Advanced Methods of Classification and Regression

Lecture Notes

Prof. Dr. Peter Filzmoser

Peter.Filzmoser@tuwien.ac.at

Institute of Statistics and Mathematical Methods in Economics

Vienna University of Technology, Austria

Vienna, August 2019

 The manuscript is essentially based on the book "The Elements of Statistical Learning" by Hastie, Tibshirani and Friedman.

Contents

Chapter 1

Preliminaries

1.1 The linear regression model

The aim of the regression model is to describe the output variable y through a linear combination of one or more input variables $x_1, x_2, \ldots, x_p$. "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. The multiple linear model has the form

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

with the linear function

$$f(x_1, x_2, \ldots, x_p) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j. \tag{1.1}$$

The goal is to find a functional relation f which justifies

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

and respectively

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

with the error term ε, which should be as small as possible. More statistical assumptions about the error term will be stated later on. The 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:

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

The predicted values $\hat{y}_i$ should be as close as possible to the real measured values y_i. The definition of "as close as possible" as well as an estimation of how good the model actually is, will be given in the next chapters.

Let us have a look at the model for every observation $i = 1, \ldots, n$. We get

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

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 variance σ^2, thus $\varepsilon_i \sim N(0, \sigma^2)$. Thus, the variance σ^2 has to be the same for every observation, and this way the errors are asked to always show the same spread.

1.2 Model evaluation by 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.

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

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).

1.2.1 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

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

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 $n_{\mathrm{Train}}/n_{\mathrm{Test}} = 2/1$ or $1/1$ is often used, this means $\frac{n-n/q}{n/q} = \frac{2}{1}$.

1.2.2 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

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

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, $\widehat{\mathrm{Err}}_{\mathrm{Boot}}$ 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.

Chapter 2

Linear regression in practice

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.

2.1 Least Squares (LS) regression

2.1.1 Parameter estimation

The multiple linear regression model has the form

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

with the n observed values

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

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 term

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

The parameters we are looking for are the regression coefficients

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}.$$

The most popular estimation method is least squares, in which we choose the coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^\top$ which minimize the residual sum of squares (RSS)

$$\begin{aligned} \mathrm{RSS}(\boldsymbol{\beta}) &= \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i))^2 \\ &= \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 \end{aligned}$$

This approach makes no assumptions about the validity of the model, it simply finds the best linear fit to the data.

How do we minimize the RSS?

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. The unique solution $\hat{\boldsymbol{\beta}}$ fro the regression coefficients $\boldsymbol{\beta}$ is

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

where we have to assume that the inverse of $\boldsymbol{X}^\top \boldsymbol{X}$ exists. This means 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).

Now the estimated regression parameters $\hat{\boldsymbol{\beta}}$, that provide the best fit in the case of the RSS minimization can be used for the prediction. If one only wants to predict for the available data $\boldsymbol{x}_i$, the estimated value is

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

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 sometimes called the "hat matrix" because it puts the hat on $\boldsymbol{y}$.

2.2 Shrinkage methods

The least-squares estimator is unbiased, which means that $\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$. In general, if we consider an estimator $\hat{\theta}$ for a parameter θ, then we have:

$$\mathrm{MSE}(\hat{\theta}) := \mathbb{E}(\hat{\theta} - \theta)^2 = \mathrm{Var}(\hat{\theta}) + [\underbrace{\mathbb{E}(\hat{\theta}) - \theta}_{\text{Bias}}]^2$$

Thus, for the least-squares estimator $\hat{\boldsymbol{\beta}}$ we have:

$$\mathrm{MSE}(\hat{\boldsymbol{\beta}}) = \mathrm{Var}(\hat{\boldsymbol{\beta}}) + \underbrace{[\mathbb{E}(\hat{\boldsymbol{\beta}}) - \boldsymbol{\beta}]^2}_{=0}$$

However, an estimator $\tilde{\boldsymbol{\beta}}$ with

$$\mathrm{MSE}(\tilde{\boldsymbol{\beta}}) < \mathrm{MSE}(\hat{\boldsymbol{\beta}})$$

could exist, with $\mathbb{E}(\tilde{\boldsymbol{\beta}}) \neq \boldsymbol{\beta}$ (biased) and $\mathrm{Var}(\tilde{\boldsymbol{\beta}}) < \mathrm{Var}(\hat{\boldsymbol{\beta}})$. A smaller MSE could thus lead to a better prediction of new values. Such estimators $\tilde{\boldsymbol{\beta}}$ can be obtained 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.

2.2.1 Ridge regression

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

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{2.1}$$

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 (2.1) in matrix form,

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

$\boldsymbol{I}$ 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

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

  thus

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

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

Section 3.2.1, page 12

2.2.2 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.

Figure 2.1 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.

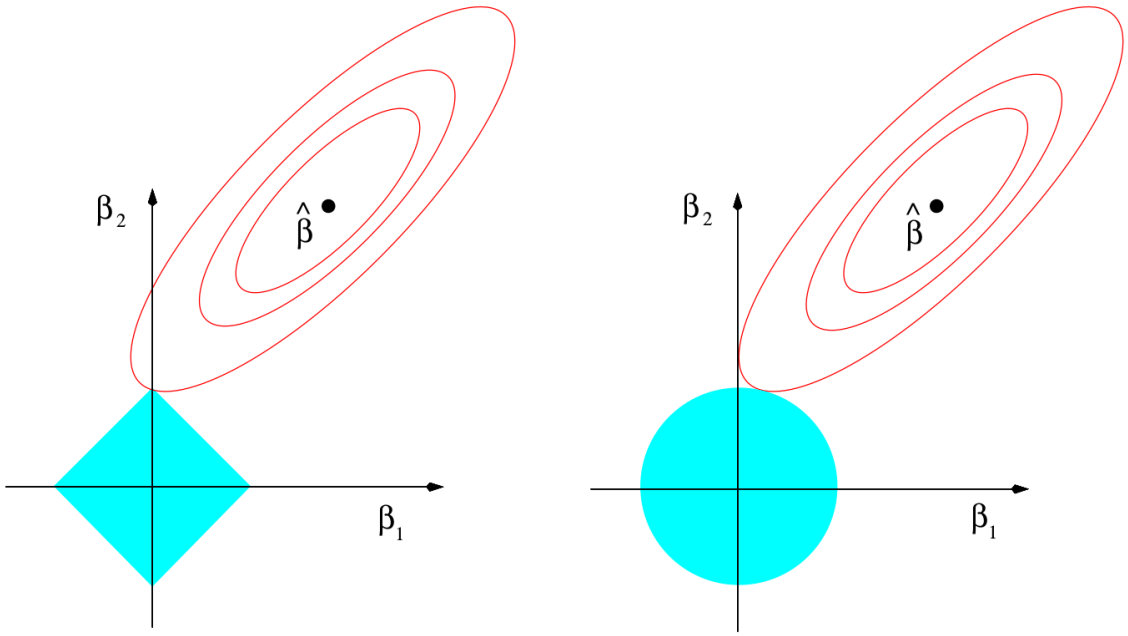

Figure 2.1: Penalties for Lasso (left) and Ridge (right) regression.

Section 3.2.2, page 14

Chapter 3

Linear regression in R

3.1 Least Squares (LS) regression in R

3.1.1 Example: 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.

3.1.2 LS regression 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 3.1 show the measured versus the predicted response variable for the test data, resulting from the full model.

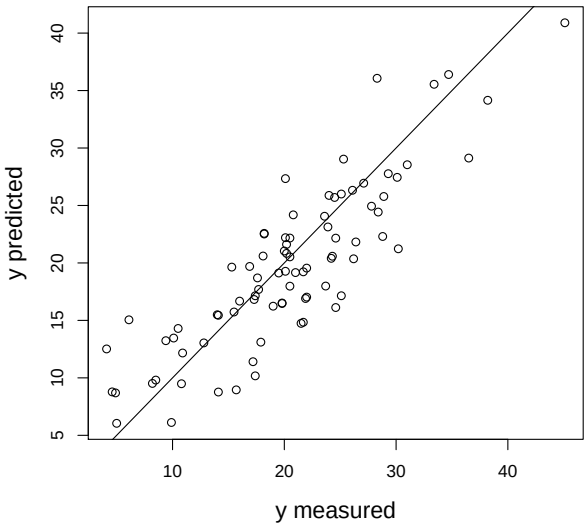

Figure 3.1: Prediction resulting from full model for test data.

3.2 Shrinkage methods in R

3.2.1 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 3.2.

```
> 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 3.2 shows a very clear minimum at a specific value of λ. This value can be identified with the `select()` function.

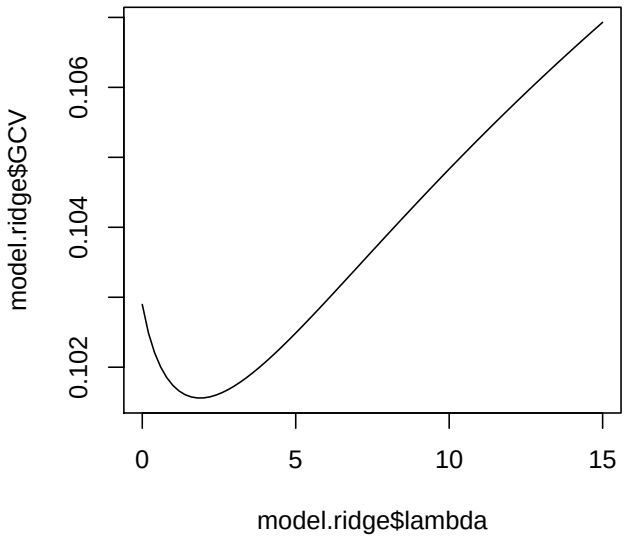

Figure 3.2: Optimal selection of the tuning parameter λ with GCV.

```
> 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 3.3, 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
 7.42343629 -1.73224089  0.89443532 -0.02883642 -0.54814379  0.25192251
   forearm       wrist
 0.58262075 -1.29940964
```

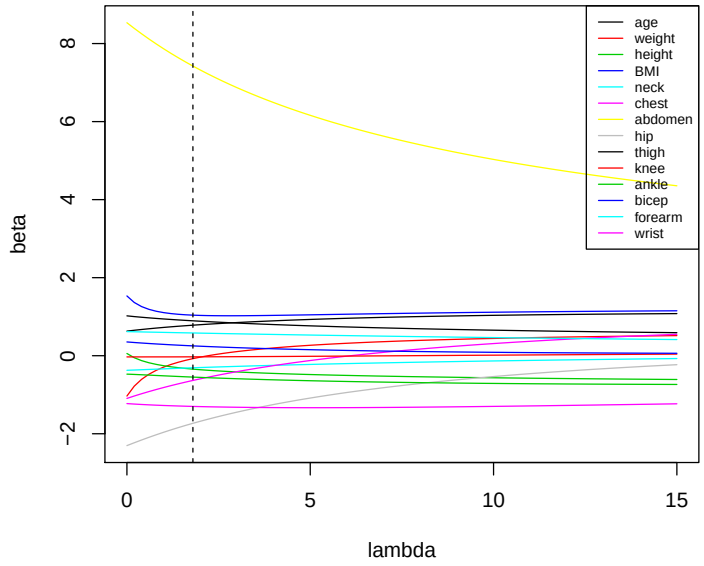

Figure 3.3: Ridge coefficients with varying λ. The optimal tuning parameter is indicated with the dashed line.

```
> 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
```

3.2.2 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 3.4.

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

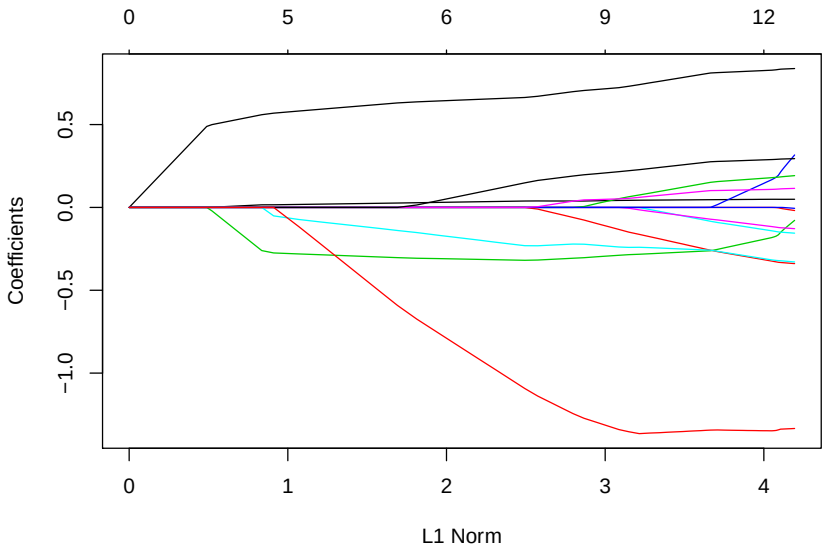

Figure 3.4: Lasso regression coefficients for varying values of the size of the L_1 norm. On top we can see the number of variables in the model.

To identify the appropriate tuning parameter, we run a cross-validation routine and plot the result, see Figure 3.5.

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

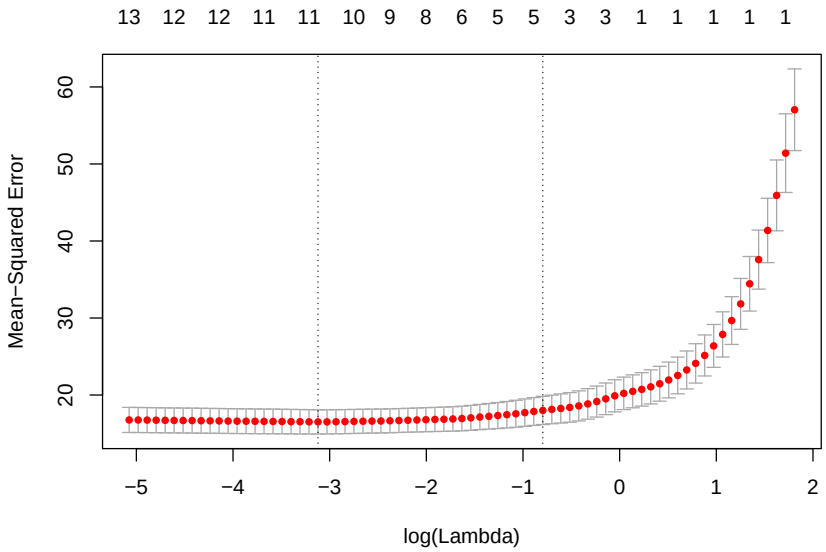

Figure 3.5: Cross-validated MSE for different values of λ in Lasso regression.

Figure 3.5 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 3.6.

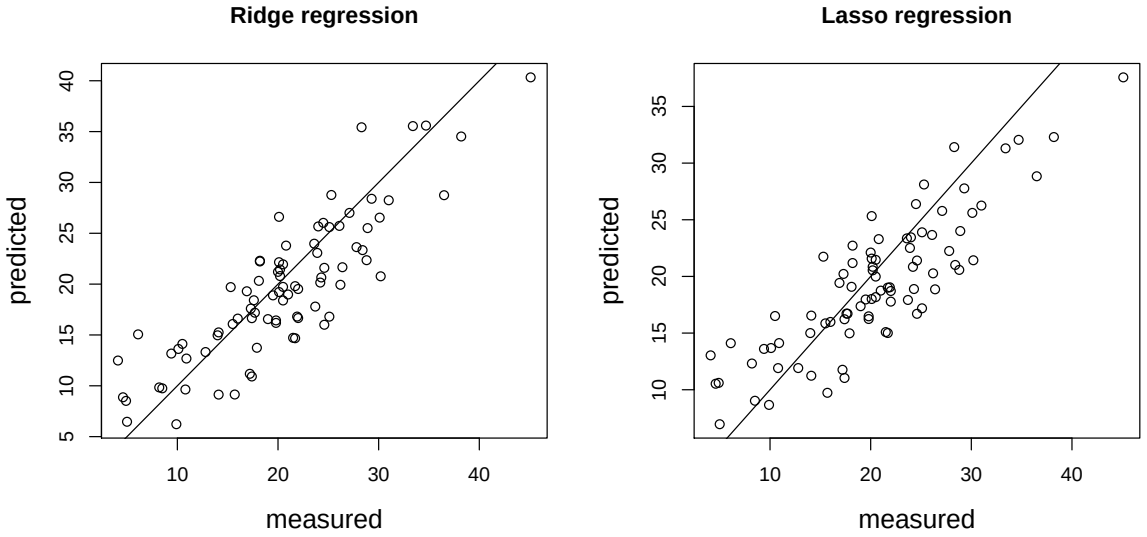

Figure 3.6: Measured versus predicted response for the test data, for Ridge (left) and LAsso (right) regression.

Chapter 4

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.

4.1 Interpolation with splines

A spline is created by joining together several functions [compare **?**]. 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.

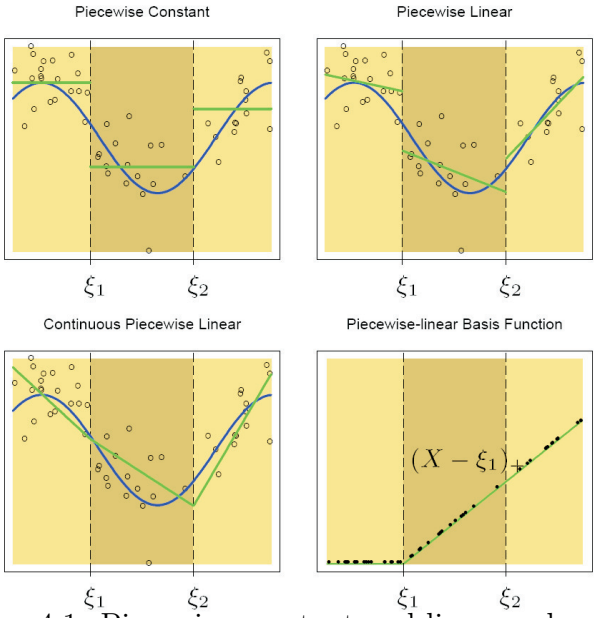

Figure 4.1: Piecewise constant and linear polynomials

Figure 4.1 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 4.1 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 4.1.

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 4.2 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.

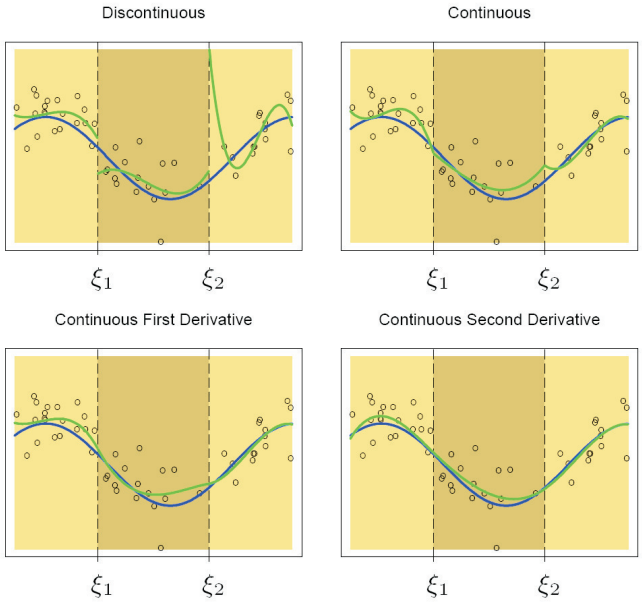

Figure 4.2: Piecewise cubic polynomials

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. This can be avoided by using natural cubic splines. 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).

4.2 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:

$$\mathrm{RSS}(f,\lambda) = \sum_{i=1}^{n}\{y_i - f(x_i)\}^2 + \lambda \int f''(t)^2 dt \tag{4.1}$$

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

(4.1) 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 (4.1) thus reduces to

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

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

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

4.2.1 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 $\boldsymbol{S_\lambda}$ is known as "smoother matrix". Due to the linearity, the computation of $\hat{\boldsymbol{f}}$ is independent from $\boldsymbol{y}$, because $\boldsymbol{S_\lambda}$ only depends on x_i and λ.

We define the "effective degrees of freedom" 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 5.5) we obtain:

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


Section 5.1, page 23

Chapter 5

Basis expansions in R

5.1 Smoothing splines in R

Consider a simulated data set according to a sine function, see R code below and Figure 5.1.

```
> 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))
```

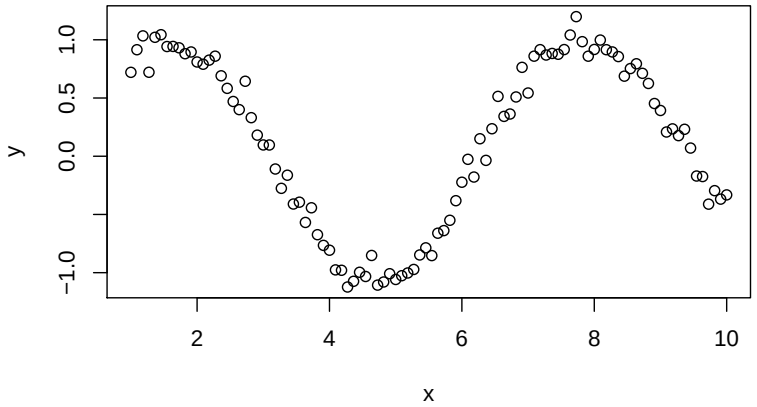

Figure 5.1: Simulated data according to a sine function

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")
```

- *Prediction outside the range*

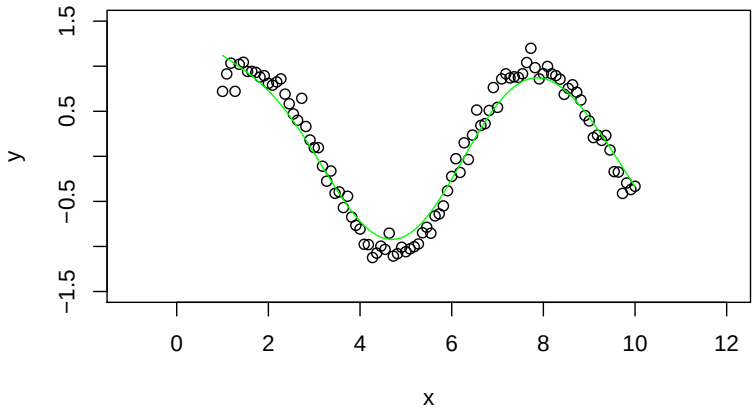

Figure 5.2: Fit with smoothing splines.

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

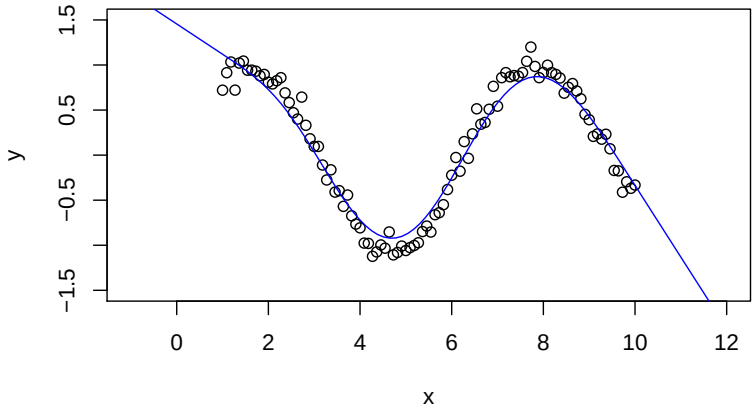

Figure 5.3: Prediction at the boundaries with smoothing splines

- *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")
```

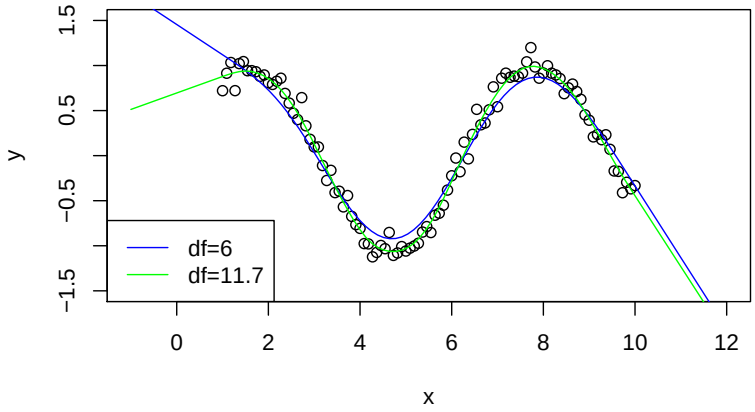

Figure 5.4: Choice of degrees of freedom

- *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)
```

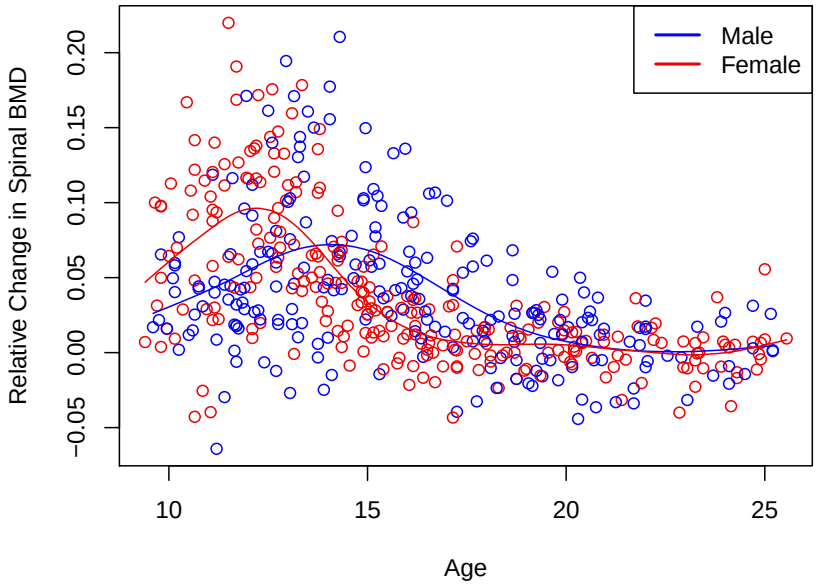

Figure 5.5: Smoothing spline fits for the bone density data.

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 6

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 **?**]. 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.

6.1 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.

6.2 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 4.2. 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 7, page 29

Chapter 7

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 7.1). The function **gam()** can be found in `library(mgcv)`

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

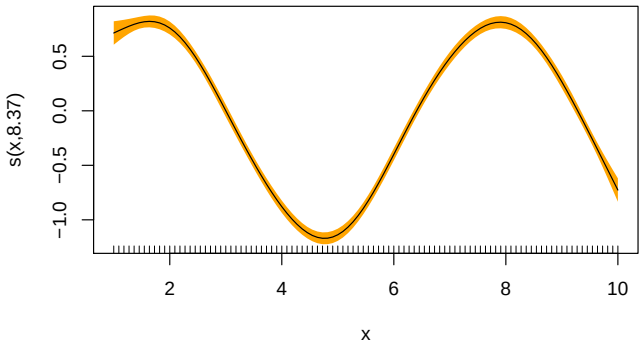

Figure 7.1: Fit with GAM

- *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)
```

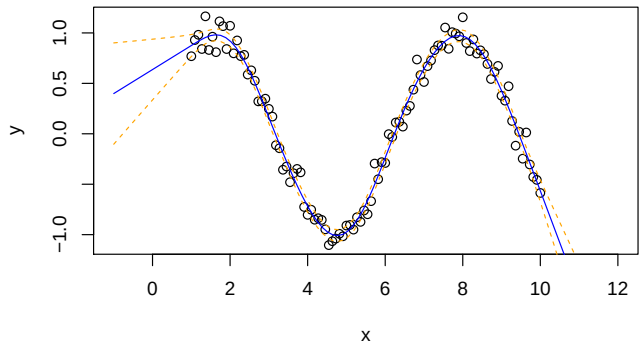

Figure 7.2: Prediction with GAM

- *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 7.3). The estimated GCV value of 0.14 indicates that the model provides a good fit.

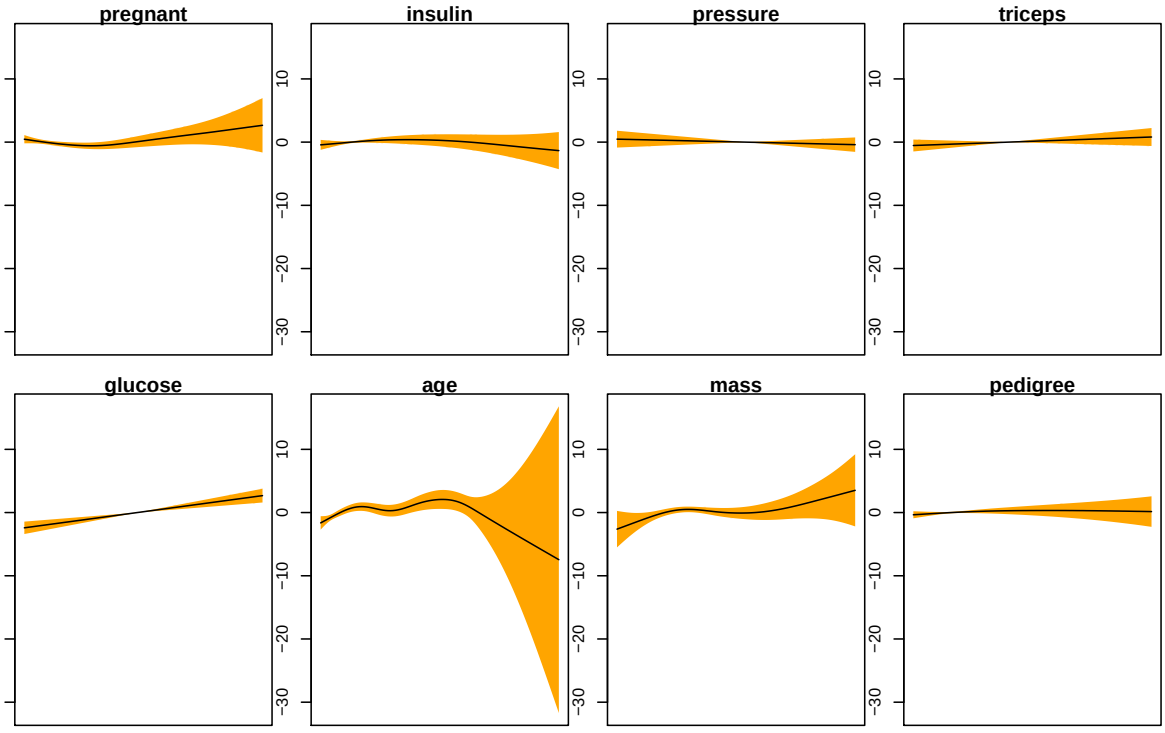

Figure 7.3: Fit of the different terms with additional 95% confidence region

- *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
```

Chapter 8

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.

8.1 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 9.1, page 38

8.2 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 8.1):

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

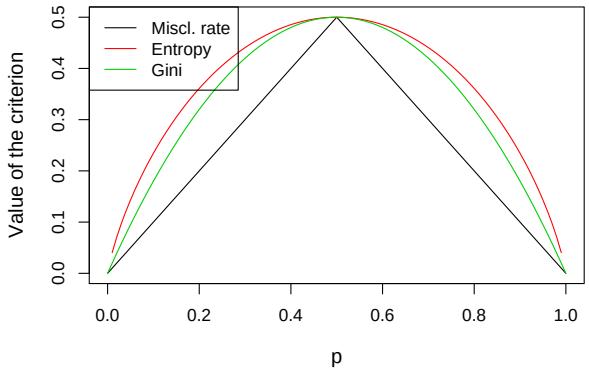

Figure 8.1: Impurity measures for two class classification

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 9.2, page 41

8.3 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 9

Tree based methods in R

9.1 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)
```

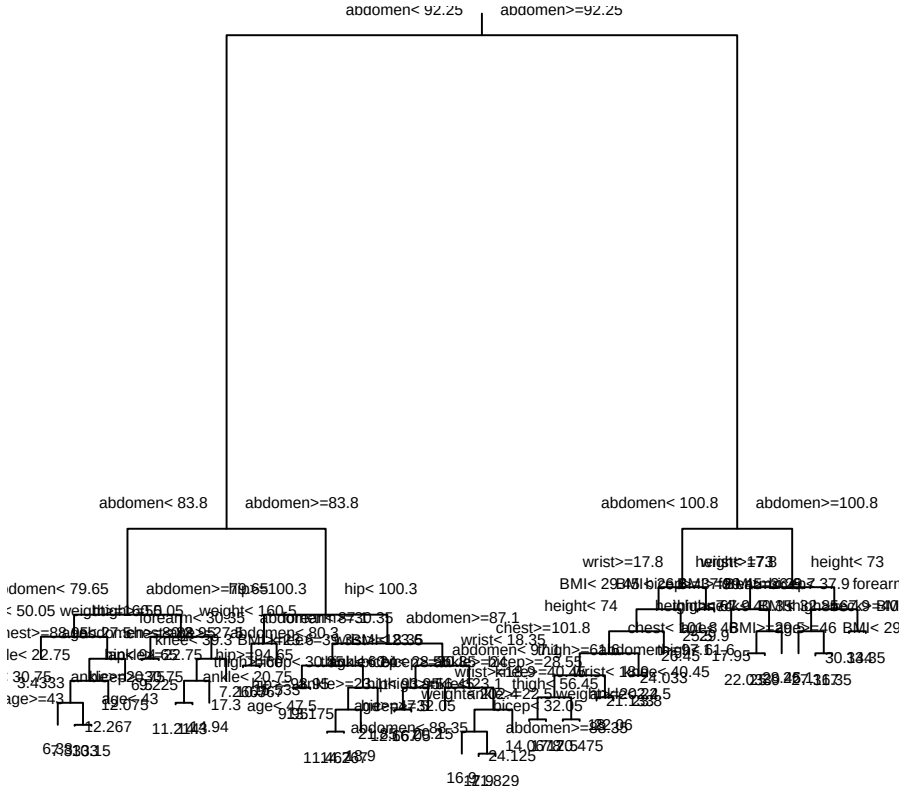

Figure 9.1: Regression tree of the body fat data

- 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)
```

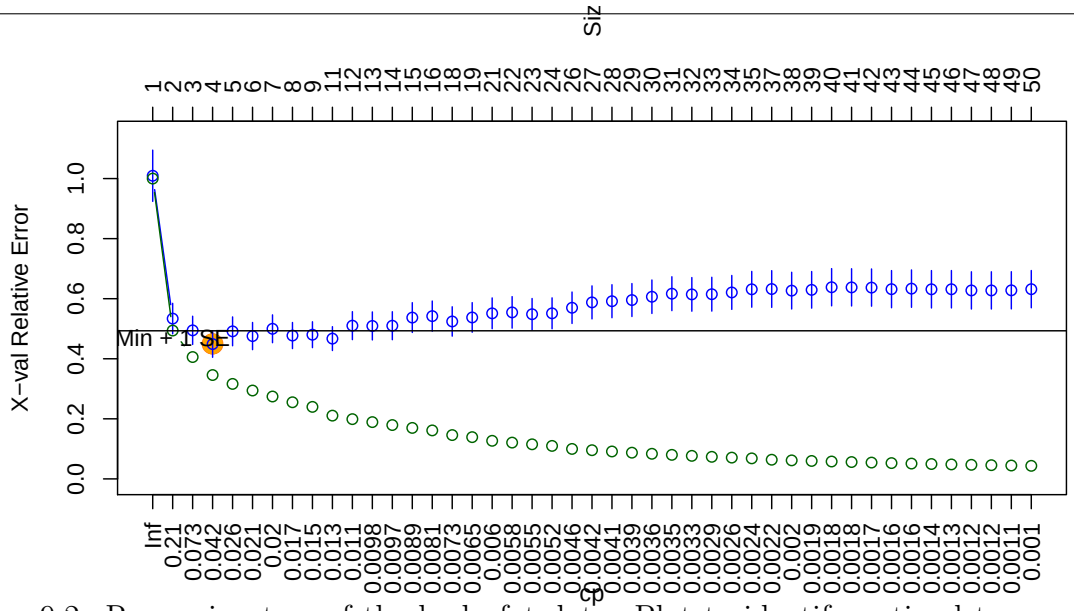

Figure 9.2: Regression tree of the body fat data: Plot to identify optimal tree complexity parameter.

Figure 9.2 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 9.3, is much smaller, and it has clearly higher predictive power.

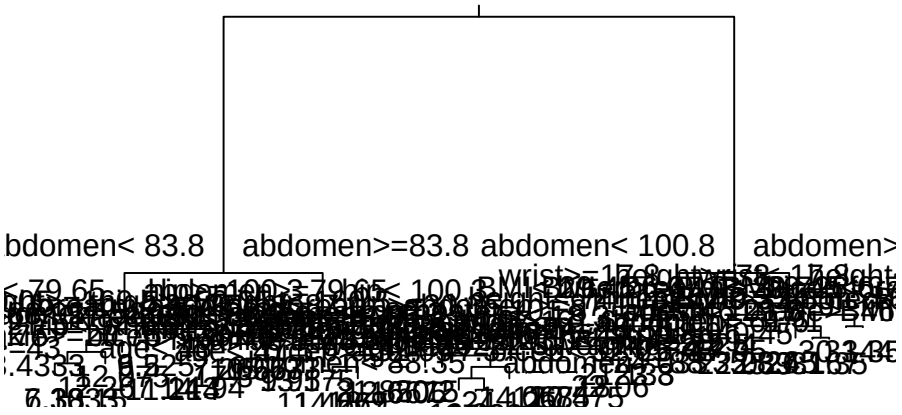

Figure 9.3: Final regression tree of the body fat data

9.2 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)
```

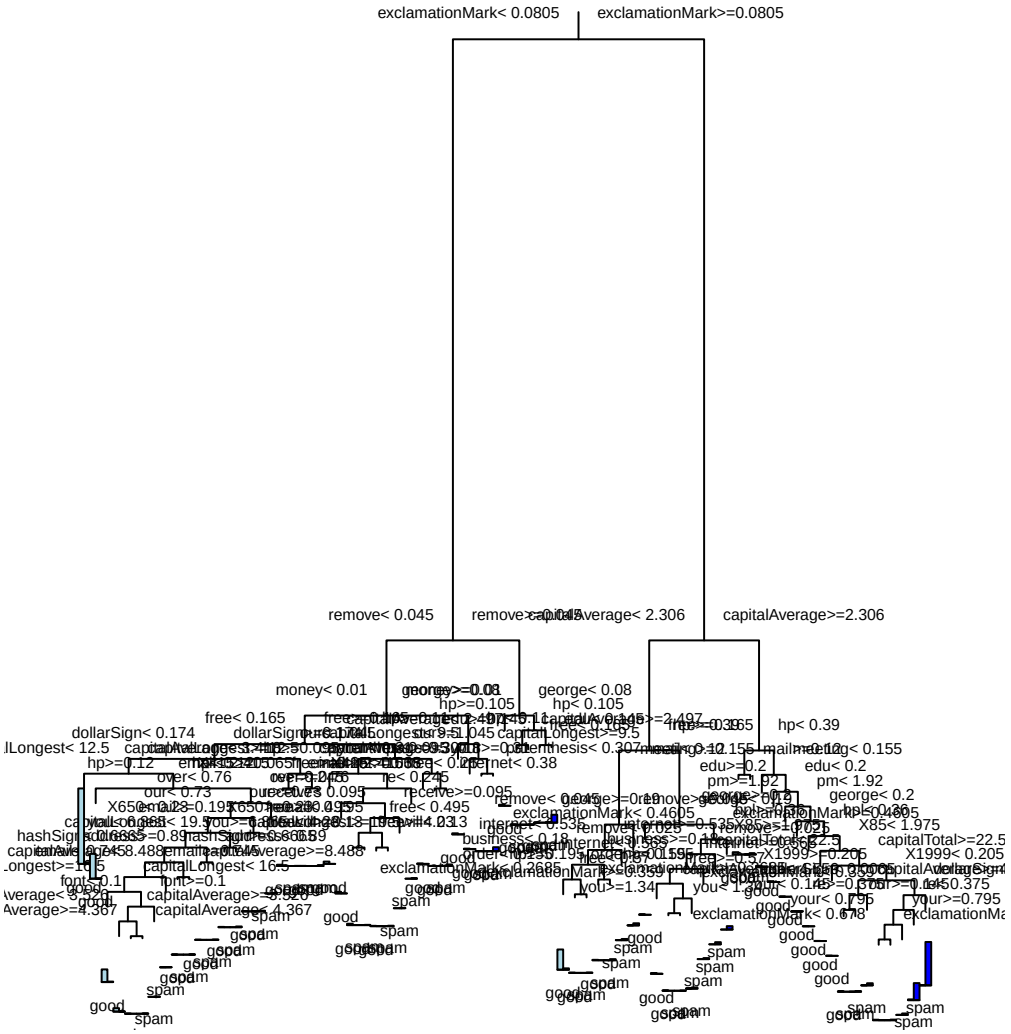

Figure 9.4: Classification tree of spam data

- *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")
```

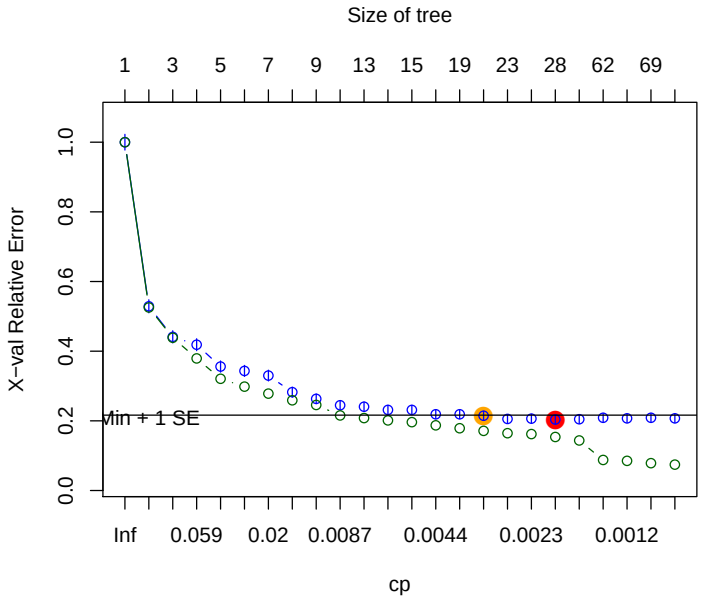

Figure 9.5: Cross-validation results of tree1: cv-error with standard error (blue) and error in test set (green)

Figure 9.5 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 $\hat{\alpha}$.

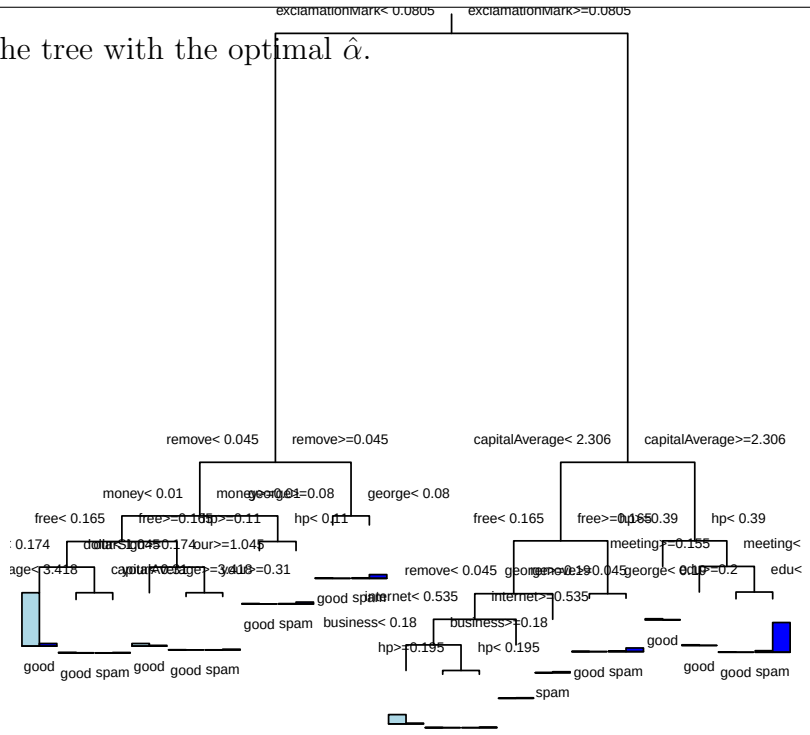

Figure 9.6: Result of the pruning of tree1

- *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
```

9.3 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)
```

Chapter 10

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.

10.1 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 10.1) 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 $f(\boldsymbol{x}) = 0$.

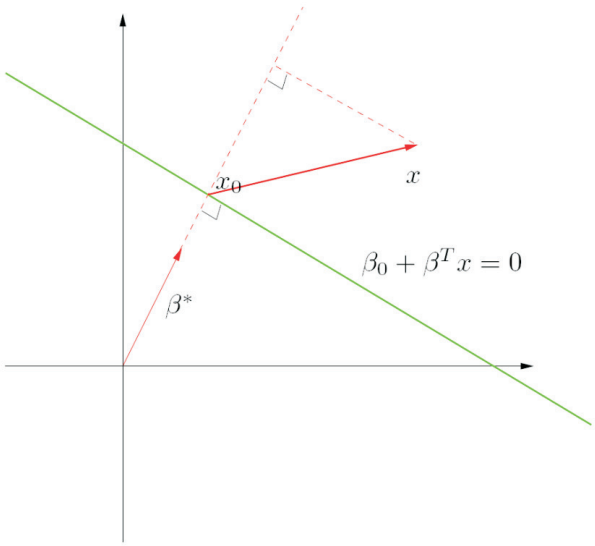

Figure 10.1: Hyperplane $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} = 0$

10.1.1 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 10.2. Depending on the starting value, different solutions will be obtained. "'Support vector machines"' or "'optimally separating hyperplanes"' are better suitable in this case [see **?**].

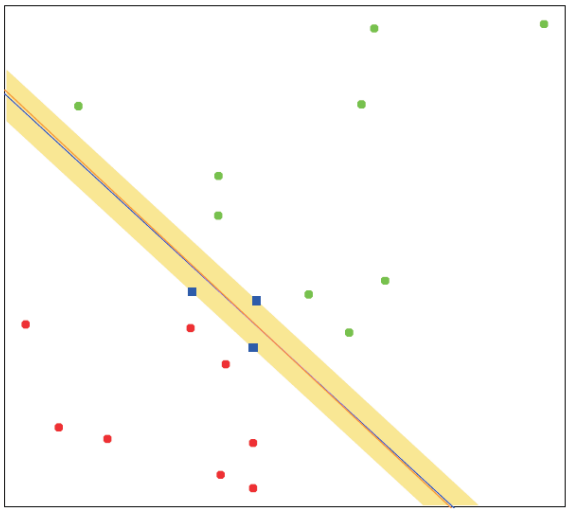

Figure 10.2: Perfect separation of two classes by a hyperplane.

10.2 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

$$\{\boldsymbol{x} \in \Omega : \underbrace{\boldsymbol{x}^\top \boldsymbol{\beta} + \beta_0}_{=: f(\boldsymbol{x})} = 0\} \tag{10.1}$$

with the corresponding classification rule

$$G(\boldsymbol{x}) = \operatorname{sgn}\left[f(\boldsymbol{x})\right] \tag{10.2}$$

(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 $g_i f(\boldsymbol{x}_i) > 0$.

10.2.1 The separable case

In the separable case we can always find a function $f : \mathbb{R}^p \to \mathbb{R}$ defined in (10.1) 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 10.3.

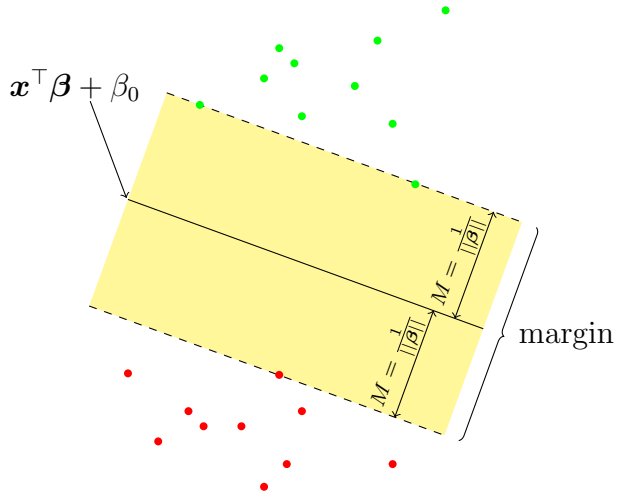

Figure 10.3: The separable case.

The yellow band on both sides of the hyperplane in Figure 10.3, 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

$$\begin{aligned} &\max_{\boldsymbol{\beta},\beta_0} M \\ &||\boldsymbol{\beta}||=1 \\ \text{s.t.} \quad &g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta} + \beta_0) \geq M \quad i = 1, \ldots, n. \end{aligned} \tag{10.3}$$

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 (10.3) 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 (10.3):

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} ||\boldsymbol{\beta}|| \\ \text{s.t.} \quad &g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta} + \beta_0) \geq 1 \quad i = 1, \ldots, n \end{aligned} \tag{10.4}$$

Problem (10.4) is a convex optimisation problem with a quadratic criterion and linear inequality constraints, and the Lagrange method can be used to solve this problem.

For that purpose, one has to minimize the *Lagrange primal function* defined as

$$L_p = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{n} \alpha_i [g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - 1] \tag{10.5}$$

with respect to $\boldsymbol{\beta}$ and β_0. The *Lagrange multipliers* α_i have to be nonnegative, $\alpha_i \geq 0$.

This leads to the *Lagrange dual function*

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \boldsymbol{x}_i^\top \boldsymbol{x}_j, \tag{10.6}$$

which gives a lower bound on the objective function (10.4). The solution of problem (10.4) is then obtained by solving the following simpler convex optimisation problem:

$$\begin{aligned} &\max_{\alpha_i} \left[\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \right] \\ &\text{s.t.} \quad \alpha_i \geq 0 \text{ for } i = 1, \ldots, n \end{aligned} \tag{10.7}$$

In order to be optimal, the solution of problem (10.7) also has to satisfy the so-called *Karush-Kuhn-Tucker conditions*

$$\sum_{i=1}^{n} \alpha_i g_i \boldsymbol{x}_i = \boldsymbol{\beta} \tag{10.8}$$

$$\sum_{i=1}^{n} \alpha_i g_i = 0 \tag{10.9}$$

$$\alpha_i \left[g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) - 1 \right] = 0 \quad i = 1, \ldots, n. \tag{10.10}$$

From the side condition of problem (10.7) and condition (10.10) 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 (10.8) we can see that the solution vector $\boldsymbol{\beta}$ from problem (10.4) is a linear combination of the support vectors: if α_i is equal to zero, the i^{th} summand in (10.8) 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 (10.10) for any of the support vectors. The separating hyperplane in Figure 10.3 has three support vectors.

10.2.2 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 10.4.

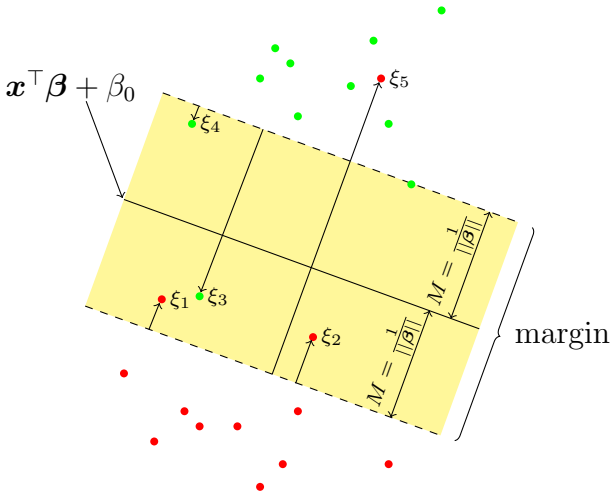

Figure 10.4: The non-separable case.

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 $\sum_{i=1}^n \xi_i \leq C$ means the total amount of misclassified training observations has to be smaller than the constant C. Then (10.4) can be written as

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} ||\boldsymbol{\beta}|| \\ \text{s.t.} \quad &\begin{cases} g_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i \text{ for } i = 1, \ldots, n \\ \xi_i \geq 0 \text{ for } i = 1, \ldots, n \\ \sum_{i=1}^n \xi_i \leq const. \end{cases} \end{aligned} \tag{10.11}$$

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 (10.11) 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 (10.11) can then be written as

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} \left[\frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{n}\xi_i\right] \\ \text{s.t.} \quad &\begin{cases} g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta}+\beta_0) \geq 1-\xi_i \text{ for } i=1,\ldots,n \\ \xi_i \geq 0 \text{ for } i=1,\ldots,n. \end{cases} \end{aligned} \tag{10.12}$$

The corresponding *Lagrange primal function*, which has to be minimised with respect to $\boldsymbol{\beta}, \beta_0$ and ξ_i, is

$$L_p = \frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta}+\beta_0) - (1-\xi_i)] - \sum_{i=1}^{n}\lambda_i\xi_i, \tag{10.13}$$

where α_i, λ_i and ξ_i have to be nonnegative. We obtain the *Lagrange dual function* as

$$L_d = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j g_i g_j \boldsymbol{x}_i^\top\boldsymbol{x}_j. \tag{10.14}$$

The solution of problem (10.11) is then obtained by solving the following simpler convex optimisation problem:

$$\begin{aligned} &\max_{\alpha_i} \left[\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j g_i g_j \boldsymbol{x}_i^\top\boldsymbol{x}_j\right] \\ \text{s.t.} \quad &0 \leq \alpha_i \leq C \text{ for } i=1,\ldots,n \end{aligned} \tag{10.15}$$

In order to be optimal, the solution of problem (10.15) also has to satisfy the *Karush-Kuhn-Tucker conditions*

$$\sum_{i=1}^{n}\alpha_i g_i \boldsymbol{x}_i = \boldsymbol{\beta} \tag{10.16}$$

$$\sum_{i=1}^{n}\alpha_i g_i = 0 \tag{10.17}$$

$$C = \alpha_i + \lambda_i \tag{10.18}$$

$$\lambda_i\xi_i = 0 \tag{10.19}$$

$$\alpha_i\left[g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta}+\beta_0) - (1-\xi_i)\right] = 0 \tag{10.20}$$

$$g_i(\boldsymbol{x}_i^\top\boldsymbol{\beta}+\beta_0) - (1-\xi_i) \geq 0. \tag{10.21}$$

Conditions (10.18)–(10.21) have to hold for $i = 1,\ldots,n$. As in the separable case we can see from constraint (10.16) 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 (10.20), 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 (10.18) and (10.19) 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 (10.19) 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*.

10.3 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 $h(\boldsymbol{x_i}) = (h_1(\boldsymbol{x_i}), \ldots, h_M(\boldsymbol{x}_i))$ instead of $\boldsymbol{x}_i$ for $i = 1, \ldots, n$. We re-define the linear function $f(\cdot)$ in (10.1) to

$$f(\boldsymbol{x}) := h(\boldsymbol{x})^\top \boldsymbol{\beta} + \beta_0, \tag{10.22}$$

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 $h(\boldsymbol{x})$. This fact causes that the fitting is computed as before, although we actually work with a larger feature space. The function (10.22) is nonlinear in $\boldsymbol{x}$ which results in a nonlinear classifier defined by

$$\widehat{G}(\boldsymbol{x}) = \operatorname{sgn}\left[\hat{f}(\boldsymbol{x})\right].$$

with $\hat{f}(\boldsymbol{x}) = h(\boldsymbol{x})^\top \hat{\boldsymbol{\beta}} + \hat{\beta}_0$ 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 (10.13) 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 $h(\boldsymbol{x}_i)$ instead of $\boldsymbol{x}_i$. Using the notation $\langle \cdot, \cdot \rangle$ for the inner product, the Langrangian dual function (10.14) can be written as

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j g_i g_j \langle h(\boldsymbol{x}_i), h(\boldsymbol{x}_j) \rangle \tag{10.23}$$

and using constraint (10.16) the solution function $f(\boldsymbol{x})$ can be written as

$$\begin{aligned} f(\boldsymbol{x}) &= h(\boldsymbol{x})^\top \boldsymbol{\beta} + \beta_0 \\ &= h(\boldsymbol{x})^\top \left(\sum_{i=1}^{n} \alpha_i g_i h(\boldsymbol{x}_i) \right) + \beta_0 \\ &= \sum_{i=1}^{n} \alpha_i g_i \langle h(\boldsymbol{x}), h(\boldsymbol{x}_i) \rangle + \beta_0. \end{aligned} \tag{10.24}$$

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, (10.23) and (10.24) involve $h(\boldsymbol{x})$ only through inner products and therefore we do not need to specify the transformation $h(\cdot)$. 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

$$K(\boldsymbol{u}, \boldsymbol{v}) = \langle h(\boldsymbol{u}), h(\boldsymbol{v}) \rangle.$$

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):
$$K(\boldsymbol{u}, \boldsymbol{v}) = \tanh(\gamma \langle \boldsymbol{u}, \boldsymbol{v} \rangle + c_0) \text{ for a constant } c_0 \text{ and } \gamma > 0$$

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 penalises 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 penalised as heavily.

Chapter 11

Support Vector Machines in R

11.1 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 11.1. 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 11.2 (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
```

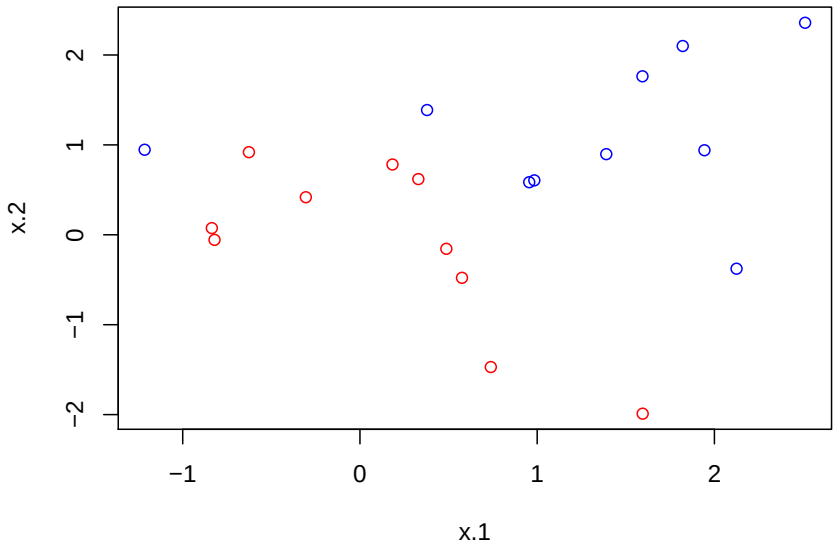

Figure 11.1: Artificial data set with 2 groups.

```
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 11.2 (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:
```

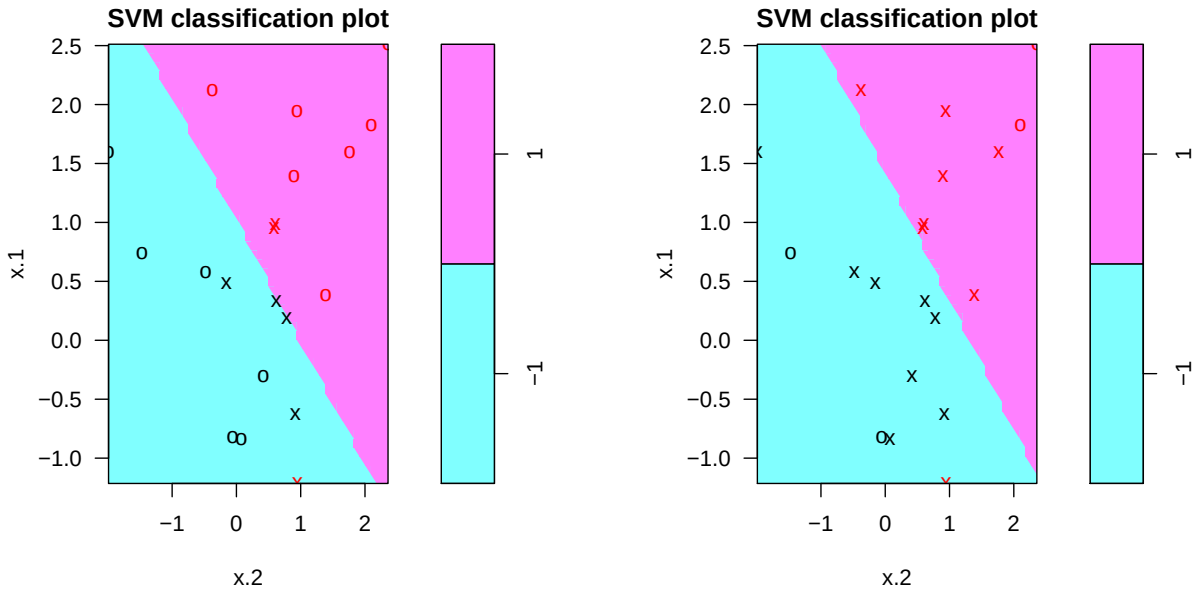

Figure 11.2: Solution of linear SVM classification for the artificial data set: left with a cost parameter of 10, right with parameter 0.1.

```
   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 11.3.

```
> 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.

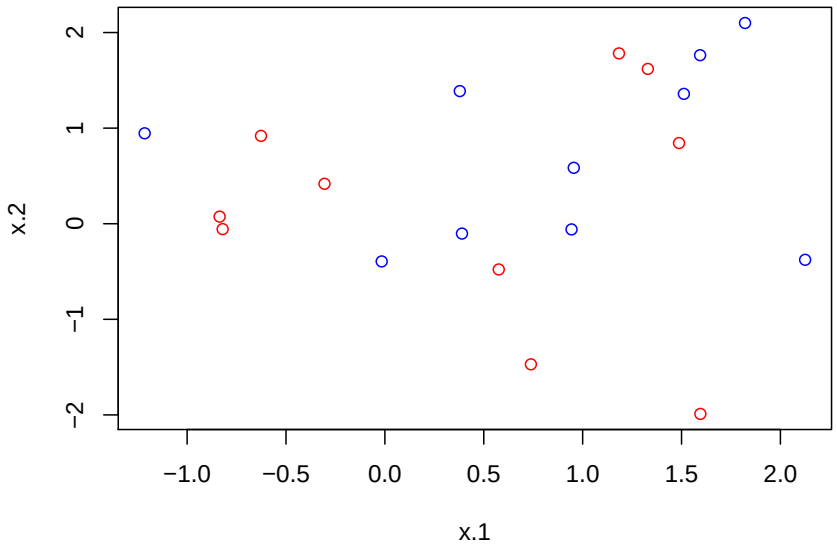

Figure 11.3: Artificial test data set with 2 groups.

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 11.4.

```
> 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")
```

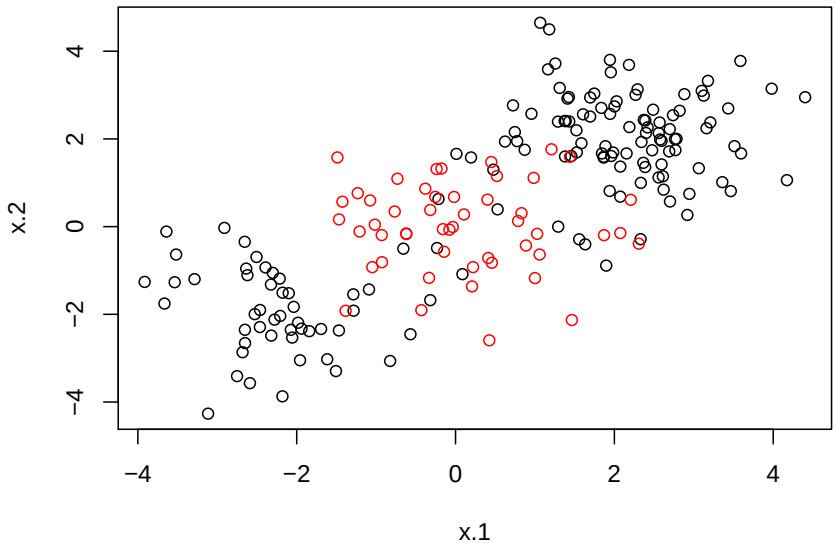

Figure 11.4: Artificial test data set with 2 groups and more difficult structure.

We use the nonlinear radial basis kernel, with default parameters for gamma and cost. This gives the separation boundary shown in Figure 11.5 (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,])
```

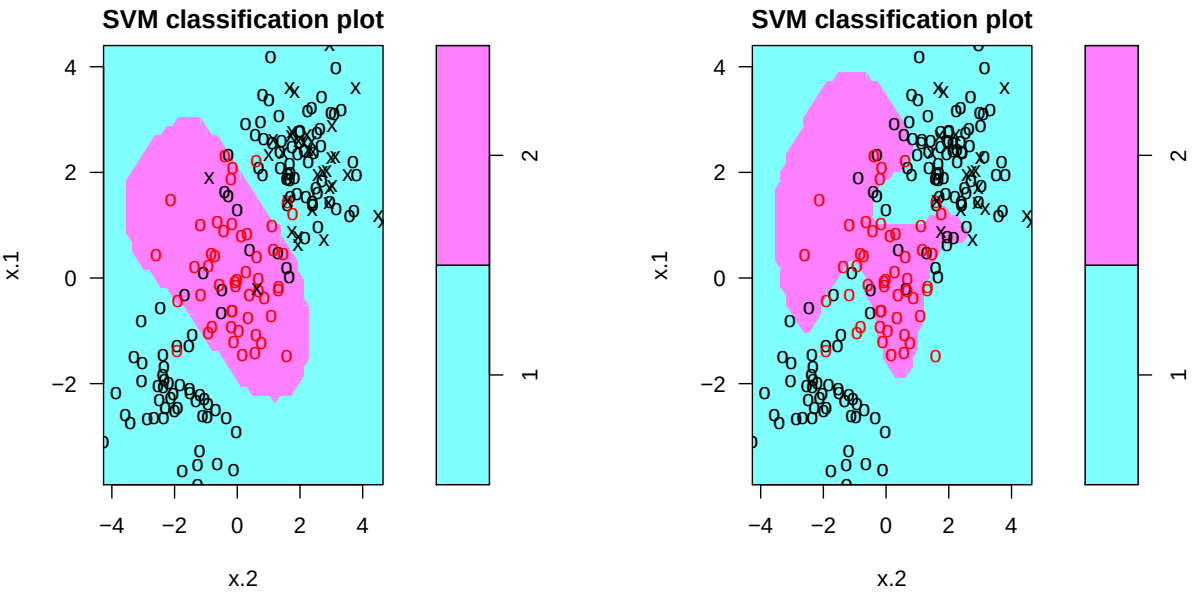

Figure 11.5: Solution of linear SVM classification for the artificial data set: left with a cost parameter of 1, right with parameter 1e5.

Figure 11.5 (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 11.6 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
```

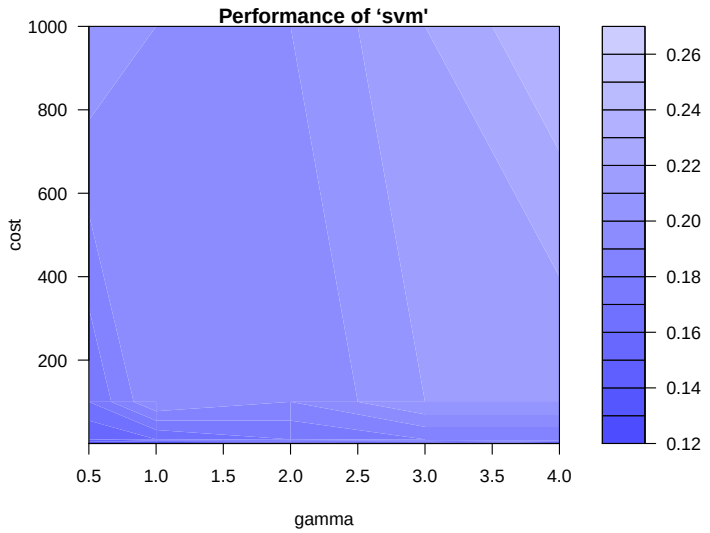

Figure 11.6: Parameter tuning for the gamma and the cost parameter.

11.2 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
```

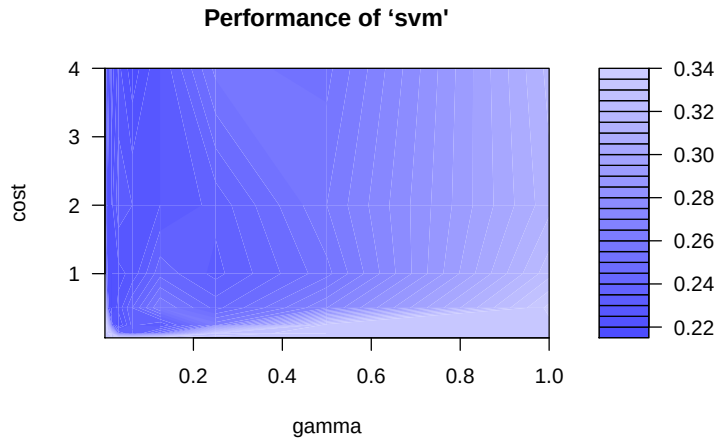

Figure 11.7: Optimimization of the parameters `gamma` and `cost`

```
 0.001953125    4

- best performance: 0.2166667

> plot(tuneSVM)
```

Figure 11.7 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
```

11.3 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,]
> 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.

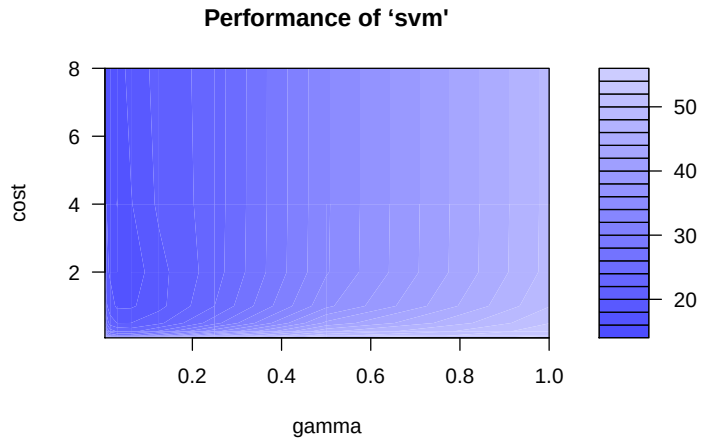

Figure 11.8: Optimimization of the parameters `gamma` and `cost`

```
> 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 11.8 corresponds to the MSE, and not to the RMSE. Using the optimized parameter from Figure 11.8, 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
```

A visual impression of measured versus predicted values is presented in Figure 11.9.

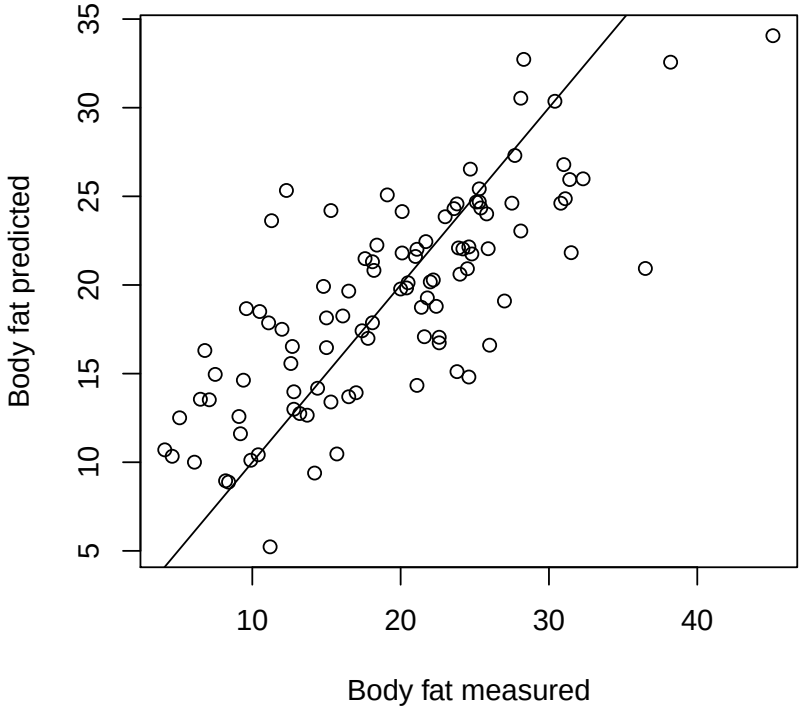

Figure 11.9: Measured versus predicted (SVM) values of *bodyfat* for the test data.