{\frac{\mathrm{RSS}_0 - \mathrm{RSS}_1}{p_1 - p_0}}{\frac{\mathrm{RSS}_1}{n - p_1 - 1}} \tag{2.1}$$

The F-statistic measures the change in residual sum of squares per additional parameter in the bigger model, and it is normalized by an estimate of σ^2. Under Gaussian assumptions $\varepsilon_i \sim N(0, \sigma^2)$ and the null hypothesis that the smaller model is correct, the F statistic will be distributed according to

$$F \sim F_{p_1-p_0, n-p_1-1}.$$

A value of the F statistic bigger than the $(1-\alpha)$-quantile $F_{p_1-p_0,n-p_1-1;1-\alpha}$, e.g. for $\alpha = 0.05$, results in a rejection of H_0.

For a better understanding of this test we can consider the sum-of-squares for the models M_0 and M_1:

$$\begin{aligned} \text{TSS} &= \text{RegSS}_0 + \text{RSS}_0 \\ &= \text{RegSS}_1 + \text{RSS}_1 \end{aligned}$$

which leads to

$$\begin{aligned} \text{RegSS}_1 - \text{RegSS}_0 &= \text{RSS}_0 - \text{RSS}_1 \\ &\geq 0 \quad \text{because } p_0 < p_1 \end{aligned}$$

If $\text{RSS}_0 - \text{RSS}_1$ is large, M_1 explains the data significantly better.

Attention: A test for $H_0 : \beta_0 = \beta_1 = \cdots = \beta_p = 0$ might not give the same result as single tests for $H_0 : \beta_j = 0$ with $j = 0, 1, \ldots, p$ - especially if the x-variables are highly correlated.

Section 4.1.2, page 28

2.2 Explained variance

The *multiple R-Square* (coefficient of determination) describes the amount of variance that is explained by the model

$$\begin{aligned} R^2 &= 1 - \frac{\text{RSS}}{\text{TSS}} = \frac{\text{RegSS}}{\text{TSS}} \\ &= 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \overline{y})^2} \\ &= \text{Cor}^2(y, \hat{y}) \in [0, 1] \end{aligned}$$

The closer R-Square is to 1, the better the fit of the regression. If the model has no intercept, $\overline{y} = 0$ is chosen.
Since the denominator remains constant, R-Square grows with the size of the model. In general we are not interested in the model with the maximum R-Square. Rather, we would like to select that model which leads only to a marginal increase in R-Square with the addition of new variables.

The *adjusted R-Square*, a reduced R-Square value, prevents the effect of getting a bigger R-Square even though the fit gets worse, by including the degrees of freedom in the calculation [see, for example, Kutner and Nachtsheim, 2004]

$$\tilde{R}^2 = 1 - \frac{\text{RSS}/(n-p-1)}{\text{TSS}/(n-1)}$$

Section 4.1.2, page 28

2.3 Information criteria

In the following we discuss the use of information criteria for model selection. The most prominent criteria are AIC (Akaike information criterion) and BIC (Bayesian information criterion). Both are applicable in settings where the model fitting is carried out by a maximization of a log-likelihood function. AIC as well as BIC include a penalty term for the number of parameters used in the model. Generally, we should then select that model which gives the smallest value of AIC (or BIC) over the set of models considered.

2.3.1 Akaike's information criterion (AIC)

The criterion of Akaike is based on the Kullback-Leibler (K-L) information $I(f, g)$ [vgl. Bozdogan, 2000]. K-L describes the loss of information when occurs when approximating a precise probability distribution $f(x)$ by a probability distribution $g(x \mid \theta)$. The K-L information is defined as:

$$I(f, g) = \int f(x) \log \left(\frac{f(x)}{g(x \mid \theta)} \right) dx \tag{2.2}$$

Alternatively, the K-L information can be interpreted as a distance between the "truth" and a model. For $I(f, g)$ we have:

- $I(f, g) > 0$ if $f(x) \neq g(x \mid \theta)$
- $I(f, g) = 0$ if $f(x) = g(x \mid \theta)$ almost everywhere

The best model is thus losing the least information compared to all other models. This is equivalent to minimizing $I(f, g)$ over all g. Since neither f nor θ are known, the K-L information cannot be used in its original version. Thus, the expected rather than the original K-L information is minimized. (2.2) can also be written as:

$$\begin{aligned} I(f, g) &= \int f(x) \log(f(x)) dx - \int f(x) \log(g(x \mid \theta)) dx \qquad \text{or} \\ I(f, g) &= \mathbb{E}_f[\log(f(x))] - \mathbb{E}_f[\log(g(x \mid \theta))] \end{aligned}$$

Since f is a fixed function, $\mathbb{E}_f[\log(f(x))]$ can be considered as a constant C. Then it follows:

$$I(f, g) = C - \mathbb{E}_f[\log(g(x|\theta))]$$

Thus, only $\mathbb{E}_f[\log(g(x \mid \theta))]$ needs to be estimated as a performance criterion for a model. Akaike has shown [vgl. Burnham and Anderson, 2004] that this is equivalent to an estimation of

$$\mathbb{E}_y \mathbb{E}_x[\log(g(x \mid \hat{\theta}(y)))]. \tag{2.3}$$

Here, x and y are independent samples of the same distribution, and $\hat{\theta}$ refers to the maximum likelihood estimation of θ based on a model g and data y. Akaike found a formal relation between the Kullback-Leibler information and the likelihood theory. According to that, the maximized log-likelihood $\log(L(\hat{\theta}|y))$ is a biased estimator of $\mathbb{E}_y \mathbb{E}_x[\log(g(x|\hat{\theta}(y)))]$, where the bias is approximatively p, the number of parameters used in the estimated model. $\log(L(\hat{\theta}|y)) - p$ thus yields an approximatively unbiased estimator for $\mathbb{E}_y \mathbb{E}_x[\log(g(x|\hat{\theta}(y)))]$.

Summarizing, the Kullback-Leibler information $I(f,g)$ can be estimated by

$$\hat{\mathbb{E}}_{\hat{\theta}}[I(f,\hat{g})] = C - \log(L(\hat{\theta}|y)) + p$$

with $\hat{g} = g(\cdot|\hat{\theta})$.

The *"'Akaike Information Criterion"' (AIC)* only considers the relative information, and thus ignores C, and after multiplication by 2 (by historical reason) it is defined as

$$AIC = -2\log(L(\hat{\theta}|Data)) + 2p.$$

From all models considered one selects now that one with minimal AIC.

AIC – an example

Consider the model $y_i = f(x_i, \boldsymbol{\theta}) + \varepsilon_i$ for $i = 1, \ldots, n$. Thus, there is a functional relation between the independent and the dependent variable by the function f, which depends on the p-dimensional parameter venctor $\boldsymbol{\theta}$. Moreover, $\varepsilon_i \sim N(0, \sigma^2)$ for all i, and the error terms are independent. The density of $\varepsilon_i = y_i - f(x_i)$ is

$$\phi(\varepsilon_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f(x_i))^2}{2\sigma^2}}.$$

Hence the likelihood function is

$$\begin{aligned} L &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{[y_i - f(x_i,\boldsymbol{\theta})]^2}{2\sigma^2}} \\ &= (2\pi\sigma^2)^{-\frac{n}{2}} \exp\left\{ -\sum_{i=1}^{n} \frac{[y_i - f(x_i,\boldsymbol{\theta})]^2}{2\sigma^2} \right\} \end{aligned}$$

and the log-likelihood function is

$$\log L = \underbrace{-\frac{n}{2}\log(2\pi\sigma^2)}_{\text{independent from data}} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} [\underbrace{y_i - f(x_i,\boldsymbol{\theta})}_{\text{residuals}}]^2.$$

Maximizing the log-likelihood-function corresponds to a minimization of the residual sum of squares RSS. If the residual variance σ^2 is unknown, it can be estimated via the maximum-likelihood method as (see also Section 1.1.1)

$$\hat{\sigma}^2 = \mathrm{RSS}/n.$$

From that we obtain

$$\max \log L = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2\hat{\sigma}^2}\mathrm{RSS}$$

which results in

$$\max \log L = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log\left(\frac{\mathrm{RSS}}{n}\right) - \frac{n}{2}.$$

After dropping the constants which are independent of the model, the AIC becomes

$$\mathrm{AIC} = n\log(\mathrm{RSS}/n) + 2p.$$

If the residual variance σ^2 is known, the AIC is

$$AIC = \frac{\text{RSS}}{\sigma^2} + 2p$$

Here, the number of model parameters $\boldsymbol{\theta}$ was equal to p. If all considered models have the same number of parameters p, then the model with minimum AIC is equivalent to selecting the model with minimum RSS, and this is the usual objective of a model selection based on least squares. Within nested models, the (log-)likelihood is monotonically increasing and the RSS monotonically decreasing.

Section 4.2.4, page 32

2.3.2 Bayes information criterion (BIC)

Similar to AIC, BIC can be used if the model selection is based on the maximization of the log-likelihood function. The BIC is defined as

$$\begin{aligned} \text{BIC} &= -2 \max \log L + \log(n)p \\ &= \frac{\text{RSS}}{\sigma^2} + \log(n)p \quad \text{if } \sigma^2 \text{ is known} \end{aligned}$$

2.3.3 Mallow's Cp

For known σ^2 the Mallow's C_p is defined as

$$C_p = \frac{\text{RSS}}{\sigma^2} + 2p - n$$

If the residual variance σ^2 is not known, it can be estimated by regression with the full model (using all variables). If a full model cannot be computed (too many variables, collinearity, etc.), a regression can be performed on the relevant principal components (see Section 3.2.1), and the variance is estimated from the resulting residuals. For the "true" model, Cp is approximately p, the number of parameters used in this model, and otherwise greater than p. Thus a model where Cp is approximately p should be selected, and preferably that model with smallest p.

2.3.4 Use of the different criteria

For more than $e^2 = 7.3$ observations, that is 8 observations, the BIC penalizes stronger than the AIC, and therefore provides smaller models.

- AIC ... for descriptive models
- BIC ... for predictive models
- The absolute values of AIC or BIC can not be interpreted.
- Mallow's C_p is mainly used for stepwise regression (adding or removing one variable at a time).

2.4 Resampling methods

After choosing a model which provides a good fit to the training data, we want to model the test data as well, with the requirement $y_{\text{Test}} \approx \hat{f}(x_{\text{Test}})$ ($\hat{f}$ was calculated based on the training data only). One could define a loss function L which measures the error between y and $\hat{f}(x)$, i.e.

$$L\left(y, \hat{f}(x)\right) = \begin{cases} (y - \hat{f}(x))^2 & \text{... quadratic error} \\ |y - \hat{f}(x)| & \text{... absolute error} \end{cases}$$

The test error is the expected predicted value of an *independent* test set

$$\text{Err} = \mathbb{E}\left[L\left(y, \hat{f}(x)\right)\right]$$

With a concrete sample, Err can be estimated using

$$\widehat{\text{Err}} = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}(\boldsymbol{x}_i)\right)$$

In case of using the loss function with quadratic error, the estimation of Err is well-known under the name *Mean Squared Error (MSE)*. So, the MSE is defined by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(\boldsymbol{x}_i)\right)^2$$

Usually there is only *one* data set available. The evaluation of the model is then done by resampling methods (i.e. cross validation, bootstrap).

2.4.1 Cross validation

A given sample is randomly divided into q parts. One part is chosen to be the test set, the other parts are defined as training data. The idea is as follows: for the kth part, $k = 1, 2, \ldots, q$, we fit the model to the other $q - 1$ parts of the data and calculate the prediction error of the fitted model when predicting the kth part of the data. In order to avoid complicated notation, $\hat{f}$ denotes the fitted function, computed with the kth part of the data removed. The functions all differ depending on the part left out. The evaluation of the model (calculation of the expected prediction error) is done on the kth part of the data set, i.e. for $k = 3$

1	2	3	4	5	$\cdots$	q
Training	Training	*Test*	Training	Training	$\cdots$	Training

$\hat{y}_i = \hat{f}(\boldsymbol{x}_i)$ represents the prediction for $\boldsymbol{x}_i$, calculated by leaving out the kth part of the data set, with $\boldsymbol{x}_i$ allocated in part k. Since each observation exists only once in each test set, we obtain n predicted values $\hat{y}_i$, $i = 1, \ldots, n$.

The estimated cross validation error is then

$$\widehat{\text{Err}}_{\text{CV}} = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{y}_i\right)$$

Choice of q

The case $q = n$ is known as "leave 1 out cross validation". In this case the fit is computed using all the data except the ith. Disadvantages of this method are

- high computational effort
- high variance due to similarity of the n "training sets".

A 5 fold or 10 fold cross validation, thus $q = 5$ or $q = 10$, should therefore be preferred. With large data sets $n_{\text{Train}}/n_{\text{Test}} = 2/1$ or $1/1$ is often used, this means $\frac{n-n/q}{n/q} = \frac{2}{1}$.

2.4.2 Bootstrap

The basic idea is to randomly draw data sets with replacement from the training data, each sample the same size as the original training set. This is done q times, producing q data sets. Then we refit the model to each of the bootstrap data sets, and examine the behavior of the fits over the q replications. The mean prediction error then is

$$\widehat{\text{Err}}_{\text{Boot}} = \frac{1}{q}\frac{1}{n}\sum_{k=1}^{q}\sum_{i=1}^{n} L\left(y_i, \hat{f}_k(\boldsymbol{x}_i)\right),$$

with $\hat{f}_k$ indicating the function assessed on sample k and $\hat{f}_k(\boldsymbol{x}_i)$ indicating the prediction of observation $\boldsymbol{x}_i$ of the kth data set.

Problem and possible improvements

Due to the large overlap in test and training sets, $\widehat{\text{Err}}_{\text{Boot}}$ is frequently too optimistic.

The probability of an observation being included in a bootstrap data set is $1 - (1 - \frac{1}{n})^n \approx 1 - e^{-1} = 0.632$. A possible improvement would be to calculate $\widehat{\text{Err}}_{\text{Boot}}$ only for those observations not included in the bootstrap data set, which is true for about $\frac{1}{3}$ of the observations.

"Leave 1 out bootstrap" offers this improvement: The prediction $\hat{f}_k(\boldsymbol{x}_i)$ is based only on the data sets which do *not* include $\boldsymbol{x}_i$:

$$\widehat{\text{Err}}_{\text{Boot}-1} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|C^{-i}|}\sum_{k\in C^{-i}} L\left(y_i, \hat{f}_k(\boldsymbol{x}_i)\right)$$

C^{-i} is the set of indices of the bootstrap samples k that do *not* contain observation i.

2.5 Procedures for variable selection

Generally speaking, smaller models are easier to interpret. Redundant or unnecessary variables should be left out. The performance of the model will rarely get worse.

So far we have defined different options for model comparison. What is still unclear is: How can we efficiently identify models which are promising for the prediction? The naive approach would be to consider all 2^p possible models. However, it is clear that this soon results in computational difficulties. For example, if $p = 20$, which is still a very reasonable number of predictors, we already would have to check more than one million different models. Therefore, an approximate procedure is required, which returns the hopefully best model.

2.5.1 Stepwise algorithms

A natural procedure is to start with the full model, including all p input variables, and to eliminate step-by-step that variable which is least important for the prediction. The problem is how to define "least important for the prediction"? A simple procedure would be to use the F-statistic defined in Equation (2.1):

- In the first step we compute the value of the F-statistic by comparing the full model with a model, where the j-th variable has been removed, for $j \in \{1, \ldots, p\}$. We remove that variable which leads to the smallest F value, if this is not already significant.
- In the next step we compute the value of the F-statistic by comparing the reduced model with a model, where one of the remaining variables is removed. Again we remove that variable which leads to the smallest F value, except if this indicates significance.
- This is done until all F-statistics are significant.

This procedure is called *forward stepwise selection*. One can also do *backward stepwise selection*, by starting from the empty model, and adding step-by-step that predictor which leads to significance and gives the highest F-value. Also a combination of forward and backward selection is possible.

Rather than performing sequences of statistical tests, it is also common to use other criteria for model comparison, such as AIC or BIC. Also cross validation would be possible, but this is computationally much more expensive.

 Section 4.2.4, page 32

2.5.2 Best subset regression

Best subset regression finds the subset of size k for each $k \in \{0, 1, \ldots, p\}$ that gives the smallest AIC (or other criteria). An efficient algorithm is the *Leaps and Bounds algorithm* which is feasible for a moderate size of p. The idea is to exclude whole branches in the graph of all possible models.

Leaps and Bounds algorithm: This algorithm creates a tree model and calculates the AIC (or other criteria) for the particular subsets. In Figure 2.1 the AIC for the first subsets is shown. Subsequently, large branches are eliminated by trying to reduce the AIC. The AIC for the model $x2 + x3 = 20$. Through the elimination of $x2$ or $x3$ we get an AIC of at least 18, since the value can reduce by $2p = 2$ at most (this follows from the formula for the $AIC := n \log(RSS/n) + 2p$). By eliminating a regressor in $x1 + x2$ a smaller AIC (>8) can be obtained. Therefore, the branch $x2 + x3$ is left out in the future analysis.

 Section 4.2.5, page 34

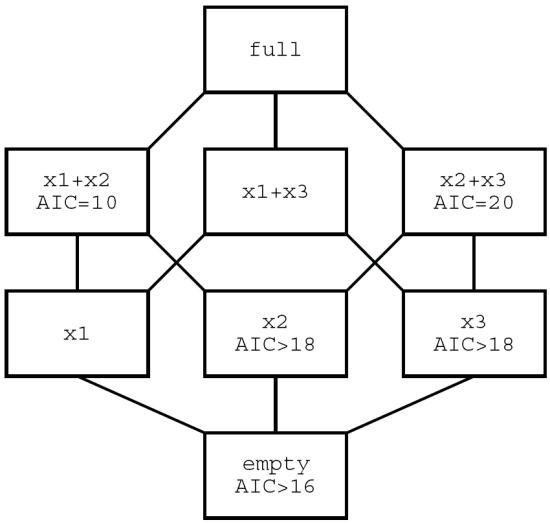

Figure 2.1: Model tree for the leaps and bound algorithm

Part II

Linear regression

Chapter 3

Linear methods

A *linear* regression model assumes that the regression function $\mathbb{E}(y|x_1, x_2, \ldots, x_p)$ is *linear* in the inputs $x_1, x_2, \ldots, x_p$. Linear models were largely developed in the pre-computer age of statistics, but even in today's computer era there are still good reasons to study and use them since they are the foundation of more advanced methods. Some important characteristics of linear models are:

- they are simple and
- often provide an adequate and
- interpretable description of how the inputs affect the outputs.
- For prediction purposes they can often outperform fancier nonlinear models, especially in situations with
  - small numbers of training data or
  - a low signal-to-noise ratio.
- Finally, linear models can be applied to transformations of the inputs and therefore be used to model nonlinear relations.

3.1 Least squares regression

This estimator has already been discussed in Section 1.1. If the model assumptions are valid, the LS estimator $\hat{\boldsymbol{\beta}}$ is the best linear unbiased estimator (BLUE). However, is this necessarily desirable? In the end we would like to have an estimator which leads to high prediction quality and simple interpretation, thus including as few input variables as necessary. It could well be that a non-linear estimator, or a biased linear estimator leads to higher prediction accuracy.

3.1.1 Bias versus variance and interpretability

The prediction quality can sometimes be improved by shrinkage of the regression coefficients or by setting some coefficients equal to zero. This way the bias increases, but the variance of the prediction reduces significantly which leads to an overall improved prediction.

Consider again the linear regression model $y = \boldsymbol{x}^\top \boldsymbol{\beta} + \varepsilon$, where here $\boldsymbol{x}^\top = (1, x_1, \ldots, x_p)$. Then the estimate is $\hat{y} = \boldsymbol{x}^\top \hat{\boldsymbol{\beta}}$, with the LS estimator $\hat{\boldsymbol{\beta}}$. Prediction quality is often measured

by the Mean Squared Error (MSE), or expected squared error, which is defined at the level of random variables as MSE= $\mathbb{E}[(y-\hat{y})^2]$. Using the model assumptions (in particular for the error term: $\mathbb{E}(\varepsilon)=0$, $\mathrm{Var}(\varepsilon)=\sigma^2$), it is not difficult to show that

$$\mathrm{MSE} = \mathbb{E}[(y-\hat{y})^2] = \sigma^2 + \mathrm{Var}(\hat{y}) + [\mathbb{E}(y-\hat{y})]^2. \tag{3.1}$$

Note that the (square-root of the) last term refers to the **bias**, because

$$\mathbb{E}(y-\hat{y}) = \mathbb{E}(\boldsymbol{x}^\top(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}})+\varepsilon) = \boldsymbol{x}^\top\mathbb{E}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}})+0.$$

For an unbiased estimator, this expression is zero, since

$$\mathbb{E}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}-\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}-\boldsymbol{\beta} = 0.$$

This means that for an unbiased estimator, the MSE is just determined by the variance of the estimates, see Equation (3.1), and of course by the irreducible error σ^2. If we consider a biased estimator with a smaller variance, it could happen that the MSE based on this estimator even gets smaller. Such estimators can be obtained from variable selection, by building linear combinations of the regressor variables (Section 3.2), or by shrinkage methods (Section 3.3).

There is also another problem with the LS estimator, especially if the input variables are highly correlated. In this case the matrix $\boldsymbol{X}^\top\boldsymbol{X}$ can become nearly singular, and thus $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$ will be numerically unstable. This expression is not only involved in the LS estimator $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$, but also in the test statistic for the hypothesis test on the regression parameters,

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{d_j}},$$

where d_j represents the jth diagonal element of $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$, see Section 1.1.2. As a consequence, also the statistical inference can become very unreliable, and we need to consider different methods which can handle (highly) correlated input variables, or settings where the number of inputs can even exceed the number of observations ($p > n$).

3.2 Methods using derived inputs as regressors

3.2.1 Principal Component Regression (PCR)

This method looks for transformations of the original data into a new set of uncorrelated variables called principal components. This transformation ranks the new variables according to their importance, which means according to the size of their variance, and eliminates those of least importance. Then a least squares regression on the reduced set of principal components is performed.

Since PCR is not scale invariant, one should scale and center the data. The idea is to construct a new matrix $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}$, which consist of linear combinations with the x-variables with coefficients defined by the matrix $\boldsymbol{V}$. The matrix $\boldsymbol{V}$ is a $(p \times p)$-matrix, and thus $\boldsymbol{Z}$ has the same dimension as $\boldsymbol{X}$. The task is to select $\boldsymbol{V}$ such that the columns of $\boldsymbol{Z}$ are uncorrelated and have maximum variance. This can be achieved by a so-called *spectral decomposition* of the covariance matrix $\mathrm{Cov}(\boldsymbol{X})$:

$$\mathrm{Cov}(\boldsymbol{X}) = \boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^\top$$

Here, $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)$ has normed and orthogonal column vectors, i.e. $\boldsymbol{v}_i^\top \boldsymbol{v}_i = 1$, and $\boldsymbol{v}_i^\top \boldsymbol{v}_j = 0$, for $i \neq j = 1, \ldots, p$, and thus $\boldsymbol{V}^\top = \boldsymbol{V}^{-1}$. The matrix $\boldsymbol{A}$ is of diagonal form, $\boldsymbol{A} = Diag(a_1, \ldots, a_p)$.

This solution results from an eigenvalue problem, where $\boldsymbol{v}_i$ are the eigenvectors of $\mathrm{Cov}(\boldsymbol{X})$, and d_i are the corresponding eigenvalues: $a_1 \geq a_2 \ldots \geq a_p$. If $\mathrm{Cov}(\boldsymbol{X})$ is positive definite, all eigenvalues are real, non negative numbers.

The columns of $\boldsymbol{Z}$ are are called *principal components*, and we obtain:

$$\mathrm{Cov}(\boldsymbol{Z}) = \boldsymbol{V}^\top \mathrm{Cov}(\boldsymbol{X}) \boldsymbol{V} = \boldsymbol{A}$$

Thus, the variance of the ith principal component is equal to the eigenvalue a_i; the variances are ranked in descending order.

In the following we will use the first q $(1 \leq q < p)$ principal components for regression. The regression model can first be rewritten by using *all* the principal components:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \\ &= \boldsymbol{X}\boldsymbol{V}\boldsymbol{V}^\top \boldsymbol{\beta} + \boldsymbol{\varepsilon} \\ &= \boldsymbol{Z}\boldsymbol{\theta} + \boldsymbol{\varepsilon} \end{aligned} \tag{3.2}$$

The new regression coefficients in this model are denoted by $\boldsymbol{\theta}$, and we have $\boldsymbol{\theta} = \boldsymbol{V}^\top \boldsymbol{\beta}$ and $\boldsymbol{\beta} = \boldsymbol{V}\boldsymbol{\theta}$. If we want to use for regression only $q < p$ principal components, then we obtain by the above relation

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{Z}_{1:q}\boldsymbol{\theta}_{1:q} + \boldsymbol{Z}_{q+1:p}\boldsymbol{\theta}_{q+1:p} + \boldsymbol{\varepsilon} \\ &= \boldsymbol{Z}_{1:q}\boldsymbol{\theta}_{1:q} + \tilde{\boldsymbol{\varepsilon}}. \end{aligned}$$

In this reduced model we obtain with LS estimation

$$\hat{\boldsymbol{\theta}}_{1:q} = (\boldsymbol{Z}_{1:q}^\top \boldsymbol{Z}_{1:q})^{-1} \boldsymbol{Z}_{1:q}^\top \boldsymbol{y}.$$

If the regression coefficients should be interpreted in terms of the original variables, then a back-transformation is possible by using the above relation,

$$\tilde{\hat{\boldsymbol{\beta}}} = \boldsymbol{V}_{1:q} \hat{\boldsymbol{\theta}}_{1:q}.$$

Note that $\tilde{\hat{\boldsymbol{\beta}}}$ does no longer correspond to the LS estimator from Equation (3.2), except if the first q principal components explain all the information of $\boldsymbol{X}$.

Section 4.3.2, page 38

3.2.2 Partial Least Squares (PLS) regression

This technique also constructs a set of linear combinations of the inputs for regression, but unlike principal components regression it uses $\boldsymbol{y}$ in addition to $\boldsymbol{X}$ for this construction. We assume that $\boldsymbol{X}$ is centered and – depending on the application – also scaled. Instead of estimating the parameters $\boldsymbol{\beta}$ in the linear model

$$y_i = \boldsymbol{x}_i^\top \boldsymbol{\beta} + \varepsilon_i$$

we estimate the parameters $\boldsymbol{\gamma}$ in the so-called latent variable model

$$y_i = \boldsymbol{t}_i^\top \boldsymbol{\gamma} + \tilde{\varepsilon}_i.$$

We assume:

- The new coefficients $\boldsymbol{\gamma}$ are of dimension $q \leq p$.
- The values $\boldsymbol{t}_i$ are arranged as rows in an $(n \times q)$ score matrix $\boldsymbol{T}$.
- Due to the reduced dimension, the regression of $\boldsymbol{y}$ on $\boldsymbol{T}$ should be more stable.
- $\boldsymbol{T}$ can not be observed directly; we obtain each column of $\boldsymbol{T}$ sequentially, for $k = 1, 2, \ldots, q$, by using the PLS criterion

  $$\boldsymbol{w}_k = \underset{\boldsymbol{w}}{\operatorname{argmax}} \operatorname{Cov}(\boldsymbol{y}, \boldsymbol{X}\boldsymbol{w})$$

  under the constraints $\|\boldsymbol{w}_k\| = 1$ and $\operatorname{Cov}(\boldsymbol{X}\boldsymbol{w}_k, \boldsymbol{X}\boldsymbol{w}_j) = 0$ for $1 \leq j < k$. The vectors $\boldsymbol{w}_k$ with $k = 1, 2, \ldots, q$ are called *loadings*, and they are collected in the columns of the matrix $\boldsymbol{W}$. The score matrix is then

  $$\boldsymbol{T} = \boldsymbol{X}\boldsymbol{W},$$

  and $\boldsymbol{y}$ can be written as:

  $$\begin{aligned} \boldsymbol{y} &= \boldsymbol{T}\boldsymbol{\gamma} + \tilde{\boldsymbol{\varepsilon}} \\ &= (\boldsymbol{X}\boldsymbol{W})\boldsymbol{\gamma} + \tilde{\boldsymbol{\varepsilon}} \\ &= \boldsymbol{X} \underbrace{(\boldsymbol{W}\boldsymbol{\gamma})}_{\tilde{\boldsymbol{\beta}}} + \tilde{\boldsymbol{\varepsilon}} \approx \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \end{aligned}$$

In other words, PLS does a regression on a weighted version of $\boldsymbol{X}$ which contains incomplete or partial information (thus the name of the method). The additional usage of the least squares method for the fit leads to the name *Partial Least Squares*.
Since PLS uses also $\boldsymbol{y}$ to determine the PLS-directions, this method is supposed to have better prediction performance than for instance PCR. In contrast to PCR, PLS is looking for directions with high variance and large correlation with $\boldsymbol{y}$.

Section 4.3.3, page 39

3.2.3 Continuum regression

This method combines LS-, PC- and PLS-regression. As for PLS regression, the regression is done using $\boldsymbol{y}$ and $\boldsymbol{T} = \boldsymbol{X}\boldsymbol{W}$, where $\boldsymbol{w}_k$ for $k = 1, 2, \ldots, q \leq p$ are obtained by

$$\boldsymbol{w}_k = \underset{\boldsymbol{w}}{\operatorname{argmax}} \left\{ [\operatorname{Cov}(\boldsymbol{y}, \boldsymbol{X}\boldsymbol{w})]^2 [\operatorname{Var}(\boldsymbol{X}\boldsymbol{w})]^{\frac{\delta}{1-\delta}-1} \right\}$$

under the constraint

$$\|\boldsymbol{w}_k\| = 1$$

and

$$\operatorname{Cov}(\boldsymbol{X}\boldsymbol{w}_k, \boldsymbol{X}\boldsymbol{w}_j) = 0 \quad \text{for } j < k.$$

δ is in the range from 0 and 1, and regulates the information content of the $\boldsymbol{X}$-part, which should be used for the prediction of the $\boldsymbol{y}$-part. We can identify three special cases:

$$\begin{aligned}
\delta = 0 \quad &: \quad \frac{\delta}{1-\delta} - 1 = -1 \\
&\Rightarrow \quad \boldsymbol{w}_k = \underset{\boldsymbol{w}}{\operatorname{argmax}} \left\{ \frac{[\operatorname{Cov}(\boldsymbol{X}\boldsymbol{w}, \boldsymbol{y})]^2}{\operatorname{Var}(\boldsymbol{X}\boldsymbol{w})} \right\} \quad \ldots \text{LS regression} \\
\delta = 0.5 \quad &: \quad \frac{\delta}{1-\delta} - 1 = 0 \\
&\Rightarrow \quad \boldsymbol{w}_k = \underset{\boldsymbol{w}}{\operatorname{argmax}} \left\{ [\operatorname{Cov}(\boldsymbol{X}\boldsymbol{w}, \boldsymbol{y})]^2 \right\} \quad \ldots \text{PLS regression} \\
\delta = 1 \quad &: \quad \frac{\delta}{1-\delta} - 1 \to \infty \\
&\Rightarrow \quad \boldsymbol{w}_k = \underset{\boldsymbol{w}}{\operatorname{argmax}} \left\{ \operatorname{Var}(\boldsymbol{X}\boldsymbol{w}) \right\} \quad \ldots \text{PC regression}
\end{aligned}$$

3.3 Shrinkage methods

Shrinkage methods keep all variables in the model and assign different (continuous) weights. In this way we obtain a smoother procedure with a smaller variability.

3.3.1 Ridge regression

Ridge regression shrinks the coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares,

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{3.3}$$

Here $\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage: the larger the value of λ, the greater the amount of shrinkage. The coefficients are shrunk towards zero (and towards each other).

An equivalent way to write the ridge problem is

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 \right\}$$

under the constraint

$$\sum_{j=1}^{p} \beta_j^2 \leq s,$$

which makes explicit the size constraint on the parameters. By penalizing the RSS we try to avoid that highly correlated regressors (e.g. x_j and x_k) cancel each other. An especially large positive coefficient β_j can be canceled by a similarly large negative coefficient β_k. By imposing a size constraint on the coefficients this phenomenon can be prevented.

The ridge solutions $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$ are *not* equivariant for different scaling of the inputs, and so one normally standardizes the inputs. In addition, notice that the intercept β_0 has been left out

of the penalty term. Penalization of the intercept would make the procedure depend on the origin chosen for $\boldsymbol{y}$; that is adding a constant c to each of the targets y_i would *not* simply result in a shift of the predictions by the same amount c.

We center the x_{ij}, each x_{ij} gets replaced by $x_{ij} - \overline{x}_j$ and estimate β_0 by $\overline{y} = \sum_{i=1}^{n} y_i/n$. The remaining coefficients get estimated by a ridge regression *without* intercept, hence the matrix $\boldsymbol{X}$ has p rather than $p+1$ columns. Rewriting (3.3) in matrix form,

$$\begin{aligned} \mathrm{RSS}(\lambda) &= (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} \quad \text{the solutions become} \\ \hat{\boldsymbol{\beta}}_{\mathrm{Ridge}} &= (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \end{aligned}$$

$\boldsymbol{I}$ is the $(p \times p)$ identity matrix. Advantages of the just described method are:

- With the choice of quadratic penalty $\boldsymbol{\beta}^\top\boldsymbol{\beta}$, the resulting ridge regression coefficients are again a linear function of $\boldsymbol{y}$.
- The solution adds a positive constant to the diagonal of $\boldsymbol{X}^\top\boldsymbol{X}$ before inversion. This makes the problem nonsingular, even if $\boldsymbol{X}^\top\boldsymbol{X}$ is not of full rank. This was the main motivation of its introduction around 1970.
- The effective degrees of freedom are

  $$\mathrm{df}(\lambda) = \mathrm{tr}(\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top),$$

  thus

  $$\begin{aligned} \text{for } \lambda = 0 &\Rightarrow \mathrm{df}(\lambda) = \mathrm{tr}(\boldsymbol{X}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}) = \mathrm{tr}(\boldsymbol{I}_p) = \mathrm{p} \\ \lambda \to \infty &\Rightarrow \mathrm{df}(\lambda) \to 0 \end{aligned}$$

In the case of orthogonal inputs, the ridge coefficients are just a scaled version of the least squares estimates, that is $\hat{\boldsymbol{\beta}}_{\mathrm{Ridge}} = \gamma\hat{\boldsymbol{\beta}}$ with $0 \le \gamma \le 1$.

Relation between ridge regression and PCR

In order to get a better insight into the principle of ridge regression, we go into more detail in the following:

Singular value decomposition
A singular value decomposition of the centered $(n \times p)$ input matrix $\boldsymbol{X}$ (here we assume $n > p$) is of the form

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top, \tag{3.4}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are ortho-normal $(n \times p)$- and $(p \times p)$ matrices, respectively. The columns of $\boldsymbol{U}$ are spanning the column space of $\boldsymbol{X}$, and the columns of $\boldsymbol{V}$ are spanning the row space of $\boldsymbol{X}$. $\boldsymbol{D}$ is a $(p \times p)$ diagonal matrix of the so-called singular values of $\boldsymbol{X}$. We have: $d_1 \ge d_2 \ge \cdots \ge d_p \ge 0$. The columns of $\boldsymbol{U}$ are the eigen vectors of $\boldsymbol{X}\boldsymbol{X}^\top$ to the eigen values d_i^2, and the columns of $\boldsymbol{V}$ are the eigen vectors of $\boldsymbol{X}^\top\boldsymbol{X}$ to the same eigen values.
The singular value decomposition of the centered matrix $\boldsymbol{X}$ is an alternative way to express the principal components of $\boldsymbol{X}$. The stimated covariance matrix is

$$\begin{aligned} \boldsymbol{S} &= \frac{1}{n-1}\boldsymbol{X}^\top\boldsymbol{X} \\ &= \frac{1}{n-1}(\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top) \\ &= \frac{1}{n-1}\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top \end{aligned}$$

and by (3.4) we obtain

$$\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top,$$

which corresponds to the eigen decomposition of $\boldsymbol{X}^\top\boldsymbol{X}$ (and thus of $\boldsymbol{S}$ - up to a factor $\frac{1}{n-1}$). $\boldsymbol{D}^2 = \boldsymbol{A}$ are the corresponding eigen values. As it was already a goal of PCR, also here the principal components of $\boldsymbol{X}$ are requiered. For the first principal components $\boldsymbol{z}_1$ we have

$$\begin{aligned} \boldsymbol{z}_1 &= \boldsymbol{X}\boldsymbol{v}_1 \\ &= \boldsymbol{u}_1 d_1 \end{aligned}$$

and this components has the largest variance out of all normalized linear compinations of the columns of $\boldsymbol{X}$. This equals

$$\begin{aligned} \mathrm{Var}(\boldsymbol{z}_1) &= \mathrm{Var}(\boldsymbol{X}\boldsymbol{v}_1) \\ &= \frac{1}{n-1}(\boldsymbol{u}_1 d_1)^\top(\boldsymbol{u}_1 d_1) \\ &= \frac{1}{n-1} d_1^2. \end{aligned}$$

Using singular decomposition, the vector of fitted values of the LS estimation can be written as

$$\begin{aligned} \boldsymbol{X}\hat{\boldsymbol{\beta}}_{\mathrm{LS}} &= \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top)^{-1}\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{D}^{-2}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{y} \end{aligned}$$

$\boldsymbol{U}^\top\boldsymbol{y}$ are coordinates of $\boldsymbol{y}$ with respect to the orthonormal basis $\boldsymbol{U}$. Now the ridge solutions can be represented as

$$\begin{aligned} \boldsymbol{X}\hat{\boldsymbol{\beta}}_{\mathrm{Ridge}} &= \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top + \lambda\boldsymbol{I})^{-1}\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{D}(\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top\boldsymbol{V} + \lambda\boldsymbol{V}^\top\boldsymbol{V})^{-1}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{D}(\boldsymbol{D}^2 + \lambda\boldsymbol{I})^{-1}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{y} \\ &= \sum_{j=1}^{p} \boldsymbol{u}_j \frac{d_j^2}{d_j^2 + \lambda}\boldsymbol{u}_j^\top\boldsymbol{y} \end{aligned}$$

with $\boldsymbol{u}_j$ as columns of $\boldsymbol{U}$, and since $\lambda \geq 0$, we have that $d_j^2/(d_j^2 + \lambda) \leq 1$. Similar to linear regression, ridge regression computed the coordinates of $\boldsymbol{y}$ with respect to the orthonormal basis $\boldsymbol{U}$. The coordinates are shrunken by the factor $d_j^2/(d_j^2 + \lambda)$. This means that the magnitude of shrinkage is increasing with decreasing d_j^2.

What is the meaning of a small value of d_j^2?

d_j^2 is proportional to the variance of the principal component. Ridge regression projects $\boldsymbol{y}$ on the component $\boldsymbol{u}_j$. The smaller d_j^2, the more penalty is used along this direction. Here we implicitely assume that $\boldsymbol{y}$ varies more in directions of higher variance of the input variables, which is not necessarily the case.

Thus, principal component regression is closely related to rigde regression: both methods make use of the principal components of the input matrix $\boldsymbol{X}$. Ridge regression shrinks

the coefficients of the principal components, and the amount of shrinkage depends on the corresponding eigen value. Principal component regression discards the components corresponding to the smallest $p - q$ eigen values.

Section 4.4.1, page 40

3.3.2 Lasso Regression

The lasso is a shrinkage method like ridge, but L_1 norm rather than the L_2 norm is used in the constraints. The lasso is defined by

$$\hat{\boldsymbol{\beta}}_{\text{Lasso}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

with the constraint

$$\sum_{j=1}^{p} |\beta_j| \leq s.$$

Just as in ridge regression we standardize the data. The solution for $\hat{\beta}_0$ is $\overline{y}$ and thereafter we fit a model without an intercept.

Lasso and ridge differ in their penalty term. The lasso solutions are nonlinear and a quadratic programming algorithm is used to compute them. Because of the nature of the constraint, making s sufficiently small will cause some of the coefficients to be exactly 0. Thus the lasso does a kind of continuous subset selection. If s is chosen larger than $s_0 = \sum_{j=1}^{p} |\hat{\beta}_j|$ (where $\hat{\beta}_j$ is the least squares estimate), then the lasso estimates are the least squares estimates. On the other hand, for $s = s_0/2$, the least squares coefficients are shrunk by about 50% on average. However, the nature of the shrinkage is not obvious. Like the subset size in subset selection, or the penalty in ridge regression, s should be adaptly chosen to minimize an estimate of expected prediction error.

Figure 3.1 visualizes the difference in the penalties in case of two parameters. The distribution of the residual sum-of-squares is visualized by the elliptical contours, which are centered at the least-squares estimator. The blue regions refer to the contours of the contraints, left for Lasso ($|\beta_1| + |\beta_2| \leq s$), and right for Ridge ($\sqrt{\beta_1^2 + \beta_2^2} \leq s$). It is clear that Lasso will more likely lead to a solution where regression coefficients are exactly zero.

Section 4.4.2, page 42

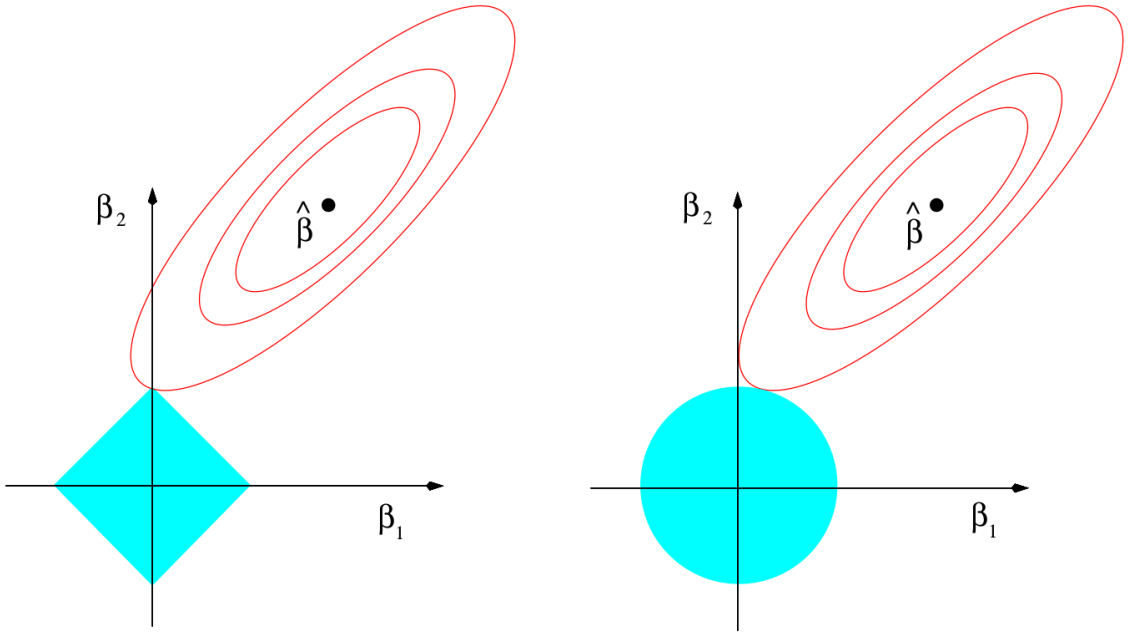

Figure 3.1: Penalties for Lasso (left) and Ridge (right) regression.

Chapter 4

Linear methods in R

4.1 Least Squares (LS) regression in R

4.1.1 Parameter estimation

Here we simulate a data set with a response and 3 explanatory variables. Therefore we know the true regression coefficients, $\beta_0 = 0$, $\beta_1 = 1$, $\beta_2 = 2$ and $\beta_3 = 0$. Accordingly, the variable x_3 has no predictive meaning, and should not be used in the model.

- *Generation of the data*

```
> set.seed(123)
> x <- matrix(runif(60), ncol = 3)
> y <- x %*% c(1, 2, 0) + 0.1 * rnorm(20)
> colnames(x) <- paste("x", 1:3, sep = "")
> d <- data.frame(x, y = y)
> plot(d)
```

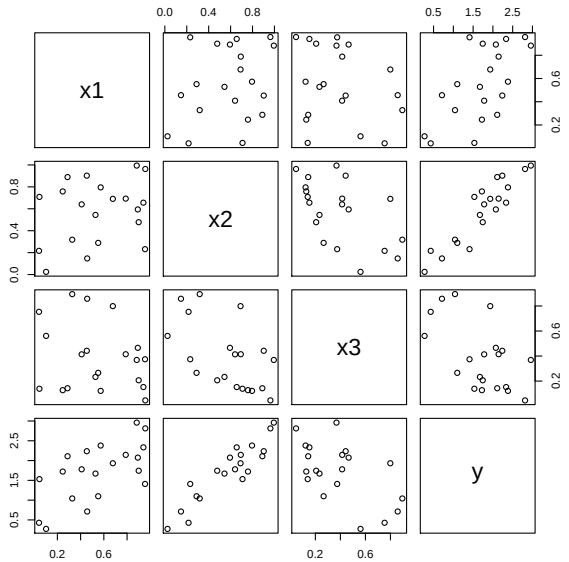

Figure 4.1: Plot of the generated data used for multiple linear regression.

- *Model using only a constant term*

```
> lm0 <- lm(y~1, data = d)
> lm0

Call:
lm(formula = y ~ 1, data = d)

Coefficients:
(Intercept)
       1.72
```

LS regression is computed by **lm()**. The estimated value of the intercept is $\beta_0 = 1.72$

- *Model with one explanatory variable*

```
> lm1 <- lm(y~x1, data = d)
> lm1

Call:
lm(formula = y ~ x1, data = d)

Coefficients:
(Intercept)           x1
     0.9157       1.4600
```

- *Fit of a full model*

```
> lm3 <- lm(y~x1+x2+x3, data = d)
> lm3

Call:
lm(formula = y ~ x1 + x2 + x3, data = d)

Coefficients:
(Intercept)           x1           x2           x3
    0.09585      0.91834      1.99804     -0.08761
```

The coefficients from the full model are close to the true coefficients. However, we would prefer a model that excludes x_3, so with $\beta_3 = 0$.

4.1.2 Tests and confidence intervals

- *Testing the coefficients for significance*

```
> summary(lm3)

Call:
lm(formula = y ~ x1 + x2 + x3, data = d)

Residuals:
     Min       1Q   Median       3Q      Max
-0.11566 -0.06133 -0.01260  0.06785  0.18004

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09585    0.08200   1.169    0.260
x1           0.91834    0.06623  13.867 2.47e-10 ***
x2           1.99804    0.08453  23.637 7.18e-14 ***
x3          -0.08761    0.09060  -0.967    0.348
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.08621 on 16 degrees of freedom
Multiple R-squared:  0.9882,        Adjusted R-squared:  0.986
F-statistic: 446.5 on 3 and 16 DF,  p-value: 1.251e-15
```

- The t statistic (see Section 1.1.2) of x_1 and x_2 is highly significant and the p-value of each variable is below 0.05. Therefore, both variables have a great impact on the explanation of the regressor and the null hypothesis can be rejected. The regressor x_3 provides no significant additional contribution.
- The model provides a good fit (R squared, see Section 2.2), 98.82% of the variance of y can be explained by the model. The value 98.6% of the adjusted R squared is very high as well.
- `> qf(0.95, 3, 16)`

  `[1] 3.238872`

  The value of the "F statistic" (see Section 2.1) of 446.5 is larger than the F quantile $F_{3,16;0.95} = 3.24$, therefore the null hypothesis $\beta_i = 0, \forall\ i = 1, \ldots, p$ can be rejected. This could also be concluded by the p-value that is close to 0.
- The test statistic from above can be used for the calculation of a confidence interval for $\hat{\beta}_j$ (see Section 1.1.2). From the approximation of the 95% confidence interval, we obtain for $\hat{\beta}_1$ the interval

$$0.91834 \pm 2 * 0.06623 = [0.78, 1.06]$$

  and for $\hat{\beta}_3$

$$-0.08761 \pm 2 * 0.09060 = [-0.27, 0.09]$$

  The interval for $\hat{\beta}_1$ does not include zero, and thus the null hypothesis can be rejected at a 95% level. The interval for $\hat{\beta}_3$ includes zero, which confirms the acceptance of the null hypothesis due to a p-value of 0.348.

4.2 Variable selection in R

4.2.1 Model comparison with anova()

```
> anova(lm3)
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq  F value    Pr(>F)
x1         1 3.9799  3.9799 535.4639 9.991e-14 ***
x2         1 5.9693  5.9693 803.1073 4.199e-15 ***
x3         1 0.0070  0.0070   0.9351    0.3479
Residuals 16 0.1189  0.0074
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

An F-test (see Section 2.1) is computed for every additional explanatory variable, starting with the empty model and following the order of the formula. Regressor x_3 does not improve the fit of the model and can be left out.

```
> lm2 <- lm(y~x1+x2, data=d)
> anova(lm0, lm1, lm2, lm3)
Analysis of Variance Table

Model 1: y ~ 1
Model 2: y ~ x1
Model 3: y ~ x1 + x2
Model 4: y ~ x1 + x2 + x3
  Res.Df     RSS Df Sum of Sq        F    Pr(>F)
1     19 10.0751
2     18  6.0951  1    3.9799 535.4639 9.991e-14 ***
3     17  0.1259  1    5.9693 803.1073 4.199e-15 ***
4     16  0.1189  1    0.0070   0.9351    0.3479
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here several nested models (with increasing size) are compared in the specified order. This allows simultaneous testing of the significance of more than one parameter. Here, again, model `lm3` does not improve the fit.

4.2.2 Body fat data

- *Scanning of the data and explanation of the variables*

```
> library("UsingR")
> data(fat)
> fat <- fat[-c(31,39,42,86), -c(1,3,4,9)]# strange values, not use all variables
> attach(fat)
```

The data set "fat" consists of 15 physical measurements of 251 men. The data can be found in the `library(UsingR)`.

- `body.fat`: percentage of body-fat calculated by Brozek's equation
- `age`: age in years
- `weight`: weight (in pounds)
- `height`: height (in inches)
- `BMI`: adiposity index
- `neck`: neck circumference (cm)
- `chest`: chest circumference (cm)
- `abdomen`: abdomen circumference (cm)
- `hip`: hip circumference (cm)
- `thigh`: thigh circumference (cm)
- `knee`: knee circumference (cm)
- `ankle`: ankle circumference (cm)
- `bicep`: extended biceps circumference (cm)
- `forearm`: forearm circumference (cm)
- `wrist`: wrist circumference (cm)

To measure the percentage of body-fat in the body, an extensive (and expensive) underwater technique has to be performed. The goal here is to establish a model which allows the prediction of the percentage of body-fat with easily measurable and collectible variables in order to avoid the underwater procedure. Nowadays, a new,

very effortless method called bio-impedance analysis provides a reliable method to determine the body-fat percentage.

4.2.3 Full model

For model evaluation later on, we first split the data randomly into training (2/3) and test data (1/3). The models will be built with the training data, and the evaluation is based on the test data.

```
> # randomly split into training and test data:
> set.seed(123)
> n <- nrow(fat)
> train <- sample(1:n,round(n*2/3))
> test <- (1:n)[-train]
> model.lm <- lm(body.fat~., data = fat, subset=train)
> summary(model.lm)

Call:
lm(formula = body.fat ~ ., data = fat, subset = train)

Residuals:
    Min      1Q  Median      3Q     Max
-9.6967 -2.5370 -0.3181  2.4634  8.9503

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.32010   42.43522  -0.125   0.9004
age          0.04946    0.03886   1.273   0.2051
weight      -0.03714    0.11980  -0.310   0.7570
height       0.02268    0.58969   0.038   0.9694
BMI          0.46329    0.86154   0.538   0.5916
neck        -0.16060    0.26965  -0.596   0.5523
chest       -0.13454    0.12691  -1.060   0.2908
abdomen      0.83996    0.11375   7.384 9.84e-12 ***
hip         -0.34536    0.16922  -2.041   0.0430 *
thigh        0.19863    0.17014   1.167   0.2449
knee        -0.01277    0.28098  -0.045   0.9638
ankle       -0.34105    0.44172  -0.772   0.4413
bicep        0.11665    0.18273   0.638   0.5242
forearm      0.29547    0.20976   1.409   0.1610
wrist       -1.32879    0.66187  -2.008   0.0465 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.955 on 150 degrees of freedom
Multiple R-squared:  0.7447,        Adjusted R-squared:  0.7208
F-statistic: 31.25 on 14 and 150 DF,  p-value: < 2.2e-16
```

The coefficients for `abdomen`, `hip` and `wrist` have a p-value below 0.05, and therefore the null hypothesis $\beta_i = 0$ should be rejected for the corresponding variables. Due to the very small p-value of the F-statistic, the null hypothesis $\beta_i = 0,\ \forall\ i = 1, \ldots, p$ should be rejected as well. With an R squared = 0.7447 we can assume that the model provides a good fit. Some measures for the prediction performance like R^2 or MSE for the test data can be computed.

```
> pred.lm <- predict(model.lm,newdata = fat[test,])
> cor(fat[test,"body.fat"],pred.lm)^2 # R^2 for test data

[1] 0.7414579

> mean((fat[test,"body.fat"]-pred.lm)^2) # MSE_test

[1] 16.62058
```

Figure 4.2 show the measured versus the predicted response variable for the test data,

resulting from the full model.

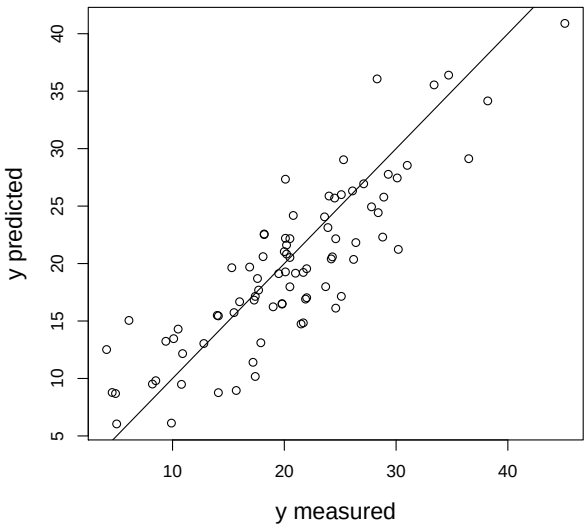

Figure 4.2: Prediction resulting from full model for test data.

4.2.4 Stepwise selection - automatic model search

– *Stepwise selection with* **drop1()**

```
> drop1(model.lm, test="F")
Single term deletions

Model:
body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist
        Df Sum of Sq    RSS    AIC F value    Pr(>F)
<none>                2346.2 468.01
age      1     25.34 2371.5 467.78  1.6200   0.20506
weight   1      1.50 2347.7 466.11  0.0961   0.75698
height   1      0.02 2346.2 466.01  0.0015   0.96938
BMI      1      4.52 2350.7 466.32  0.2892   0.59155
neck     1      5.55 2351.7 466.40  0.3547   0.55234
chest    1     17.58 2363.7 467.24  1.1239   0.29079
abdomen  1    852.89 3199.0 517.17 54.5288 9.839e-12 ***
hip      1     65.15 2411.3 470.53  4.1652   0.04302 *
thigh    1     21.32 2367.5 467.50  1.3629   0.24489
knee     1      0.03 2346.2 466.01  0.0021   0.96382
ankle    1      9.32 2355.5 466.66  0.5961   0.44127
bicep    1      6.37 2352.5 466.45  0.4075   0.52422
forearm  1     31.03 2377.2 468.18  1.9841   0.16103
wrist    1     63.04 2409.2 470.38  4.0306   0.04648 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> summary(update(model.lm,.~.-knee))

Call:
lm(formula = body.fat ~ age + weight + height + BMI + neck +
    chest + abdomen + hip + thigh + ankle + bicep + forearm +
    wrist, data = fat, subset = train)

Residuals:
    Min      1Q  Median      3Q     Max
-9.6903 -2.5408 -0.3137  2.4703  8.9383
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.30457   42.29339  -0.125   0.9004
age          0.04883    0.03620   1.349   0.1794
weight      -0.03730    0.11936  -0.312   0.7551
height       0.01955    0.58373   0.033   0.9733
BMI          0.46113    0.85739   0.538   0.5915
neck        -0.15963    0.26790  -0.596   0.5522
chest       -0.13417    0.12622  -1.063   0.2895
abdomen      0.84015    0.11329   7.416 8.09e-12 ***
hip         -0.34589    0.16825  -2.056   0.0415 *
thigh        0.19643    0.16258   1.208   0.2288
ankle       -0.34570    0.42828  -0.807   0.4208
bicep        0.11683    0.18208   0.642   0.5221
forearm      0.29542    0.20907   1.413   0.1597
wrist       -1.32794    0.65942  -2.014   0.0458 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.942 on 151 degrees of freedom
Multiple R-squared:  0.7447,        Adjusted R-squared:  0.7227
F-statistic: 33.88 on 13 and 151 DF,  p-value: < 2.2e-16
```

Elimination of the least significant variable, in this case `knee` is excluded from the model. The R squared (and adjusted R squared) do not change, the fit remains the same.

– *Automatic model search with* **step()**

```
> model.lmstep <- step(model.lm)
Start:  AIC=468.01
body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist

          Df Sum of Sq    RSS    AIC
- height   1      0.02 2346.2 466.01
- knee     1      0.03 2346.2 466.01
- weight   1      1.50 2347.7 466.11
- BMI      1      4.52 2350.7 466.32
- neck     1      5.55 2351.7 466.40
- bicep    1      6.37 2352.5 466.45
- ankle    1      9.32 2355.5 466.66
- chest    1     17.58 2363.7 467.24
- thigh    1     21.32 2367.5 467.50
- age      1     25.34 2371.5 467.78
<none>                 2346.2 468.01
- forearm  1     31.03 2377.2 468.18
- wrist    1     63.04 2409.2 470.38
- hip      1     65.15 2411.3 470.53
- abdomen  1    852.89 3199.0 517.17

Step:  AIC=466.01
body.fat ~ age + weight + BMI + neck + chest + abdomen + hip +
    thigh + knee + ankle + bicep + forearm + wrist

          Df Sum of Sq    RSS    AIC
- knee     1      0.03 2346.2 464.01
- weight   1      5.08 2351.3 464.37
- neck     1      5.54 2351.7 464.40
- bicep    1      6.38 2352.6 464.46
- ankle    1      9.43 2355.6 464.67
- chest    1     17.60 2363.8 465.24
- thigh    1     21.43 2367.6 465.51
- age      1     25.32 2371.5 465.78
- BMI      1     28.47 2374.6 466.00
<none>                 2346.2 466.01
- forearm  1     31.11 2377.3 466.18
- wrist    1     63.20 2409.4 468.39
```

```
- hip      1     65.24 2411.4 468.53
- abdomen  1    855.35 3201.5 515.30
```

⋮

```
Step:  AIC=457.89
body.fat ~ age + BMI + chest + abdomen + hip + forearm + wrist

          Df Sum of Sq    RSS    AIC
<none>                  2402.0 457.89
- forearm  1     33.75 2435.8 458.19
- age      1     35.00 2437.0 458.28
- chest    1     42.28 2444.3 458.77
- BMI      1     59.35 2461.4 459.92
- hip      1    115.34 2517.4 463.63
- wrist    1    163.81 2565.9 466.78
- abdomen  1    944.72 3346.8 510.62
```

step() calls **add1()** and **drop1()** as long as the AIC cannot be reduced further.

– *Comparison of the models with* **anova()**

```
> anova(model.lmstep, model.lm1,model.lm)

Analysis of Variance Table

Model 1: body.fat ~ age + BMI + chest + abdomen + hip + forearm + wrist
Model 2: body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + ankle + bicep + forearm + wrist
Model 3: body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1    157 2402.0
2    151 2346.2  6    55.857 0.5952 0.7338
3    150 2346.2  1     0.032 0.0021 0.9638
```

By using the smaller model model.lmstep, no essential information is lost, therefore it can be used for the prediction instead of model.lm.

```
> pred.lmstep <- predict(model.lmstep,newdata = fat[test,])
> cor(fat[test,"body.fat"],pred.lmstep)^2 # R^2 for test data

[1] 0.7372035

> mean((fat[test,"body.fat"]-pred.lmstep)^2) # MSE_test

[1] 16.87865
```

In this case, however, we seem to have almost the same performance as for the full model.

4.2.5 Best subset regression with Leaps and Bound algorithm

```
> library(leaps)
> lm.regsubset <- regsubsets(body.fat~., data=fat, nbest = 1, subset=train)
> summary(lm.regsubset)

Subset selection object
Call: regsubsets.formula(body.fat ~ ., data = fat, nbest = 1, subset = train)
14 Variables  (and intercept)
        Forced in Forced out
age         FALSE      FALSE
weight      FALSE      FALSE
height      FALSE      FALSE
BMI         FALSE      FALSE
neck        FALSE      FALSE
```

```
chest       FALSE      FALSE
abdomen     FALSE      FALSE
hip         FALSE      FALSE
thigh       FALSE      FALSE
knee        FALSE      FALSE
ankle       FALSE      FALSE
bicep       FALSE      FALSE
forearm     FALSE      FALSE
wrist       FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         age weight height BMI neck chest abdomen hip thigh knee ankle bicep
1  ( 1 ) " " " "    " "    " " " " " "   "*"     " " " " " "   " "  " "
2  ( 1 ) " " "*"    " "    " " " " " "   "*"     " " " " " "   " "  " "
3  ( 1 ) " " " "    " "    " " " " " "   "*"     "*" " " " "   " "  " "
4  ( 1 ) " " " "    "*"    " " " " " "   "*"     "*" " " " "   " "  " "
5  ( 1 ) " " " "    "*"    " " " " " "   "*"     "*" " " " "   " "  " "
6  ( 1 ) "*" " "    "*"    " " " " " "   "*"     "*" " " " "   " "  " "
7  ( 1 ) "*" " "    "*"    " " " " " "   "*"     "*" "*"  " "   " "  " "
8  ( 1 ) "*" " "    "*"    " " " " "*"   "*"     "*" "*"  " "   " "  " "
         forearm wrist
1  ( 1 ) " "     " "
2  ( 1 ) " "     " "
3  ( 1 ) " "     "*"
4  ( 1 ) " "     "*"
5  ( 1 ) "*"     "*"
6  ( 1 ) "*"     "*"
7  ( 1 ) "*"     "*"
8  ( 1 ) "*"     "*"
```

regsubsets() in `library(leaps)` provides the "best" model for different sizes of subsets. Here only one "best" model per subset size was considered. The ranking of the models is done using the BIC measure.

```
> plot(lm.regsubset)
```

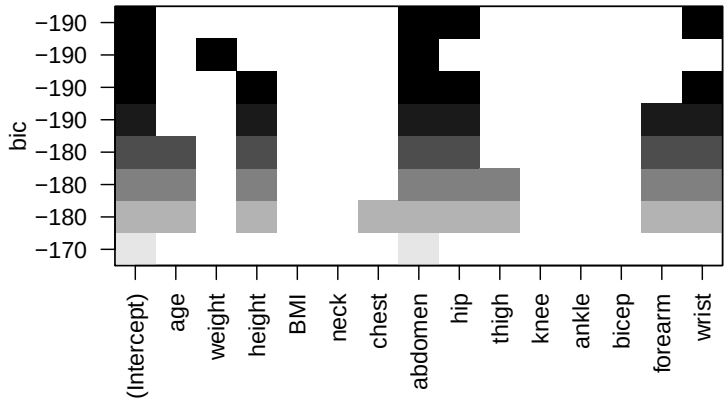

Figure 4.3: Model selection with **leaps()**

This plot shows the resulting models and their BIC value, coded in grey scale. The BIC does not improve after the fifth stage (starting from the bottom, see Figure 4.3). The optimal model can then be chosen from the models with "saturated" grey, and preferably that model is taken with the smallest number of variables.

A more intuitive plot might be that in Figure 4.4, where the BIC values are directly show. On the horizontal axis is the model size. The optimal model is the 2-variable model, with "weight" and "abdomen".

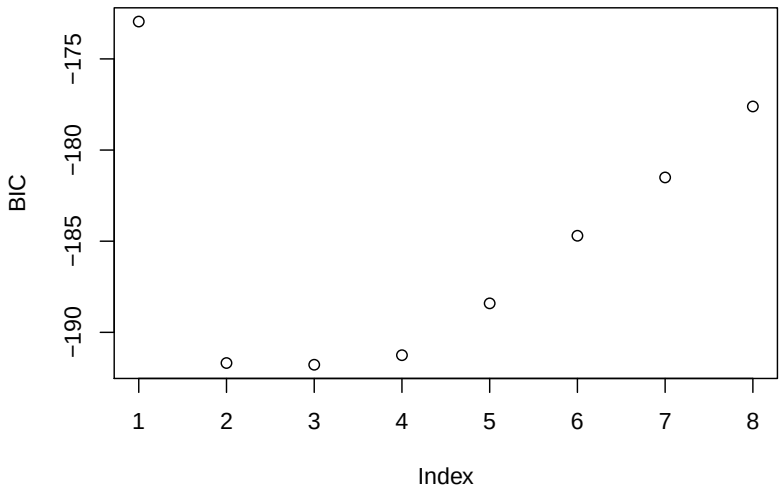

Figure 4.4: Model selection with **leaps()**

```
> modregsubset.lm <- lm(body.fat~weight+abdomen,data=fat,subset=train)
> pred.regsubset <- predict(modregsubset.lm,newdata = fat[test,])
> cor(fat[test,"body.fat"],pred.regsubset)^2 # R^2 for test data
[1] 0.7371287
> mean((fat[test,"body.fat"]-pred.regsubset)^2) # MSE_test
[1] 16.92699
```

Here again, the quality of the model does not quite change.

4.3 Methods using derived inputs as regressors in R

4.3.1 The problem of correlated regressors

The problem that occurs using correlated regressors is demonstrated using the following regression model:

$$y = X + \varepsilon, \quad \varepsilon \sim N(0, 0.25)$$

with the regressors

$$x_1 \sim U(0, 1)$$
$$x_2 \sim U(0, 1)$$
$$x_3 = x_1 + \nu_3, \quad \nu_3 \sim U(0, 0.1)$$

```
> x1 = runif(100)
> y = x1 + 0.5 * rnorm(100)
> summary(lm(y ~ x1))
Call:
lm(formula = y ~ x1)

Residuals:
     Min       1Q   Median       3Q      Max
-1.14872 -0.33275 -0.01504  0.31066  1.32760

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.2633     0.1219  -2.160   0.0332 *
```

```
x1            1.3987     0.2048   6.831 7.15e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5292 on 98 degrees of freedom
Multiple R-squared:  0.3226,        Adjusted R-squared:  0.3157
F-statistic: 46.67 on 1 and 98 DF,  p-value: 7.147e-10
```

```
> x2 = runif(100)
> summary(lm(y ~ x1 + x2))

Call:
lm(formula = y ~ x1 + x2)

Residuals:
     Min       1Q   Median       3Q      Max
-1.16328 -0.31706 -0.00776  0.30290  1.32335

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.22602    0.16554  -1.365    0.175
x1           1.38593    0.20922   6.624 1.95e-09 ***
x2          -0.06155    0.18402  -0.334    0.739
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5316 on 97 degrees of freedom
Multiple R-squared:  0.3234,        Adjusted R-squared:  0.3094
F-statistic: 23.18 on 2 and 97 DF,  p-value: 5.918e-09
```

- *Adding a highly correlated variable*

```
> x3 = x1 + 0.1 * runif(100)
> summary(lm(y ~ x1 + x3))

Call:
lm(formula = y ~ x1 + x3)

Residuals:
     Min       1Q   Median       3Q      Max
-1.14745 -0.31870 -0.01526  0.31558  1.33702

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.1730     0.1526  -1.134   0.2596
x1            3.1738     1.8166   1.747   0.0838 .
x3           -1.7631     1.7929  -0.983   0.3279
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5293 on 97 degrees of freedom
Multiple R-squared:  0.3293,        Adjusted R-squared:  0.3154
F-statistic: 23.81 on 2 and 97 DF,  p-value: 3.868e-09
```

The fit of the model (R squared) does not change when x_3 is added, but now none of the coefficients is significant. To estimate the amount of collinearity, one could look at the correlation between the x-variables. Another option is the function `alias()`, which identifies exact linear dependencies.

```
> alias(lm(y ~ x1 + x3))

Model :
y ~ x1 + x3
```

Here, there are no exact linear dependencies, but we can create some to test the method:

```
> alias(lm(body.fat~., data=fat))

Model :
body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist

> fat.mod <- fat
> fat.mod$nonsense <- fat$neck+2*fat$chest-3*fat$abdomen
> alias(lm(body.fat~., data=fat.mod))

Model :
body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist + nonsense

Complete :
         (Intercept) age weight height BMI neck chest abdomen hip thigh knee
nonsense  0           0   0      0      0   1    2    -3       0   0     0
         ankle bicep forearm wrist
nonsense  0     0     0       0
```

4.3.2 PCR

```
> library(pls)
> model.pcr <- pcr(body.fat~., data=fat, scale=TRUE, subset=train,
+                  validation="CV", segments=10, segment.type="random")
> summary(model.pcr)

Data:         X dimension: 165 14
       Y dimension: 165 1
Fit method: svdpc
Number of components considered: 14

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           7.508    5.893    4.896    4.854    4.636    4.612    4.502
adjCV        7.508    5.888    4.890    4.852    4.626    4.602    4.497
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       4.435    4.402    4.444     4.474     4.529     4.083     4.113
adjCV    4.417    4.389    4.430     4.458     4.512     4.065     4.095
       14 comps
CV        4.141
adjCV     4.121

TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X           65.45    76.56    82.56    87.26    90.43    92.92    95.17
body.fat    39.77    58.07    59.44    63.13    64.67    67.48    68.48
          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X           96.75    97.96     98.84     99.38     99.75     99.96    100.00
body.fat    68.51    68.56     68.77     68.79     74.43     74.44     74.47
```

Note that the explanatory variables need to be scaled, therefore the option `scale=TRUE`. The plot (see Figure 4.5 left) suggests to use 12 components. The resulting predictions for the training data are shown in Figure 4.5 (right).

The predictions from the test data and some evaluation measure are shown below.

```
> pred.pcr <- predict(model.pcr,newdata=fat[test,],ncomp=12)
> cor(fat[test,"body.fat"],pred.pcr)^2 # R^2 for test data

[1] 0.7385471

> mean((fat[test,"body.fat"]-pred.pcr)^2) # MSE_test

[1] 16.81261
```

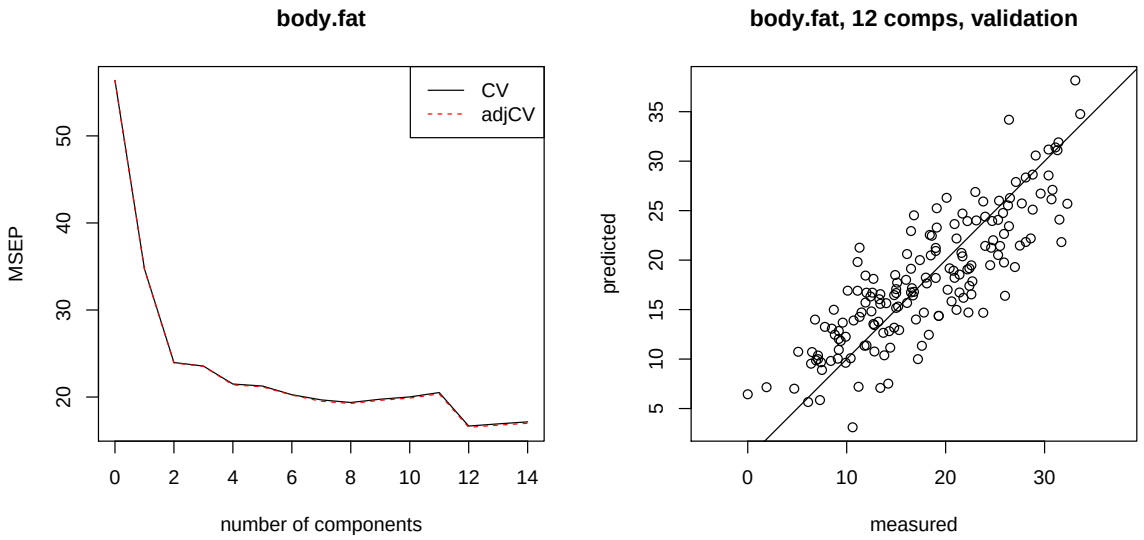

Figure 4.5: Validation plot, and measured versus predicted response for PCR.

4.3.3 PLS regression

```
> library(pls)
> model.pls <- plsr(body.fat~., data=fat, scale=TRUE, subset=train,
+                   validation="CV", segments=10, segment.type="random")
> summary(model.pls)
Data:         X dimension: 165 14
        Y dimension: 165 1
Fit method: kernelpls
Number of components considered: 14

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           7.508    5.491    4.553    4.391    4.258    4.198    4.124
adjCV        7.508    5.489    4.547    4.375    4.246    4.182    4.102
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       4.091    4.073    4.077     4.074     4.091     4.112     4.120
adjCV    4.073    4.057    4.061     4.059     4.075     4.094     4.101
       14 comps
CV        4.121
adjCV     4.102

TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X           64.68    75.99    80.62    84.30    87.41    90.23    92.93
body.fat    46.95    65.13    69.68    71.43    73.17    74.29    74.41
          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X           94.35    96.27     97.62     98.28     99.25     99.72    100.00
body.fat    74.45    74.45     74.45     74.46     74.46     74.47     74.47
```

Again, the explanatory variables need to be scaled, therefore the option `scale=TRUE`. The plot (see Figure 4.6 left) suggests to use something like 7 components. The resulting predictions for the training data are shown in Figure 4.6 (right).

The predictions from the test data and some evaluation measure are shown below.

```
> pred.pls <- predict(model.pls,newdata=fat[test,],ncomp=7)
> cor(fat[test,"body.fat"],pred.pls)^2 # R^2 for test data
[1] 0.7395234
> mean((fat[test,"body.fat"]-pred.pls)^2) # MSE_test
```

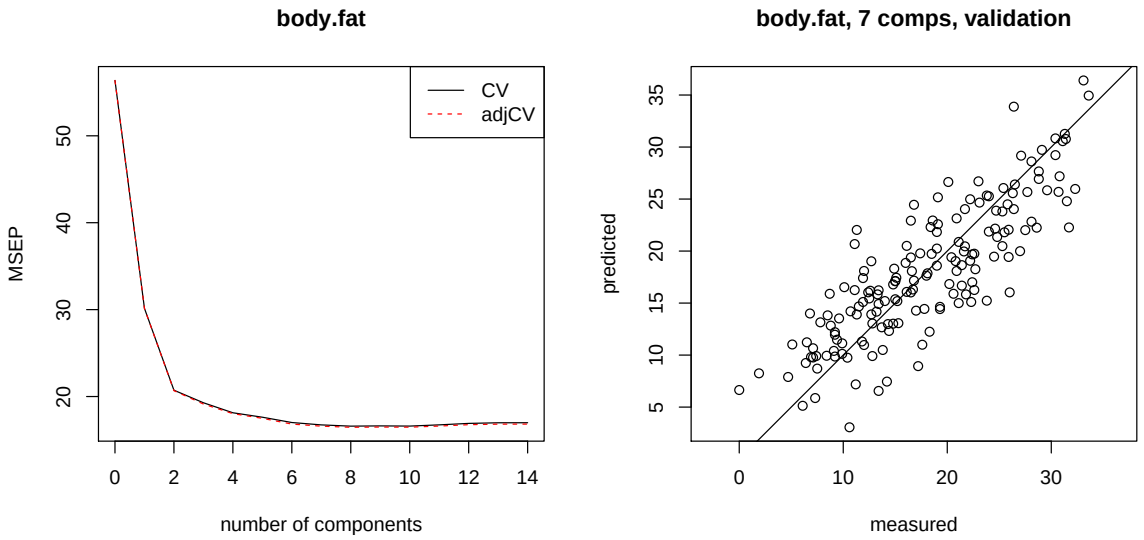

Figure 4.6: Validation plot, and measured versus predicted response for PLS.

```
[1] 16.74858
```

4.4 Shrinkage methods in R

4.4.1 Ridge regression

For a given grid of possible values of λ, the optimal tuning parameter has to be selected. Within the function `lm.ridge()`, generalized cross-validation (GCV) is used. This is a good approximation of leave-one-out (LOO) cross-validation by linear fits with quadratic errors. For a linear fit we have $\hat{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{y}$ with the corresponding transformation matrix $\boldsymbol{S}$. In classical linear regression $\boldsymbol{S}$ is the hat matrix. One can show that the following equation holds:

$$\frac{1}{n}\sum_{i=1}^{n}\left[y_i - \hat{f}^{-i}(\boldsymbol{x}_i)\right]^2 = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{y_i - \hat{f}(\boldsymbol{x}_i)}{1 - S_{ii}}\right]^2$$

Here, $\hat{f}^{-i}(\boldsymbol{x}_i)$ is a fit without observation i, $\hat{f}(\boldsymbol{x}_i)$ is a fit to all data, and S_{ii} is the i-th diagonal element of $\boldsymbol{S}$. The GCV approximation is

$$\text{GCV} = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{y_i - \hat{f}(\boldsymbol{x}_i)}{1 - trace(\boldsymbol{S})/n}\right]^2,$$

where $trace(\boldsymbol{S}) = \sum_{i=1}^{n} S_{ii}$. The idea is thus to replace S_{ii} by an average value, and since $trace(\boldsymbol{S})$ is the effective number of parameter, one has a computational advantage if this trace is easier to compute than the individual elements S_{ii}.

Now the model is fit to the training body fat data, and the resulting GCV errors are shown in Figure 4.7.

```
> library(MASS)
> model.ridge <- lm.ridge(body.fat~., data=fat, lambda=seq(0,15, by=0.2), subset=train)
> plot(model.ridge$lambda,model.ridge$GCV,type="l")
```

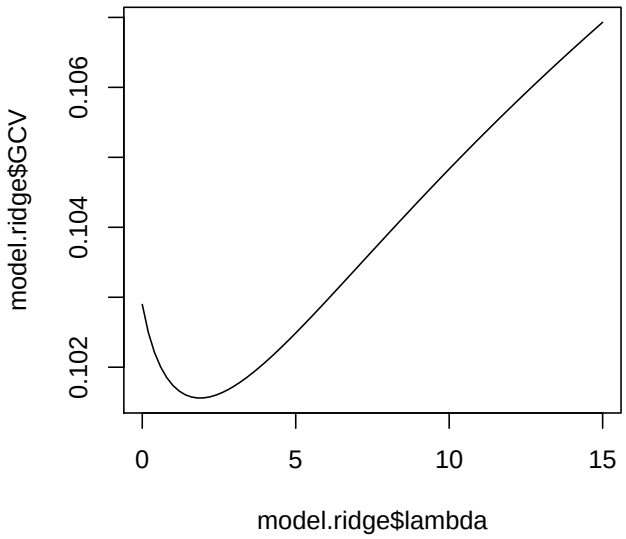

Figure 4.7: Optimal selection of the tuning parameter λ with GCV.

Figure 4.7 shows a very clear minimum at a specific value of λ. This value can be identified with the `select()` function.

```
> library(MASS)
> select(model.ridge)

modified HKB estimator is 2.168309
modified L-W estimator is 4.525899
smallest value of GCV  at 1.8

> lambda.opt <- model.ridge$lambda[which.min(model.ridge$GCV)]
```

This gives now the optimal tuning parameter, and some other characteristics which have been proposed for tuning parameter selection (HBK = Hoerl-Kennard coefficient, LW = Lawless-Wang coefficient) are returned as well.

One can also look at the whole solution path for each coefficient, depending on the values of the tuning parameter λ. This is shown in Figure 4.8, where each line corresponds to the resulting coefficients for one specific variable.

With the optimal values of λ, we want to predict the outcome variable for the test data. Let us first again compute the model for the optimized λ.

```
> # Prediction with Ridge:
> mod.ridge <- lm.ridge(body.fat~., data=fat, lambda = lambda.opt, subset=train)
```

Be careful here: The resulting coefficients are for **scaled** x-variables. For the prediction on unscaled data we need to back-transform them to the original scale. This is done with `coef()`, applied on the result object, where the scales are still stored, as well as the mean of the response, which is the intercept:

```
> mod.ridge$coef # coefficients for scaled x

       age      weight      height         BMI        neck       chest
0.78235616 -0.07457382 -0.34385066  1.03933922 -0.30950292 -0.63353943
   abdomen         hip       thigh        knee       ankle       bicep
```

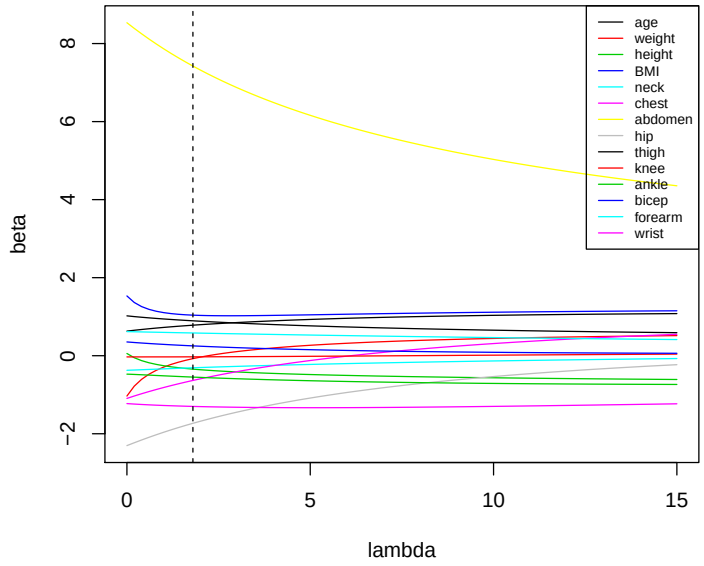

Figure 4.8: Ridge coefficients with varying λ. The optimal tuning parameter is indicated with the dashed line.

```
 7.42343629 -1.73224089  0.89443532 -0.02883642 -0.54814379  0.25192251
   forearm       wrist
 0.58262075 -1.29940964
> ridge.coef <- coef(mod.ridge)
> ridge.coef # coefficients in original scale + intercept
                     age       weight       height          BMI         neck
 3.367460057  0.061373666 -0.002680338 -0.135686607  0.314311555 -0.132886946
       chest      abdomen          hip        thigh         knee        ankle
-0.077906397  0.730676903 -0.259680747  0.173865322 -0.012053075 -0.396166795
       bicep      forearm        wrist
 0.083260870  0.278848086 -1.407762749
```

Now we predict the outcome variable for the test data. It turns out that the performance of the model is quite competitive.

```
> pred.ridge <- as.matrix(cbind(rep(1,length(test)),fat[test,-1]))%*%ridge.coef
> # plot(fat[test,"body.fat"],pred.ridge)
> # abline(c(0,1))
> cor(fat[test,"body.fat"],pred.ridge)^2 # R^2 for test data
         [,1]
[1,] 0.740998
> mean((fat[test,"body.fat"]-pred.ridge)^2) # MSE_test
[1] 16.78654
```

4.4.2 Lasso regression

We recommend to use the implementation in the package `glmnet`, which is very flexible. One could also do Ridge regression with this package, or extend Lasso regression to the ENET (Elastic Net), combining the L_1 and L_2 penalties.

Let us fit a Lasso model for the training data of body fat. The whole solution path for different values of the L_1 norm is computed, and it is visualized in Figure 4.9.

```
> library(glmnet)
> res <- glmnet(as.matrix(fat[train,-1]),fat[train,1])
> # print(res)
> plot(res)
```

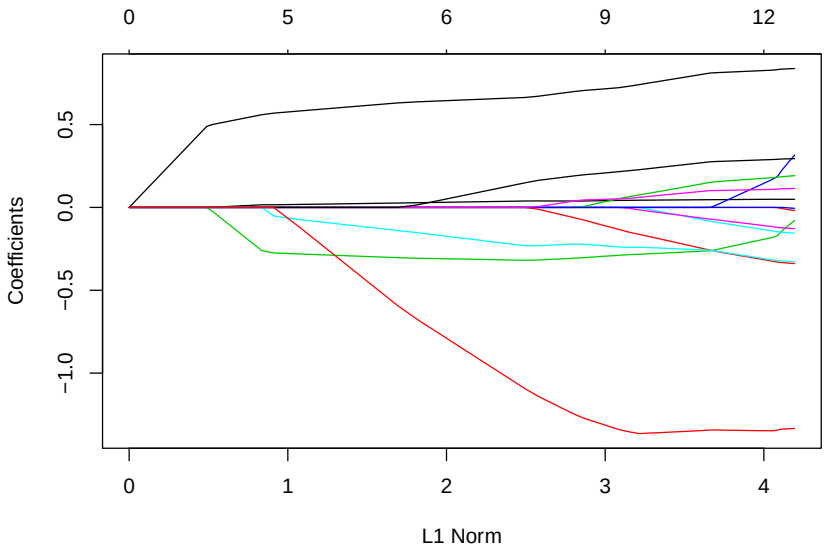

Figure 4.9: Lasso regression coefficients for varying values of the size of the L_1 norm. On top we can see the number of variables in the model.

To identify the appropriate tuning parameter, we run a cross-validation routine and plot the result, see Figure 4.10.

```
> res.cv <- cv.glmnet(as.matrix(fat[train,-1]),fat[train,1])
```

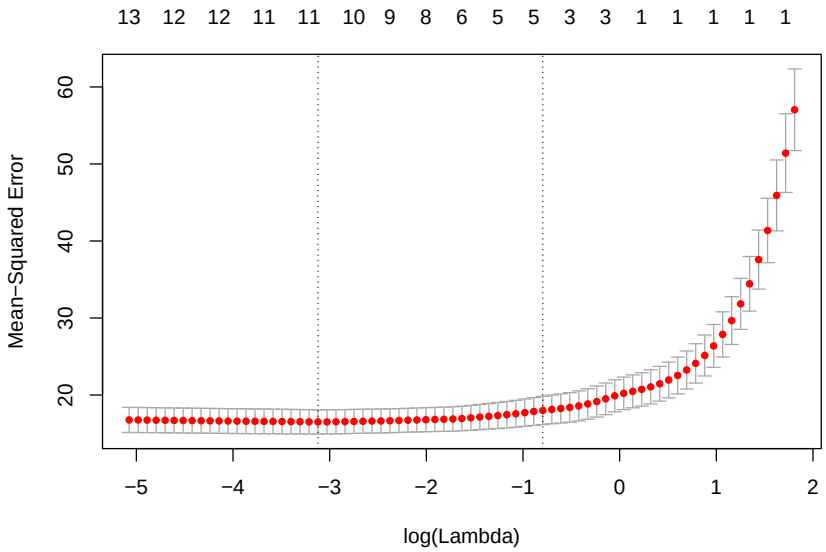

Figure 4.10: Cross-validated MSE for different values of λ in Lasso regression.

Figure 4.10 shows the MSE together with their standard errors. The left dashed line indicates the smallest MSE, and the right dashed line points at the optimal λ for with the MSE is still below the bound defined by the smallest MSE plus its standard error. This λ is selected if we go for the "one-standard error rule" (default). The resulting coefficients are:

```
> coef(res.cv,s="lambda.1se")

15 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept) -12.56008443
age           0.01790817
weight        .
height       -0.28074139
BMI           .
neck          .
chest         .
abdomen       0.58165319
hip           .
thigh         .
knee          .
ankle        -0.07271589
bicep         .
forearm       .
wrist        -0.11538101
```

We can see that the resulting coefficient vector is sparse, we obtain several zero entries and thus variable selection.

Finally, with the optimized model we predict the test data, and derive some evaluation characteristis. The model is competitive.

```
> pred.lasso <- predict(res.cv,newx=as.matrix(fat[test,-1]),s="lambda.1se")
> cor(fat[test,"body.fat"],pred.lasso)^2 # R^2 for test data

             1
[1,] 0.7299078

> mean((fat[test,"body.fat"]-pred.lasso)^2) # MSE_test

[1] 18.58005

> # plot(fat[test,"body.fat"],pred.lasso)
> # abline(c(0,1))
```

The resulting predictions from Ridge and Lasso regression are compared in Figure 4.11.

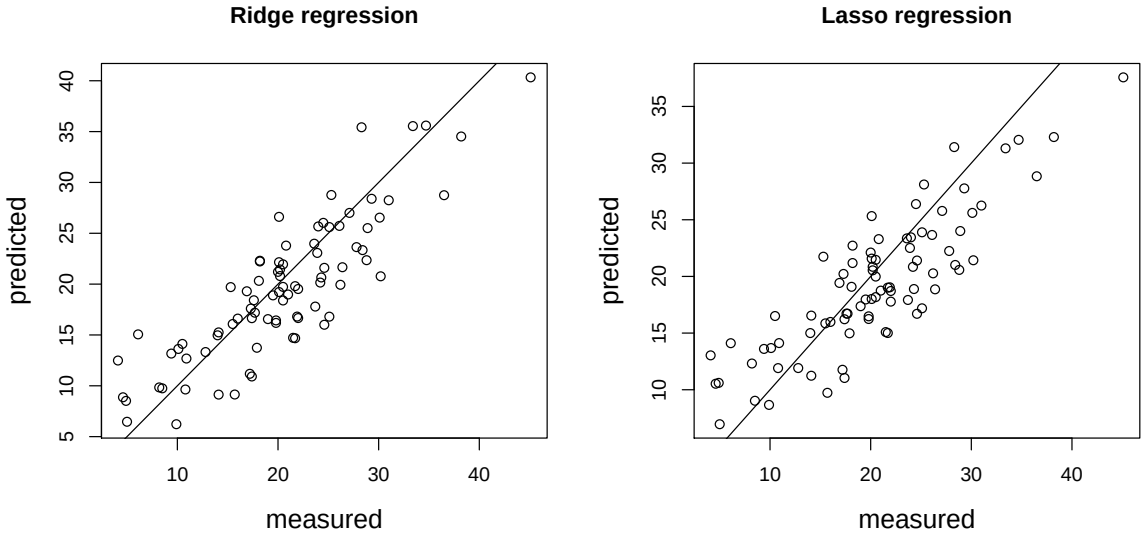

Figure 4.11: Measured versus predicted response for the test data, for Ridge (left) and LAsso (right) regression.

Part III

Linear classification

Chapter 5

Linear methods for classification

Linear classification tries to find linear functions of the form (1.1) which separate the observations into different classes. Observations within one group should have as many similar features as possible whereas observations of different groups should have little in common. If the predictor $G(\boldsymbol{x})$ takes on values in a discrete set $\mathcal{G}$, we can always divide the input space into different regions corresponding to the classification. The decision boundaries can either be smooth or rough. Assuming that we have K groups and the fitted linear model for the kth regressor variable is $\hat{f}_k(\boldsymbol{x}) = \hat{\beta}_{k0} + \hat{\boldsymbol{\beta}}_k^\top \boldsymbol{x}$. Then the decision boundary between class k and l is the set of points for which $\hat{f}_k(\boldsymbol{x}) = \hat{f}_l(\boldsymbol{x})$, that is, $\{\boldsymbol{x} : (\hat{\beta}_{k0} - \hat{\beta}_{l0}) + (\hat{\boldsymbol{\beta}}_k - \hat{\boldsymbol{\beta}}_l)^\top \boldsymbol{x} = 0\}$. New data points $\boldsymbol{x}$ can then be classified with this predictor.

5.1 Linear regression of an indicator matrix

Here each of the response categories is coded by an indicator variable. Thus if $\mathcal{G}$ has K classes, there will be K such indicators

$$y_k \quad \text{with } k = 1, 2, \ldots, K$$

with

$$y_k = \begin{cases} 1 & \text{for } G = k \\ 0 & \text{otherwise} \end{cases}$$

These response variables are collected in a vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_K)$, and the n training instances form an $(n \times K)$ indicator response matrix $\boldsymbol{Y}$, with 0/1 values as indicators for the class membership. We fit a linear regression model to each of the columns $\boldsymbol{y}_k$ of $\boldsymbol{Y}$ which is given by

$$\hat{\boldsymbol{\beta}}_k = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}_k \quad \text{with } k = 1, 2, \ldots, K$$

The $\hat{\boldsymbol{\beta}}_k$ can be combined in a $((p+1) \times K)$ matrix $\hat{\boldsymbol{B}}$ since

$$\begin{aligned} \hat{\boldsymbol{B}} &= (\hat{\boldsymbol{\beta}}_1, \hat{\boldsymbol{\beta}}_2, \ldots, \hat{\boldsymbol{\beta}}_K) \\ &= (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_K) \\ &= (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}. \end{aligned}$$

The estimated values are then

$$\begin{aligned} \hat{\boldsymbol{Y}} &= \boldsymbol{X} \hat{\boldsymbol{B}} \\ &= \boldsymbol{X} (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y} \end{aligned}$$

A new observation $\boldsymbol{x}$ can be classified as follows:

- Compute the K vector of the fitted values

$$\hat{f}(\boldsymbol{x}) = \left[(1, \boldsymbol{x}^\top)\hat{\boldsymbol{B}}\right]^\top = \left[\hat{f}_1(\boldsymbol{x}), \ldots, \hat{f}_K(\boldsymbol{x})\right]^\top$$

- identify the largest component $\hat{f}(\boldsymbol{x})$ and classify $\boldsymbol{x}$ accordingly:

$$\hat{G}(\boldsymbol{x}) = \underset{k \in \mathcal{G}}{\operatorname{argmax}}\, \hat{f}_k(\boldsymbol{x})$$

Section 6.1, page 55

5.2 Linear discriminant analysis (LDA)

5.2.1 Classical LDA

$\mathcal{G}$ consists of K classes. Let the probability of an observation belonging to class k be (the prior probability) π_k, $k = 1, 2, \ldots, K$ with $\sum_{k=1}^{K} \pi_k = 1$. Suppose $h_k(\boldsymbol{x})$ is the density function of $\boldsymbol{x}$ in class $G = k$. Then

$$P(G = k|\boldsymbol{x}) = \frac{h_k(\boldsymbol{x})\pi_k}{\sum_{l=1}^{K} h_l(\boldsymbol{x})\pi_l}$$

is the conditional probability that with a given observation $\boldsymbol{x}$ the random variable G is k. $h_k(\boldsymbol{x})$ is often assumed to be the density of a multivariate normal distribution φ_k

$$\varphi_k(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}_k|}} \exp\left\{-\frac{(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)}{2}\right\}$$

LDA arises in the special case when we assume that the classes have a common covariance $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$, $k = 1, 2, \ldots, K$.

Comparing these two classes, it is sufficient to look at the log-ratio

$$\begin{aligned}
\log \frac{P(G = k|\boldsymbol{x})}{P(G = l|\boldsymbol{x})} &= \log \frac{\varphi_k(\boldsymbol{x})\pi_k}{\varphi_l(\boldsymbol{x})\pi_l} = \log \frac{\varphi_k(\boldsymbol{x})}{\varphi_l(\boldsymbol{x})} + \log \frac{\pi_k}{\pi_l} \\
&= \log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_l)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_l) + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_l)
\end{aligned}$$

The decision boundary between the classes k and l is

$$P(G = k|\boldsymbol{x}) = P(G = l|\boldsymbol{x})$$

which is linear in $\boldsymbol{x}$ (in p dimensions this is a hyperplane).

From the log-ratio we get the **linear discriminant function:**

$$\begin{aligned}
\log \frac{P(G=k|\boldsymbol{x})}{P(G=l|\boldsymbol{x})} &= \log(1) = 0 = \log \frac{\varphi_k(\boldsymbol{x})\pi_k}{\varphi_l(\boldsymbol{x})\pi_l} \\
&= \log \varphi_k(\boldsymbol{x}) + \log \pi_k - \log \varphi_l(\boldsymbol{x}) - \log \pi_l \\
&= \log \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) + \log \pi_k \\
&- \log \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_l)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_l) - \log \pi_l \\
&= -\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} \underbrace{+\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log \pi_k}_{\delta_k(\boldsymbol{x})} \\
&\quad + \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} \underbrace{-\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_l + \frac{1}{2}\boldsymbol{\mu}_l^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_l - \log \pi_l}_{-\delta_l(\boldsymbol{x})}
\end{aligned}$$

The linear discriminant function

$$\delta_k(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log \pi_k$$

provides an equivalent description of the decision rule

$$G(\boldsymbol{x}) = \operatorname*{argmax}_k \delta_k(\boldsymbol{x})$$

The observation $\boldsymbol{x}$ is classified to the group where $\delta_k(\boldsymbol{x}),\ k = 1, 2, \ldots, K$ is the largest. $\boldsymbol{x}$ is classified to class k if

$$\delta_k(\boldsymbol{x}) > \delta_l(\boldsymbol{x}) \quad \Longleftrightarrow \quad \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_l) - \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_l)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_l) > \log \frac{\pi_l}{\pi_k}$$

The parameters of the distribution $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ as well as the prior probabilities π_k are usually unknown and need to be estimated from the training data:

$$\begin{aligned}
\hat{\pi}_k &= \frac{n_k}{n} \qquad \text{with } n_k \ldots \text{ amount of observations in group } k \\
\hat{\boldsymbol{\mu}}_k &= \sum_{g_i=k} \frac{\boldsymbol{x}_i}{n_k} \\
\hat{\boldsymbol{\Sigma}} &= \frac{1}{n-K} \sum_{k=1}^{K} \sum_{g_i=k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top
\end{aligned}$$

g_i indicates the true group number of observation $\boldsymbol{x}_i$. Using the linear discriminant function we now can estimate the group membership of $\boldsymbol{x}_i$. This gives a value for $\hat{G}(\boldsymbol{x}_i)$, which can now be compared to g_i. The aim is correct classification of as many observations as possible. The misclassification rate provides the relative amount of incorrectly classified observations.

Section 6.2.1, page 56

5.2.2 Quadratic discriminant analysis (QDA)

Just as in LDA we assume a prior probability π_k for class k, $k = 1, 2, \ldots, K$ with $\sum_{k=1}^{K} \pi_k = 1$. The conditional probability that G takes on the value k is

$$P(G = k|\boldsymbol{x}) = \frac{\varphi_k(\boldsymbol{x})\pi_k}{\sum_{l=1}^{K} \varphi_l(\boldsymbol{x})\pi_l}$$

(φ is the density of the multivariate normal distribution). QDA does not assume the covariance matrices to be equal, which complicates the formulas. After some calculation we obtain the quadratic discriminant function

$$\delta_k(\boldsymbol{x}) = -\frac{1}{2}\log|\boldsymbol{\Sigma}_k| - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) + \log \pi_k.$$

Observation $\boldsymbol{x}$ is classified to that group which yields the maximal value of the discriminant function. $\boldsymbol{\Sigma}_k$, $\boldsymbol{\mu}_k$ and π_k can be estimated from the training data.

Comparison QDA and LDA

With LDA we need to estimate much less parameters as with QDA (each covariance matrix $\hat{\boldsymbol{\Sigma}}_k$). Both methods work surprisingly well, even if the classes are not normally distributed and even if the equality of the covariance matrices is not given. The reason is most likely that the data can only support simple decision boundaries such as linear or quadratic, and the estimates provided via Gaussian models are stable.

Section 6.2.2, page 59

5.2.3 Regularized discriminant analysis

This method is a compromise between linear and quadratic discriminant analysis, that allows to shrink the separate covariances of QDA towards a common covariance as in LDA. These common (pooled) covariance matrices have the form

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{\sum_{k=1}^{K} n_k}\left(\sum_{k=1}^{K} n_k \hat{\boldsymbol{\Sigma}}_{\boldsymbol{k}}\right)$$

with n_k as the number of observations per group. With the pooled covariance matrices the regularized covariance matrices have the form

$$\hat{\boldsymbol{\Sigma}}_k(\alpha) = \alpha\hat{\boldsymbol{\Sigma}}_k + (1 - \alpha)\hat{\boldsymbol{\Sigma}}$$

with $\alpha \in [0, 1]$. α provides a compromise between LDA ($\alpha = 0$) and QDA ($\alpha = 1$). The idea is to keep the degrees of freedom flexible. α can be estimated using cross validation.

Section 6.2.3, page 59

5.3 Logistic regression

Logistic regression also deals with the problem of classifying observations that originate from 2 or more groups. The difference to the previous classification methods is that the output of logistic regression includes an inference statistic which provides information about which variables are well suitable for separating the groups, and which provide no contribution to this goal.

In logistic regression the posterior probabilities of the K classes are modeled by linear functions in $\boldsymbol{x}$, with the constraint that the probabilities remain in the interval $[0, 1]$ and that they sum up to 1. Let us consider the following models:

$$\begin{aligned}
\log \frac{P(G=1|\boldsymbol{x})}{P(G=K|\boldsymbol{x})} &= \beta_{10} + \boldsymbol{\beta}_1^\top \boldsymbol{x} \\
\log \frac{P(G=2|\boldsymbol{x})}{P(G=K|\boldsymbol{x})} &= \beta_{20} + \boldsymbol{\beta}_2^\top \boldsymbol{x} \\
&\vdots \\
\log \frac{P(G=K-1|\boldsymbol{x})}{P(G=K|\boldsymbol{x})} &= \beta_{(K-1)0} + \boldsymbol{\beta}_{K-1}^\top \boldsymbol{x}
\end{aligned}$$

Here we chose class K to be the denominator, but the choice of the denominator is arbitrary in the sense that the estimates are equivariant under that choice. After a few calculations we get

$$P(G=k|\boldsymbol{x}) = P(G=K|\boldsymbol{x}) \exp\left\{\beta_{k0} + \boldsymbol{\beta}_k^\top \boldsymbol{x}\right\} \quad \text{for } k = 1, 2, \ldots, K-1$$

$$\sum_{k=1}^{K} P(G=k|\boldsymbol{x}) = 1 = P(G=K|\boldsymbol{x}) \left[1 + \sum_{k=1}^{K-1} \exp\left\{\beta_{k0} + \boldsymbol{\beta}_k^\top \boldsymbol{x}\right\}\right]$$

$$\begin{aligned}
P(G=K|\boldsymbol{x}) &= \frac{1}{1 + \sum_{l=1}^{K-1} \exp\left\{\beta_{l0} + \boldsymbol{\beta}_l^\top \boldsymbol{x}\right\}} \\
P(G=k|\boldsymbol{x}) &= \frac{\exp\left\{\beta_{k0} + \boldsymbol{\beta}_k^\top \boldsymbol{x}\right\}}{1 + \sum_{l=1}^{K-1} \exp\left\{\beta_{l0} + \boldsymbol{\beta}_l^\top \boldsymbol{x}\right\}} \quad \text{for } k = 1, 2, \ldots, K-1
\end{aligned}$$

Special case of $K = 2$ classes: In this case, the model is

$$\log \frac{P(G=1|\boldsymbol{x})}{P(G=2|\boldsymbol{x})} = \beta_0 + \boldsymbol{\beta}_1^\top \boldsymbol{x}$$

or

$$\begin{aligned}
P(G=1|\boldsymbol{x}) &= \frac{\exp\{\beta_0 + \boldsymbol{\beta}_1^\top \boldsymbol{x}\}}{1 + \exp\{\beta_0 + \boldsymbol{\beta}_1^\top \boldsymbol{x}\}} \\
P(G=2|\boldsymbol{x}) &= \frac{1}{1 + \exp\{\beta_0 + \boldsymbol{\beta}_1^\top \boldsymbol{x}\}} \\
\underbrace{P(G=1|\boldsymbol{x})}_{p_1(\boldsymbol{x})} + \underbrace{P(G=2|\boldsymbol{x})}_{p_2(\boldsymbol{x})} &= 1
\end{aligned}$$

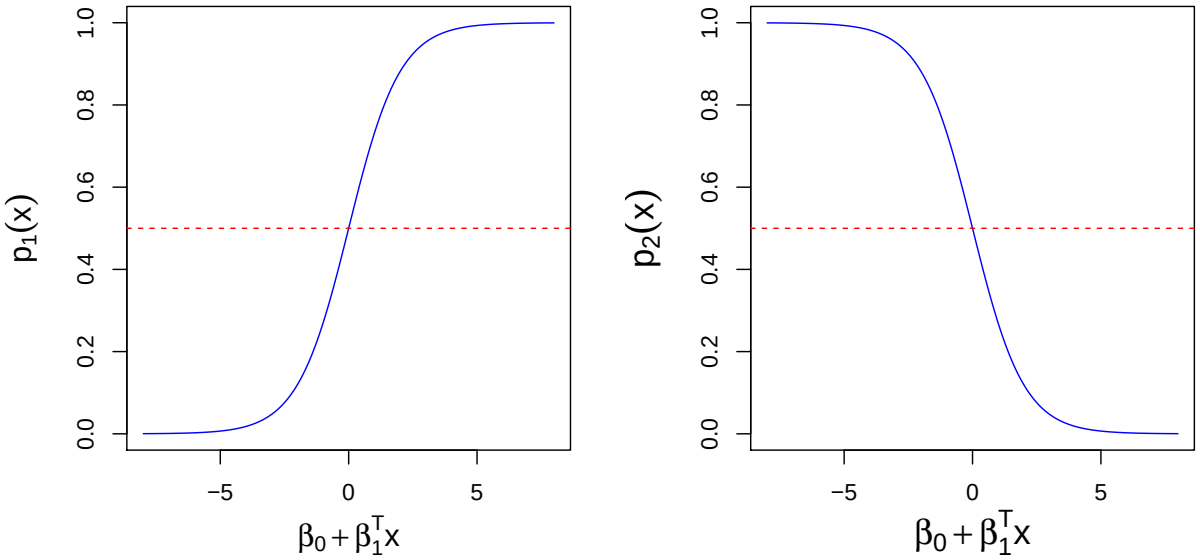

Figure 5.1: Logistic functions $p_1(\boldsymbol{x})$ (left) and $p_2(\boldsymbol{x})$ (right). Values above the dashed lines would be assigned to the corresponding groups.

Figure 5.1 visualizes the so-called *logistic functions*, i.e. the probabilities $p_1(\boldsymbol{x})$ (left) and $p_2(\boldsymbol{x})$ (right). A natural cut-off for the assignment of an observations to one of the groups would be 0.5 (dashed lines).

The parameters can be estimated using the ML method. The log-likelihood function is

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} \log p_{g_i}(\boldsymbol{x}_i; \boldsymbol{\beta}),$$

where

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \boldsymbol{\beta}_1 \end{pmatrix}$$

and $\boldsymbol{x}_i$ includes the intercept. p_{g_i} is the probability that observation $\boldsymbol{x}_i$ belongs to class $g_i = 1, 2$. With

$$p(\boldsymbol{x}; \boldsymbol{\beta}) = p_1(\boldsymbol{x}; \boldsymbol{\beta}) = 1 - p_2(\boldsymbol{x}; \boldsymbol{\beta})$$

and an indicator $y_i = 1$ for $g_i = 1$ and $y_i = 0$ for $g_i = 2$, $l(\boldsymbol{\beta})$ can be written as

$$\begin{aligned}
l(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \{y_i \log p(\boldsymbol{x}_i; \boldsymbol{\beta}) + (1 - y_i) \log [1 - p(\boldsymbol{x}_i; \boldsymbol{\beta})]\} \\
&= \sum_{i=1}^{n} \left\{ y_i \log \left[\frac{\exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i)}{1 + \exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i)} \right] + (1 - y_i) \log \left[1 - \frac{\exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i)}{1 + \exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i)} \right] \right\} \\
&= \sum_{i=1}^{n} \left\{ y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - y_i \log \left[1 + \exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i) \right] - (1 - y_i) \log \left[1 + \exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i \right] \right\} \\
&= \sum_{i=1}^{n} \left\{ y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - \log \left[1 + \exp(\boldsymbol{\beta}^\top \boldsymbol{x}_i) \right] \right\}
\end{aligned}$$

To maximize the log-likelihood we set its derivative equal to zero. These score equations are $p+1$ nonlinear equations in $\boldsymbol{\beta}$ of the form

$$\begin{aligned}\frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \sum_{i=1}^{n}\left\{y_i\boldsymbol{x}_i - \frac{1}{1+\exp(\boldsymbol{\beta}^\top\boldsymbol{x}_i)}\exp(\boldsymbol{\beta}^\top\boldsymbol{x}_i)\boldsymbol{x}_i\right\}\\ &= \sum_{i=1}^{n}\left[y_i - p(\boldsymbol{x}_i;\boldsymbol{\beta})\right]\boldsymbol{x}_i = \mathbf{0}.\end{aligned}$$

To solve these equations, we use the *Newton-Raphson algorithm*, which requires the second derivative or Hessian matrix:

$$\frac{\partial^2 l(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^\top} = \ldots = -\sum_{i=1}^{n} p(\boldsymbol{x}_i;\boldsymbol{\beta})[1-p(\boldsymbol{x}_i;\boldsymbol{\beta})]\boldsymbol{x}_i\boldsymbol{x}_i^\top$$

Starting with $\boldsymbol{\beta}_{old}$ we get

$$\boldsymbol{\beta}_{new} = \boldsymbol{\beta}_{old} - \left(\frac{\partial^2 l(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^\top}\right)^{-1}\frac{\partial l(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}},$$

where the derivatives are evaluated at $\boldsymbol{\beta}_{old}$.
Matrix notation provides a better insight in that method:

$\boldsymbol{y}\ldots(n\times 1)$ vector of the y_i
$\boldsymbol{X}\ldots(n\times(p+1))$ matrix of observations $\boldsymbol{x}_i$
$\boldsymbol{p}\ldots(n\times 1)$ vector of estimated probabilities $p(\boldsymbol{x}_i,\boldsymbol{\beta}_{old})$
$\boldsymbol{W}\ldots(n\times n)$ diagonal matrix with weights $p(\boldsymbol{x}_i,\boldsymbol{\beta}_{old})(1-p(\boldsymbol{x}_i,\boldsymbol{\beta}_{old}))$ in the diagonal

and thus:

$$\begin{aligned}\frac{\partial l(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} &= \boldsymbol{X}^\top(\boldsymbol{y}-\boldsymbol{p})\\ \frac{\partial^2 l(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^\top} &= -\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X}\end{aligned}$$

The Newton-Raphson algorithm has the form

$$\begin{aligned}\boldsymbol{\beta}_{new} &= \boldsymbol{\beta}_{old} + (\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top(\boldsymbol{y}-\boldsymbol{p})\\ &= (\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{W}(\boldsymbol{X}\boldsymbol{\beta}_{old} + \boldsymbol{W}^{-1}(\boldsymbol{y}-\boldsymbol{p}))\\ &= \underbrace{(\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{W}}_{\text{weighted LS}}\boldsymbol{z}\end{aligned}$$

with the adjusted response

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{\beta}_{old} + \underbrace{\boldsymbol{W}^{-1}(\boldsymbol{y}-\boldsymbol{p})}_{\text{adjustment}}).$$

This algorithm is also referred to as IRLS (iteratively reweighted LS), since each iteration solves the weighted least squares problem

$$\boldsymbol{\beta}_{new} \longleftarrow \underset{\boldsymbol{\beta}}{\operatorname{argmin}}(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{\beta})^\top\boldsymbol{W}(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{\beta}).$$

Some remarks:

- $\boldsymbol{\beta} = \mathbf{0}$ is a good starting value for the iterative procedure, although convergence is never guaranteed.
- For $K \geq 3$ the Newton-Raphson algorithm can also be expressed as an IRLS algorithm, but in this case $\boldsymbol{W}$ is no longer a diagonal matrix.
- The logistic regression is mostly used for modeling and inference. The goal is to understand the role of the input variables in explaining the outcome.

Comparison of logistic regression and LDA

Logistic regression uses

$$\log \frac{P(G = k|\boldsymbol{x})}{P(G = q|\boldsymbol{x})} = \beta_{k0} + \boldsymbol{\beta}_k^\top \boldsymbol{x},$$

whereas LDA uses

$$\begin{aligned} \log \frac{P(G = k|\boldsymbol{x})}{P(G = K|\boldsymbol{x})} &= \log \frac{\pi_k}{\pi_K} - \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_K)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_K) + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_K) \\ &= \alpha_{k0} - \boldsymbol{\alpha}_k^\top \boldsymbol{x} \end{aligned}$$

The linearity in $\boldsymbol{x}$ in LDA is achieved by:

- the assumption of equal covariance matrices
- the assumption of multivariate normally distributed groups

Even though the models have the same form, the coefficients are estimated differently. The joint distribution of $\boldsymbol{x}$ and G can be expressed as

$$P(\boldsymbol{x}, G = k) = P(\boldsymbol{x})P(G = k|\boldsymbol{x})$$

Logistic regression does not specify $P(\boldsymbol{x})$, LDA assumes a mixture model (with φ as a normal distribution):

$$P(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \varphi(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma})$$


Section 6.3, page 59

Chapter 6

Linear methods for classification in R

6.1 Linear regression of an indicator matrix in R

- *Classification with indicator matrix based on the "Pima Indian" data*

```
> library(mlbench)
> data(PimaIndiansDiabetes2)
> #plot(PimaIndiansDiabetes2)
> pid <- na.omit(PimaIndiansDiabetes2)
```

The data consist of a population of 392 women with "Pima Indian" heritage, who live in the area of Phoenix, Arizona.They were tested for diabetes. The goal is to get a classification rule for the diagnosis of diabetes. The variables are:

- `npreg`: number of pregnancies
- `glu`: plasma glucose concentration (glucose tolerance test)
- `bp`: diastolic blood pressure (mm Hg)
- `skin`: triceps skin thickness (mm)
- `bmi`: BMI
- `ped`: diabetes pedigree function
- `age`: age in years
- `type`: "pos" or "neg" for diabetes diagnosis

Generation of the indicator matrix for regression:

```
> pidind <- pid
> ind <- ifelse(pid$diabetes=="neg",0,1)
> pidind$diabetes <- cbind(1-ind,ind)
```

It would be sufficient to consider only one indicator variable since we have a symmetric problem.

Random selection (using a fixed random seed) of the training data, and fitting of a linear model:

```
> set.seed(101)
> train <- sample(1:nrow(pid), 300)
> mod.ind <- lm(diabetes~., data=pidind[train,])
> mod.ind
```

```
Call:
lm(formula = diabetes ~ ., data = pidind[train, ])

Coefficients:
                        ind
(Intercept)   2.059e+00  -1.059e+00
pregnant     -1.292e-02   1.292e-02
glucose      -6.046e-03   6.046e-03
pressure      3.797e-04  -3.797e-04
triceps      -3.315e-03   3.315e-03
insulin       6.719e-05  -6.719e-05
mass         -7.820e-03   7.820e-03
pedigree     -8.852e-02   8.852e-02
age          -7.654e-03   7.654e-03
```

Fitting a linear model to the data yields a regression coefficients $\hat{\boldsymbol{\beta}}_k$ for each $\boldsymbol{y}_k$.

The model is applied to the test data, and the resulting misclassification rate is computed.

```
> mod.pred <- predict(mod.ind, newdata=pidind[-train,])
> class.pred <- apply(mod.pred,1,which.max) # class prediction
> TAB <- table(pid$diabetes[-train], class.pred)
> mklrate<-1-sum(diag(TAB))/sum(TAB)
> mklrate
[1] 0.2391304
```

The resulting misclassification rate is 0.239.
The misclassification rates of different classification methods will be collected in a table and then compared and analyzed throughout the rest of this manuscript.

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239						

6.2 Linear Discriminant Analysis in R

6.2.1 Classical LDA

- *LDA for 2-dimensional multivariate normally distributed data with parameters* $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}$

```
> mu1<-c(0,0)
> mu2<-c(3.5,1)
> sig<-matrix(c(1.5,1,1,1.5),ncol=2)
```

The ellipses in Figure 6.1 are so-called tolerance ellipses which (in case of multivariate normal distribution) include the inner 95% of the data of each group. The LDA separation line was determined using the known population parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$ and $\boldsymbol{\Sigma}$. The command **lda()** can be found in `library(MASS)`.

- *Change of the prior probabilities – see Figure 6.2*

  When changing the prior probabilities, the LDA separation line is moved towards the group with smaller prior probability (see Figure 6.2). This is done because LDA minimizes the probability of misclassification.

- *Classification of the PimaIndianDiabetes data with* **lda()**

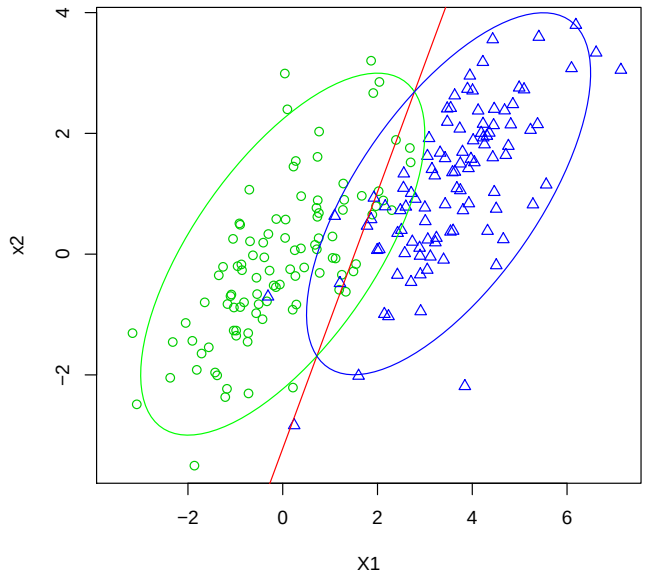

Figure 6.1: LDA separation line using the population parameters and $\pi_1 = \pi_2$

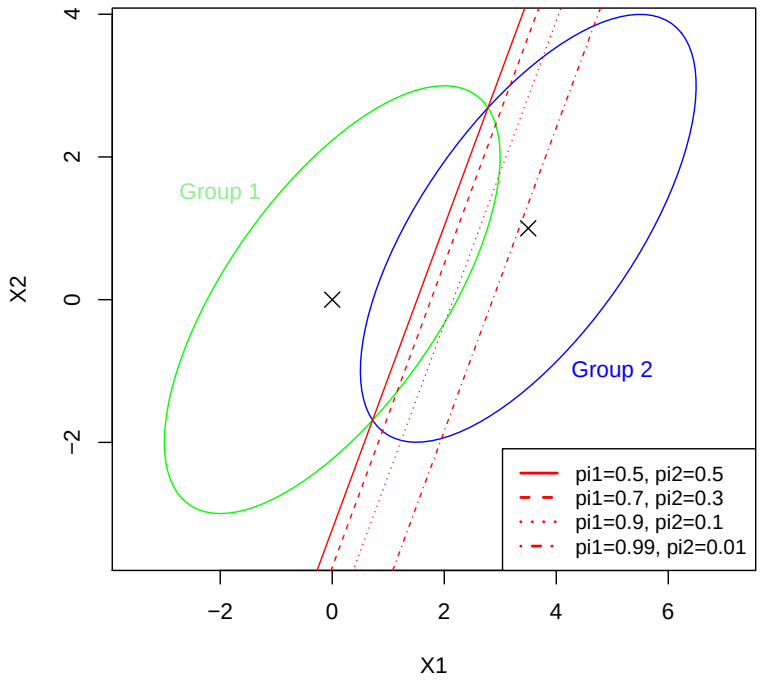

Figure 6.2: Change of the prior probabilities

An advanced form of an evaluation (we could think of an even more advanced form) is as follows:

```
> library(MASS)
> mypred <- function(object, newdata) UseMethod("mypred", object)
> mypred.lda <- function(object, newdata){
+              predict(object, newdata = newdata)$class
+ }
> library(ipred)
> CEE <-  control.errorest(k = 5, nboot=10)
> ldacvest <- errorest(diabetes~., data=pid[train,], model=lda, predict=mypred,
+          est.para=CEE)
> ldacvest

Call:
errorest.data.frame(formula = diabetes ~ ., data = pid[train,
    ], model = lda, predict = mypred, est.para = CEE)

         5-fold cross-validation estimator of misclassification error

Misclassification error:  0.2267

> ldabest <- errorest(diabetes~., data=pid[train,], model=lda,predict=mypred,
+          estimator="boot", est.para=CEE)
> ldabest

Call:
errorest.data.frame(formula = diabetes ~ ., data = pid[train,
    ], model = lda, predict = mypred, estimator = "boot", est.para = CEE)

         Bootstrap estimator of misclassification error
         with 10 bootstrap replications

Misclassification error:  0.2391
Standard deviation: 0.0124
```

The command **errorest()** can be found in `library(ipred)` and can be used to estimate the misclassification rate with cv or bootstrap.

A simple evaluation based on just one training and test data set can give quite unreliable results:

```
> set.seed(101)
> train <- sample(1:nrow(pid), 300)
> mod.lda <- lda(diabetes~.,data=pid[train,])
> TAB <- table(pid[-train,]$diabetes,mypred(mod.lda,pid[-train,]))
> mkrlda <- 1-sum(diag(TAB))/sum(TAB)
> mkrlda

[1] 0.2391304
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239					

6.2.2 QDA

- *Classification of the PimaIndianDiabetes data with* **qda()**

```
> set.seed(101)
> train <- sample(1:nrow(pid), 300)
> mod.qda <- qda(diabetes~.,data=pid[train,])
> predictqda <- predict(mod.qda, pid[-train,])
> TAB <- table(pid$diabetes[-train],predictqda$class)
> mkrqda <- 1-sum(diag(TAB))/sum(TAB)
> mkrqda
[1] 0.25
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25				

6.2.3 Regularized discriminant analysis

- *Classification of the PimaIndianDiabetes data with* **rda()**

```
> library(klaR)
> mod.rda <- rda(diabetes~.,data=pid[train,])
> predictrda <- predict(mod.rda, pid[-train,])
> TAB <- table(pid$diabetes[-train], predictrda$class)
> mkrrda <- 1-sum(diag(TAB))/sum(TAB)
> mkrrda
[1] 0.2391304
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.239			

6.3 Logistic regression in R

- *Fit of a model with* **glm()**

  Logistic regression can be carried out with the function **glm()** under the model of the binomial family. Syntax and interpretation of the inference statistic are in analogy to **lm()**. The tests for the single coefficients are similar to those from the linear model, but here they are based on the asymptotic normality of the parameter estimates that are MLEs. The standard errors are based on the second derivative of the log-likelihood function at the maximum, which is an indication of how rapidly the function decreases as one moves away from the peak.

```
> set.seed(101)
> train <- sample(1:nrow(pid), 300)
> modelglm <- glm(diabetes~.,data=pid, family=binomial, subset=train)
> summary(modelglm)

Call:
glm(formula = diabetes ~ ., family = binomial, data = pid, subset = train)
```

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4400  -0.6729  -0.3530   0.6820   2.4859

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -9.7862578  1.3765057  -7.109 1.16e-12 ***
pregnant     0.0813129  0.0637407   1.276   0.2021
glucose      0.0370238  0.0066143   5.598 2.17e-08 ***
pressure    -0.0045533  0.0135442  -0.336   0.7367
triceps      0.0219238  0.0191473   1.145   0.2522
insulin     -0.0005762  0.0014721  -0.391   0.6955
mass         0.0608347  0.0300131   2.027   0.0427 *
pedigree     0.5898194  0.4901525   1.203   0.2288
age          0.0444517  0.0209460   2.122   0.0338 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 380.51  on 299  degrees of freedom
Residual deviance: 262.09  on 291  degrees of freedom
AIC: 280.09

Number of Fisher Scoring iterations: 5
```

The deviances are the negative contributions of each observation to the log-likelihood function. The "Null deviance" refers to the empty model, the "Residual deviance" to the full model.

- *Model selection with* **step()**: This is done in the same spirit as for the linear model.

```
> mod.glm <- step(modelglm,direction="both")

Start:  AIC=280.09
diabetes ~ pregnant + glucose + pressure + triceps + insulin +
    mass + pedigree + age

           Df Deviance    AIC
- pressure  1   262.20 278.20
- insulin   1   262.25 278.25
- triceps   1   263.41 279.41
- pedigree  1   263.61 279.61
- pregnant  1   263.73 279.73
<none>          262.09 280.09
- mass      1   266.27 282.27
- age       1   266.91 282.91
- glucose   1   299.22 315.22

                    ⋮

Step:  AIC=274.91
diabetes ~ glucose + mass + age

           Df Deviance    AIC
<none>          266.91 274.91
+ triceps   1   265.34 275.34
+ pregnant  1   265.44 275.44
+ pedigree  1   265.50 275.50
+ insulin   1   266.70 276.70
+ pressure  1   266.78 276.78
- mass      1   280.12 286.12
- age       1   285.74 291.74
- glucose   1   315.97 321.97
```

The result after stepwise logistic regression is a smaller model that should be able to have a better prediction performance than the full model. The inference statistic tells which variables are significantly contributing to the group separation:

```
> summary(mod.glm)

Call:
glm(formula = diabetes ~ glucose + mass + age, family = binomial,
    data = pid, subset = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6686  -0.6674  -0.3642   0.6503   2.5019

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -9.931492   1.205761  -8.237  < 2e-16 ***
glucose      0.035407   0.005616   6.304 2.90e-10 ***
mass         0.078319   0.022551   3.473 0.000515 ***
age          0.063726   0.015525   4.105 4.05e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 380.51  on 299  degrees of freedom
Residual deviance: 266.91  on 296  degrees of freedom
AIC: 274.91

Number of Fisher Scoring iterations: 5
```

The two resulting models can be compared within a test which is making use of the fact the approximately, the sum of squared deviances is χ^2 distributed with $n-(p+1)$ degrees of freedom, where p is the number of explanatory variables.

```
> anova(mod.glm,modelglm,test="Chisq")

Analysis of Deviance Table

Model 1: diabetes ~ glucose + mass + age
Model 2: diabetes ~ pregnant + glucose + pressure + triceps + insulin +
    mass + pedigree + age
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1       296     266.91
2       291     262.09  5   4.8196   0.4383
```

The null hypothesis is that the smaller Model 1 is the true one, and this cannot be rejected.

- *Prediction of the class membership and presentation of the results, see Figure 6.3*

```
> plot(predict(mod.glm, pid[-train,]),col= as.numeric(pid$diabetes[-train])+2)
```

Note that by default (of the function `predict.glm()`), the predictions are returned in the scale of the linear predictor, and thus zero is the decision boundary. One could also get predictions in the scale of the response variable (with `type="response"`).

- *Comparison of LDA and logistic regression*
- Logistic regression makes no assumption about the distribution. LDA assumes Gaussian distributions with equal covariances.
- A comparison of the LDA and logistic regression outcome is in Figure 6.4.

```
> modlda <- lda(diabetes~., data = pid[train,])
> plot(predict(mod.glm, pid[-train,]), col=as.numeric(pid$diabetes[-train])+2,
+      pch=as.numeric(predict(modlda,pid[-train,])$class))
```

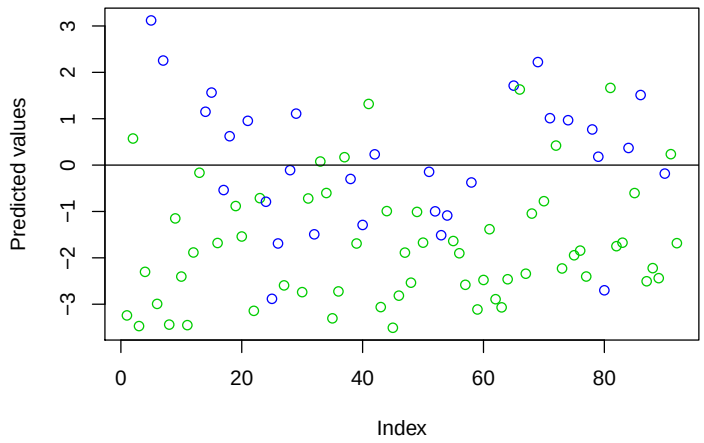

Figure 6.3: Prediction of the group memberships; the line is the separation from logistic regression, the color is the true group membership

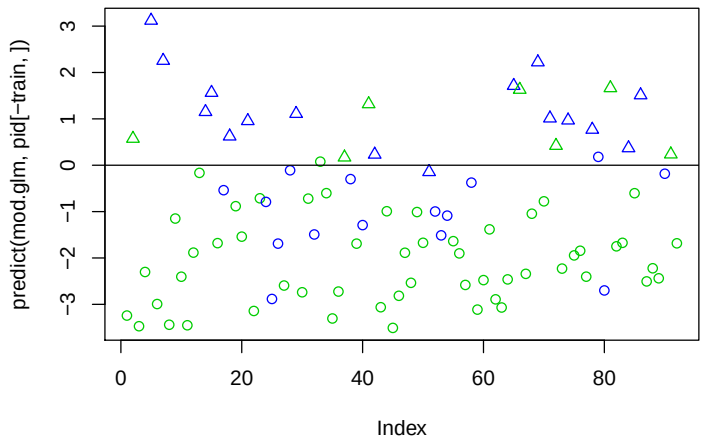

Figure 6.4: LDA (symbols) versus logistic regression (line); the color corresponds to the true classes

- *Prediction of the PimaIndianDiabetes data with logistic regression*

Here are again two evaluation schemes, the first based on cross-validation, the second based on bootstrap, and the final simple evaluation based on just one training and test set:

```
> library(ipred)
> mypred <- function(object, newdata) UseMethod("mypred", object)
> mypred.glm = function(object, newdata){
+     LEV = levels(object$model[,1])
+     as.factor(LEV[(predict(object, newdata=newdata, type="response")>0.5)+1])
+ }
> CEE <- control.errorest(k = 5, nboot=10)
> logcvest <- errorest(diabetes~., data=pid[train,], model=glm,
+                      family=binomial(),  predict=mypred, est.para=CEE)
> logcvest

Call:
errorest.data.frame(formula = diabetes ~ ., data = pid[train,
    ], model = glm, predict = mypred, est.para = CEE, family = binomial())

         5-fold cross-validation estimator of misclassification error

Misclassification error:  0.2133

> logbest <- errorest(diabetes~., data=pid[train,], model=glm,
+                     family=binomial(),  predict=mypred, estimator="boot", est.para=CEE)
> logbest

Call:
errorest.data.frame(formula = diabetes ~ ., data = pid[train,
    ], model = glm, predict = mypred, estimator = "boot", est.para = CEE,
    family = binomial())

         Bootstrap estimator of misclassification error
         with 10 bootstrap replications

Misclassification error:  0.2355
Standard deviation: 0.0103
```

Simpler (but less reliable) evaluation of the misclassification rate:

```
> TAB <- table(pid$diabetes[-train],mypred(mod.glm, pid[-train,]))
> mkrlog <- 1-sum(diag(TAB))/sum(TAB)
> mkrlog

[1] 0.25
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.239	0.25		

Part IV

Nonlinear methods

So far we have used linear models for both regression and classification because

- they are easy to interpret and have a closed solution;
- with small n and/or large p, linear models are often the only possibility to avoid overfitting.

However, in reality there is often no linear relation, and the errors are not normally distributed. Methods that are not based on linearity are:

- Generalized Linear Models (GLM): The expansion of normally distributed errors to the family of exponential distributions like the Gamma or Poisson distribution
- Mixed models: population comes from k different latent classes, which have different parameters for the same regression model
- Nonlinear regression: parametric nonlinear relation between regressor and regressand, e.g. $y = ae^{bx} + \varepsilon$

Chapter 7

Basis expansions

The idea is to augment/replace the vectors of inputs $\boldsymbol{x}$ with transformations of $\boldsymbol{x}$, and then use linear models in this new space of derived input features. Denote by $h_m(\boldsymbol{x}) : \mathbb{R}^p \longmapsto \mathbb{R}$ the mth transformation of $\boldsymbol{x}$, $m = 1, \ldots, M$. We then model

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}).$$

Some widely used examples are:

- $h_m(\boldsymbol{x}) = x_m, \quad m = 1, \ldots, p \quad \ldots$ describes the original linear model
- $h_m(\boldsymbol{x}) = x_j^2$ or $h_m(\boldsymbol{x}) = x_j x_k \quad \ldots$ quadratic transformation
- $h_m(\boldsymbol{x}) = \log(x_i), \sqrt{x_j} \quad \ldots$ nonlinear transformation
- $h_m(\boldsymbol{x}) = I(L_m \leq x_k < U_m) \quad \ldots$ indicator function, results in models with a constant contribution for x_k in the interval $[L_m, U_m)$, or piecewise constant in case of more non-overlapping regions.

7.1 Interpolation with splines

A spline is created by joining together several functions [compare Hansen et al., 2006]. The names comes from a tool called "spline" (a tool for curves). This thin flexible rod is fixed by weights and then used to draw curves through the given points. Since polynomials are one of the easiest functions, they are often preferred as basic elements for splines.

Piecewise polynomials

We assume x to be univariate. A piecewise polynomial function $f(x)$ is obtained by dividing the domain of x into $k+1$ disjoint intervals and defining for each interval $(-\infty, \xi_1)$, $[\xi_1, \xi_2)$, $\ldots$, $[\xi_{k-1}, \xi_k)$, $[\xi_k, \infty)$ an own polynomial function of order $\leq M$. The boundaries of the intervals are called *knots*. Piecewise constant functions are the easiest of all piecewise polynomials. Piecewise polynomials of order $M = 2, 3, 4$ are called piecewise linear (quadratic, cubic, etc.) polynomials. To determine a piecewise polynomial function of order M with k knots $\xi_1, \ldots, \xi_k$ we need $M(k+1)$ parameters, since each of the $k+1$ polynomials consists of M coefficients.

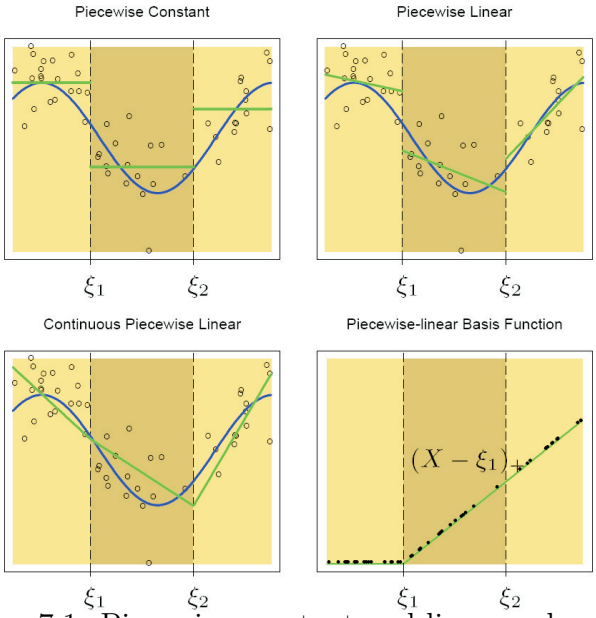

Figure 7.1: Piecewise constant and linear polynomials

Figure 7.1 shows data generated from the model of the blue continuous line with additional random noise. The data range is split into 3 regions, thus we obtain the knots ξ_1 and ξ_2. In the upper left picture we fit in each interval a constant function. The basis functions are thus:

$$h_1(x) = I(x < \xi_1), \quad h_2(x) = I(\xi_1 \leq x < \xi_2), \quad h_3(x) = I(\xi_2 \leq x)$$

It is easy to see that the LS solutions for the model $f(x) = \sum_{m=1}^{3} \beta_m h_m(x)$ are the arithmetic means $\hat{\beta}_m = \bar{y}_m$ of the y-values in each region.

Figure 7.1 upper right shows a piecewise fit by linear functions. Thus we need 3 additional basis functions, namely $h_{m+3} = h_m x$ for $m = 1, 2, 3$. In the lower left plot we also use piecewise linear functions but with the constraints of continuity at the knots. These constraints also lead to constraints on the parameters. For example, at the first knot we require $f(\xi_1^-) = f(\xi_1^+)$, and this means that $\beta_1 + \xi_1\beta_4 = \beta_2 + \xi_1\beta_5$. Thus the number of parameters is reduced by 1, for 2 knots by 2, and we end up with 4 free parameters in the model.

These basis functions and the constraints can be obtained in a more direct way by using the following definitions:

$$h_1(x) = 1, \quad h_2(x) = x, \quad h_3(x) = (x - \xi_1)_+, \quad h_4(x) = (x - \xi_2)_+$$

where t_+ denotes the positive part. The function h_3 is shown in the lower right panel of Figure 7.1.

Splines

A spline of order M with knots $\xi_i,\ i = 1, \ldots, k$ is a piecewise polynomial function of order M which has continuous derivatives up to order $M - 2$. The general form of the basis functions for splines is

$$\begin{aligned} h_j(x) &= x^{j-1}, \quad j = 1, \ldots, M, \\ h_{M+l}(x) &= (x - \xi_l)_+^{M-1}, \quad l = 1, \ldots, k. \end{aligned}$$

We have: number of basis functions $= M + k =$ number of parameters $(=$ df$)$.

A cubic spline $(M = 4)$ with 2 knots has the following basis functions:

$$\begin{aligned} h_1(x) &= 1, \quad h_3(x) = x^2, \quad h_5(x) = (x - \xi_1)^3_+ \\ h_2(x) &= x, \quad h_4(x) = x^3, \quad h_6(x) = (x - \xi_2)^3_+ \end{aligned}$$

Figure 7.2 shows piecewise cubic polynomials, with increasing order of continuity in the knots. The curve in the lower right picture has continuous derivatives of order 1 and 2, and thus it is a cubic spline.

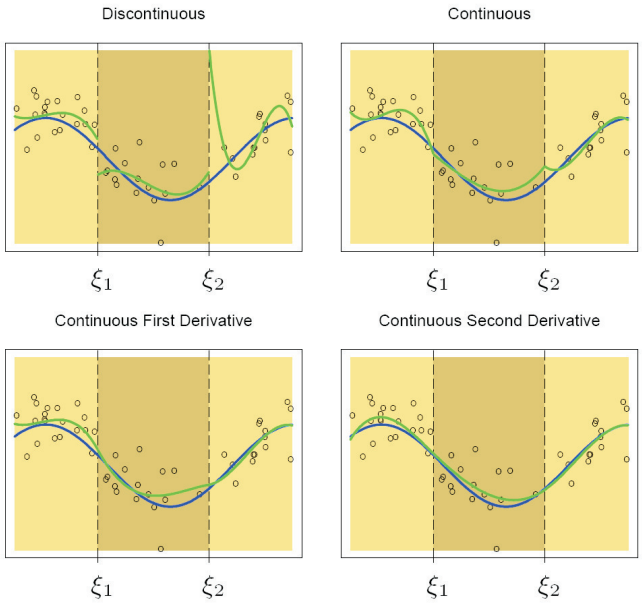

Figure 7.2: Piecewise cubic polynomials

Usually there is no need to go beyond cubic splines. In practice the most widely used orders are $M = 1, M = 2$ and $M = 4$.

The following parameters have to be chosen:

- order M of the splines
- number of knots
- placement of knots: chosen by the user, for instance at the appropriate percentiles of x (e.g. for $k = 3$ at the percentiles $25, 50, 75\%$).

The spline bases functions suggested above are not too attractive numerically. The so-called *B-spline* basis is numerically more suitable, and it is an equivalent form of the basis. In R this function is called `bs()`, and the argument `df` allows to select the number of spline basis functions.

Natural cubic splines

Polynomials tend to be erratic near the lower and upper data range, which can result in poor approximations (Figure 8.7). This can be avoided by using natural cubic splines (Figure 8.8). They have the additional constraint that the function has to be linear beyond the boundary knots. In this way we get back 4 degrees of freedom (two constraints each in both boundary

regions), which can then be "invested" in a larger number of knots. Natural cubic splines thus have $M + k - 4 = 4 + k - 4 = k$ basis functions (degrees of freedom).

Section 8.1, page 74

7.2 Smoothing splines

This spline method avoids the knot selection problem by controlling the complexity of the fit through regularization.
Consider the following problem: among all functions $f(x)$ with two continuous derivatives, find the one that minimizes the penalized residual sum of squares:

$$\text{RSS}(f, \lambda) = \sum_{i=1}^{n} \{y_i - f(x_i)\}^2 + \lambda \int f''(t)^2 dt \tag{7.1}$$

The first term measures closeness to the data, the second term penalizes curvature in the function. The smoothing parameter λ establishes a tradeoff between the two. There are two special cases:

- $\lambda = 0$: f is any function that interpolates the data
- $\lambda = \infty$: the simple least square fit, where no second derivative can be tolerated

(7.1) has a unique minimizer, which is a natural cubic spline with knots at the values $x_i, i = 1, \ldots, n$. It seems that this solution is over parameterized, since n knots correspond to n degrees of freedom. However, the penalty term reduces them, since the spline coefficients are shrunk towards the linear fit.
Since the solution is a natural cubic spline, we can write it as

$$f(x) = \sum_{j=1}^{n} N_j(x)\theta_j$$

where N_j is the jth spline basis function. The criterion (7.1) thus reduces to

$$\text{RSS}(\boldsymbol{\theta}, \lambda) = (\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\theta})^\top (\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\theta}) + \lambda \boldsymbol{\theta}^\top \boldsymbol{\Omega}_N \boldsymbol{\theta}$$

with $\{\boldsymbol{N}\}_{ij} = N_j(x_i)$ and $\{\boldsymbol{\Omega}_N\}_{jk} = \int N_j''(t) N_k''(t) dt$.
The solution is obtained as

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{N}^\top \boldsymbol{N} + \lambda \boldsymbol{\Omega}_N)^{-1} \boldsymbol{N}^\top \boldsymbol{y}$$

which is a generalized form of Ridge regression. The fit of the smoothing spline is given by

$$\hat{f}(x) = \sum_{j=1}^{n} N_j(x)\hat{\theta}_j$$

7.2.1 Choice of the degrees of freedom

Up to now we did not mention how the parameter λ for the smoothing splines should be chosen. The choice can be based on usual techniques like cross-validation. In the following we show an intuitive way how this parameter can be pre-specified.

A smoothing spline with given λ is an example for a "linear smoother", since the estimated parameters are a linear combination of the y_i. $\hat{\boldsymbol{f}}$ can be computed by

$$\begin{aligned}\hat{\boldsymbol{f}} &= \boldsymbol{N}(\boldsymbol{N}^\top\boldsymbol{N} + \lambda\boldsymbol{\Omega}_N)^{-1}\boldsymbol{N}^\top\boldsymbol{y} \\ &= \boldsymbol{S}_\lambda\boldsymbol{y}\end{aligned}$$

The fit is linear in $\boldsymbol{y}$ and the finite operator $\boldsymbol{S}_{\boldsymbol{\lambda}}$ is known as "smoother matrix". Due to the linearity, the computation of $\hat{\boldsymbol{f}}$ is independent from $\boldsymbol{y}$, because $\boldsymbol{S}_{\boldsymbol{\lambda}}$ only depends on x_i and λ. Let $\boldsymbol{B}_\xi$ denote a $n \times M$ matrix of M cubic spline basis functions, evaluated at the n training points x_i, with knot sequence ξ and $M << n$. Then

$$\begin{aligned}\hat{\boldsymbol{f}} &= \boldsymbol{B}_\xi\left(\boldsymbol{B}_\xi^\top\boldsymbol{B}_\xi\right)^{-1}\boldsymbol{B}_\xi^\top\boldsymbol{y} \\ &= \boldsymbol{H}_\xi\boldsymbol{y}\end{aligned}$$

The linear operator $\boldsymbol{H}_\xi$ is a projection operator, also known as "hat matrix". For $\boldsymbol{H}_\xi$ and $\boldsymbol{S}_\lambda$ we have:

- Both are symmetric, positive semidefinite matrices;
- $\boldsymbol{H}_\xi\boldsymbol{H}_\xi = \boldsymbol{H}_\xi$ (idempotent), while$\boldsymbol{S}_\lambda\boldsymbol{S}_\lambda = \boldsymbol{A}\boldsymbol{S}_\lambda$ with a positive semidefinite matrix $\boldsymbol{A}$;
- $\text{rank}(\boldsymbol{H}_\xi) = M$, $\text{rank}(\boldsymbol{S}_\lambda) = n$.

The dimension of the projection space is given by $\text{trace}(\boldsymbol{H}_\xi) = \text{trace}(\boldsymbol{B}_\xi^\top\boldsymbol{B}_\xi(\boldsymbol{B}_\xi^\top)^{-1}\boldsymbol{B}_\xi) = \text{trace}(\boldsymbol{I}_M) = M$. M also is the number of basis functions, and hence the number of parameters. By analogy we define the "effective degrees of freedom" of $\boldsymbol{S}$ by

$$df_\lambda = \text{trace}\left(\boldsymbol{S}_\lambda\right),$$

the sum of the diagonal elements. With an initial guess of the degrees of freedom we can then derive λ by numerical optimization.

In the example of the bone density data (Figure 8.12) we obtain:

$$df(\lambda) = \text{trace}(\boldsymbol{S}_\lambda) = 12 \Longrightarrow \text{ numerical solution: } \lambda = 0.00022$$

Section 8.2, page 78

Chapter 8

Basis expansions in R

There are several possibilities in R to fit a nonlinear function. It is important to distinguish the case where (1) a function can be explicitly stated, and based on that a nonlinear optimization is carried out, or (2) where no functional relationship can be defined, and the task is to "automatically" fit a nonlinear function. In case of (1) one can use the nonlinear least-squares method `nls()`, or the more general optimization routine `optimize()`. For the case (2) we have the spline interpolation methods, which are treated in Section 8.1.

Example for case (1), where an explicit function can be set up: Consider the data set `wtloss` from `library(MASS)`, which consists of 2 variables with 52 observations each.

- **Weight:** weight of the patient (in kg)
- **Days:** number of days that the patient has been in therapy

The patient is undergoing a diet which lasts for 8 months. It will be interesting to predict the future weight is the diet continues.

- *Fit of a linear model*

```
> library(MASS)
> data(wtloss)
> lm1 <- lm(Weight ~ Days, data=wtloss)
> plot(Weight ~ Days, data=wtloss)
> abline(lm1, col="blue")
```

- *Fit of a linear model with a quadratic term*

```
> lm2 <- lm(Weight ~ Days + I(Days^2), data=wtloss)
> lines(wtloss$Days, predict.lm(lm2), col="green")
```

Quadratic regression or, more generally, regression with polynomials allows a good approximation of the data (Figure 8.1, left). However, often this only works in the range of the data and cannot be used for prediction (Figure 8.1, right).

```
> plot(Weight~Days, data=wtloss, xlim=c(0,1000), ylim=c(0,400))
> abline(lm1, col="blue")
> x=c(wtloss$Days, seq(300,1000,length=50))
> lines(x,lm2$coef[1]+x*lm2$coef[2]+x^2*lm2$coef[3], col="green")
```

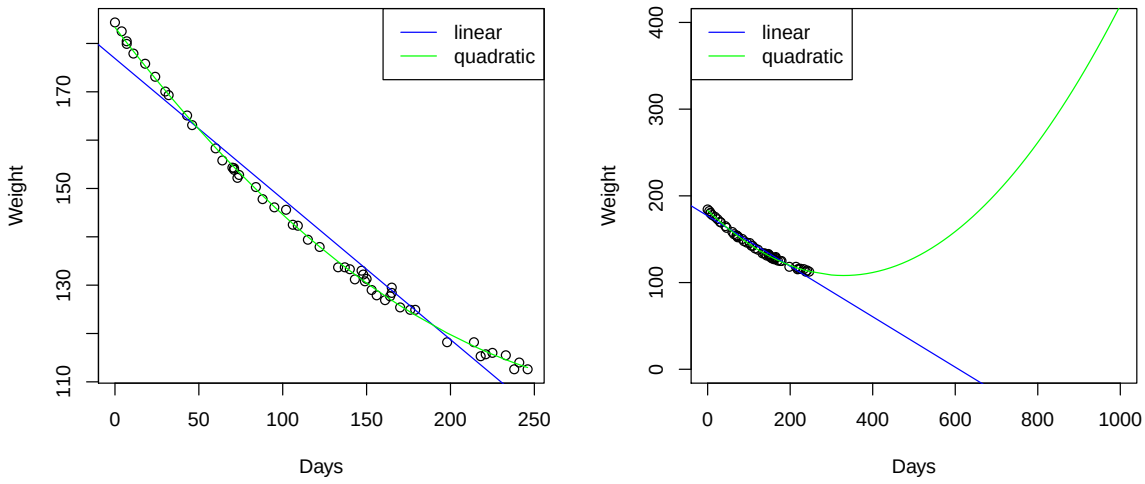

Figure 8.1: Graphical presentation of the linear and the quadratic fit (left), and their predictions for future observations (right).

As a way out, the following nonlinear function with an exponential decay is defined, since weight loss is neither linear nor quadratic:

$$y = \beta_0 + \beta_1 2^{-t/\theta} + \varepsilon$$

β_0 . . . asymptotic final weight (2nd term $\to 0$ for $t \to \infty$)
β_1 . . . final weight loss (starting weight at $t = 0$ is $\beta_0 + \beta_1$)
θ . . . time to reach half of the final weight loss (for $t = \theta$ we get $\beta_0 + \beta_1/2$)

- *Nonlinear regression with* **nls()**: needs starting values

```
> mod.start <- c(b0=100, b1=85, theta=100)   # seems to be reasonable
> mod.nls <- nls(Weight ~ b0 + b1 * 2^(-Days/theta), data=wtloss, start=mod.start, trace=TRUE)

162.2385 :  100  85 100
47.24374 :   86.38065  97.54973 131.36239
39.27375 :   81.67936 102.37229 141.32577
39.2447 :   81.37396 102.68386 141.91059
39.2447 :   81.37382 102.68411 141.91036
```

  - The formula for the nonlinear model contains the variables as well as the parameters.
  - Parameters are estimated through iterative numeric optimization, the starting values have to be chosen. All variable names in the formula that do not get starting values are treated as data variables.
  - Arguments `control` and `algorithm` in **nls()** allow for a finer control of the numeric optimization.
  - Faster convergence can be obtained through specification of the first partial derivatives.

```
> lines(x, predict(mod.nls, list(Days=x)), col="orange")
> abline(h=81.37 , lty=3, col = "red")
```

The resulting fit and prediction is shown in Figure 8.2.

- *General optimization with* **optim()**
  - The function **optim()** allows for a minimization of any (univariate) function.
  - Specification of the function and (optionally) of the gradient of the function.
  - Several algorithms can be chosen, i.e. quasi Newton, conjugate gradient or simulated annealing.
  - The algorithm `L-BFGS-B` allows a restriction of the parameter space to a hypercube: for every variable, an upper and lower boundary can be determined.

Minimization of the residual sum of squares with **optim()**

```
> funSSR <- function(p){ sum((wtloss$Weight - (p[1] + p[2] * 2^(-wtloss$Days/p[3])))^2) }
> mod.opt1 <- optim(mod.start, funSSR)
> mod.opt1
$par
       b0        b1     theta
 81.37586 102.68225 141.90608

$value
[1] 39.2447

$counts
function gradient
     130       NA

$convergence
[1] 0

$message
NULL
```

Again, the resulting fit and prediction is shown in Figure 8.2. As it should be, there is practically no difference in the outcome of `nls()` the `optim()`, since in this case both are based on minimizing the sum of squared residuals.

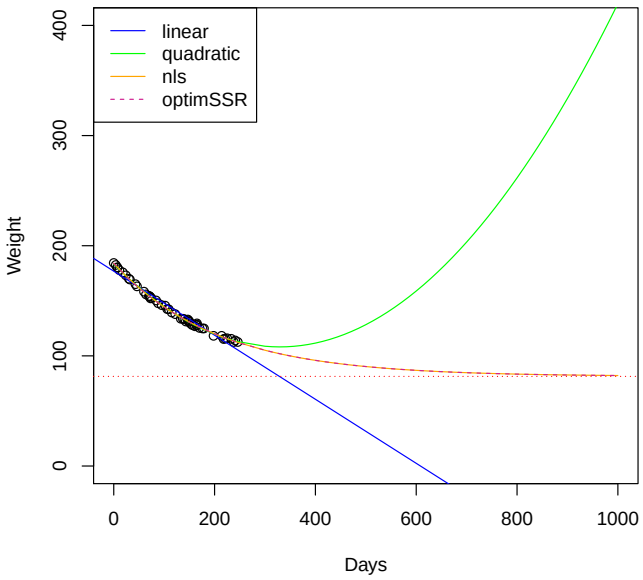

Figure 8.2: Prediction with the different models and routines.

- *Comparison of the functions* **nls()** *and* **optim()**
  - The function **nls()** provides the usual methods such as **summary()**, **predict()**, etc. Tests for the parameters can be obtained as well.

```
> summary(mod.nls)

Formula: Weight ~ b0 + b1 * 2^(-Days/theta)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
b0      81.374      2.269   35.86   <2e-16 ***
b1     102.684      2.083   49.30   <2e-16 ***
theta  141.910      5.295   26.80   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8949 on 49 degrees of freedom

Number of iterations to convergence: 4
Achieved convergence tolerance: 1.167e-07
```

  - Those options are not provided in **optim()**:

```
> summary(mod.opt1)

            Length Class  Mode
par         3      -none- numeric
value       1      -none- numeric
counts      2      -none- numeric
convergence 1      -none- numeric
message     0      -none- NULL
```

    In exchange, **optim()** allows for more flexibility in terms of the specification of the model and provides a wider range of optimization algorithms.

8.1 Interpolation with splines in R

- *Spline basis functions as defined in Section 7.1:* The function below allows to construct the spline basis functions as they have been defined in the theoretical part.

```
> lecturespl <- function(x, nknots=2, M=4){
+   # nknots ... number of knots -> placed at regular quantiles
+   # M ... M-1 is the degree of the polynomial
+   n <- length(x)
+   # X will not get an intercept column
+   X <- matrix(NA,nrow=n,ncol=(M-1)+nknots)
+   for (i in 1:(M-1)){ X[,i] <- x^i }
+   # now the basis functions for the constraints:
+   quant <- seq(0,1,1/(nknots+1))[c(2:(nknots+1))]
+   qu <- quantile(x,quant)
+   for (i in M:(M+nknots-1)){
+     X[,i] <- ifelse(x-qu[i-M+1]<0,0,(x-qu[i-M+1])^(M-1))
+   }
+   list(X=X,quantiles=quant,xquantiles=qu)
+ }
```

Consider a simulated data set according to a sine function, see R code below and Figure 8.3.

```
> x <- seq(1,10,length=100)
> y <- sin(x) + 0.1 * rnorm(x)
> x1 <- seq(-1,12,length=100)
> plot(x, y, xlim = range(x1))
```

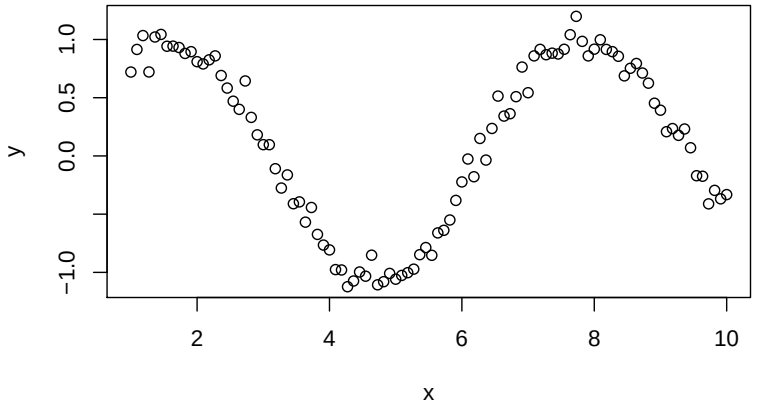

Figure 8.3: Simulated data according to a sine function

```
> spl <- lecturespl(x, nknots=2, M=4)    # generate the bases based on the x data
> dim(spl$X)       # generated matrix with spline basis functions
[1] 100   5
> spl$quantiles    # quantiles of the knots
[1] 0.3333333 0.6666667
> spl$xquantiles   # corresponding x-positions
33.33333% 66.66667%
        4         7
```

With 2 knots and $M = 4$ we obtain 5 basis functions. Note that no basis function for the intercept is constructed, since the intercept can be directly obtained in `lm()`. The corresponding spline basis functions are shown in Figure 8.4.

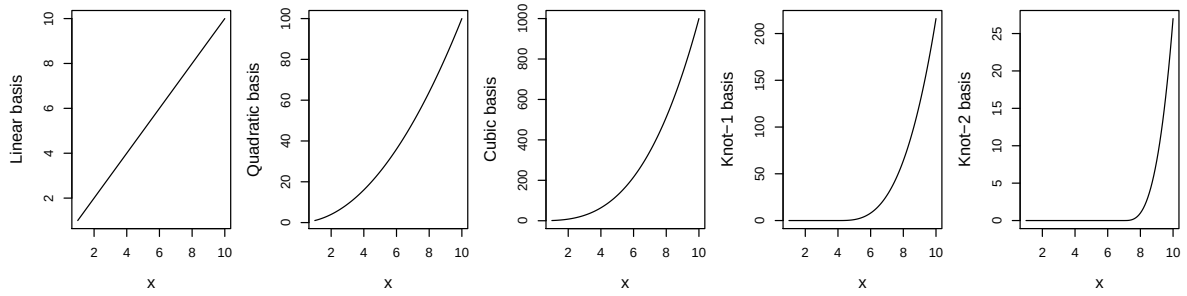

Figure 8.4: Spline basis functions as defined in the course notes (no function for the intercept) for 2 knots and $M = 4$.

The constructed basis functions can now be used to fit the sine function. Since we are looking for a linear relationship between the y variable and the spline bases, the function **`lm()`** can be used.

```
> spl <- lecturespl(x, nknots=2, M=4)
> lm1 <-  lm(y ~ spl$X)
> lines(x, predict(lm1, newdata=data.frame(x)), col="blue")
```

The resulting fit is shown in Figure 8.5 as blue line. Obviously, this is still an underfit, and we need to use more spline basis functions. The green line is constructed with 9 basis functions, resulting from using 6 knots and $M = 4$.

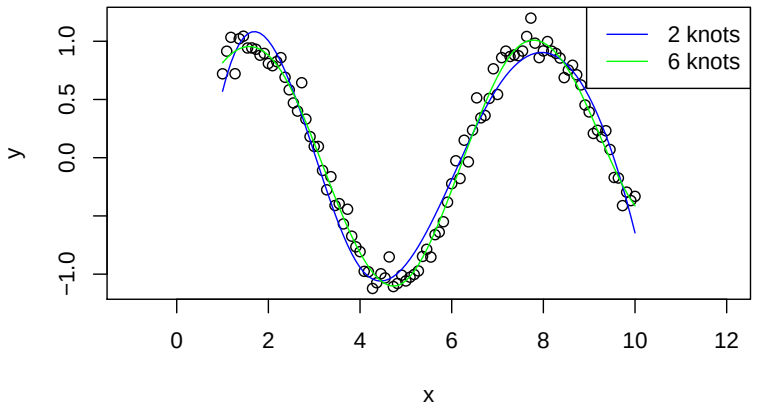

Figure 8.5: Sine function fitted by spline basis functions as defined in the course notes.

- *Model fit with B-splines*

  THe spline basis functions as defined in the course notes are numerically problematic, because huge values could be produced, in particular for the basis functions corresponding to higher order. Figure 8.4 already shows that the cubic basis function has large values. B-splines are an alternative way to generate spline basis functions, but they are numerically more stable. The following code shows some examples, and Figure 8.6 shows the resulting B-spline basis functions.

  – Change of the order of the polynomials:

```
> library(splines)
> matplot(x, bs(x, knots=5, degree=2), type="l",lty=1)
```

  – Change of the degrees of freedoms (number of spline basis functions):

```
> matplot(x, bs(x, df=4, degree=3), type="l",lty=1)
```

  – Change of the knots:

```
> matplot(x, bs(x, knots=c(3,7), degree=3), type="l",lty=1)
```

- *Estimation of the parameters*

  As before, **lm()** can be used to fit the model.

```
> lm1B <- lm(y ~ bs(x, df=4))
> lines(x1, predict.lm(lm1B, list(x=x1)), col="blue")
```

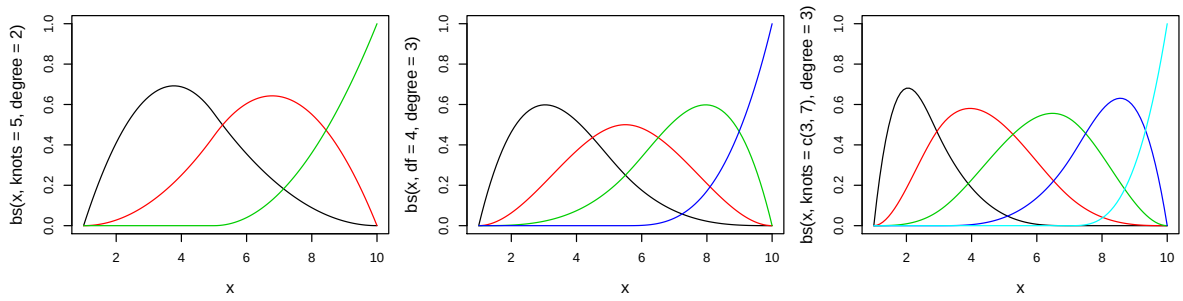

Figure 8.6: B-spline basis functions with varying degree, knots, and df.

Here we also use the model for the prediction to an extended x-range * (extrapolation). The result is shown in Figure 8.7 as blue line. The green line shows the same thing, but with more basis functions (`df=6`).

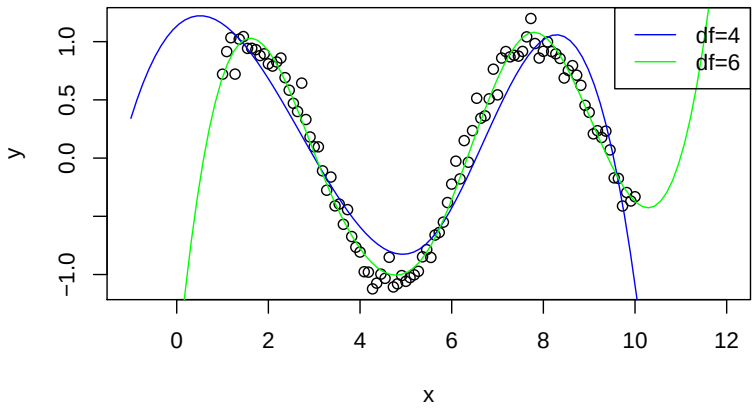

Figure 8.7: Fit and prediction with B-spline basis functions.

- *Usage of natural cubic splines (Figure 8.8)*

  The extrapolation outside of the boundary knots is done as linear function, which allows to save parameters.

```
> lm3N <- lm(y ~ ns(x, df=6))
> lines(x1, predict.lm(lm3N, list(x=x1)), col="orange")
```

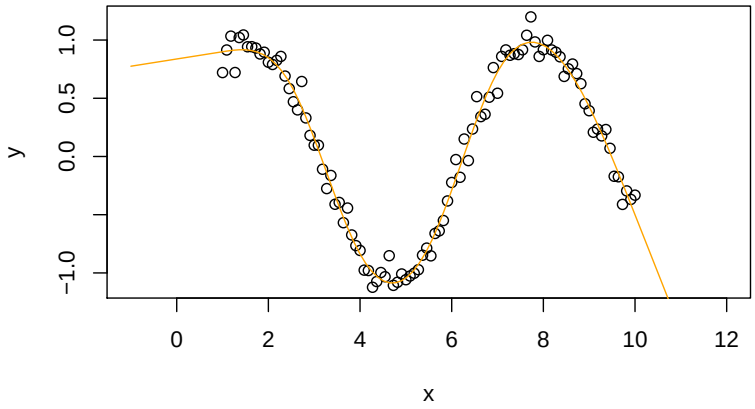

Figure 8.8: Fit and prediction with natural cubic splines.

8.2 Smoothing splines in R

Smoothing splines solve the knot selection problem. They consist of natural cubic splines with knots at every x data value. However, the resulting matrix with spline basis functions is shrunken according to the tuning parameter. This shrinkage can also be controlled by the parameter `df`.

- *Fit with* **smooth.spline()**

```
> m1 <- smooth.spline(x, y, df=6)
> plot(x, y, xlim = range(x1), ylim=c(-1.5, 1.5))
> lines(m1, col="green")
```

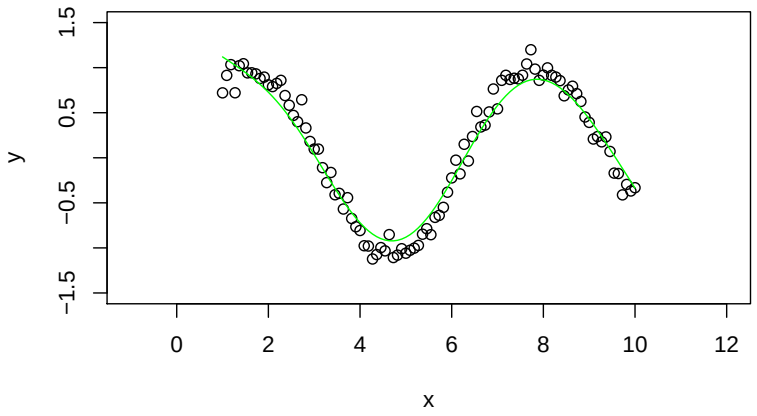

Figure 8.9: Fit with smoothing splines.

- *Prediction outside the range*

```
> lines(predict(m1, x1), col="blue")
```

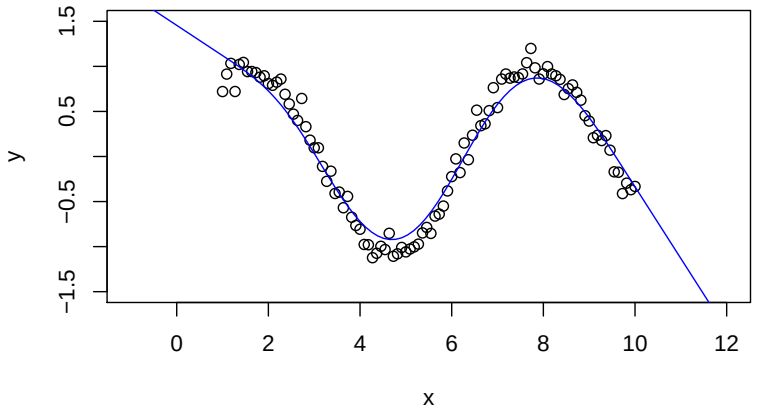

Figure 8.10: Prediction at the boundaries with smoothing splines

- *Choice of degrees of freedom with cross-validation*

```
> m2 <- smooth.spline(x, y, cv=TRUE)
> plot(x, y, xlim = range(x1), ylim=c(-1.5, 1.5))
> lines(predict(m2, x1), col="green")
```

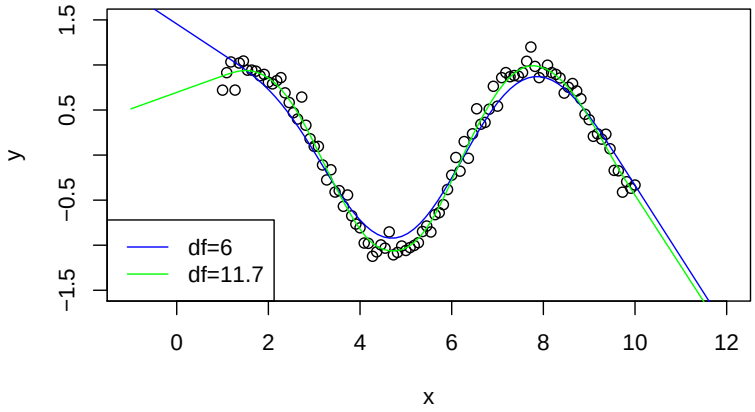

Figure 8.11: Choice of degrees of freedom

- *Example: smoothing splines with the bone density data*

```
> library(ElemStatLearn)
> data(bone)
```

Bone density data of 261 teens from North America. The average age of the teens (**age**) and the relative change in the bone density (**spnbmd**) were measured at two consecutive visits.

```
> plot(spnbmd ~ age, data=bone, col = ifelse(gender=="male", "blue", "red2"),
+      xlab="Age", ylab="Relative Change in Spinal BMD")
> bone.spline.male <- with(subset(bone,gender=="male"), smooth.spline(age, spnbmd, cv=TRUE))
> bone.spline.female <- with(subset(bone, gender=="female"), smooth.spline(age, spnbmd, cv=TRUE))
> lines(bone.spline.male, col="blue")
> lines(bone.spline.female, col="red2")
> legend("topright", legend=c("Male", "Female"), col=c("blue", "red2"), lwd=2)
```

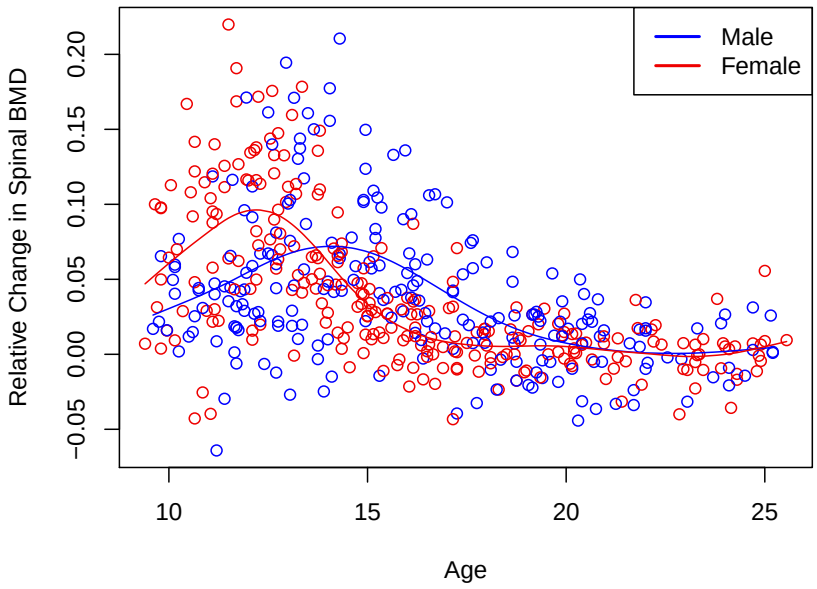

Figure 8.12: Smoothing spline fits for the bone density data.

The tuning parameters for both fits were selected by cross-validation. For the males we obtain $df = 5.62$, corresponding to $\lambda = 0.0073027$, and for the femails we get $df = 8.17$, corresponding to $\lambda = 0.0013912$.

Chapter 9

Generalized Additive Models (GAM)

For GAM's the weighted sum of the regressor variables is replaced by a weighted sum of transformed regressor variables [see Hand et al., 2001]. In order to achieve more flexibility, the relations between y and x are modeled in a non-parametric way, for instance by cubic splines. This allows to identify and characterize nonlinear effects in a better way.

9.1 General aspects on GAM

GAM's are generalizations of *Generalized Linear Models* (GLM) to nonlinear functions. Let us consider the special case of multiple linear regression. There, we model the conditional expectation of y by a linear function:

$$\mathbb{E}(y|x_1, x_2, \ldots, x_p) = \alpha + \beta_1 x_1 + \ldots + \beta_p x_p$$

A generalization is to use unspecified nonlinear (but smooth) functions f_j instead of the original regressor variables x_j.

$$\mathbb{E}(y|x_1, x_2, \ldots, x_p) = \alpha + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

Another regression model is the logistic regression model, which is in case of 2 groups

$$\log\left(\frac{\mu(\boldsymbol{x})}{1-\mu(\boldsymbol{x})}\right) = \alpha + \beta_1 x_1 + \ldots + \beta_p x_p,$$

where $\mu(\boldsymbol{x}) = P(y = 1|\boldsymbol{x})$. The generalization with nonlinear functions is called *additive logistic regression model*, and it replaces the linear terms by

$$\log\left(\frac{\mu(\boldsymbol{x})}{1-\mu(\boldsymbol{x})}\right) = \alpha + f_1(x_1) + \ldots + f_p(x_p).$$

For GLM's, and analogously for GAM's, the "left hand side" is replaced by different other functions. The right hand side remains a linear combination of the input variables for GLM, or of nonlinear functions of the input variables for GAM.

Generally we have for GAM:

The conditional expectation $\mu(\boldsymbol{x})$ of y is related to an additive function of the predictors via a link function g:

$$g\left[\mu(\boldsymbol{x})\right] = \alpha + f_1(x_1) + \ldots + f_p(x_p)$$

Examples for classical link functions are:

- $g(\mu) = \mu$: is the identity link, used for linear and additive models of Gaussian response data
- $g(\mu) = \text{logit}(\mu)$ or $g(\mu) = \text{probit}(\mu)$; the probit link function ($\text{probit}(\mu) = \Phi^{-1}(\mu)$) is used for modeling binomial probabilities
- $g(\mu) = \log(\mu)$ log-linear or log-additive models for Poisson count data

All these link functions arise from the exponential family, which forms the class of generalized linear models. Those are all extended in the same way to generalized additive models.

9.2 Parameter estimation with GAM

The model has the form

$$y_i = \alpha + \sum_{j=1}^{p} f_j(x_{ij}) + \varepsilon_i, \qquad i = 1, \ldots, n$$

with $\mathbb{E}(\varepsilon_i) = 0$. Given observations $(\boldsymbol{x}_i, y_i)$, a criterion like the penalized residual sum of squares (PRSS) can be specified for this problem:

$$\text{PRSS}(\alpha, f_1, f_2, \ldots, f_p) = \sum_{i=1}^{n} \left\{ y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij}) \right\}^2 + \sum_{j=1}^{p} \lambda_j \int f_j''(t_j)^2 dt_j$$

$\int f_j''(t_j)^2$ is an indicator for how much the function is ≥ 0. With linear f_j, the integral is 0, nonlinear f_j have values larger than 0. λ_j are tuning parameters. They regularize the tradeoff between the fit of the model and the roughness of the function. The larger the λ_j, the smoother the function. It can be shown, that (independent of the choice of λ_j) an **additive model with cubic splines minimizes the PRSS**. Each of the functions f_j is a cubic spline in the component x_j with knot $x_{ij}, i = 1, \ldots, n$. Without the following restrictions, no uniqueness can be obtained:

$$\sum_{i=1}^{n} f_j(x_{ij}) = 0 \quad \forall j \quad \Longrightarrow \quad \hat{\alpha} = \frac{1}{n} \sum_{i=1}^{n} y_i =: ave(y_i)$$

and the non-singularity of $\boldsymbol{X}$.

Iterative algorithm for finding the solution:

1. Initialization of $\hat{\alpha} = \text{ave}(y_i)$, $\hat{f}_j \equiv 0 \quad \forall i, j$
2. For the cycle $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots, p, \ldots$
   - $\hat{f}_j \quad \leftarrow \quad S_j \left[\left\{ y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik}) \right\} \right], \quad i = 1, \ldots, n$
   - $\hat{f}_j \quad \leftarrow \quad \hat{f}_j - \frac{1}{n} \sum_{i=1}^{n} \hat{f}_j(x_{ij})$

   until all $\hat{f}_j$ are stabilized.

The functions $S_j[x] = \sum_{j=1}^{n} N_j(x)\theta_j$ denote cubic smoothing splines, see Section 7.2. This algorithm is also known as *backfitting algorithm.*

Let us have a closer look at Step 2:

$j = 1$**:** In the first iteration, all functions are still set to 0, thus $\hat{f}_1 \equiv \ldots \equiv \hat{f}_p \equiv 0$. Thus we estimate f_1 from:

$$\hat{f}_1 \quad \leftarrow \quad S_1 \Big[\Big\{ y_i - \hat{\alpha} \Big\}, \; i = 1, \ldots, n \Big]$$

This means that the objective function

$$\sum_{i=1}^{n} \{ y_i - \hat{\alpha} - f_1(x_{i1}) \}^2 + \lambda_1 \int f_1''(t_1)^2 dt_1$$

is minimized, with the solution $\hat{f}_1$ as cubic smoothing spline.

$j = 2$**:** After centering $\hat{f}_1$, we estimate f_2 by:

$$\hat{f}_2 \quad \leftarrow \quad S_2 \Big[\Big\{ y_i - \hat{\alpha} - \hat{f}_1(x_{i1}) \Big\}, \; i = 1, \ldots, n \Big]$$

This means that the objective function

$$\sum_{i=1}^{n} \Big\{ y_i - \hat{\alpha} - \hat{f}_1(x_{i1}) - f_2(x_{i2}) \Big\}^2 + \lambda_2 \int f_2''(t_2)^2 dt_2$$

is minimized, with the solution $\hat{f}_2$ as cubic smoothing spline. S_2 is thus only applied to the residulas of $y_i - \hat{\alpha} - f_1(x_{i1})$, and $\hat{f}_2$ will thus explain the information which is not yet explained by $\hat{f}_1$.

$j \geq 3$**:** Similarly, the subsequent functions are estimated such that the spline functions are determined according to the residuals. In the next cycle, the estimattions are improved step by step.

Section 10, page 83

Chapter 10

Generalized additive models in R

- *Interpolation with GAM*

```
> library(mgcv)
> m1=gam(y ~ s(x))
> summary(m1)

Family: gaussian
Link function: identity

Formula:
y ~ s(x)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.16112    0.01005   16.03   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
      edf Ref.df   F p-value
s(x) 8.37   8.89 521  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.979   Deviance explained = 98.1%
GCV = 0.011141  Scale est. = 0.010097  n = 100
```

The spline interpolation fits a new spline basis with least squares. First, a smoothing algorithm is used and then an adequate algorithm estimates all p functions simultaneously (Figure 10.1). The function **gam()** can be found in `library(mgcv)`

```
> plot(m1, shade=T, shade.col="orange")
```

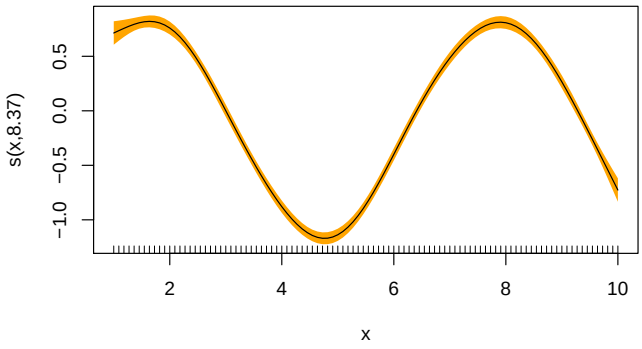

Figure 10.1: Fit with GAM

- *Prediction with* **gam()**

```
> plot(x, y, xlim=range(x1))
> m1.pred = predict(m1, se.fit=TRUE, data.frame(x = x1))
> lines(x1, m1.pred$fit, col="blue")
> lines(x1, m1.pred$fit+2*m1.pred$se, col="orange",lty=2)
> lines(x1, m1.pred$fit-2*m1.pred$se, col="orange",lty=2)
```

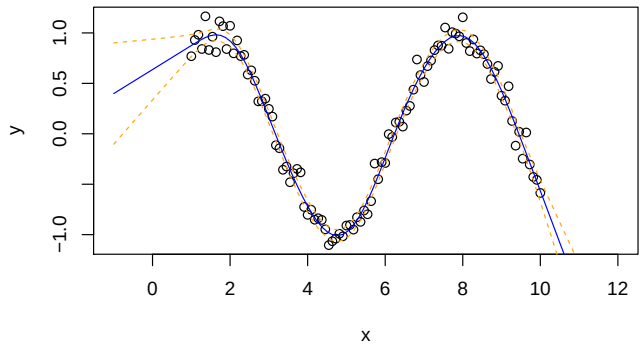

Figure 10.2: Prediction with GAM

- *Fit of a model to the PimaIndianDiabetes data with* **gam()**

```
> set.seed(101)
> train = sample(1:nrow(pid),300)
> mod.gam <-gam(diabetes ~ s(pregnant)+s(insulin)+s(pressure)+s(triceps)+s(glucose)+
        s(age)+s(mass)+s(pedigree), data=pid, family="binomial", subset=train)
> summary(mod.gam)

Family: binomial
Link function: logit

Formula:
diabetes ~ s(pregnant) + s(insulin) + s(pressure) + s(triceps) +
    s(glucose) + s(age) + s(mass) + s(pedigree)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -1.307      0.214  -6.108 1.01e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
              edf Ref.df Chi.sq  p-value
```

```
s(pregnant) 2.628  3.296  7.523  0.07344 .
s(insulin)  1.961  2.470  3.159  0.28428
s(pressure) 1.000  1.000  0.497  0.48087
s(triceps)  1.000  1.000  1.297  0.25479
s(glucose)  1.000  1.000 24.235 8.54e-07 ***
s(age)      4.997  6.007 19.917  0.00289 **
s(mass)     3.639  4.586  7.883  0.13186
s(pedigree) 1.507  1.843  1.094  0.43316
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.441   Deviance explained = 41.9%
UBRE = -0.13846  Scale est. = 1         n = 300
```

The estimated degrees of freedom ("edf") for each term have been computed by GCV. In this case, 3.043 edf's have been used for the term `pregnant`. The degrees of freedom of `pressure` and `triceps` are each 1. This suggests, that both fits prepresent a straight line (Figure 10.3). The estimated GCV value of 0.14 indicates that the model provides a good fit.

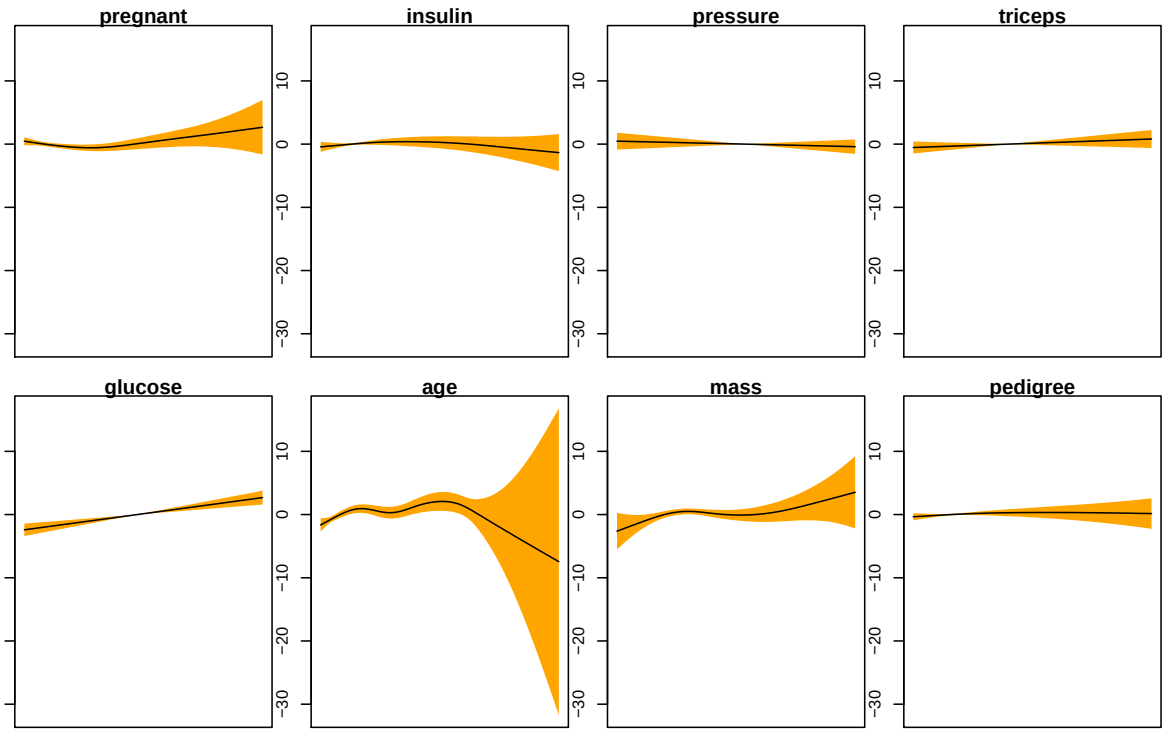

Figure 10.3: Fit of the different terms with additional 95% confidence region

- *Prediction with* **gam()**

```
> gam.res <- predict(mod.gam, pid[-train,])>0.5
> gam.TAB <- table(pid$diabetes[-train],as.numeric(gam.res))
> gam.TAB

    0  1
0 55  6
1 16 15
```

- *Misclassification rate for test set*

```
> mkrgam<-1-sum(diag(gam.TAB))/sum(gam.TAB)
> mkrgam
[1] 0.2391304
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.239	0.25	0.239	

Chapter 11

Tree-based methods

Tree-based methods partition the feature space of the x-variables into a set of rectangular regions which should be as homogeneous as possible, and then fit a simple model in each one. In each step, a decision rule is determined by a split variable and a split point which afterwards is used to assign an observation to the corresponding partition. Then a simple model (i.e. a constant) is fit to every region. To simplify matters, we restrict attention to binary partitions, therefore we always have only 2 branches. Mostly, the result is presented in form of a tree and is easy to understand and interpret. Tree models are nonparametric estimation methods, since no assumptions about the distribution of the regressors is made. They are very flexible in application which also makes them computationally intensive, and the results are highly dependent on the observed data. Even a small change in the observations can result in a severe change of the tree structure.

11.1 Regression trees

We have data of the form $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$ with $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$. The algorithm has to make decision about the:

- Split variable
- Split point
- Form of the tree

Suppose that we have a partition into M regions $R_1, \ldots, R_M$ and we model the response as a constant c_m in each region: $f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m I(\boldsymbol{x} \in R_m)$. If we adopt as our criterion minimization of the sum of squares $\sum (y_i - f(\boldsymbol{x}_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in each region:

$$\hat{c}_m = \text{ave}(y_i | \boldsymbol{x}_i \in R_m)$$

Now finding the best binary partition in terms of minimization of the above criterion of the sum of squares is generally computationally infeasible. Hence we look for an approximation of the solution.

Approximative solution:

- Consider a split variable x_j and a split point s and define a pair of half planes by:

$$R_1(j, s) = \{\boldsymbol{x} | x_j \leq s\}; \quad R_2(j, s) = \{\boldsymbol{x} | x_j > s\}$$

- Search for the splitting variable x_j and for the split point s that solves

$$\min_{j,s}\left[\min_{c_1}\sum_{\boldsymbol{x}_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{\boldsymbol{x}_i\in R_2(j,s)}(y_i-c_2)^2\right].$$

  For any choice of j and s, the inner minimization is solved by

$$\hat{c}_1=\text{ave}(y_i|\boldsymbol{x}_i\in R_1(j,s))\quad\text{and}\quad\hat{c}_2=\text{ave}(y_i|\boldsymbol{x}_i\in R_2(j,s)).$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, the determination of the best pair (j,s) is feasible. Afterwards, this algorithm is applied on all the other regions. In order to avoid overfitting, we use a tuning parameter, which tries to regulate the model's complexity. The strategy is to grow a large tree T_0, stopping the splitting process when some minimum node size is reached. Then this large tree is pruned using cost complexity pruning. By pruning (thus reduction of inner nodes) of T_0, we get a "sub" tree T. We index terminal nodes by m representing region R_m. Let $|T|$ denote the number of terminal nodes in T and

$$\begin{aligned}\hat{c}_m &= \frac{1}{n_m}\sum_{\boldsymbol{x}_i\in R_m}y_i \qquad n_m\ldots\text{ number of observations in the space }R_m\\ Q_m(T) &= \frac{1}{n_m}\sum_{\boldsymbol{x}_i\in R_m}(y_i-\hat{c}_m)^2\end{aligned}$$

and thus we define the cost complexity criterion

$$c_\alpha(T)=\sum_{m=1}^{|T|}n_mQ_m(T)+\alpha|T|$$

which has to be minimized. The tuning parameter $\alpha\geq 0$ regulates the compromise between tree size (large α results in a small tree) and goodness of fit ($\alpha=0$ results in a full tree T_0). The optional value $\hat{\alpha}$ for the final tree $T_{\hat{\alpha}}$ can be chosen by cross-validation.

It can be shown that for every α there exists a unique smallest sub-tree $T_\alpha\subseteq T_0$ which minimizes c_α. For finding T_α, we eliminate successively that internal node which yields the smallest increase (per node) of $\sum_m n_mQ_m(T)$. This is done as long as no node is left. It can be shown that this sequence of sub-trees must include T_α.

Section 12.1, page 92

11.2 Classification trees

The goal is to partition the x-variables into $1,\ldots,K$ classes. They are classified by the known output variable y with values between $1,\ldots,K$. Afterwards, new data should be assigned to the corresponding class.

In a node m representing a region R_m with n_m observations, let

$$\hat{p}_{mk}=\frac{1}{n_m}\sum_{\boldsymbol{x}_i\in R_m}I(y_i=k)$$

be the proportion of class k in node m. We classify the observations in node m to that class $k(m)$ for which $k(m) = \text{argmax}_k \hat{p}_{mk}$. So, the observation if assigned to the majority class in node m.

Different measures $Q_m(T)$ of node impurity include the following:

1. Misclassification error: $\frac{1}{n_m} \sum_{\boldsymbol{x}_i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$
2. Gini index: $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
3. Cross-entropy or deviance: $\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

Examples for those criteria for two classes with the proportion p of observations in the second class are (see Figure 11.1):

1. $1 - \max(p, 1 - p)$
2. $2p(1 - p)$
3. $-p \log p - (1 - p) \log(1 - p)$

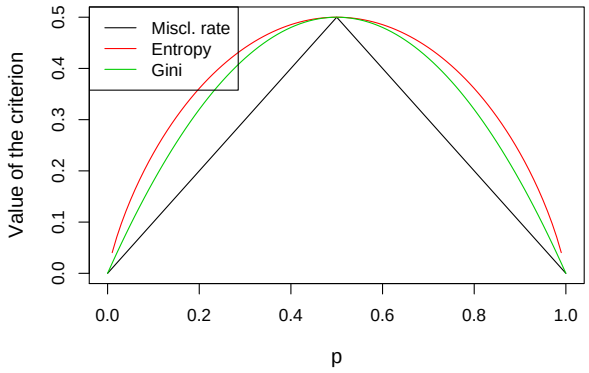

Figure 11.1: Impurity measures for two class classification

The three criteria are very similar, but cross-entropy and Gini index are differentiable, and hence more amenable to numerical optimization. In addition, cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate. For this reason, either Gini index or cross-entropy should be used when growing a tree.

Section 12.2, page 95

11.3 Random Forests

It turns out that classification and regression trees are very sensitive to small data changes. For example, it could happen that the inclusion of an additional observation could lead to a different decision already at the first knots, and thus to a completely different structure of the tree.

Random Forests consist of many classification (regression) trees which are used to make a decision – thus the terminology “forest”. They are “random” because the trees are generated “randomly” by

- using only bootstrap samples instead of the whole data set,
- at each knot only a random selection of the variables is available in order to produce very diverse trees.

The structure of the algorithm is as follows:
Step 1:

- Start with $b = 1$ (first tree).
- Take a random sample with replacement (= bootstrap sample) of size n, where n is the number of observations of the original data set.
- n_1 denotes the number of distinct observations ($n_1 \approx 2/3n$).
- The remaining $n_2 = n - n_1$ samples form the "Out of Bag" (OOB) data.

Step 2:

- Use this first learning sample to generate a tree.
- Hereby use at each knot only a random selection of the p variables:
  - classification: $\sqrt{p}$
  - regression: $p/3$

Step 3: Only with OOB data:

- Compute the tree impurity of the whole tree T_b as $\pi_b = \sum_{m=1}^{|T_b|} Q_m(T_b)$.
- Permute for each variable $j = 1, \ldots, p$ the values x_j and compute the resulting tree impurity, denoted as π_{b_j}.
  Define the measure for **variable importance** as $\delta_{b_j} = \pi_{b_j} - \pi_b$.

Step 4: Repeat Steps 1–3 for $b = 2, \ldots, B$, and compute for each j the measure $\delta_{1_j}, \ldots, \delta_{B_j}$.

Step 5: Compute the overall variable importance score for the j-th variable as:

$$\hat{\theta}_j = \frac{1}{B} \sum_{b=1}^{B} \delta_{b_j}$$

Advantages of Random Forests over single trees:

- A single tree has a high variability. Thus, a wrong decision in an "early" knot has concequences for all preceeding knots. Many trees reduce this effect.

Decision:

- *Classification:* majority decision, i.e. assign an observation to the class which is predicted by the majority of the trees.
- *Regression:* assign the average of all tree predictions.

Prediction in regression: compute the MSE for the OOB data as follows:

$$\text{MSE}_{\text{OOB}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y}_i^{\text{OOB}})^2$$

Here, $\bar{y}_i^{\text{OOB}}$ is the average of all OOB predictions for the i-th observation.
The proportion of explained variance is

$$1 - \frac{\text{MSE}_{\text{OOB}}}{\hat{\sigma}_y^2},$$

with the estimated variance $\hat{\sigma}_y^2$ of the response y.

Random Forests are implemented for instance in the R package `randomForest` as function `randomForest()`.

Chapter 12

Tree based methods in R

12.1 Regression trees in R

Use the body fat data for an illustration of regression trees:

```
> library("UsingR")
> data(fat)
> fat <- fat[-c(31,39,42,86), -c(1,3,4,9)] # strange values, not use all variables
> # randomly split into training and test data:
> set.seed(123)
> n <- nrow(fat)
> train <- sample(1:n,round(n*2/3))
> test <- (1:n)[-train]
```

- *Growing a regression tree with* **rpart()**

```
> library(rpart)
> mod.tree <- rpart(body.fat~.,data=fat, cp=0.001, xval=20, subset=train)
```

  - `library(mvpart)` or `library(tree)` can be used as well for growing a tree.
  - **rpart()** does cv internally. The number of cv steps can be controlled by the `control` parameters.

- *Output of the tree*

```
> mod.tree
n= 165

node), split, n, deviance, yval
      * denotes terminal node

  1) root 165 9.188831e+03 18.013330
    2) abdomen< 92.25 96 2.733025e+03 13.512500
      4) abdomen< 83.8 43 6.569642e+02 10.288370
        8) abdomen< 79.65 20 2.672720e+02  8.480000
         16) thigh< 50.05 3 3.880667e+01  3.433333 *
         17) thigh>=50.05 17 1.385753e+02  9.370588
           34) chest>=88.95 13 9.421077e+01  8.538462
             68) ankle< 22.75 10 2.557600e+01  7.420000
              136) bicep< 30.75 5 4.788000e+00  6.380000 *
              137) bicep>=30.75 5 9.972000e+00  8.460000
                274) age>=43 3 3.266667e-01  7.333333 *
                275) age< 43 2 1.250000e-01 10.150000 *
                    ⋮
```

```
      31) forearm>=29.7 11 3.716182e+01 30.727270
        62) BMI< 32.85 9 2.022222e+01 30.144440 *
        63) BMI>=32.85 2 1.250000e-01 33.350000 *
```

For each branch of the tree we obtain results in the following order:

- branch number
- split
- number of data following the split
- deviations associated with the split
- predicted value
- "*", if node is a terminal node

- *Plot of the tree*

```
> plot(mod.tree)
> text(mod.tree)
```

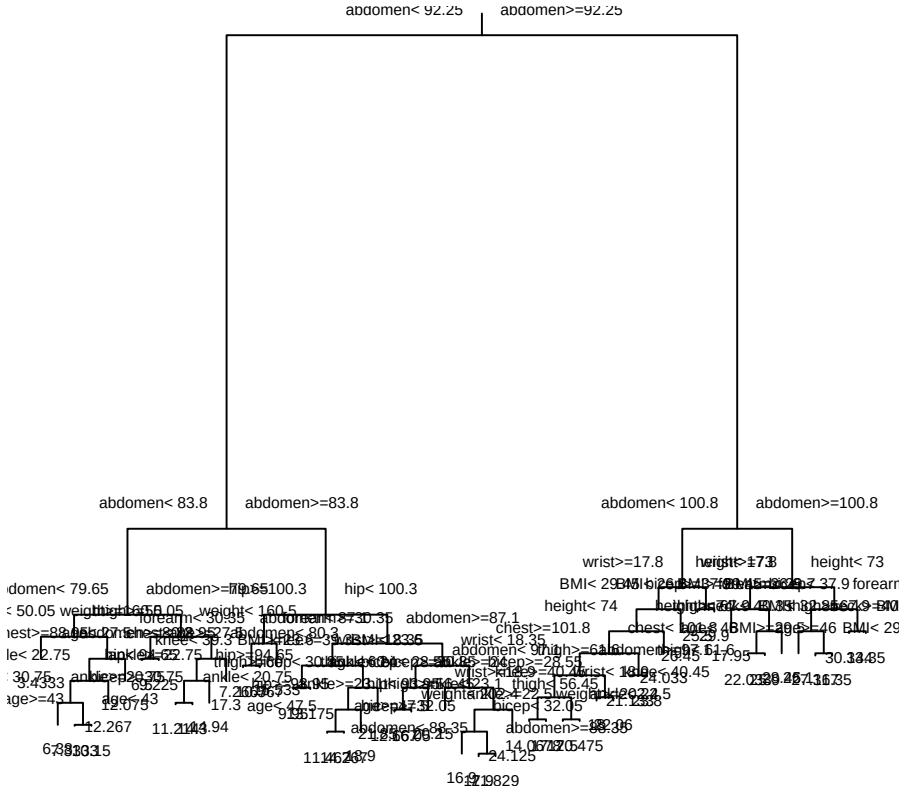

Figure 12.1: Regression tree of the body fat data

- Predict the response for the test set observations and compute the Root Mean Squared Error (RMSE):

```
> mod.tree.pred <- predict(mod.tree,newdata=fat[test,])
> RMSE <- sqrt(mean((fat$body.fat[test]-mod.tree.pred)^2))
> RMSE

[1] 6.363439
```

- Identify the optimal tree complexity to prune the tree:

```
> plotcp(mod.tree)
```

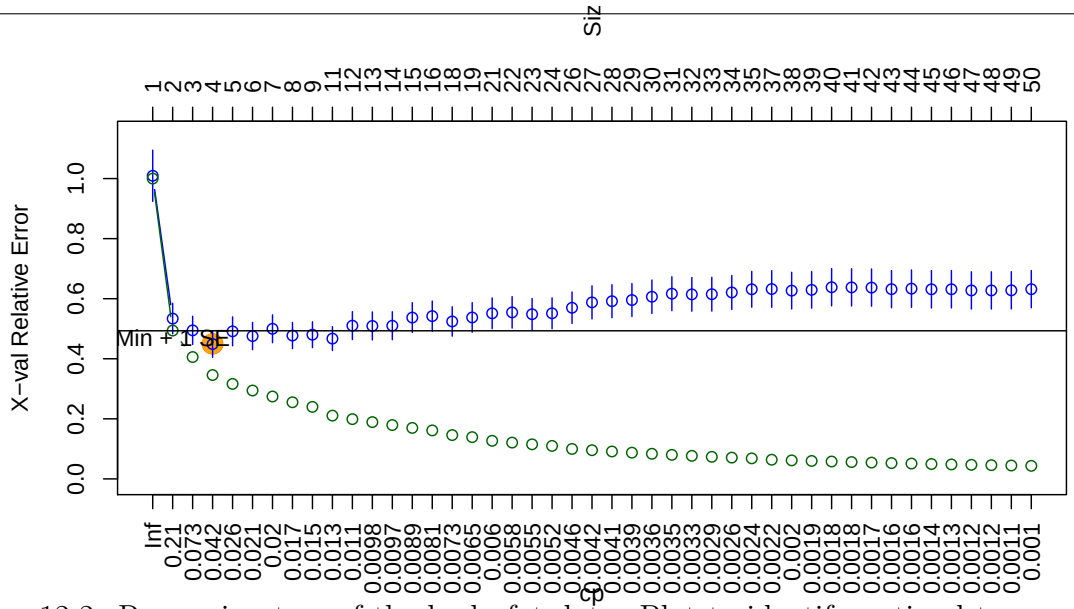

Figure 12.2: Regression tree of the body fat data: Plot to identify optimal tree complexity parameter.

Figure 12.2 reveals (although here not quite clearly) that the optimal tree complexity parameter is 0.042.

- Prune the tree to the optimal complexity, and predict the response for the test set observations and the RMSE:

```
> mod2.tree <- prune(mod.tree,cp=0.042)
> mod2.tree.pred <- predict(mod2.tree,newdata=fat[test,])
> RMSE <- sqrt(mean((fat$body.fat[test]-mod2.tree.pred)^2))
> RMSE

[1] 5.062448
```

- Show the final pruned regeression tree:

```
> plot(mod2.tree)
> text(mod2.tree)
```

The final regression tree, see Figure 12.3, is much smaller, and it has clearly higher predictive power.

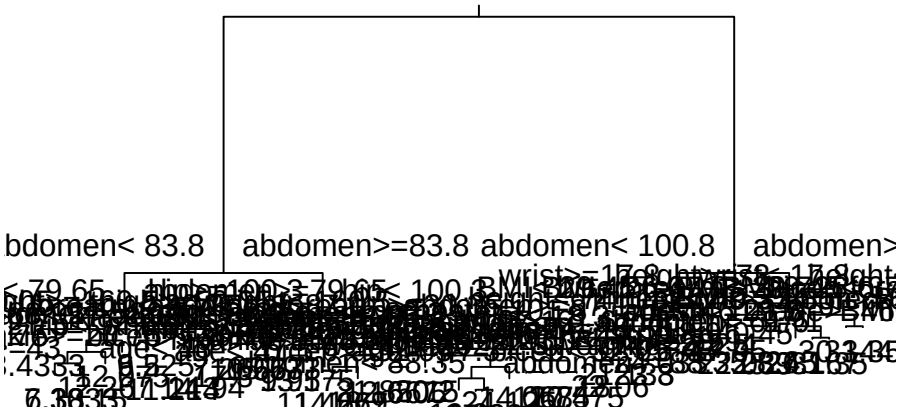

Figure 12.3: Final regression tree of the body fat data

12.2 Classification trees in R

The data set `Spam` consists of 4601 observations and 58 variables. The goal is to divide the variable `Email` in "good" and "spam" emails with classification trees.

```
> load("Spam.RData")
> names(Spam)
 [1] "make"            "address"         "all"             "X3d"
 [5] "our"             "over"            "remove"          "internet"
 [9] "order"           "mail"            "receive"         "will"
[13] "people"          "report"          "addresses"       "free"
[17] "business"        "email"           "you"             "credit"
[21] "your"            "font"            "X000"            "money"
[25] "hp"              "hpl"             "george"          "X650"
[29] "lab"             "labs"            "telnet"          "X857"
[33] "data"            "X415"            "X85"             "technology"
[37] "X1999"           "parts"           "pm"              "direct"
[41] "cs"              "meeting"         "original"        "project"
[45] "re"              "edu"             "table"           "conference"
[49] "semicolon"       "parenthesis"     "bracket"         "exclamationMark"
[53] "dollarSign"      "hashSign"        "capitalAverage"  "capitalLongest"
[57] "capitalTotal"    "class"
```

```
> set.seed(100)
> train <- sample(1:nrow(Spam), 3065)
> library(mvpart) # is already archaived, but one can install the .tar.gz file
> tree1 <- rpart(class~., data=Spam, subset=train, method="class",cp=0.001,xval=20)
> plot(tree1)
> text(tree1)
```

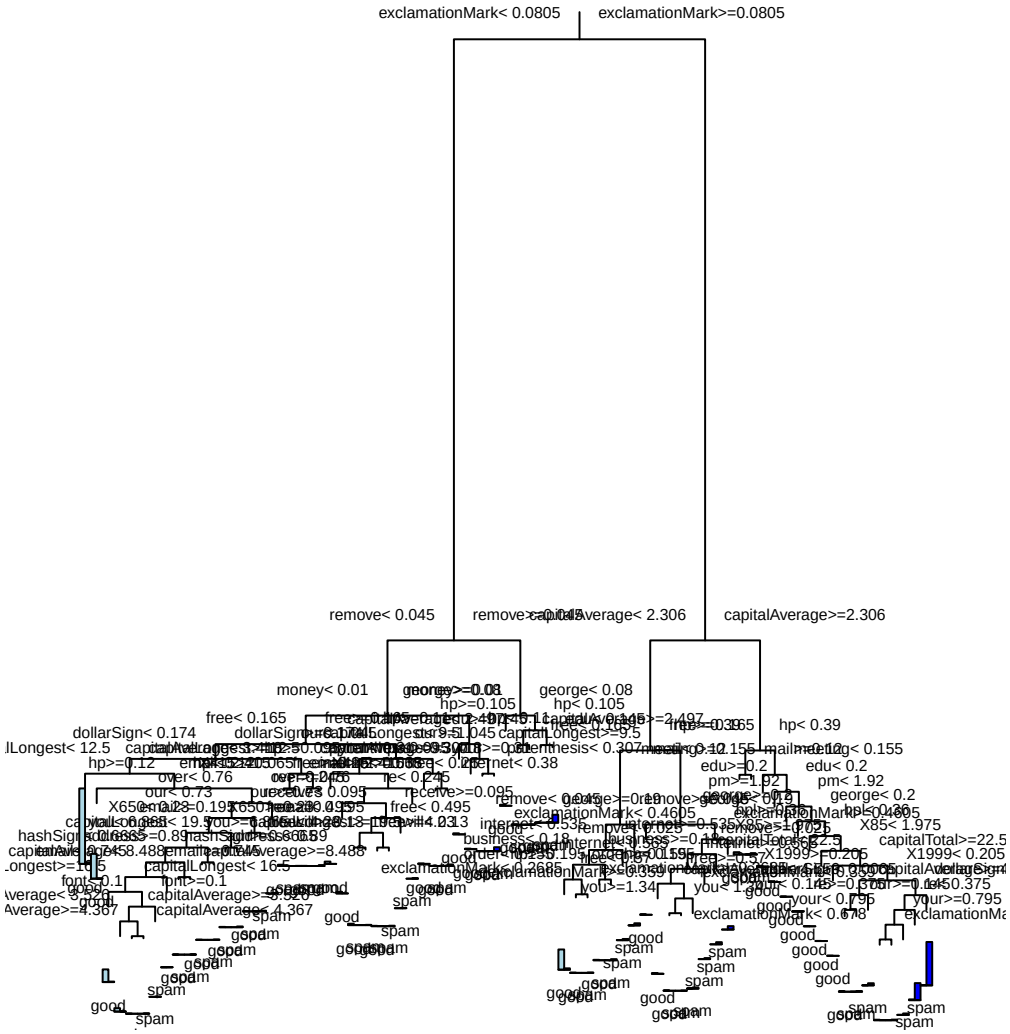

Figure 12.4: Classification tree of spam data

- *Prediction with the classification tree*

```
> tree1.pred <- predict(tree1, Spam[-train,],type="class")
> tree1.tab <- table(Spam[-train, "class"], tree1.pred)
> tree1.tab

      tree1.pred
       good spam
  good  866   54
  spam   69  547
```

- *Misclassification rate for the test set*

```
> 1-sum(diag(tree1.tab))/sum(tree1.tab)

[1] 0.08007812
```

- *Results of the cv*

```
> printcp(tree1)

Classification tree:
rpart(formula = class ~ ., data = Spam, subset = train, method = "class",
    cp = 0.001, xval = 20)

Variables actually used in tree construction:
 [1] address         business        capitalAverage  capitalLongest
 [5] capitalTotal    dollarSign      edu             email
 [9] exclamationMark font            free            george
[13] hashSign        hp              hpl             internet
[17] mail            meeting         money           order
[21] our             over            parenthesis     pm
[25] re              receive         remove          will
[29] X1999           X415            X650            X85
[33] you             your

Root node error: 1197/3065 = 0.39054

n= 3065

          CP nsplit rel error  xerror     xstd
1  0.4745196      0  1.000000 1.00000 0.022565
2  0.0868839      1  0.525480 0.52882 0.018723
3  0.0593150      2  0.438596 0.44110 0.017465
4  0.0584795      3  0.379282 0.41855 0.017103
5  0.0225564      4  0.320802 0.35589 0.016000
6  0.0200501      5  0.298246 0.34336 0.015760
7  0.0192147      6  0.278195 0.32999 0.015497
8  0.0133668      7  0.258981 0.28237 0.014487
9  0.0100251      8  0.245614 0.26316 0.014045
10 0.0075188     11  0.215539 0.24478 0.013599
11 0.0066834     12  0.208020 0.24060 0.013495
12 0.0050125     13  0.201337 0.23141 0.013261
13 0.0045948     14  0.196324 0.23141 0.013261
14 0.0041771     16  0.187135 0.21888 0.012932
15 0.0037594     18  0.178780 0.21888 0.012932
16 0.0033417     20  0.171261 0.21470 0.012819
17 0.0025063     22  0.164578 0.20551 0.012566
18 0.0020886     23  0.162072 0.20635 0.012590
19 0.0018797     27  0.153718 0.20384 0.012520
20 0.0016708     32  0.143693 0.20468 0.012543
21 0.0012531     61  0.087719 0.20886 0.012659
22 0.0011139     63  0.085213 0.20718 0.012613
23 0.0010443     68  0.078530 0.20886 0.012659
24 0.0010000     72  0.074353 0.20718 0.012613

> plotcp(tree1,upper="size")
```

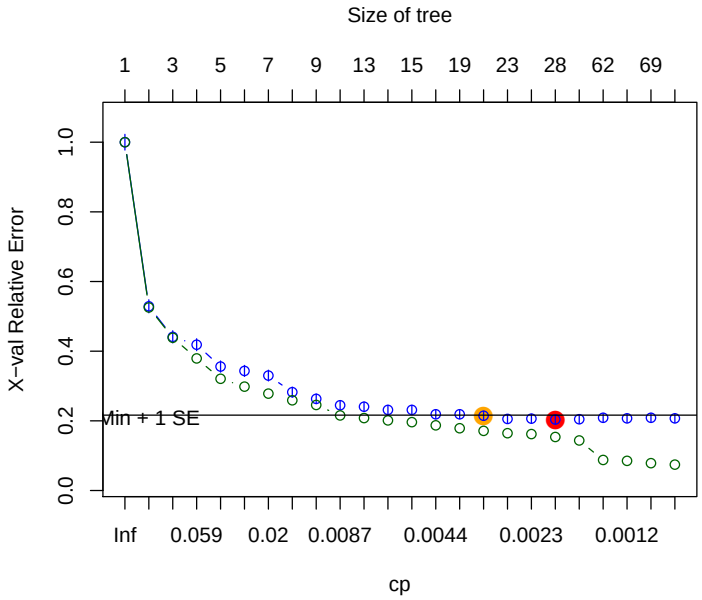

Figure 12.5: Cross-validation results of tree1: cv-error with standard error (blue) and error in test set (green)

Figure 12.5 shows the complexity of the tree and the cv estimate. The optimal, by cv computed value for α is 0.0035. The optimal size of the tree is approximately 20 nodes.

- *Pruning of the tree*

```
> tree2 <- prune(tree1, cp=0.0035)
> plot(tree2)
> text(tree2)
```

Pruning of the tree with the optimal $\hat{\alpha}$.

- *Prediction with the new tree*

```
> tree2.pred <- predict(tree2, Spam[-train,],type="class")
> tree2.tab <- table(Spam[-train, "class"], tree2.pred)
> tree2.tab
      tree2.pred
       good spam
  good  868   52
  spam   80  536
```

12.3 Random Forests in R

```
> rf <- randomForest(class~., data=Spam, subset=train, importance=TRUE)
> plot(rf)
> varImpPlot(rf)
> rf.pred <- predict(rf,Spam[-train,])
> rf.tab <- table(Spam[-train, "class"], rf.pred)
> 1-sum(diag(rf.tab))/sum(rf.tab)
```

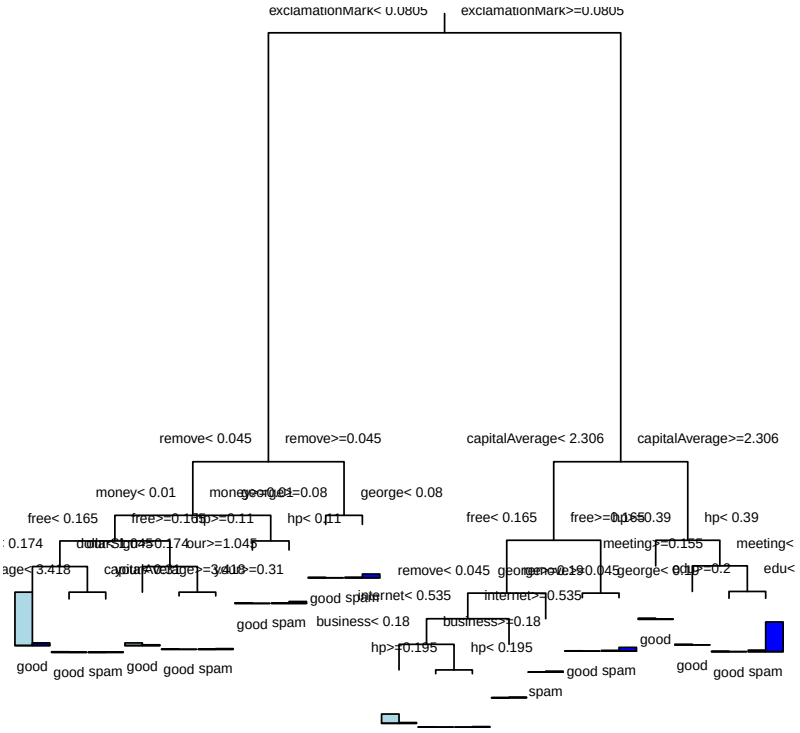

Figure 12.6: Result of the pruning of tree1

Chapter 13

Support Vector Machine (SVM)

Here we consider the case of K=2, which means there are two groups. When the data clouds of the two groups overlap, the classes are non-separable and linear decision boundaries, among others, perform poorly. Support Vector Machines transform the feature space into a higher-dimensional space and construct linear decision boundaries there.

13.1 Separating hyperplanes

Similar to LDA and logistic regression, also here we construct linear decision boundaries based on separating hyperplanes, which should be able to separate different groups in the best possible way. Classifications which are using linear combinations of the input variables and return the sign for the group membership are called "perceptrons" (this expression frequently appears for *neural networks*).

Mathematical background

A hyperplane (see Figure 13.1) is characterized by a set $\mathcal{L}$, defined by

$$f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} = 0$$

with the following properties:

- For any two points $\boldsymbol{x}_1$ und $\boldsymbol{x}_2$ lying in $\mathcal{L}$, we have that

  $$\boldsymbol{\beta}^\top (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0,$$

  i.e. $\boldsymbol{\beta}^* = \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|}$ is orthogonal to $\mathcal{L}$.
- For a point $\boldsymbol{x}_0$ from $\mathcal{L}$ we have $\boldsymbol{\beta}^\top \boldsymbol{x}_0 = -\beta_0$.
- As a distance measure (with sign) for any point $\boldsymbol{x}$ to $\mathcal{L}$ we can consider

  $$\boldsymbol{\beta}^{*\top}(\boldsymbol{x} - \boldsymbol{x}_0) = \frac{1}{\|\boldsymbol{\beta}\|}(\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0) = \frac{1}{\|f'(\boldsymbol{x})\|} f(\boldsymbol{x}),$$

  i.e. $f(\boldsymbol{x})$ is proportional to the distance from $\boldsymbol{x}$ to the hyperplane defined by $f(\boldsymbol{x}) = 0$.

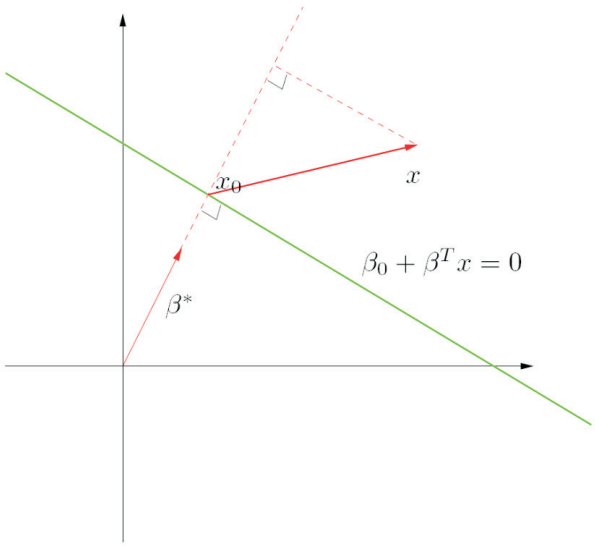

Figure 13.1: Hyperplane $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} = 0$

13.1.1 Perceptron learning algorithm of Rosenblatt

The perceptron learning algorithm of Rosenblatt looks for a separating hyperplane by minimizing the distance of misclassified points to the decision boundary. If a response $y_i = 1$ is misclassified, then we have that $\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta} < 0$; if $y_i = -1$ is misclassified, then we have $\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta} > 0$. We thus minimize

$$D(\beta_0, \boldsymbol{\beta}) = -\sum_{i \in \mathcal{M}} y_i (\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta})$$

where $\mathcal{M}$ contains the indexes of the misclassified points. $D(\beta_0, \boldsymbol{\beta})$ is non-negative and proportional to the distance of the misclassified points to the decision boundary, given by $\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} = 0$. The gradient is

$$\begin{aligned} \frac{\partial D(\beta_0, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= -\sum_{i \in \mathcal{M}} y_i \boldsymbol{x}_i \\ \frac{\partial D(\beta_0, \boldsymbol{\beta})}{\partial \beta_0} &= -\sum_{i \in \mathcal{M}} y_i \end{aligned}$$

Practically, one is using a stochastic gradient algorithm, where not the sum of the gradients but the contributions of the observations are considered, and in each iteration we take a step in the direction of the negative gradient. Thus, we obtain the new parameter estimates by

$$\begin{pmatrix} \boldsymbol{\beta} \\ \beta_0 \end{pmatrix} \leftarrow \begin{pmatrix} \boldsymbol{\beta} \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} y_i \boldsymbol{x}_i \\ y_i \end{pmatrix}$$

with the learning rate ρ (e.g. equal to 1).

A drawback of this algorithm is: If the groups can be perfectly separated by a hyperplane, then we obtain infinitely many solutions, see Figure 13.2. Depending on the starting value, different solutions will be obtained. "'Support vector machines"' or "'optimally separating hyperplanes"' are better suitable in this case [see Hastie et al., 2001].

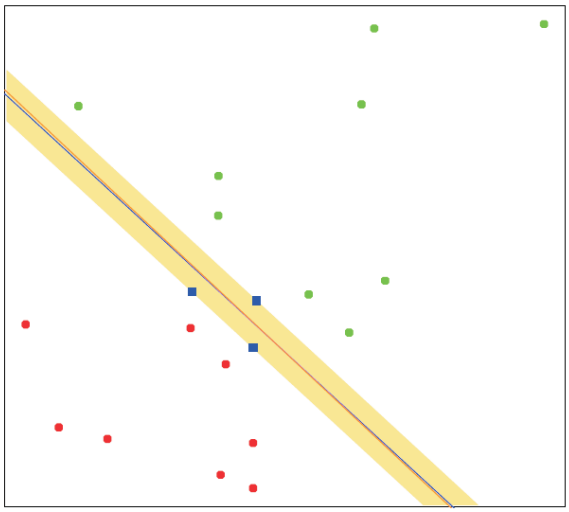

Figure 13.2: Perfect separation of two classes by a hyperplane.

13.2 Linear Hyperplanes

Assuming that training data is available, our training data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ has the following form:

$$\mathbf{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_i \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \dots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix}$$

The class membership is denoted by the vector $\boldsymbol{g} \in \mathbb{R}^n$ with $g_i \in \{-1, 1\}$ for $i = 1 \dots, n$. We can summarise the training information to n pairs

$$(\boldsymbol{x}_1, g_1), (\boldsymbol{x}_2, g_2), \dots, (\boldsymbol{x}_n, g_n).$$

Assuming $\boldsymbol{\beta}$ is a unit vector ($||\boldsymbol{\beta}|| = 1$), a hyperplane can be characterised by the affine set

$$\{\boldsymbol{x} \in \Omega : \underbrace{\boldsymbol{x}^\top \boldsymbol{\beta} + \beta_0}_{=: f(\boldsymbol{x})} = 0\} \tag{13.1}$$

with the corresponding classification rule

$$G(\boldsymbol{x}) = \operatorname{sgn}\left[f(\boldsymbol{x})\right] \tag{13.2}$$

(sgn stands for the signum function). The function value $f(\boldsymbol{x}_0)$ denotes the signed distance of the observation $\boldsymbol{x}_0$ to the hyperplane $f(\boldsymbol{x}) = 0$ (in the case $||\boldsymbol{\beta}|| \neq 0$ the signed distance would be $\frac{1}{||\boldsymbol{\beta}||} f(\boldsymbol{x})$). If an observation $\boldsymbol{x}_i$ is correctly classified, then $g_i = \operatorname{sgn}\left[f(\boldsymbol{x}_i)\right]$ and therefore $g_i f(\boldsymbol{x}_i) > 0$.

13.2.1 The separable case

In the separable case we can always find a function $f : \mathbb{R}^p \to \mathbb{R}$ defined in (13.1) with $g_i f(\boldsymbol{x}_i) > 0 \; \forall i \in \{1, \ldots, n\}$, which means the observations can be completely separated by a hyperplane. The distance M is defined as the minimal distance of a point to the hyperplane taken over all observations:

$$M = \min_{i=1,\ldots,n} g_i f(\boldsymbol{x_i})$$

An example for the two-dimensional case is shown in Figure 13.3.

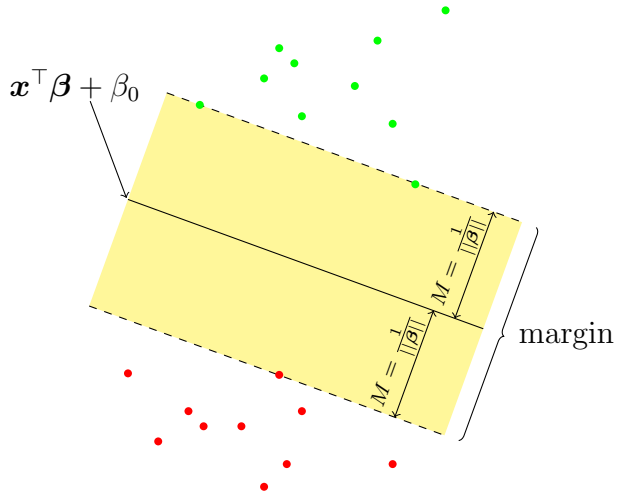

Figure 13.3: The separable case.

The yellow band on both sides of the hyperplane in Figure 13.3, which is exactly $2M$ units wide, is called the *margin*. The aim is to find the biggest margin between the training data of the two classes by solving the optimisation problem

$$\begin{aligned} &\max_{\boldsymbol{\beta},\beta_0} M \\ &||\boldsymbol{\beta}||=1 \\ \text{s.t.} \quad &g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta} + \beta_0) \geq M \quad i = 1, \ldots, n. \end{aligned} \tag{13.3}$$

This approach provides a unique solution for the problem of finding a separating hyperplane between the classes. The side conditions ensure that the distance of each data point to the decision boundary is at least M and the main condition seeks the maximal M over the parameters $\boldsymbol{\beta}$ and β_0, which define the hyperplane.
We still have the constraint that $\boldsymbol{\beta}$ has to be normed, thus $||\boldsymbol{\beta}|| = 1$, but we can avoid it by replacing the side conditions in (13.3) by

$$\frac{1}{||\boldsymbol{\beta}||} g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta} + \beta_0) \geq M.$$

This leads to new coefficients $\tilde{\boldsymbol{\beta}}$ and $\tilde{\beta}_0$ with $||\tilde{\boldsymbol{\beta}}|| = 1$. Since for any $\boldsymbol{\beta}$ and β_0 satisfying this inequality, any positively scaled multiple satisfies it too, we can arbitrarily set $M = 1/||\boldsymbol{\beta}||$, which leads to an equivalent formulation of problem (13.3):

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} ||\boldsymbol{\beta}|| \\ \text{s.t.} \quad &g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta} + \beta_0) \geq 1 \quad i = 1, \ldots, n \end{aligned} \tag{13.4}$$

Problem (13.4) is a convex optimisation problem with a quadratic criterion and linear inequality constraints. When using the Lagrange method to solve it, one has to minimize the *Lagrange primal function* defined as

$$L_p = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{n} \alpha_i [g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - 1] \tag{13.5}$$

with respect to $\boldsymbol{\beta}$ and β_0. The *Lagrange multipliers* α_i have to be nonnegative, $\alpha_i \geq 0$. As norm functions only take values in $\mathbb{R}_+$ we can square the target function and add the costant $\frac{1}{2}$ without changing the resulting minimum. Assuming $||.||$ stands for the Euclidean norm, which means $||\boldsymbol{\beta}||^2 = \boldsymbol{\beta}^\top \boldsymbol{\beta}$, this modification makes the derivation easier:

$$\frac{\partial L_p}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{n} \alpha_i g_i \boldsymbol{x}_i$$
$$\frac{\partial L_p}{\partial \beta_0} = \sum_{i=1}^{n} \alpha_i g_i$$

Setting these derivatives to zero and substituting them in (13.5) results in the *Lagrange dual function*

$$\begin{aligned} L_d &= \frac{1}{2}(\sum_{i=1}^{n} \alpha_i g_i \boldsymbol{x}_i)^\top (\sum_{j=1}^{n} \alpha_j g_j \boldsymbol{x}_j) - (\sum_{i=1}^{n} \alpha_i g_i \boldsymbol{x}_i)^\top (\sum_{j=1}^{n} \alpha_j g_j \boldsymbol{x}_j) - \beta_0 \underbrace{\sum_{i=1}^{n} \alpha_i g_i}_{=0} + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \boldsymbol{x}_i^\top \boldsymbol{x}_j, \end{aligned} \tag{13.6}$$

which gives a lower bound on the objective function (13.4). The solution of problem (13.4) is then obtained by solving the following simpler convex optimisation problem:

$$\begin{aligned} &\max_{\alpha_i} \left[\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \right] \\ &\text{s.t.} \quad \alpha_i \geq 0 \text{ for } i = 1, \ldots, n \end{aligned} \tag{13.7}$$

In order to be optimal, the solution of problem (13.7) also has to satisfy the so-called *Karush-Kuhn-Tucker conditions*

$$\sum_{i=1}^{n} \alpha_i g_i \boldsymbol{x}_i = \boldsymbol{\beta} \tag{13.8}$$
$$\sum_{i=1}^{n} \alpha_i g_i = 0 \tag{13.9}$$
$$\alpha_i \left[g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - 1 \right] = 0 \quad i = 1, \ldots, n. \tag{13.10}$$

From the side condition of problem (13.7) and condition (13.10) we can see that the following implications are true for all $i = 1, \ldots, n$:

- $\alpha_i > 0 \Rightarrow g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) = 1$: the observation $\boldsymbol{x}_i$ lies on one of the boundaries of the margin; these points are called *support vectors*

- $g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) > 1 \Rightarrow \alpha_i = 0$: the observation $\boldsymbol{x}_i$ does not lie on one of the boundaries, but outside of the margin

The case that an observation lies within the margin cannot occur in the separable case. Having a closer look at condition (13.8) we can see that the solution vector $\boldsymbol{\beta}$ from problem (13.4) is a linear combination of the support vectors: if α_i is equal to zero, the i^{th} summand in (13.8) is zero and therefore only the summands of the support vectors do not vanish. Finally, the intercept β_0 can be computed by solving equation (13.10) for any of the support vectors. The separating hyperplane in Figure 13.3 has three support vectors.

13.2.2 The non-separable case

In the non-separable case the two classes overlap and we have to take into account that we cannot find a hyperplane where all points are on the correct side. There exist two types of misclassification:

- an observation $\boldsymbol{x}_i$ with $g_i = 1$ is misclassified when $f(\boldsymbol{x}_i) < 0$
- an observation $\boldsymbol{x}_j$ with $g_j = -1$ is misclassified when $f(\boldsymbol{x}_j) > 0$

In order to include unavoidable misclassified points in our optimisation problem, a so-called *slack variable* $\xi_i \geq 0$ is introduced for each side condition. The value of ξ_i is the violation of the corresponding side condition: $\xi_i > 0$ means the point $\boldsymbol{x}_i$ is on the wrong side of its margin by the amount of $M\xi_i$, $g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) < M$. Points $\boldsymbol{x}_j$ on the correct side of the margin have slack variables $\xi_j = 0$. An example is given in Figure 13.4.

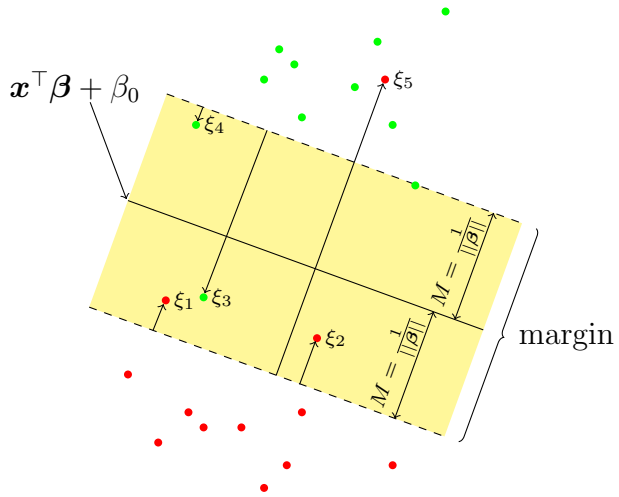

Figure 13.4: The non-separable case.

The new side conditions are $g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq M(1 - \xi_i)$, which measure the overlap of obervations on the wrong side of the margin in relative distance. As the sum of violations should be as small as possible, $\sum_{i=1}^n \xi_i \leq const$ is added to the optimisation problem. When $\xi_i > 1$ the observation $\boldsymbol{x}_i$ is misclassified, therefore the restriction $\sum_{i=1}^n \xi_i \leq const$ means the total amount of misclassified training observations has to be smaller than the constant

const. Then (13.4) can be written as

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} ||\boldsymbol{\beta}|| \\ \text{s.t.} \quad &\begin{cases} g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i \text{ for } i = 1, \ldots, n \\ \xi_i \geq 0 \text{ for } i = 1, \ldots, n \\ \sum_{i=1}^n \xi_i \leq const. \end{cases} \end{aligned} \tag{13.11}$$

Points clearly outside the margins do not play an important role for determining $\boldsymbol{\beta}$ and β_0 and thus can be ignored for shaping the class boundary. In linear discriminant analysis, by contrast, all points have influence on the decision rule through the mean vectors and covariance matrices. This property of the SVM can be useful in the presence of outliers in the data which do not lie near the decision boundary.

Problem (13.11) is also a convex optimisation problem and can be again solved by using Lagrange multipliers. As in the separable case we replace $||\boldsymbol{\beta}||$ by $\frac{1}{2}||\boldsymbol{\beta}||^2$ for easier derivation. The side condition $\sum_{i=1}^n \xi_i \leq const$ is included in the main condition by adding the term $C \sum_{i=1}^n \xi_i$, where the C is the so-called *cost parameter*. In the separable case C is set to ∞. Problem (13.11) can then be written as

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} \left[\frac{1}{2}||\boldsymbol{\beta}||^2 + C \sum_{i=1}^n \xi_i \right] \\ \text{s.t.} \quad &\begin{cases} g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i \text{ for } i = 1, \ldots, n \\ \xi_i \geq 0 \text{ for } i = 1, \ldots, n. \end{cases} \end{aligned} \tag{13.12}$$

The corresponding *Lagrange primal function*, which has to be minimized with respect to $\boldsymbol{\beta}, \beta_0$ and ξ_i, is

$$L_p = \frac{1}{2}||\boldsymbol{\beta}||^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^n \lambda_i \xi_i \tag{13.13}$$

and the required derivatives of L_p are

$$\begin{aligned} \frac{\partial L_p}{\partial \boldsymbol{\beta}} &= \boldsymbol{\beta} - \sum_{i=1}^n \alpha_i g_i \boldsymbol{x}_i \\ \frac{\partial L_p}{\partial \beta_0} &= \sum_{i=1}^n \alpha_i g_i \\ \frac{\partial L_p}{\partial \xi_i} &= C - \alpha_i - \lambda_i \quad \forall i = 1, \ldots, n. \end{aligned}$$

Additionally α_i, λ_i and ξ_i have to be nonnegative. Setting these derivatives to zero and substituting them in (13.13) we obtain the *Lagrange dual function*

$$\begin{aligned} L_d =& \frac{1}{2}(\sum_{i=1}^n \alpha_i g_i \boldsymbol{x}_i)^\top (\sum_{j=1}^n \alpha_j g_j \boldsymbol{x}_j) + \cancel{\sum_{i=1}^n (\alpha_i + \lambda_i)\xi_i} - (\sum_{i=1}^n \alpha_i g_i \boldsymbol{x}_i)^\top (\sum_{j=1}^n \alpha_j g_j \boldsymbol{x}_j) + \\ &- \beta_0 \underbrace{\sum_{i=1}^n \alpha_i g_i}_{=0} + \sum_{i=1}^n \alpha_i - \cancel{\sum_{i=1}^n \alpha_i \xi_i} - \cancel{\sum_{i=1}^n \lambda_i \xi_i} \\ =& \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j g_i g_j \boldsymbol{x}_i^\top \boldsymbol{x}_j. \end{aligned} \tag{13.14}$$

The solution of problem (13.11) is then obtained by solving the following simpler convex optimisation problem:

$$\begin{aligned} &\max_{\alpha_i} \left[\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \right] \\ &\text{s.t.} \quad 0 \leq \alpha_i \leq C \text{ for } i = 1, \ldots, n \end{aligned} \tag{13.15}$$

In order to be optimal, the solution of problem (13.15) also has to satisfy the *Karush-Kuhn-Tucker conditions*

$$\sum_{i=1}^{n} \alpha_i g_i \boldsymbol{x}_i = \boldsymbol{\beta} \tag{13.16}$$

$$\sum_{i=1}^{n} \alpha_i g_i = 0 \tag{13.17}$$

$$C = \alpha_i + \lambda_i \tag{13.18}$$

$$\lambda_i \xi_i = 0 \tag{13.19}$$

$$\alpha_i \left[g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - (1 - \xi_i) \right] = 0 \tag{13.20}$$

$$g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - (1 - \xi_i) \geq 0. \tag{13.21}$$

Conditions (13.18)–(13.21) have to hold for $i = 1, \ldots, n$. As in the separable case we can see from constraint (13.16) that the solution of $\boldsymbol{\beta}$ is a linear combination of those $\boldsymbol{x}_i$ for which $\alpha_i > 0$, the other summands are zero. These observations $\boldsymbol{x}_i$ with positive α_i are again called *support vectors* as $\boldsymbol{\beta}$ is only constructed out of them. According to condition (13.20), if $\alpha_i > 0$ then $g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) = (1 - \xi_i)$ which leads to two kinds of support vectors:

- $\xi_i = 0$: the observation $\boldsymbol{x}_i$ lies on one of the two boundaries of the margin; due to conditions (13.18) and (13.19) these vectors are characterised by $0 < \alpha_i < C$
- $\xi_i > 0$: the observation $\boldsymbol{x}_i$ does not lie on one of the two boundaries of the margin; as condition (13.19) implies $\xi_i > 0 \Rightarrow \lambda_i = 0$, these vectors are characterised by $\alpha_i = C$

Any of the margin points with $\alpha_i > 0$ and $\xi_i = 0$ can be used to determine the intercept β_0 by solving the equation $g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) = 1$. In order to obtain numerical stability, the average over all of the solutions can be computed. By changing the cost parameter C the amount of support vectors and the width of the margin changes and therefore C is a so-called *tuning parameter*.

13.3 Moving beyond linearity

Often linear hyperplanes are just a convenient approximation of a much better separation of the classes. Moreover, linear models are easy to calculate and do not easily overfit. A possible compromise between a linear model and a nonlinear decision boundary can be achieved by using transformations of the original data $\boldsymbol{x} = (x_1, \ldots, x_p)$ as input.
Let $h_m(\boldsymbol{x})$ be the m^{th} transformation of $\boldsymbol{x}$ with $h_m : \mathbb{R}^p \to \mathbb{R},\ m = 1, \ldots, M$. A *linear basis expansion* of $\boldsymbol{x}$ is then defined as

$$H(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m h_m(\boldsymbol{x})$$

Let $p < M$ and $h : \mathbb{R}^p \to \mathbb{R}^M$ be the transformation in a higher-dimensional feature space, then our new input features are $\boldsymbol{h}(\boldsymbol{x_i}) = (h_1(\boldsymbol{x_i}), \ldots, h_M(\boldsymbol{x_i}))^\top$ instead of $\boldsymbol{x}_i$ for $i = 1, \ldots, n$. We re-define the linear function $f(\cdot)$ in (13.1) to

$$f(\boldsymbol{x}) := \boldsymbol{h}(\boldsymbol{x})^\top \boldsymbol{\beta} + \beta_0, \tag{13.22}$$

with $\boldsymbol{\beta} \in \mathbb{R}^M$ and $\beta_0 \in \mathbb{R}$. As the basis function h_m are fixed, the model is linear in the new variables $\boldsymbol{h}(\boldsymbol{x})$. This fact causes that the fitting is computed as before, although we actually work with a larger feature space. The function (13.22) is nonlinear in $\boldsymbol{x}$ which results in a nonlinear classifier defined by

$$\widehat{G}(\boldsymbol{x}) = \text{sgn}\left[\hat{f}(\boldsymbol{x})\right].$$

with $\hat{f}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})^\top \hat{\boldsymbol{\beta}} + \hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}, \hat{\beta}_0$ being the estimated coefficients. In the case of Support Vector Machines typical basis expansions are polynomials and splines.

The optimisation problem (13.13) and its solution can be represented in a way that only involves the new input features as inner products:
In the following we work with the transformed feature vectors $\boldsymbol{h}(\boldsymbol{x}_i)$ instead of $\boldsymbol{x}_i$. Using the notation $\langle \cdot, \cdot \rangle$ for the inner product, the Langrangian dual function (13.14) can be written as

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \langle \boldsymbol{h}(\boldsymbol{x}_i), \boldsymbol{h}(\boldsymbol{x}_j) \rangle \tag{13.23}$$

and using constraint (13.16) the solution function $f(\boldsymbol{x})$ can be written as

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{h}(\boldsymbol{x})^\top \boldsymbol{\beta} + \beta_0 \\ &= \boldsymbol{h}(\boldsymbol{x})^\top \left(\sum_{i=1}^{n} \alpha_i g_i \boldsymbol{h}(\boldsymbol{x}_i) \right) + \beta_0 \\ &= \sum_{i=1}^{n} \alpha_i g_i \langle \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x}_i) \rangle + \beta_0. \end{aligned} \tag{13.24}$$

The intercept β_0 is again determined by solving the equation $g_i f(\boldsymbol{x}_i) = 1$ for any $\boldsymbol{x}_i$ with $0 < \alpha_i < C$. As we can see, (13.23) and (13.24) involve $\boldsymbol{h}(\boldsymbol{x})$ only through inner products and therefore we do not need to specify the transformation $\boldsymbol{h}(\cdot)$. It is enough to know a special symmetric positive (semi-) definite function $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$, the so-called *kernel function*, with

$$K(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{h}(\boldsymbol{u}), \boldsymbol{h}(\boldsymbol{v}) \rangle.$$

K computes inner products in the transformed feature space. The following three choices for K are implemented in the R-function `svm()` from the package `e1071`:

- **linear kernel**:
$$K(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u}^\top \boldsymbol{v}$$
- (d^{th}-degree) **polynomial kernel**:
$$K(\boldsymbol{u}, \boldsymbol{v}) = (c_0 + \gamma \langle \boldsymbol{u}, \boldsymbol{v} \rangle)^d \text{ for a constant } c_0 \text{ and } \gamma > 0$$
- **Radial basis kernel** (also called *RBF* (from Radial Basis Function) or *Gaussian* kernel):
$$K(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\gamma ||\boldsymbol{u} - \boldsymbol{v}||^2) \quad \text{with } \gamma > 0$$

- **Sigmoid kernel** (also called *neural network* or *hyperbolic tangent* kernel):

$$K(\boldsymbol{u}, \boldsymbol{v}) = \tanh(\gamma\langle \boldsymbol{u}, \boldsymbol{v}\rangle + c_0) \text{ for a constant } c_0 \text{ and } \gamma > 0$$

Example: In the two-dimensional case, the *polynomial kernel* with $d = 2$ and $c_0 = \gamma = 1$ has the following form for two observations $\boldsymbol{x} = (x_1, x_2)^\top$ and $\boldsymbol{x}' = (x_1', x_2')^\top$:

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{x}') &= (1 + \langle \boldsymbol{x}, \boldsymbol{x}'\rangle)^2 \\ &= (1 + x_1x_1' + x_2x_2')^2 \\ &= 1 + 2x_1x_1' + 2x_2x_2' + (x_1x_1')^2 + (x_2x_2')^2 + 2x_1x_1'x_2x_2' \end{aligned} \tag{13.25}$$

In this case we have the following basis functions:

$$h_1(\boldsymbol{x}) = 1, \quad h_2(\boldsymbol{x}) = \sqrt{2}x_1, \quad h_3(\boldsymbol{x}) = \sqrt{2}x_2, \quad h_4(\boldsymbol{x}) = x_1^2, \quad h_5(\boldsymbol{x}) = x_2^2, \quad h_6(\boldsymbol{x}) = \sqrt{2}x_1x_2$$

Taking all basis function together as $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}, \ldots, h_6(\boldsymbol{x}))^\top$ yields the same result (13.25) directly by applying the kernel function on the basis expansion:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x}')\rangle)$$

After selecting a kernel, choosing the right parameters is often a difficult task. Methods like k-fold cross validation can be used to search for them in a set of possible values.

In the nonlinear case, the cost parameter C plays an even more important role than in the linear case: A large value of C penalizes observations on the wrong side of the margin heavily and therefore only a few ξ_i, if any, will be positive. This results in a small margin and a sinuous and overfit decision boundary in the original feature space. A small value of C causes a wider margin and a smoother decision boundary, as observations on the wrong side are not penalized as heavily.

Chapter 14

Support Vector Machines with R

14.1 Introductory examples

We start with a simple 2-dimensional data set where we can see the details. The code below generates 10 observations from each of two groups, see Figure 14.1. Obviously, it will not be possible to find a perfect separating line.

```
> set.seed(1)
> x <- matrix(rnorm(20*2), ncol=2)
> y <- c(rep(-1,10), rep(1,10))
> x[y==1,] <- x[y==1,] + 1
> plot(x, col=y+3, xlab="x.1", ylab="x.2")
```

After arranging the data in an appropriate data frame, the function `svm()` from the package `e1071` can be used to fit the SVM. We have chosen a linear kernel, and the cost parameter as 10. Thus, there is a strong penalty on the slack variables.

```
> dat <- data.frame(x=x, y=as.factor(y))
> library(e1071)
> res <- svm(y~., data=dat, kernel="linear",cost=10,scale=FALSE)
> plot(res, dat) #  1 misclassified, support vectors are crosses
```

Figure 14.2 (left) shows the resulting linear separation line. Support vectors are indicated by crosses. There is one misclassified observation.

The result object contains information such as `$index` with the indexes of the support vectors. The summary command provides detailed insight.

```
> res$index # support vectors
[1]  1  2  5  7 14 16 17
> summary(res)
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5
```

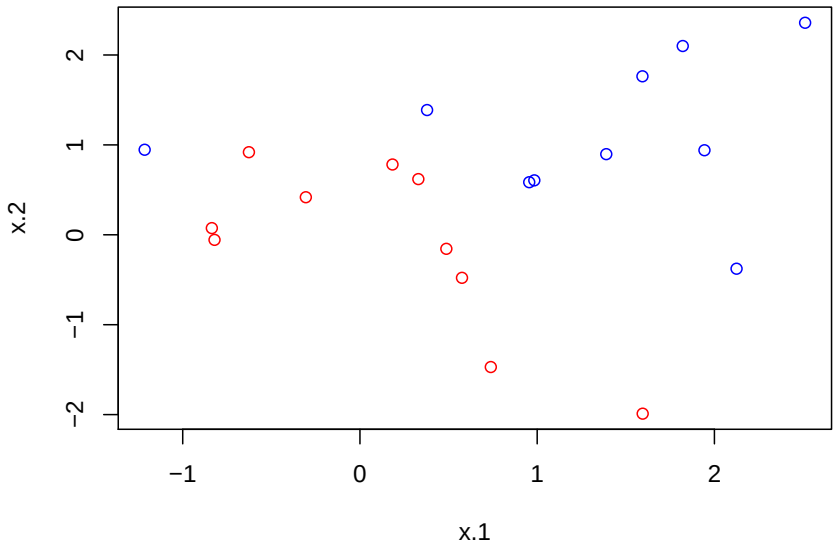

Figure 14.1: Artificial data set with 2 groups.

```
Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

Now we change the cost parameter to a much smaller value of 0.1. This means that we are less concerned about slack variables, and the margin gets cosiderably wider. This can be seen in Figure 14.2 (right), where many more observations are seen as support vectors (crosses).

```
> res1 <- svm(y~., data=dat, kernel="linear",cost=0.1,scale=FALSE)
> plot(res1, dat)
```

This makes it clear that the cost parameter is important for shaping the decision boundary. Thus we try to tune this parameter in the following, by providing a range of values for the cost parameter.

```
> set.seed(1)
> res2 <- tune.svm(y~., data=dat, kernel="linear",
+              cost=c(0.001,0.01,0.1,1,5,10,100))
> summary(res2)
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.1

- Detailed performance results:
```

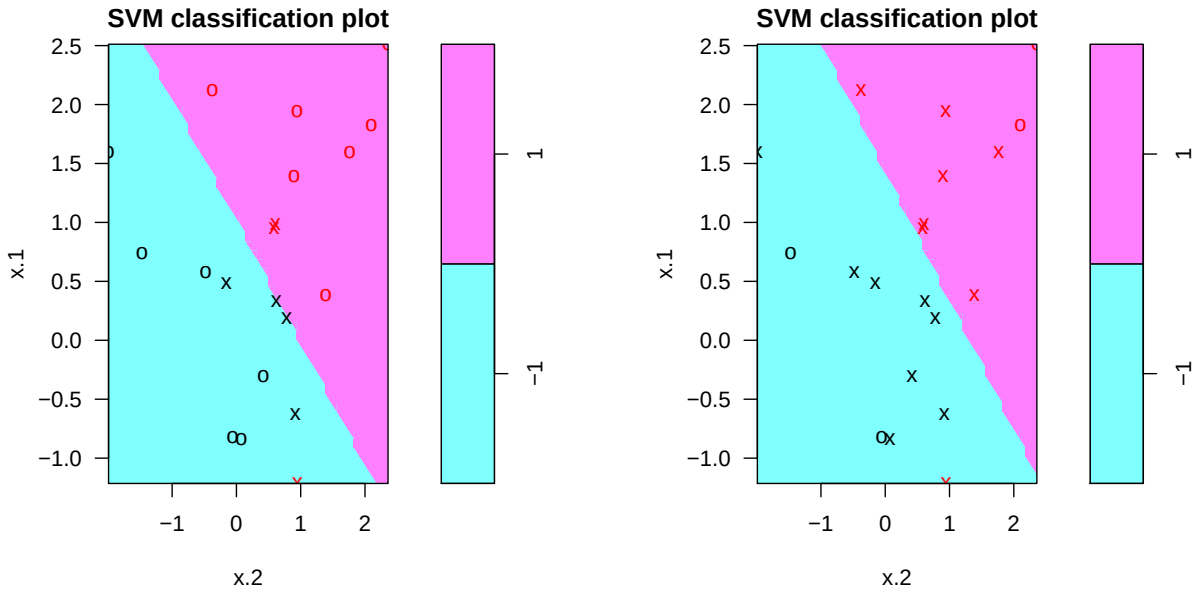

Figure 14.2: Solution of linear SVM classification for the artificial data set: left with a cost parameter of 10, right with parameter 0.1.

```
   cost error dispersion
1 1e-03  0.70  0.4216370
2 1e-02  0.70  0.4216370
3 1e-01  0.10  0.2108185
4 1e+00  0.15  0.2415229
5 5e+00  0.15  0.2415229
6 1e+01  0.15  0.2415229
7 1e+02  0.15  0.2415229
```

The summary gives details about the best choice for the cost parameter.

Now the best model can be selected and details can be derived.

```
> res2$best.model

Call:
best.svm(x = y ~ ., data = dat, cost = c(0.001, 0.01, 0.1, 1, 5,
    10, 100), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5

Number of Support Vectors:  16

> summary(res2$best.model)

Call:
best.svm(x = y ~ ., data = dat, cost = c(0.001, 0.01, 0.1, 1, 5,
    10, 100), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5

Number of Support Vectors:  16
```

```
 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

With the best model we want to do prediction. This we generate new artificial test data according to the same scheme as before, see Figure 14.3.

```
> set.seed(1)
> xtest <- matrix(rnorm(20*2), ncol=2)
> ytest <- sample(c(-1,1),20,rep=TRUE)
> xtest[ytest==1,] <- xtest[ytest==1,] +1
> plot(xtest, col=ytest+3, xlab="x.1", ylab="x.2")
> testdat <- data.frame(x=xtest, y=as.factor(ytest))
> ypred <- predict(res2$best.model, testdat)
> table(truth=ypred, prediction=testdat$y)
     prediction
truth -1  1
   -1 10  1
   1   1  8
```

From the classification table above it can be seen that 2 observations out of 20 have been misclassified.

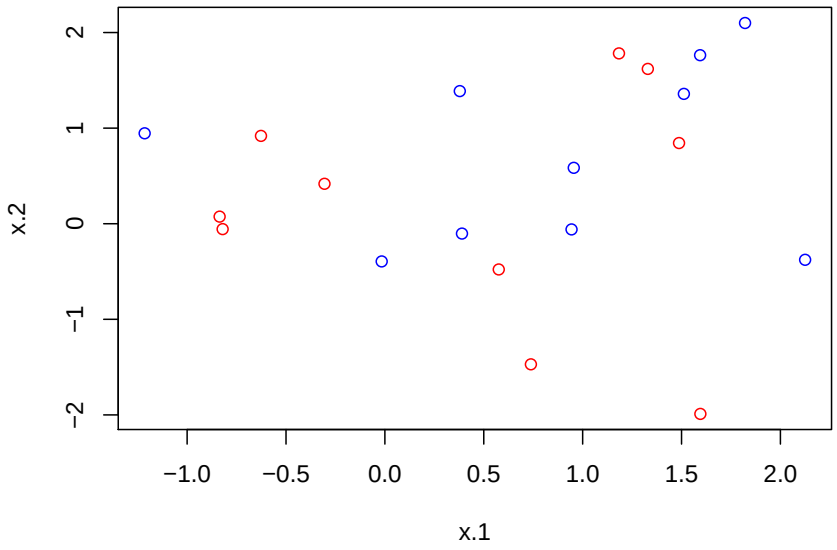

Figure 14.3: Artificial test data set with 2 groups.

Now let us proceed to nonlinear SVMs. We start generating an artificial data example with a more difficult data structure, see code below and plot in Figure 14.4.

```
> set.seed(1)
> x <- matrix(rnorm(200*2), ncol=2)
> x[1:100,] <- x[1:100,]+2
> x[101:150,] <- x[101:150,] -2
> y <- c(rep(1,150), rep(2,50))
> plot(x, col=y, xlab="x.1",ylab="x.2")
```

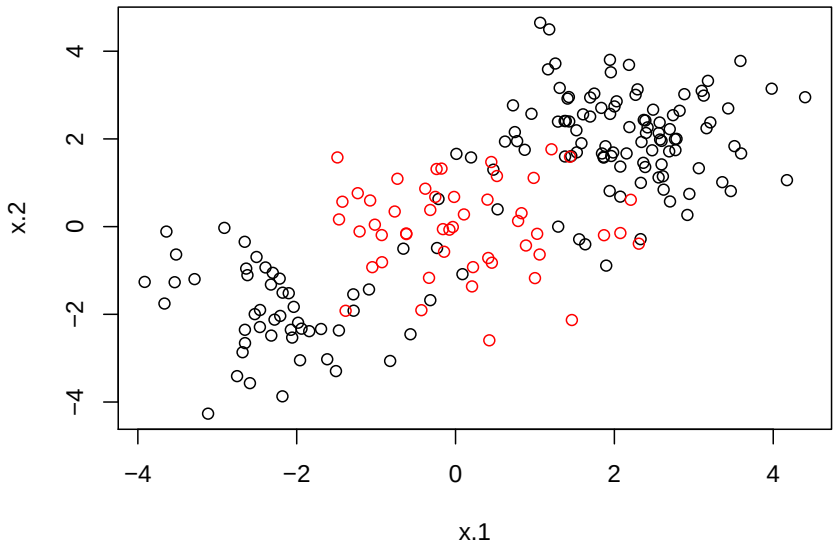

Figure 14.4: Artificial test data set with 2 groups and more difficult structure.

We use the nonlinear radial basis kernel, with default parameters for gamma and cost. This gives the separation boundary shown in Figure 14.5 (left).

```
> dat <- data.frame(x=x, y=as.factor(y))
> train <- sample(200,100)
> res <- svm(y~., data=dat[train,], kernel="radial", gamma=1, cost=1)
> plot(res, dat[train,])
```

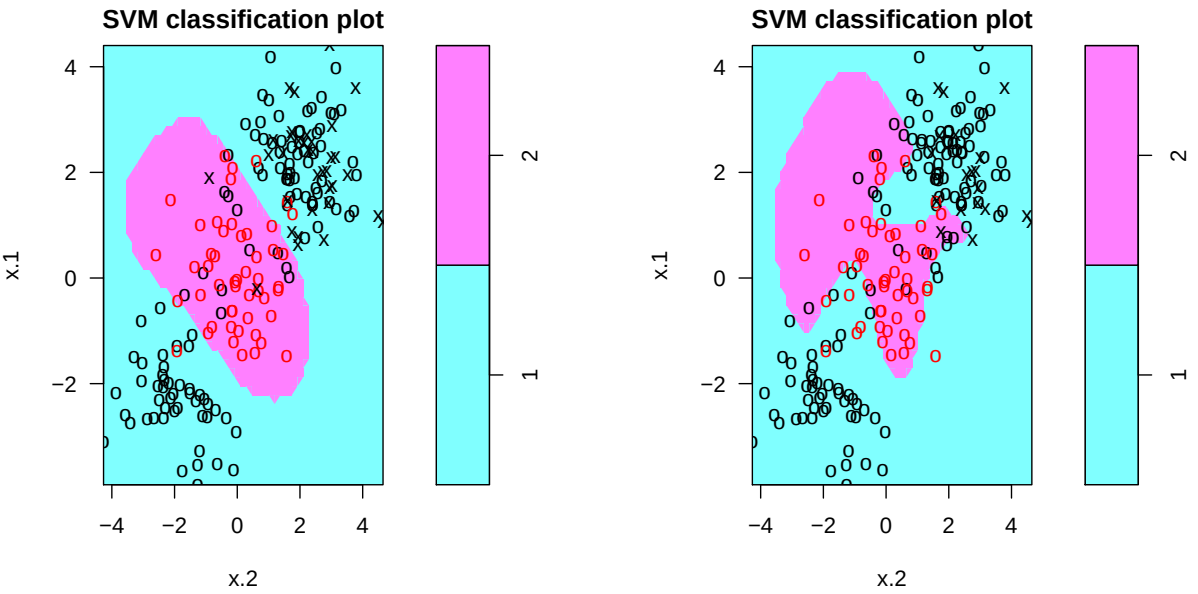

Figure 14.5: Solution of linear SVM classification for the artificial data set: left with a cost parameter of 1, right with parameter 1e5.

Figure 14.5 (right) shows the result when modifying the cost parameter to a huge value (1e5). This means, we are very concerned about support vectors, and the result is a highly nonlinear decision boundary.

```
> res1 <- svm(y~., data=dat[train,], kernel="radial", gamma=1, cost=1e5)
> plot(res1, dat[train,])
```

Finally, we tune both the gamma and the cost parameter by providing a range of values. Each comination is a potential candidate.

```
> set.seed(1)
> res2 <- tune.svm(y~., data=dat[train,], kernel="radial",
+                  cost=c(0.1,1,10,100,1000), gamma=c(0.5,1,2,3,4))
> summary(res2)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
     2    1

- best performance: 0.12

- Detailed performance results:
   gamma  cost error dispersion
1    0.5 1e-01  0.27 0.11595018
2    1.0 1e-01  0.25 0.13540064
3    2.0 1e-01  0.25 0.12692955
4    3.0 1e-01  0.27 0.11595018
5    4.0 1e-01  0.27 0.11595018
6    0.5 1e+00  0.13 0.08232726
7    1.0 1e+00  0.13 0.08232726
8    2.0 1e+00  0.12 0.09189366
9    3.0 1e+00  0.13 0.09486833
10   4.0 1e+00  0.15 0.10801234
11   0.5 1e+01  0.15 0.07071068
12   1.0 1e+01  0.16 0.06992059
13   2.0 1e+01  0.17 0.09486833
14   3.0 1e+01  0.18 0.10327956
15   4.0 1e+01  0.18 0.11352924
16   0.5 1e+02  0.17 0.08232726
17   1.0 1e+02  0.20 0.09428090
18   2.0 1e+02  0.19 0.09944289
19   3.0 1e+02  0.21 0.08755950
20   4.0 1e+02  0.21 0.08755950
21   0.5 1e+03  0.21 0.09944289
22   1.0 1e+03  0.20 0.08164966
23   2.0 1e+03  0.20 0.09428090
24   3.0 1e+03  0.22 0.10327956
25   4.0 1e+03  0.24 0.10749677

> plot(res2)
```

Figure 14.6 shows the results of parameter tuning.

The best model is finally used for prediction of the test set data.

```
> ypred <- predict(res2$best.model, dat[-train,])
> table(truth=ypred, prediction=dat$y[-train])

     prediction
truth  1  2
    1 74  7
    2  3 16
```

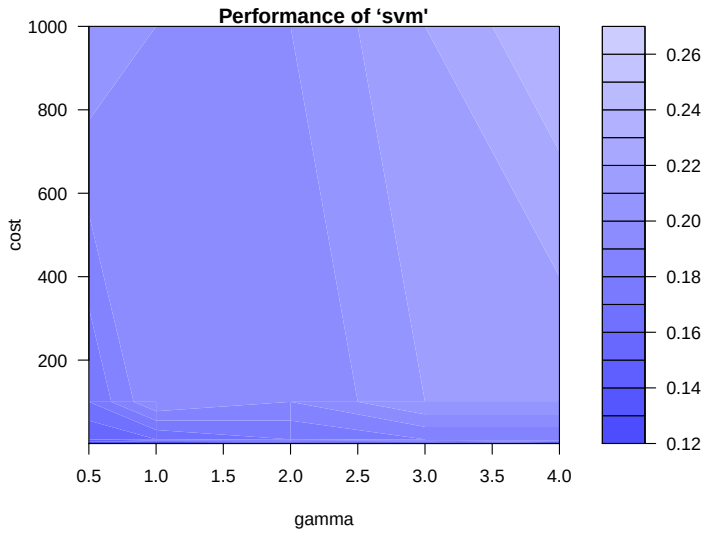

Figure 14.6: Parameter tuning for the gamma and the cost parameter.

14.2 Classification example

We consider our Pima Indian Diabetes data set with the grouping variable *diabetes*. For the computation we use the function `svm` from the `library(e1071)`. The grouping variable is defined as a factor, which causes that `svm` automatically recognizes this as a classification task.

```
> grp <- as.factor(pid[,9])
> x <- pid[,1:8]
> set.seed(100)
> train <- sample(1:nrow(pid),300)
> library(e1071)
> resSVM <- svm(x[train,],grp[train],kernel="radial")
> predSVM <- predict(resSVM,newdata=x[-train,])
```

We use a *radial basis kernel* and the default value for the parameter `gamma` within the radial basis function, which is set to `1/ncol(x)`, as well as the default for the `cost` parameter C (which is 1).

```
> TAB1 <- table(predSVM,pid[-train,9])
> mkrSVM <- 1-sum(diag(TAB1))/sum(TAB1)
> mkrSVM
[1] 0.3043478
```

The parameter γ and C are important for the usefulness of the classification model, and thus we try to optimize these parameters. For this purpose we consider various ranges for the parameters.

```
> tuneSVM <- tune.svm(x[train,],grp[train],gamma=2^(-10:0),cost=2^(-4:2),kernel="radial")
> tuneSVM
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
      gamma cost
```

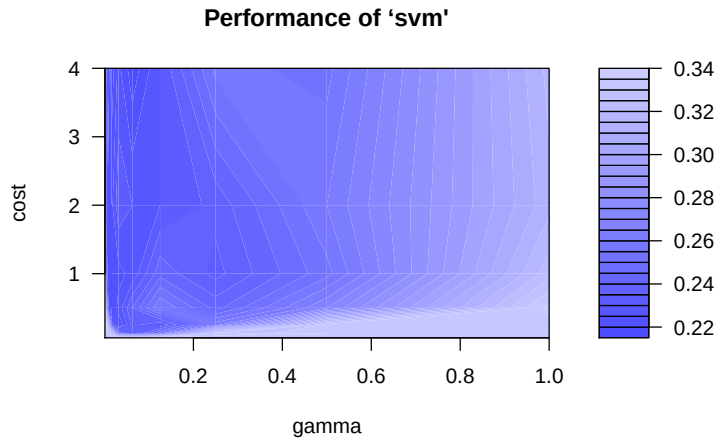

Figure 14.7: Optimimization of the parameters `gamma` and `cost`

```
 0.001953125    4

- best performance: 0.2166667

> plot(tuneSVM)
```

Figure 14.7 shows a representation of the resulting misclassification rates from the cross-validation carried out internally in `tune.svm`, where both γ and C are varied. Using the optimal values for γ and C, we evaluate the SVM on the test set.

```
> resSVM <- svm(x[train,],grp[train],kernel="radial",gamma=2^-9,cost=2^2)
> predSVM <- predict(resSVM,newdata=x[-train,])
> TAB1 <- table(predSVM,pid[-train,9])
> mkrSVM <- 1-sum(diag(TAB1))/sum(TAB1)
> mkrSVM

[1] 0.2282609
```

	INDR	LDA	QDA	RDA	GLM	GAM	SVM
MKR	0.239	0.239	0.25	0.217	0.217	0.283	0.228

The comparison with the other methods is not completely correct, because different evaluation methods have been used. Nevertheless, SVM seems to work quite well.

14.3 Regression example

We use the body fat data to illustrate that SVMs can also be used for regression. The data preprocessing is done as earlier.

```
> library("UsingR")
> data(fat)
> attach(fat)
> fat$body.fat[fat$body.fat==0]<-NA
> fat<-fat[,-cbind(1,3,4,9)]
> fat<-fat[-42,]
```

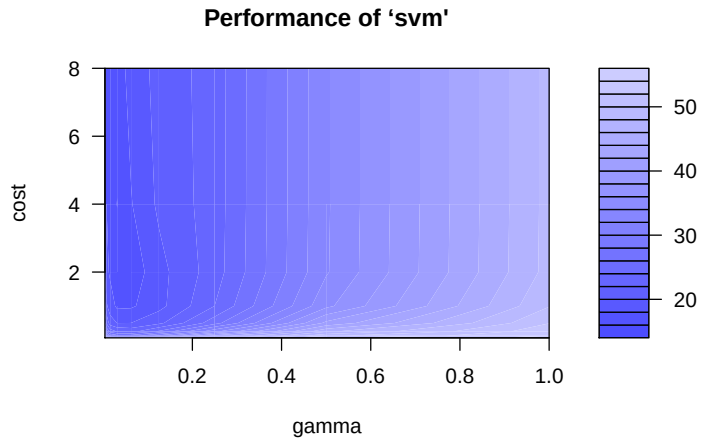

Figure 14.8: Optimimization of the parameters `gamma` and `cost`

```
> fat[,4]<-fat[,4]*2.54
> fat <- na.omit(fat)
```

We randomly select a training set of 150 observations, and evaluate the prediction quality of the model on the remaining test set of about 100 observations.

```
> set.seed(100)
> train=sample(1:nrow(fat),150)
```

Now we tune the parameters `gamma` and `cost` by means of cross-validation.

```
> tuneSVM <- tune.svm(fat[train,-1],fat[train,1],gamma=2^(-8:0),cost=2^(-4:3),kernel="radial")
> tuneSVM

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
   gamma cost
 0.03125    4

- best performance: 15.9481

> plot(tuneSVM)
```

Note that the scale used in Figure 14.8 corresponds to the MSE, and not to the RMSE. Using the optimized parameter from Figure 14.8, we use this model to fit the test data set.

```
> resSVM <- svm(body.fat~.,data=fat,subset=train,kernel="radial",gamma=2^-5,cost=2^2)
> predSVM <- predict(resSVM,newdata=fat[-train,])
> RMSEtest <- sqrt(mean((fat$body.fat[-train]-predSVM)^2))
> RMSEtest

[1] 5.1158
```

Compared to previous results (PCR, PLS), SVM regression has not improved the prediction. On the other hand, previously we did not use training and test data. A visual impression of measured versus predicted values is presented in Figure 14.9.

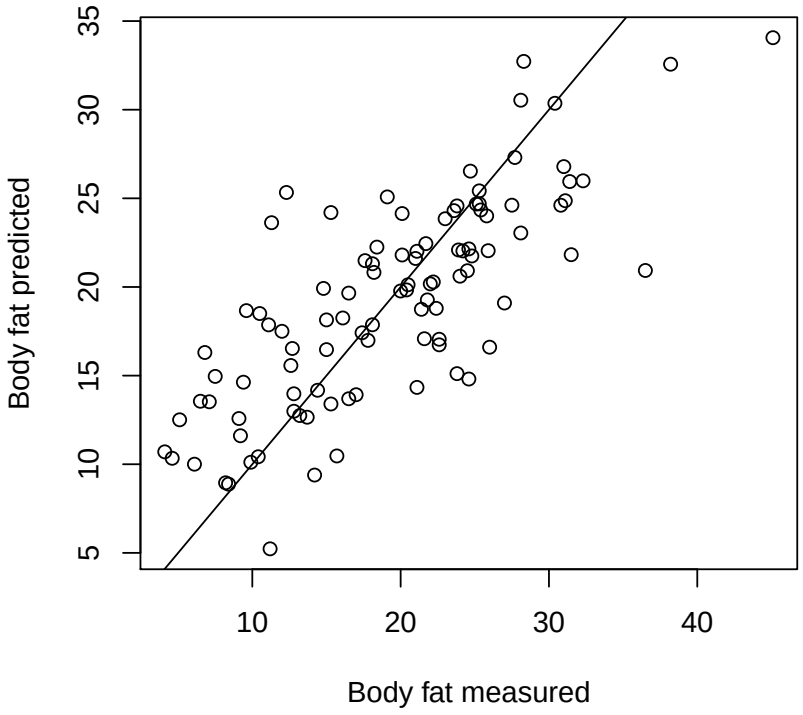

Figure 14.9: Measured versus predicted (SVM) values of *bodyfat* for the test data.

Bibliography

H. Bozdogan. Akaike's Information Criterion and Recent Developments in Information Complexity. *Journal of Mathematical Psychology*, 44:62 – 91, 2000.

K.P. Burnham and D.R. Anderson. Multimodel Inference: Understanding AIC and BIC in model selection. *Sociological Methods Research*, 33:261 – 304, 2004.

D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. A Bradford Book, The MIT Press, Cambridge, Massachusetts, 2001.

M. H. Hansen, J. Z. Huang, C. Kooperberg, C. Stone, and Y. K. Truong. Statistical modeling with spline functions: Methodology and theory. http://bear.fhcrc.org/ clk/monopdf/mono.html, January 2006.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer - Verlag, New York, 2001.

M.H. Kutner and C.J. Nachtsheim. *Applied Linear Regression Models*. McGraw-Hill / Irwin, Chicago, 2004.

P. Schönfeld. *Methoden der Ökonometrie, Band I*. Verlag Franz Vahlen GmbH, Berlin, 1969.