Advanced Regression and Classification

Lecture Notes

Prof. Dr. Peter Filzmoser

P.Filzmoser@tuwien.ac.at

Institute of Statistics and Probability Theory

Vienna University of Technology, Austria

Vienna, February 2008

Preface

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

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

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

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

This manuscript would not exist in this form without the help of two very engaged (former) students, Wolfgang Huf and Tanja Lumplecker. Both invested a lot of time, and I am very grateful for their careful work.

Contents

Preface

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

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

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

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

This manuscript would not exist in this form without the help of two very engaged (former) students, Wolfgang Huf and Tanja Lumplecker. Both invested a lot of time, and I am very grateful for their careful work.

Contents

Part I

Fundamentals

Chapter 1

The classical linear regression model

The aim of the classical 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 classical 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.

 The R-logo refers to demonstrations and examples in R and gives the section and page where they can be found. Examples to the classical linear model are provided in 4.1.1 on page 30.

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.

Chapter 2

Comparison of models and model selection

Let us consider a statistical model which combines several input variables $x_1, x_2, \ldots, x_p$ linearly in order to explain an output variable y as good as possible. One question often asked is what "as good as possible" really means and if there would be another model giving an explanation of y as good as the one just found. A lot of times the differences of the quadratic error of the observed values y_i and the estimated values $\hat{y}_i$ are used to measure the performance of a model.Those differences $y_i - \hat{y}_i$ are called *residuals*.

There are different strategies to compare several models: One could be interested in reducing the number of input variables used to explain the output variable since this could simplify the model, making it easier to understand. In addition, the measurements of variables is often expensive, a smaller model would therefore be cheaper. Another strategy is to use all of the input variables and derive a small number of new input variables from them (see Chapter 3). Both strategies are aimed at describing the output variable as good as possible, not just for already measured and available values but also for values acquired in the future.

2.1 Test for several coefficients to be zero

Let us assume that we have 2 models of different size

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

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

$$\mathrm{RSS}_0 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p_0} \beta_j x_{ij} \right)^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

for model M_0 and respectively

$$\mathrm{RSS}_1 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p_1} \beta_j x_{ij} \right)^2$$

for model M_1. This leads to the following test statistic

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

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

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

For a better understanding of this test statistic we use the fact that the Total Sum of Squares (TSS) can be expressed by the sum of two variations: the sum of squares explained by the regression (Regression Sum of Squares - RegSS) and the Residual Sum of Squares - RSS:

- Total Sum of Squares $\mathrm{TSS} = \sum_{i=1}^{n} (y_i - \overline{y})^2$
- Residual Sum of Squares $\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
- Regression Sum of Squares $\mathrm{RegSS} = \sum_{i=1}^{n} (\hat{y}_i - \overline{y})^2$
- $\mathrm{TSS} = \mathrm{RegSS} + \mathrm{RSS}$ (proof: see Section 3.1.5)

For the models M_0 and M_1 we have

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

which leads to

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

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

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

Section 4.1.2, page 31

2.2 Explained variance

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

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

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

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

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

Section 4.1.2, page 31

2.3 Information criteria

We now want to approach the analysis of nested models $M_1, \ldots, M_q$. The models are ranked with $M_1 \prec M_2 \prec \cdots \prec M_q$, therefore if $M_i \prec M_j$, the model M_i can be seen as a special case of M_j which can be obtained by setting some parameters equal to zero, i.e.

$$\begin{aligned} M_1 &: y = \beta_0 + \beta_1 x_1 + \varepsilon \\ M_2 &: y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon \end{aligned}$$

Of course we have $M_1 < M_2$, since M_1 is a special case of M_2 where $\beta_2 = 0$.

A possibility to evaluate nested models are information criteria. The most popular criteria are AIC and BIC, which try in different ways to combine the model complexity with a penalty term for the number of parameters used in the model.

2.3.1 Akaike's information criterion (AIC)

The criterion of Akaike is based on the Kullback-Leibler (K-L) information $I(f, g)$ [compare Bozdogan, 2000]. The latter describes the loss of information that occurs by approximating a true probability distribution $f(x)$ by a probability distribution $g(x \mid \theta)$. It is defined by

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

Alternatively, the K-L information can also be interpreted as a distance between full reality and a model. We have that

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

Clearly, the best model looses the least information relative to other models in the set, which is equivalent to minimizing $I(f,g)$ over g. Since neither f nor θ are known, the K-L information can not be used in the original version, hence we minimize the expected rather than the original K-L information. (2.2) can then be expressed as

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

Since f is a fixed function, $\mathbb{E}_f[\log(f(x))]$ can be seen as a constant C. This leaves

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

One only has to estimate $\mathbb{E}_f[\log(g(x\,|\,\theta))]$ as a performance index for a model. Akaike [compare Burnham and Anderson, 2004] showed that this is equivalent to the estimation of

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

x and y can be conceptualized as independent random samples from the same distribution and $\hat{\theta}$ stands for the maximized likelihood estimation of θ based on a model g and data y. Akaike found a formal relationship between K-L information and likelihood theory. He found that the maximized log-likelihood $\max\log(L(\hat{\theta}|y))$ is a biased estimate of $\mathbb{E}_y\mathbb{E}_x[\log(g(x|\hat{\theta}(y)))]$, where this bias is approximately equal to p, the number of parameters estimated in the approximating model. Thus, $\max\log(L(\hat{\theta}|y)) - p$ provides an approximately unbiased estimator of $\mathbb{E}_y\mathbb{E}_x[\log(g(x|\hat{\theta}(y)))]$.

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

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

with $\hat{g} = g(\cdot|\hat{\theta})$. Thus, an estimate of the expected relative K-L information which measures the difference to $\mathbb{E}_f[\log(g(x\,|\,\theta))]$ is

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

After multiplying with -2 (for historical reasons), this became the *"Akaike Information Criterion" (AIC)*

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

Out of all estimated nested models we now choose the one with the smallest value of AIC.

AIC in linear models with normal distribution

The density of the univariate normal distribution with expectation μ and variance σ^2 is defined as

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Hence the likelihood function of a sample $y_i = f(x_i) + \varepsilon_i$ for $i = 1, \ldots, n$ and $\varepsilon_i \sim N(0, \sigma^2)$, independent, is

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

and the log-likelihood function is

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

Therefore, a maximization of the log-likelihood equals a minimization of the residual sum of squares (RSS), and we have (see Section 3.1.1)

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{ML} &= \hat{\beta}_{LS} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top y \\ \hat{\sigma}^2_{ML} &= \mathrm{RSS}/n \end{aligned}$$

It follows that

$$\max \log L(\hat{\boldsymbol{\beta}}_{ML}, \hat{\sigma}_{ML}) = -\frac{n}{2}(1 + \log(2\pi)) - \frac{n}{2}\log\left(\frac{\mathrm{RSS}}{n}\right)$$

After dropping the constant the AIC becomes

$$\begin{aligned} \mathrm{AIC} &= n\log(\mathrm{RSS}/n) + 2p \\ &= \frac{\mathrm{RSS}}{\sigma^2} + 2p \quad \text{if } \sigma^2 \text{ is known} \end{aligned}$$

Of course this relationship also holds in the case of several input variables $x_1, \ldots, x_p$. Within nested models the (log-)likelihood is monotonically increasing and the RSS monotonically decreasing.


Section 4.2.5, page 36

2.3.2 Bayes information criterion (BIC)

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

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

2.3.3 Mallow's Cp

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

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

If the residual variance σ^2 is not known, it can be estimated by regression with the full model (using all variables). If a full model cannot be computed (too many variables, collinearity, etc.), a regression can be performed on the relevant principal components (see Section 3.3.1), and the variance is estimated from the resulting residuals. For the "true" model, Cp is approximately p, the number of parameters used in this model, and otherwise greater than p. Thus a model where Cp is approximately p should be selected, and preferably that model with smallest p.

2.3.4 Use of the different criteria

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

- AIC ... for descriptive models
- BIC ... for predictive models
- The usage of both models is in principle only allowed in nested models. However, in practice they are also used for not nested models. Absolute values can not be interpreted.
- Mallow's C_p is mainly used for stepwise regression (adding or removing one variable at a time).

Other possibilities for the evaluation of linear models are the F-test or ANOVA. Those methods decide whether the difference in the log-likelihood between two models is significant (modulo the different number of parameters).

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

Possibilities for the optimal choice of regressor variables are:

- *Complete testing* of all possible subsets
  - The computational effort increases with increasing p exponentially and gets infeasible with large p (2^p possible models).
  - This approach is especially time consuming if mixed effects are considered as well.
- *Stepwise algorithms* start with a minimal (respectively maximal) model and add (respectively delete) variables with each step; this approach might only lead to a local optimum.
- The global optimum can be found with *efficient* algorithms (like the "leaps and bound" algorithms). The common idea is the exclusion of whole branches in the graph of all models.

2.4 Resampling methods

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

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

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

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

With a concrete sample, Err can be estimated using

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

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

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

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

2.4.1 Cross validation

A given sample is randomly divided into q parts. One part is chosen to be the test set, the other parts are defined as training data. The idea is as follows: for the kth part, $k = 1, 2, \dots, q$, 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	...	q
Training	Training	*Test*	Training	Training	...	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 $\hat{y}_i$, $i = 1, \dots, n$.

The estimated cross validation error is then

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

Choice of q

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

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

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

2.4.2 Bootstrap

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

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

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

Problem and possible improvements

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

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

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

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

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

Part II

Linear regression

Chapter 3

Linear methods

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

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

3.1 Least Squares (LS) regression

3.1.1 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. Differentiating with respect to $\boldsymbol{\beta}$ yields

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

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

$$\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = \boldsymbol{0} \tag{3.1}$$

and then obtain the normal equations

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

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

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

Now the estimated regression parameters $\hat{\boldsymbol{\beta}}$, that provide the best fit in the 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 $\boldsymbol{y}$.

A geometric interpretation of the least squares estimate is the projection of $\boldsymbol{y}$ on the column space of $\boldsymbol{X}$. We minimize $\text{RSS}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$ by choosing $\hat{\boldsymbol{\beta}}$ in order for the residual vector $\hat{\boldsymbol{\varepsilon}} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ to be orthogonal to this subspace. This is also expressed by (3.1). The resulting estimate $\hat{\boldsymbol{y}}$ is hence the orthogonal projection of $\boldsymbol{y}$ onto this subspace. The hat matrix computes the orthogonal projection and hence is also known as a projection matrix.

It might happen that the columns of $\boldsymbol{X}$ are not linearly independent (i.e. if two input variables perfectly correlate), so that $\boldsymbol{X}$ is not of full rank. Then $\boldsymbol{X}^\top \boldsymbol{X}$ is singular and the least squares coefficients $\hat{\boldsymbol{\beta}}$ are not uniquely defined. However, the fitted values $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$ are still the projection of $\boldsymbol{y}$ on the column space of $\boldsymbol{X}$, there is just more than one solution. This case appears most often when one or more qualitative inputs are coded in a redundant fashion (recoding mostly eliminates the correlation) or in signal and image analysis, where the number of inputs p can exceed the number of training cases n (the features are typically reduced by filtering or regularization).

Section 4.1.1, page 30

3.1.2 Characteristics of the residuals

The estimated residuals are $\hat{\varepsilon}_i = y_i - \hat{y}_i$ with $i = 1, 2, \ldots, n$. We have:

1. the estimated residuals are centered.
2. the estimated residuals are orthogonal to $\hat{\boldsymbol{y}}$

Proof:

1. Assume that the residuals calculated by minimizing the RSS are not centered, thus

$$\bar{\hat{\varepsilon}} = \frac{1}{n}\sum_{i=1}^{n}\hat{\varepsilon}_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) = a > 0,$$

then we have

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)-a &= 0\\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i-a)\\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i-(\hat{y}_i+a)).\end{aligned}$$

By inserting those "new residuals" in the least squares criterion we get

$$\begin{aligned}\sum_{i=1}^{n}(y_i-(\hat{y}_i+a))^2 &= \sum_{i=1}^{n}((y_i-\hat{y}_i)-a)^2\\ &= \sum_{i=1}^{n}((y_i-\hat{y}_i)^2-2(y_i-\hat{y}_i)a+a^2)\\ &= \sum_{i=1}^{n}(y_i-\hat{y}_i)^2-2\underbrace{\sum_{i=1}^{n}(y_i-\hat{y}_i)\,a}_{na}+na^2\\ &= \sum_{i=1}^{n}(y_i-\hat{y}_i)^2-na^2<\sum_{i=1}^{n}(y_i-\hat{y}_i)^2\end{aligned}$$

which is a contradiction to our original assumption of an already minimized RSS. Generally we have

$$\sum_{i=1}^{n}\hat{y}_i=\sum_{i=1}^{n}y_i=n\overline{y}$$

2.

$$\hat{\boldsymbol{y}}^\top\hat{\boldsymbol{\varepsilon}}=\hat{\boldsymbol{y}}^\top(\boldsymbol{y}-\hat{\boldsymbol{y}})=\hat{\boldsymbol{\beta}}^\top(\boldsymbol{X}^\top\boldsymbol{y}-\boldsymbol{X}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y})=\hat{\boldsymbol{\beta}}^\top(\boldsymbol{X}^\top\boldsymbol{y}-\boldsymbol{X}^\top\boldsymbol{y})=0$$

□

3.1.3 Characteristics of the parameters

Up to now, no assumptions have been made about the true distribution of the data. In order to determine the sampling properties of $\hat{\boldsymbol{\beta}}$ (the parameters are taken to be stochastic variables), we now assume that

- the stochastic observations y_i are uncorrelated,
- have a constant variance σ^2 and
- the $\boldsymbol{x}_i$ are fixed (non random).

The variance-covariance matrix of the least squares parameter estimates is given by

$$\begin{aligned}\mathrm{Cov}(\hat{\boldsymbol{\beta}}) &= \mathrm{Cov}[(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}]\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\underbrace{\mathrm{Cov}(\boldsymbol{y})}_{\sigma^2\boldsymbol{I}_n}\left[(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\right]^\top\\ &= \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\\ &= \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\end{aligned}$$

Typically one estimates the variance σ^2 by

$$\begin{aligned}\hat{\sigma}^2 &= \frac{1}{n-p-1}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2 \\ &= \frac{1}{n-p-1}(\boldsymbol{y}-\boldsymbol{X}\hat{\boldsymbol{\beta}})^\top(\boldsymbol{y}-\boldsymbol{X}\hat{\boldsymbol{\beta}})\end{aligned}$$

The $n-p-1$ rather than n in the denominator makes $\hat{\sigma}^2$ an unbiased estimate of σ^2; so $\mathbb{E}(\hat{\sigma}^2)=\sigma^2$.

To draw inferences about the parameters and the model, additional assumptions have to be made:

- (1.1) needs to be a correct model for the mean, the conditional expectation of y is linear in $x_1, x_2, \ldots, x_p$ or in other words: $\mathbb{E}(\boldsymbol{\varepsilon})=\mathbf{0}$.
- Additionally we assume that the deviations of y around its expectation are additive and normally distributed.

Hence

$$\begin{aligned}y_i &= \mathbb{E}(y_i|\boldsymbol{x}_i)+\varepsilon_i \\ &= \beta_0+\sum_{j=1}^{p}x_{ij}\beta_j+\varepsilon_i,\end{aligned}$$

where the errors $\varepsilon_1,\ldots,\varepsilon_n$ are independent normally distributed random variables with expectation zero and variance σ^2,

$$\boldsymbol{\varepsilon}=(\varepsilon_1,\ldots,\varepsilon_n)^\top \sim N_n(\mathbf{0},\sigma^2 I_n)$$

With these assumptions we get

$$\hat{\boldsymbol{\beta}} \sim N(\boldsymbol{\beta}, \underbrace{(\sigma^2\boldsymbol{X}^\top\boldsymbol{X})^{-1}}_{\mathrm{Cov}(\hat{\boldsymbol{\beta}})}) \tag{3.2}$$

Proof of Equation (3.2):

$$\begin{aligned}\mathbb{E}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}[(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}] \\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathbb{E}(\boldsymbol{y}) \\ &= \mathbb{E}[(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta}]+(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathbb{E}(\boldsymbol{\varepsilon}) \\ &= \boldsymbol{\beta} \\ Cov(\hat{\boldsymbol{\beta}}) &= \ldots=\sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\end{aligned}$$

□

Theorem 3.1.3.1 (Gauss-Markov Theorem) *Under the model assumption*

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

and the condition that the input variables are fixed, the least squares estimate $\hat{\boldsymbol{\beta}}$ is the best, linear, unbiased estimator (BLUE).

Proof: see, e.g., Schönfeld 1969

3.1.4 Tests and confidence intervals

Besides the already known property $\hat{\boldsymbol{\beta}} \sim N_{p+1}(\boldsymbol{\beta}, \sigma^2 \boldsymbol{X}^\top \boldsymbol{X})^{-1})$ we have furthermore:

- $(n-p-1)\hat{\sigma}^2 \sim \sigma^2 \chi^2_{n-p-1}$
- $\hat{\boldsymbol{\beta}}$ and $\hat{\sigma}^2$ are statistically independent.

Proof see e.g. Schönfeld 1969

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

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

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

For comparison

$$x_i \overset{i.i.d.}{\sim} N(\mu, \sigma^2) \quad \text{with } i = 1, 2, \ldots, n \quad \Longrightarrow \quad \overline{x} \sim N\left(\mu, \frac{\sigma^2}{n}\right)$$

$$H_0 : \mu = \mu_0 = 0$$

$$H_1 : \mu \neq \mu_0$$

With σ unknown

$$T = \frac{\overline{x} - \mu_0}{s/\sqrt{n}} \sim t_{n-1}$$

With σ known

$$T = \frac{\overline{x} - \mu_0}{\sigma/\sqrt{n}} \sim N(0,1)$$

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

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

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

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

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

An approximative 95%-confidence interval is:

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

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

Section 4.1.2, page 31

3.1.5 Decomposition of the variance of y

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

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

Proof : With

$$y_i - \overline{y} = (y_i - \hat{y}_i) + (\hat{y}_i - \overline{y})$$

, the total variation (TSS) of $\boldsymbol{y}$ can be decomposed as follows:

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

where

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

□

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

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

	DF	MeanSS
RegSS	p	RegSS$/p$
RSS	$n-p-1$	RSS$/(n-p-1)$

with

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

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

Section 4.2.1, page 32

3.2 Variable selection

3.2.1 Bias versus variance and interpretability

There are two reasons why a least square estimate of the parameters in the full model is unsatisfying:

1. *prediction quality*: Least square estimates often have a small bias but a high variance. The prediction quality can sometimes be improved by shrinkage of the regression coefficients or by setting some coefficients equal to zero. This way the bias increases, but the variance of the prediction reduces significantly which leads to an overall improved prediction.

   Theorem 3.2.1.1 *Let $\hat{\theta}$ be an estimate for θ, then:*

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

   Proof:

   $$\begin{aligned}
   \mathbb{E}(\hat{\theta}-\theta)^2 &= \mathbb{E}(\hat{\theta}^2 - 2\hat{\theta}\theta + \theta^2) \\
   &= \mathbb{E}(\hat{\theta}^2) - 2\theta\mathbb{E}(\hat{\theta}) + \theta^2 \\
   \text{Var}(\hat{\theta}) + [\mathbb{E}(\hat{\theta})-\theta]^2 &= \mathbb{E}(\hat{\theta}^2) - [\mathbb{E}(\hat{\theta})]^2 + [\mathbb{E}(\hat{\theta})]^2 - 2\theta\mathbb{E}(\hat{\theta}) + \theta^2 \\
   &= \mathbb{E}(\hat{\theta}^2) - 2\theta\mathbb{E}(\hat{\theta}) + \theta^2
   \end{aligned}$$

   □

   For our BLUE $\hat{\boldsymbol{\beta}}$ we have

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

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

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

   could exist, with $\mathbb{E}(\tilde{\boldsymbol{\beta}}) \neq \boldsymbol{\beta}$ (biased) and $\text{Var}(\tilde{\boldsymbol{\beta}}) < \text{Var}(\hat{\boldsymbol{\beta}})$. A smaller MSE could thus lead to a better prediction of new values. Such estimators $\tilde{\boldsymbol{\beta}}$ can be obtained from variable selection, by building linear combinations of the regressor variables (Section 3.3), or by shrinkage methods (Section 3.4).

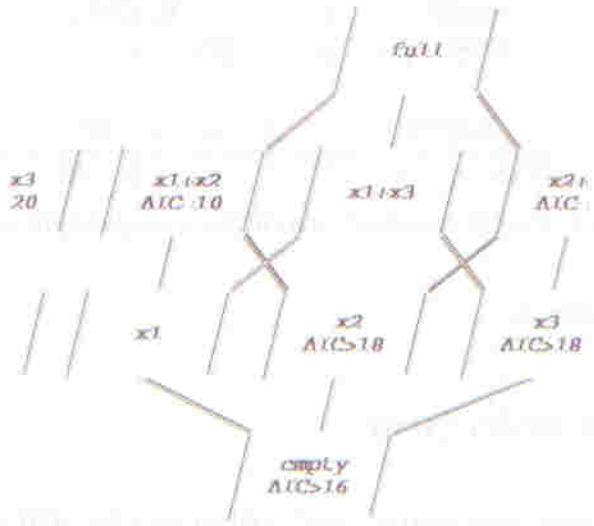


Figure 3.1: Model tree

2. *Interpretability*: In the case of large amounts of predictors it makes sense to identify the ones who have the largest influence and to set the ones to zero that are not relevant for the prediction. Thus we eliminate variables that will only explain some details, but we keep those which allow for the major explanation of the response variable.

With variable selection only a subset of all input variables is used, the rest is eliminated from the model. For this subset, the least squares estimate is used for parameter estimation. There are many different strategies to choose a subset.

3.2.2 Best subset regression

The *Best subset regression* finds the subset of size k for each $k \in 0, 1, \ldots, p$ that gives the smallest RSS. An efficient algorithm is the already mentioned *Leaps and Bounds algorithm* in Chapter 2 who is feasible for $p \leq 40$.

Leaps and Bounds algorithm: This algorithm creates a tree model and calculates the RSS (or other criteria) for the particular subsets. In Figure 3.1 the AIC for the first subsets is shown. Subsequently, large branches are eliminated by trying to reduce the RSS. The AIC for the model $x2 + x3 = 20$. Through the elimination of $x2$ or $x3$ we get an AIC of at least 18, since the value can reduce by $2p = 2$ at most (this follows from the formula for the $AIC := n \log(RSS/n) + 2p$). By eliminating a regressor in $x1 + x2$ a smaller AIC (>8) can be obtained. Therefore, the branch $x2 + x3$ is left out in the future analysis.

Section 4.2.4, page 35

3.2.3 Stepwise algorithms

With data sets larger than 40 input variables a search through all possible subsets becomes infeasible. Following algorithms are used instead:

- *Forward stepwise selection:* starts with the intercept and then sequentially adds into the model the predictor that most improves the fit. The improvement of fit is often based on the F-statistic

$$F = \frac{\text{RSS}(\hat{\beta}) - \text{RSS}(\tilde{\beta})}{\text{RSS}(\tilde{\beta})/(n-k-2)}.$$

  $\hat{\beta}$ is the parameter vector of the current model with k parameters, $\tilde{\beta}$ the parameter vector of the model with $k+1$ parameters. Typically, predictors are added sequentially until no predictor produces an F ratio greater than the $90th$ or $95th$ percentile of the $F_{1,n-k-2}$ distribution.

- *Backward stepwise selection:* starts with the full model and sequentially deletes the predictors. It typically uses the same F ratio like forward stepwise selection. Backward selection can only be used when $n > p$ in order to have a well defined model.

- There are also hybrid stepwise strategies that consider both forward and backward moves at each stage, and make the "best" move.

Section 4.2.5, page 36

3.3 Methods using derived inputs as regressors

In many situations we have a large number of inputs, often highly correlated. For the model

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

with $\boldsymbol{X}$ fixed and $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}, \sigma^2\boldsymbol{I})$ we have: $\hat{\boldsymbol{\beta}}_{\text{LS}} \sim N(\boldsymbol{\beta}, \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1})$.

In the case of correlated regressors, $\boldsymbol{X}^\top\boldsymbol{X}$ is almost singular and $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$ therefore very large which leads to a numerically unstable $\hat{\boldsymbol{\beta}}$. As a consequence, tests like the one in Section 3.1.4 become unreliable:

$$\begin{aligned} H_0 &: \beta_j = 0 \\ H_1 &: \beta_j \neq 0 \end{aligned}$$

with

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

where d_j represents the jth diagonal element of $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$.

Section 4.3.1, page 39

To avoid this problem, we use methods that use derived input directions. The methods in this section produce a small number of linear combinations $\boldsymbol{z}_k, k = 1, 2, \ldots, q$ of the original inputs $\boldsymbol{x}_j$ which are then used as inputs in the regression. The methods differ in how the linear combinations are constructed.

3.3.1 Principal Component Regression (PCR)

This method looks for transformations of the original data into a new set of uncorrelated variables called principal components. This transformation ranks the new variables according to their importance, which means according to the size of their variance [see, e.g., Fekedulegn et al., 2002]), and eliminates those of least importance. Then a least squares regression on the reduced set of principal components is performed.
Since PCR is not scale invariant, one should scale and center the data. Given a p-dimensional random vector $\boldsymbol{x} = (x_1, \ldots, x_p)^\top$ with covariance matrix $\boldsymbol{\Sigma}$. We assume that $\boldsymbol{\Sigma}$ is positive definite. Let $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)$ be a $(p \times p)$-matrix with orthonormal column vectors, i.e. $\boldsymbol{v}_i^\top \boldsymbol{v}_i = 1$, $i = 1, \ldots, p$, and $\boldsymbol{V}^\top = \boldsymbol{V}^{-1}$. We are looking for the linear transformation

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{V}^\top \boldsymbol{x} \\ z_i &= \boldsymbol{v}_i^\top \boldsymbol{x} \end{aligned}$$

The variance of the random variable z_i is given by

$$\mathrm{Var}(z_i) = \mathbb{E}[\boldsymbol{v}_i^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{v}_i] = \boldsymbol{v}_i^\top \boldsymbol{\Sigma} \boldsymbol{v}_i$$

Maximizing the variance $\mathrm{Var}(z_i)$ under the conditions $\boldsymbol{v}_i^\top \boldsymbol{v}_i = 1$ with Lagrange gives

$$\phi_i = \boldsymbol{v}_i^\top \boldsymbol{\Sigma} \boldsymbol{v}_i - a_i(\boldsymbol{v}_i^\top \boldsymbol{v}_i - 1)$$

Setting the partial derivation to zero we get

$$\frac{\partial \phi_i}{\partial \boldsymbol{v_i}} = 2\boldsymbol{\Sigma} \boldsymbol{v}_i - 2a_i \boldsymbol{v}_i = \boldsymbol{0}$$

which is

$$(\boldsymbol{\Sigma} - a_i \boldsymbol{I})\boldsymbol{v}_i = \boldsymbol{0}$$

In matrix form we have

$$\boldsymbol{\Sigma V} = \boldsymbol{VA}$$

or

$$\boldsymbol{\Sigma} = \boldsymbol{VAV}^\top$$

where $\boldsymbol{A} = Diag(a_1, \ldots, a_p)$. This is known as the eigenvalue problem, $\boldsymbol{v}_i$ are the eigenvectors of $\boldsymbol{\Sigma}$ and a_i the corresponding eigenvalues: $a_1 \geq a_2 \ldots \geq a_p$. Since $\boldsymbol{\Sigma}$ is positive definite, all eigenvalues are real, non negative numbers.

z_i is named the ith principal component of $\boldsymbol{x}$, and we have:

$$Cov(\boldsymbol{z}) = \boldsymbol{V}^\top Cov(\boldsymbol{x}) \boldsymbol{V} = \boldsymbol{V}^\top \boldsymbol{\Sigma} \boldsymbol{V} = \boldsymbol{A}$$

The variance of the ith principal component matches the eigenvalue a_i, the variances are ranked in descending order. The last principal component has the smallest variance. The principal components are orthogonal to all the other principal components (they are even uncorrelated) since $\boldsymbol{A}$ is a diagonal matrix.

In the following we will use q $(1 \leq q \leq p)$ principal components for regression. The regression model for observed data $\boldsymbol{X}$ and $\boldsymbol{y}$ can then be expressed as

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

with the $n \times q$ matrix of the empirical principal components $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}$ and the new regression coefficients $\boldsymbol{\theta} = \boldsymbol{V}^\top \boldsymbol{\beta}$. The solution of the least squares estimation is

$$\hat{\theta}_k = (\boldsymbol{z}_k^\top \boldsymbol{z}_k)^{-1} \boldsymbol{z}_k^\top \boldsymbol{y}$$

and

$$\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_q)^\top.$$

Since the $\boldsymbol{z}_k$ are orthogonal, the regression is just a sum of univariate regressions

$$\hat{\boldsymbol{y}}_{\text{PCR}} = \sum_{k=1}^{q} \hat{\theta}_k \boldsymbol{z}_k$$

Since the $\boldsymbol{z}_k$ are linear combinations of the original $\boldsymbol{x}_j$, we can express the solution in terms of coefficients of the $\boldsymbol{x}_j$

$$\hat{\boldsymbol{\beta}}_{\text{PCR}}(q) = \sum_{k=1}^{q} \hat{\theta}_k \boldsymbol{v}_k = \boldsymbol{V}\hat{\boldsymbol{\theta}}$$

Note that if $q = p$, we would just get back the usual least squares estimates for the full model. For $q < p$ we get a "reduced" regression.

Section 4.3.2, page 42

3.3.2 Partial Least Squares (PLS) regression

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

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

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

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

We assume:

- The new coefficients $\boldsymbol{\gamma}$ are of dimension $q \leq p$.
- The values $\boldsymbol{t}_i$ of the variables are put together in a $(n \times q)$ score matrix $\boldsymbol{T}$.
- Due to the reduced dimension, the regression of $\boldsymbol{y}$ on $\boldsymbol{T}$ should be more stable.
- $\boldsymbol{T}$ can not be observed directly; we get each $\boldsymbol{T}$ sequentially for $k = 1, 2, \ldots, q$ through the PLS criteria

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

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

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

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

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

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

Section 4.3.3, page 43

3.3.3 Continuum regression

This approach is a combination of least squares regression, PCR and PLS regression. Just like with PLS we regress $\boldsymbol{y}$ on $\boldsymbol{T} = \boldsymbol{X}\boldsymbol{A}$, and obtain the $\boldsymbol{a}_k$ for $k = 1, 2, \ldots, q \leq p$ through

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

under the condition

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

and

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

δ lies between 0 and 1 and represents the information content of $\boldsymbol{X}$ that is used for the prediction of $\boldsymbol{y}$. Three special cases can be identified:

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

Section 4.3.4, page 44

3.4 Shrinkage methods

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

3.4.1 Ridge regression

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

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

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

An equivalent way to write the ridge problem is

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

under the constraint

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

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

The ridge solutions $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$ are *not* equivariant for different scaling of the inputs, and so one normally standardizes the inputs. In addition, notice that the intercept β_0 has been left out of the penalty term. Penalization of the intercept would make the procedure depend on the origin chosen for $\boldsymbol{y}$; that is adding a constant c to each of the targets y_i would *not* simply result in a shift of the predictions by the same amount c.

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

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

$\boldsymbol{I}$ 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

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

thus

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

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

What does ridge regression really do?

To get some additional insight in the nature of ridge regression we first need to mention Singular Value Decomposition:

Excursion: **Singular Value Decomposition (SVD)**

The singular value decomposition of the centered $(n \times p)$ input matrix $\boldsymbol{X}$ (here $n > p$) has the form

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

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal $(n \times p)$- and $(p \times p)$ matrices. The columns of $\boldsymbol{U}$ span the column space of $\boldsymbol{X}$, the rows of $\boldsymbol{V}$ span the row space of $\boldsymbol{X}$. $\boldsymbol{D}$ is a $(p \times p)$ diagonal matrix, with diagonal entries $d_1 \geq d_2 \geq \cdots \geq d_p \geq 0$ called the singular values of $\boldsymbol{X}$. The columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^\top$ to the eigenvalues d_i^2, and the columns of $\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{X}^\top\boldsymbol{X}$ to the same eigenvalues.

The singular value decomposition of the centered matrix $\boldsymbol{X}$ is just another way to express the principal components of $\boldsymbol{X}$. The estimate of the covariance matrix is

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

and with (3.4) we have

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

This is the eigenvalue decomposition of $\boldsymbol{X}^\top\boldsymbol{X}$ (and of $\boldsymbol{S}$ - up to a factor $\frac{1}{n-1}$, $\boldsymbol{D}^2 = \boldsymbol{A}$ are the corresponding eigenvalues). Just as in PCR we are looking for the principal components of $\boldsymbol{X}$. For the first principal component $\boldsymbol{z}_1$ we get

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

and it has the maximum variance of all normalized linear combinations of the columns of $\boldsymbol{X}$. The variance is seen to be

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

Using singular value decomposition we can write the estimated y-values that have been fitted by least squares as

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

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

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

where the $\boldsymbol{u}_j$ are the columns of $\boldsymbol{U}$. Note that since $\lambda \geq 0$, we have $d_j^2/(d_j^2+\lambda) \leq 1$. Like linear regression, ridge regression computes the coordinates of $\boldsymbol{y}$ with respect to the orthonormal basis $\boldsymbol{U}$. It then shrinks these coordinates by the factor $d_j^2/(d_j^2+\lambda)$. This means that a greater amount of shrinkage is applied to basis vectors with smaller d_j^2.

What does a small value of d_j^2 mean?

d_j^2 is proportional to the variance of the principal components. Ridge regression projects $\boldsymbol{y}$ on the component $\boldsymbol{u}_j$. The smaller d_j^2, the more penalization in that direction is made. We assume implicitly, that $\boldsymbol{y}$ will tend to vary more in the direction of high variance of the inputs. This may not always be true.

PCR is very similar to ridge regression: both methods use the principal components of the input matrix $\boldsymbol{X}$. Ridge regression shrinks the coefficients of the principal components, the shrinkage depends on the corresponding eigenvalues; PCR completely discards the components to the smallest $p-q$ eigenvalues.

Section 4.4.1, page 45

3.4.2 Lasso Regression

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

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

with the constraint

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

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

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

Section 4.4.2, page 47

Chapter 4

Linear methods in R

4.1 Least Squares (LS) regression in R

4.1.1 Parameter estimation

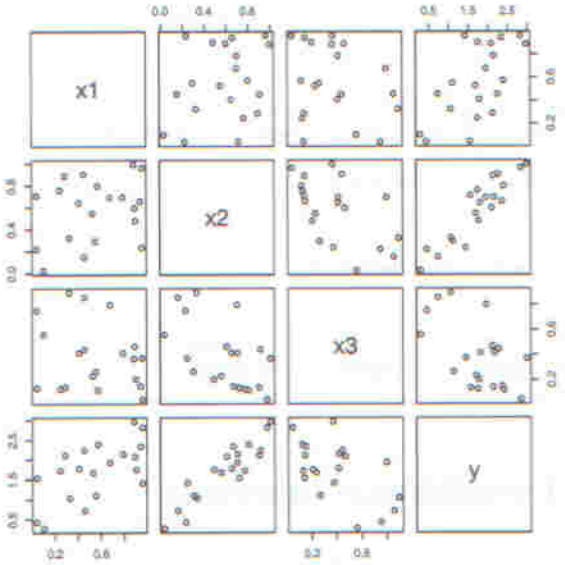


Figure 4.1: Multiple regression with simulated data: regression of y on three x-variables

- *Generation of the data*

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

- *Model using only a constant term*

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

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

Coefficients:
(Intercept)
       1.72
```

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

- *Model with one explanatory variable*

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

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

Coefficients:
(Intercept)           x1
     0.9157       1.4600
```

- *Fit of a full model*

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

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

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

4.1.2 Tests and confidence intervals

- *Testing the coefficients for significance*

```
> summary(lm3)

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

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

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09585    0.08200   1.169    0.260
x1           0.91834    0.06623  13.867 2.47e-10 ***
x2           1.99804    0.08453  23.637 7.18e-14 ***
x3          -0.08761    0.09060  -0.967    0.348
---
Signif. codes:  0 ãŸ***ãŹ 0.001 ãŸ**ãŹ 0.01 ãŸ*ãŹ 0.05 ãŸ.ãŹ 0.1 ãŸ ãŹ 1

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

- The t statistic (see Section 3.1.4) of $x1$ and $x2$ is highly significant and the p-value of each variable is below 0.05. Therefore, both variables have a great impact on the explanation of the regressor and the null hypothesis can be rejected. The regressor $x3$ provides no significant additional contribution.
- The model provides a good fit (R squared, see Section 2.2), 98.82% of the variance of y can be explained by the model. The value 98.6% of the adjusted R squared is very high as well.
- ```
  > qf(0.95, 3, 16)
  [1] 3.238872
  ```

  The value of the "F statistic" (see Section 2.1) of 446.5 is larger than the F quantile $F_{3,16;0.95} = 3.24$, therefore the null hypothesis $\beta_i = 0, \forall\ i = 1, \ldots, p$ can be rejected. This could also be concluded by the p-value that is close to 0.
- The test statistic from above can be used for the calculation of a confidence interval for $\hat{\beta}_j$ (see Section 3.1.4). From the approximation of the 95% confidence interval, we obtain for $\hat{\beta}_1$ the interval

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

  and for $\hat{\beta}_3$

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

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

4.2 Variable selection in R

4.2.1 Model comparison with anova()

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

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

An F-test (see Section 2.1) is computed for every additional explanatory variable, starting with the empty model and following the order of the formula. Regressor x3 does not improve the fit of the model and can be left out.

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

```
Analysis of Variance Table

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

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

4.2.2 Body fat data

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

```
> library("UsingR")
> data(fat)
> attach(fat)
> fat$body.fat[fat$body.fat == 0] <- NA
# exclude observations that are not used for the analysis
> fat <- fat[, -cbind(1, 3, 4, 9)]
# exclude a sample with wrong body height
> fat <- fat[-42, ]
# transform the body height in centimeter
> fat[, 4] <- fat[, 4] * 2.54
```

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)

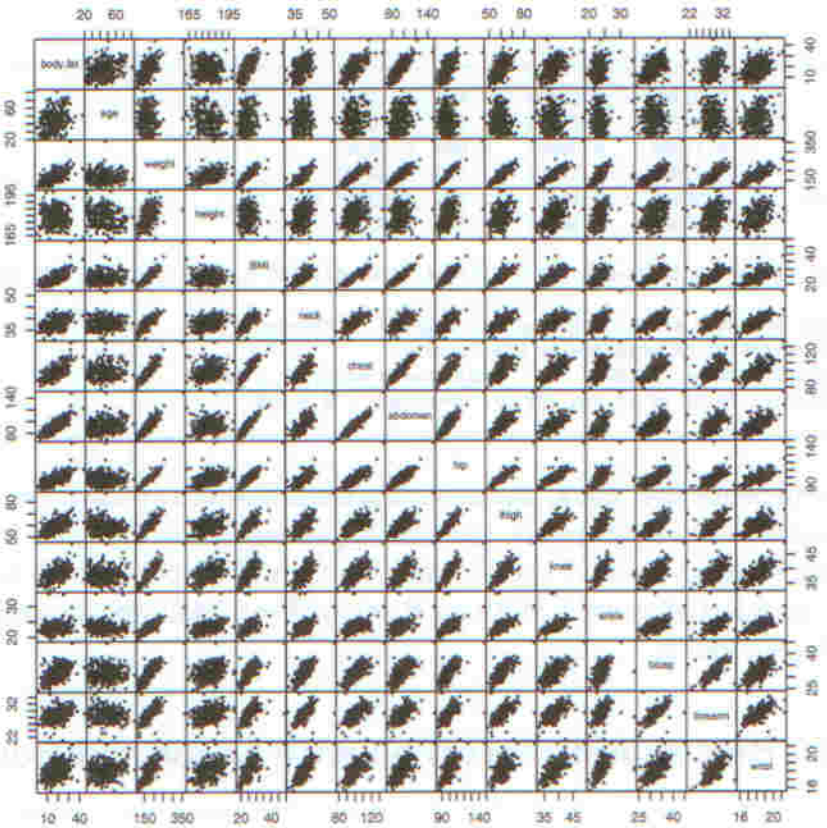


Figure 4.2: Scatterplot matrix of the Body-fat data

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

4.2.3 Full model

```
> model.lm <- lm(body.fat ~ ., data = fat)
> summary(model.lm)

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

Residuals:
     Min       1Q   Median       3Q      Max
-10.1062  -2.6605  -0.2011   2.8920   9.2619

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -44.91075   36.67739  -1.224  0.22200
age           0.05740    0.03004   1.911  0.05725 .
weight       -0.16239    0.10076  -1.612  0.10838
height        0.17192    0.20001   0.860  0.39089
BMI           0.75340    0.73339   1.027  0.30534
neck         -0.42594    0.21857  -1.949  0.05251 .
chest        -0.05969    0.09907  -0.603  0.54740
abdomen       0.87126    0.08569  10.168  < 2e-16 ***
hip          -0.22543    0.13796  -1.634  0.10359
```

```
thigh         0.21780    0.13660   1.594  0.11220
knee         -0.01257    0.22965  -0.055  0.95639
ankle         0.12398    0.20837   0.595  0.55243
bicep         0.16357    0.16000   1.022  0.30769
forearm       0.39166    0.18627   2.103  0.03656 *
wrist        -1.49585    0.49586  -3.017  0.00284 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.988 on 235 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-Squared: 0.7432,     Adjusted R-squared: 0.7279
F-statistic: 48.58 on 14 and 235 DF,  p-value: < 2.2e-16
```

The coefficients `age`, `neck`, `abdomen`, `forearm` and `wrist` have very large t-values and very small p-values, therefore the null hypothesis $\beta_i = 0$ should be rejected. 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 $= 0.7432$ we can assume that the model provides a good fit.

4.2.4 Best subset regression with Leaps and Bound algorithm

```
> library(leaps)
> lm.regsubset <- regsubsets(body.fat ~ ., data = fat, nbest = 1,
+     nvmax = 8)
> summary(lm.regsubset)

Subset selection object
Call: regsubsets.formula(body.fat ~ ., data = fat, nbest = 1, nvmax = 8)
14 Variables  (and intercept)
        Forced in Forced out
age         FALSE      FALSE
weight      FALSE      FALSE
height      FALSE      FALSE
BMI         FALSE      FALSE
neck        FALSE      FALSE
chest       FALSE      FALSE
abdomen     FALSE      FALSE
hip         FALSE      FALSE
thigh       FALSE      FALSE
knee        FALSE      FALSE
ankle       FALSE      FALSE
bicep       FALSE      FALSE
forearm     FALSE      FALSE
wrist       FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         age weight height BMI neck chest abdomen hip thigh knee ankle bicep
1  ( 1 ) " " " "    " "    " " "*"  " "   "*"     " " " "   " "  " "   " "
2  ( 1 ) " " " "    " "    " " "*"  " "   "*"     " " "*"   " "  " "   " "
3  ( 1 ) " " "*"    " "    " " "*"  " "   "*"     " " "*"   " "  " "   " "
4  ( 1 ) " " "*"    " "    " " "*"  " "   "*"     " " "*"   " "  " "   " "
5  ( 1 ) " " "*"    " "    " " "*"  " "   "*"     " " "*"   " "  " "   " "
6  ( 1 ) " " "*"    " "    " " "*"  " "   "*"     " " "*"   " "  " "   "*"
7  ( 1 ) "*" "*"    " "    " " "*"  " "   "*"     " " "*"   " "  " "   " "
8  ( 1 ) "*" "*"    " "    " " "*"  " "   "*"     "*" "*"   " "  " "   " "
         forearm wrist
1  ( 1 ) " "     " "
2  ( 1 ) " "     " "
3  ( 1 ) " "     "*"
4  ( 1 ) "*"     "*"
5  ( 1 ) "*"     "*"
6  ( 1 ) "*"     "*"
7  ( 1 ) "*"     "*"
8  ( 1 ) "*"     "*"
```

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

```
> lm.regsubset2 <- regsubsets(body.fat ~ ., data = fat, nbest = 2,
+     nvmax = 8)
> plot(lm.regsubset2)
```

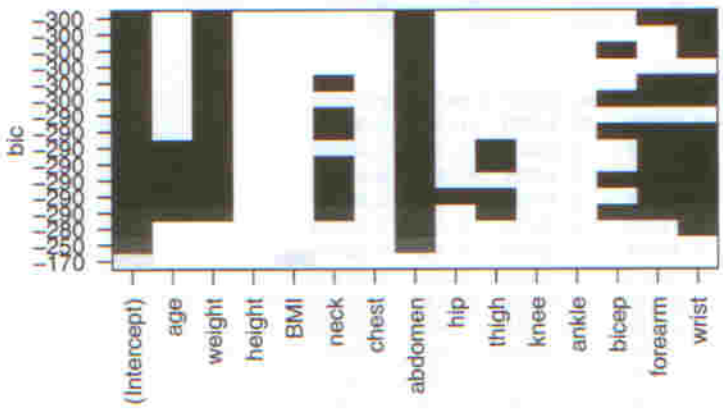


Figure 4.3: Model selection with **leaps()**

This plot shows the two best, by **regsubsets()** computed models with 1-8 regressors each. The BIC, coded in grey scale, does not improve after the fifth stage (starting from the bottom, see Figure 4.3). The optimal model can then be chosen from the models with "saturated" grey, and preferable that model is taken with the smallest number of variables.

4.2.5 Stepwise selection - automatic model search

- *Stepwise selection with* **drop1()**

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

Model:
body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist
        Df Sum of Sq    RSS    AIC  F value    Pr(F)
<none>                3738.3  706.2
age      1      58.1 3796.4  708.1   3.6511 0.057249 .
weight   1      41.3 3779.6  707.0   2.5974 0.108379
height   1      11.8 3750.0  705.0   0.7389 0.390892
BMI      1      16.8 3755.1  705.4   1.0553 0.305339
neck     1      60.4 3798.7  708.2   3.7978 0.052509 .
chest    1       5.8 3744.1  704.6   0.3630 0.547401
abdomen  1    1644.6 5382.9  795.4 103.3844 < 2.2e-16 ***
hip      1      42.5 3780.8  707.1   2.6700 0.103596
thigh    1      40.4 3778.7  706.9   2.5419 0.112202
knee     1   0.04766 3738.3  704.2   0.0030 0.956396
ankle    1       5.6 3743.9  704.6   0.3540 0.552429
bicep    1      16.6 3754.9  705.3   1.0451 0.307694
```

```
forearm  1      70.3 3808.6  708.9   4.4213  0.036558 *
wrist    1     144.8 3883.1  713.7   9.1002  0.002837 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

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

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

Residuals:
     Min       1Q   Median       3Q      Max
-10.0922  -2.6545  -0.1914   2.9011   9.2520

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -45.01721   36.54833  -1.232  0.21928
age           0.05699    0.02907   1.961  0.05107 .
weight       -0.16288    0.10014  -1.627  0.10516
height        0.17148    0.19942   0.860  0.39072
BMI           0.75481    0.73139   1.032  0.30312
neck         -0.42464    0.21682  -1.959  0.05135 .
chest        -0.05961    0.09885  -0.603  0.54704
abdomen       0.87123    0.08551  10.189  < 2e-16 ***
hip          -0.22594    0.13735  -1.645  0.10132
thigh         0.21554    0.12999   1.658  0.09862 .
ankle         0.12186    0.20432   0.596  0.55147
bicep         0.16398    0.15949   1.028  0.30491
forearm       0.39080    0.18520   2.110  0.03590 *
wrist        -1.49797    0.49329  -3.037  0.00266 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 3.98 on 236 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-Squared: 0.7432,         Adjusted R-squared: 0.7291
F-statistic: 52.54 on 13 and 236 DF,  p-value: < 2.2e-16
```

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

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

```
>model.lmstep<-step(model.lm)
Start:  AIC=706.23
body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist

          Df Sum of Sq    RSS    AIC
- knee     1   0.04766 3738.3  704.2
- ankle    1       5.6 3743.9  704.6
- chest    1       5.8 3744.1  704.6
- height   1      11.8 3750.0  705.0
- bicep    1      16.6 3754.9  705.3
- BMI      1      16.8 3755.1  705.4
<none>                3738.3  706.2
- thigh    1      40.4 3778.7  706.9
- weight   1      41.3 3779.6  707.0
- hip      1      42.5 3780.8  707.1
- age      1      58.1 3796.4  708.1
- neck     1      60.4 3798.7  708.2
- forearm  1      70.3 3808.6  708.9
- wrist    1     144.8 3883.1  713.7
- abdomen  1    1644.6 5382.9  795.4

Step:  AIC=704.23
```

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

          Df Sum of Sq    RSS    AIC
- ankle    1       5.6 3744.0  702.6
- chest    1       5.8 3744.1  702.6
- height   1      11.7 3750.1  703.0
- bicep    1      16.7 3755.1  703.4
- BMI      1      16.9 3755.2  703.4
<none>                3738.3  704.2
- weight   1      41.9 3780.3  705.0
- hip      1      42.9 3781.2  705.1
- thigh    1      43.6 3781.9  705.1
- neck     1      60.8 3799.1  706.3
- age      1      60.9 3799.3  706.3
- forearm  1      70.5 3808.9  706.9
- wrist    1     146.1 3884.4  711.8
- abdomen  1    1644.6 5382.9  793.4

                    ⋮

Step:  AIC=697.41
body.fat ~ age + weight + neck + abdomen + hip + thigh + forearm +
    wrist

          Df Sum of Sq    RSS    AIC
<none>                3786.2  697.4
- hip      1      37.4 3823.6  697.9
- age      1      59.3 3845.5  699.3
- neck     1      61.2 3847.4  699.4
- weight   1      74.7 3860.9  700.3
- thigh    1      77.5 3863.7  700.5
- forearm  1     114.0 3900.2  702.8
- wrist    1     135.8 3922.1  704.2
- abdomen  1    2712.5 6498.7  830.5

Call:
lm(formula = body.fat ~ age + weight + neck + abdomen + hip +
 thigh + forearm + wrist, data = fat)

Coefficients:
(Intercept)          age       weight         neck      abdomen          hip
  -18.46826      0.05577     -0.08081     -0.41183      0.87775     -0.20063
      thigh      forearm        wrist
    0.26719      0.46567     -1.39341
```

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

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

```
> anova(model.lm, model.lm1, model.lmstep)
Analysis of Variance Table

Model 1: body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + knee + ankle + bicep + forearm + wrist
Model 2: body.fat ~ age + weight + height + BMI + neck + chest + abdomen +
    hip + thigh + ankle + bicep + forearm + wrist
Model 3: body.fat ~ age + weight + neck + abdomen + hip + thigh + forearm +
    wrist
  Res.Df    RSS  Df Sum of Sq      F Pr(>F)
1    235 3738.3
2    236 3738.3  -1  -0.04766 0.0030 0.9564
3    241 3786.2  -5     -47.9 0.6017 0.6987
```

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

4.3 Methods using derived inputs as regressors in R

4.3.1 The problem of correlated regressors

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

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

with the regressors

$$x_1 \sim U(0, 1)$$

$$x_2 \sim U(0, 1)$$

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

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

Call:
lm(formula = y ~ x1)

Residuals:
     Min       1Q   Median       3Q      Max
-0.97272 -0.34061 -0.05724  0.29269  1.67159

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.04379    0.10689  -0.410    0.683
x1           0.98059    0.18515   5.296 7.26e-07 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 0.4859 on 98 degrees of freedom
Multiple R-Squared: 0.2225,        Adjusted R-squared: 0.2146
F-statistic: 28.05 on 1 and 98 DF,  p-value: 7.259e-07

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

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

Residuals:
     Min       1Q   Median       3Q      Max
-0.93258 -0.37100 -0.04242  0.28951  1.72204

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.1186     0.1331  -0.892    0.375
x1            0.9751     0.1853   5.261 8.56e-07 ***
x2            0.1600     0.1693   0.945    0.347
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 0.4862 on 97 degrees of freedom
Multiple R-Squared: 0.2296,        Adjusted R-squared: 0.2137
F-statistic: 14.46 on 2 and 97 DF,  p-value: 3.199e-06
```

- *Adding a highly correlated variable*

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

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

Residuals:
     Min       1Q   Median       3Q      Max
-0.92394 -0.33728 -0.03346  0.27951  1.70392

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.09655    0.13758  -0.702    0.485
x1          -0.03652    1.67213  -0.022    0.983
x3           1.01705    1.66168   0.612    0.542

Residual standard error: 0.4875 on 97 degrees of freedom
Multiple R-Squared: 0.2255,        Adjusted R-squared: 0.2096
F-statistic: 14.12 on 2 and 97 DF,  p-value: 4.139e-06
```

The fit of the model (R squared) does not change when x3 is added, none of the coefficients is significant. To estimate the amount of collinearity, x_i is regressed on all the other explanatory variables. The resulting R squared is a measurement of the magnitude of the relation.

```
> summary(lm(x3 ~ x1))

Call:
lm(formula = x3 ~ x1)

Residuals:
      Min        1Q    Median        3Q       Max
-0.050270 -0.021671  0.000931  0.028588  0.047027

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.051872   0.006519   7.957 3.13e-12 ***
x1          1.000065   0.011291  88.568  < 2e-16 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 0.02963 on 98 degrees of freedom
Multiple R-Squared: 0.9877,        Adjusted R-squared: 0.9875
F-statistic:  7844 on 1 and 98 DF,  p-value: < 2.2e-16
```

The R squared measure is 0.9877 in this model which means that x_1 and x_3 are highly correlated with each other.

- *Examination of the data* `fat` *for correlated regressors*

```
> round(cor(fat[, -1]), 3)

          age weight height   BMI  neck chest abdomen    hip  thigh  knee
age     1.000 -0.013 -0.245 0.120 0.113 0.177   0.231 -0.050 -0.201 0.018
weight -0.013  1.000  0.489 0.887 0.835 0.894   0.888  0.944  0.870 0.853
height -0.245  0.489  1.000 0.041 0.321 0.228   0.192  0.378  0.344 0.505
BMI     0.120  0.887  0.041 1.000 0.784 0.912   0.924  0.884  0.812 0.712
neck    0.113  0.835  0.321 0.784 1.000 0.787   0.759  0.748  0.708 0.680
chest   0.177  0.894  0.228 0.912 0.787 1.000   0.916  0.833  0.732 0.720
abdomen 0.231  0.888  0.192 0.924 0.759 0.916   1.000  0.875  0.766 0.736
hip    -0.050  0.944  0.378 0.884 0.748 0.833   0.875  1.000  0.894 0.821
thigh  -0.201  0.870  0.344 0.812 0.708 0.732   0.766  0.894  1.000 0.797
knee    0.018  0.853  0.505 0.712 0.680 0.720   0.736  0.821  0.797 1.000
ankle  -0.105  0.814  0.394 0.500 0.479 0.483   0.453  0.561  0.542 0.613
bicep  -0.041  0.800  0.319 0.747 0.733 0.728   0.685  0.743  0.765 0.680
```

```
forearm -0.085  0.631  0.322 0.560 0.624 0.581   0.504  0.550  0.572 0.559
wrist    0.214  0.735  0.397 0.633 0.744 0.664   0.626  0.645  0.572 0.675
          ankle  bicep forearm wrist
age      -0.105 -0.041  -0.085 0.214
weight    0.614  0.800   0.631 0.735
height    0.394  0.319   0.322 0.397
BMI       0.500  0.747   0.560 0.633
neck      0.479  0.733   0.624 0.744
chest     0.483  0.728   0.581 0.664
abdomen   0.453  0.685   0.504 0.626
hip       0.561  0.743   0.550 0.645
thigh     0.542  0.765   0.572 0.572
knee      0.613  0.680   0.559 0.675
ankle     1.000  0.485   0.419 0.568
bicep     0.485  1.000   0.678 0.635
forearm   0.419  0.678   1.000 0.587
wrist     0.568  0.635   0.587 1.000
```

Several variables (i.e. `abdomen` and `BMI`) are highly correlated with each other.

- *Regression of the variables against each other*

```
> summary(lm(weight ~ . - body.fat, data = fat))$r.squared
[1] 0.992625
> summary(lm(BMI ~ . - body.fat, data = fat))$r.squared
[1] 0.9909463
> summary(lm(chest ~ . - body.fat, data = fat))$r.squared
[1] 0.9065757
> summary(lm(abdomen ~ . - body.fat, data = fat))$r.squared
[1] 0.9240081
> summary(lm(hip ~ . - body.fat, data = fat))$r.squared
[1] 0.9326881
> summary(lm(body.fat ~ . - weight - BMI - chest - hip, data = fat))

Call:
lm(formula = body.fat ~ . - weight - BMI - chest - hip, data = fat)

Residuals:
      Min        1Q    Median        3Q       Max
-12.356507  -2.848101   0.008792   3.153073   9.261398

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.55144    7.97519   0.821 0.412193
age          0.08040    0.02928   2.746 0.006494 **
height      -0.09606    0.05028  -1.910 0.057284 .
neck        -0.56296    0.20591  -2.734 0.006725 **
abdomen      0.69792    0.05098  13.691  < 2e-16 ***
thigh        0.06441    0.12208   0.528 0.598266
knee        -0.14553    0.22914  -0.635 0.525958
ankle        0.05462    0.20409   0.268 0.789208
bicep        0.07944    0.15631   0.508 0.611743
forearm      0.43483    0.18504   2.350 0.019592 *
wrist       -1.74218    0.49509  -3.519 0.000519 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 4.047 on 239 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-Squared: 0.7311,     Adjusted R-squared: 0.7199
F-statistic: 64.99 on 10 and 239 DF,  p-value: < 2.2e-16
```

The fit of the original model does not change if the variables `weight`, `BMI`, `chest` and `hip` are discarded.

4.3.2 PCR

```
> x <- scale(fat[, -1], T, T)
> y <- scale(fat[, 1], T, F)
> library(pls)
> model.pcr <- pcr(y ~ x, ncomp = 14, validation = "CV", segments = 10,
+     segment.type = "random")
> summary(model.pcr)
Data:         X dimension: 250 14
        Y dimension: 250 1
Fit method: svdpc
Number of components considered: 14

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           7.661    6.039    5.148    4.988    4.963    5.010    4.856
adjCV        7.661    6.035    5.142    4.981    4.947    5.001    4.830
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       4.750    4.664    4.677     4.691     4.714     4.143     4.168
adjCV    4.724    4.650    4.662     4.675     4.697     4.127     4.152
       14 comps
CV        4.215
adjCV     4.196

TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X    63.66    74.63    81.46    86.36    90.46    92.75    94.73    96.51
y    38.96    55.87    58.79    60.27    60.78    64.95    66.00    66.53
   9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X    97.85     98.81     99.35     99.76     99.97    100.00
y    66.60     66.92     67.04     74.05     74.14     74.32
> plot(model.pcr, plottype = "validation", val.type = "MSEP", legend = "topright")
```

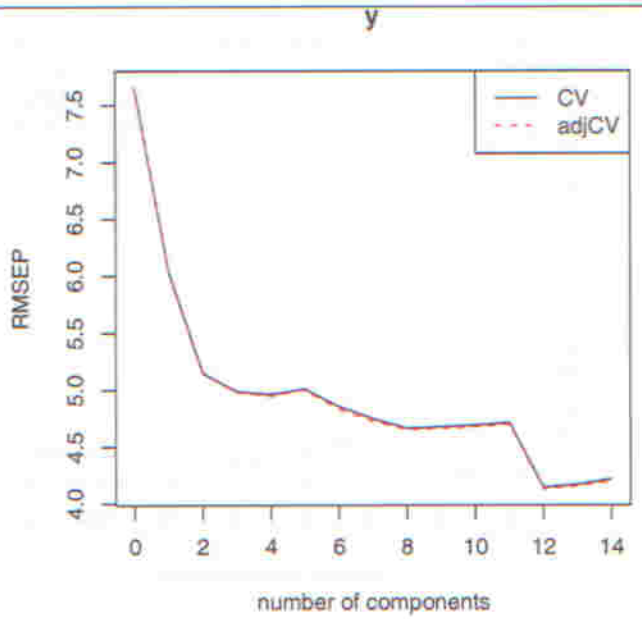


Figure 4.4: Determination of the number of components for PCR

The plot (see Figure 4.4) suggests to use 12 components. The RMSEP is defined as

$$\text{RMSEP} = \sqrt{\widehat{\text{Err}}_{\text{CV}}}$$

$\widehat{\text{Err}}_{\text{CV}}$ is the mean prediction error of Section 2.4.1 with the quadratic function L. RMSEP has a value of 4.143.

4.3.3 PLS regression

```
> model.plsr <- plsr(y ~ x, ncomp = 14, validation = "CV", segments = 10,
+     segment.type = "random")
> plot(RMSEP(model.plsr), legendpos = "topright")
> summary(model.plsr)
Data:         X dimension: 250 14
        Y dimension: 250 1
Fit method: kernelpls
Number of components considered: 14

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           7.661    5.684    4.766    4.566    4.422    4.302    4.211
adjCV        7.661    5.680    4.760    4.552    4.405    4.288    4.189
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       4.177    4.157    4.158     4.180     4.186     4.198     4.204
adjCV    4.160    4.141    4.143     4.163     4.169     4.180     4.186
       14 comps
CV        4.202
adjCV     4.183

TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X    62.91    74.04    78.96    83.24    86.75    88.97    92.33    94.57
y    45.89    63.22    68.09    70.57    72.38    73.99    74.11    74.16
   9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X    95.96     96.87     97.95     98.97     99.67    100.00
y    74.18     74.22     74.25     74.28     74.30     74.32
```

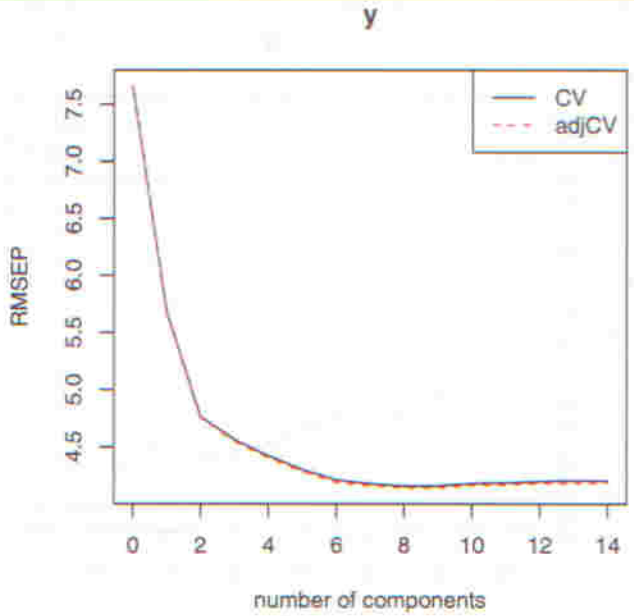
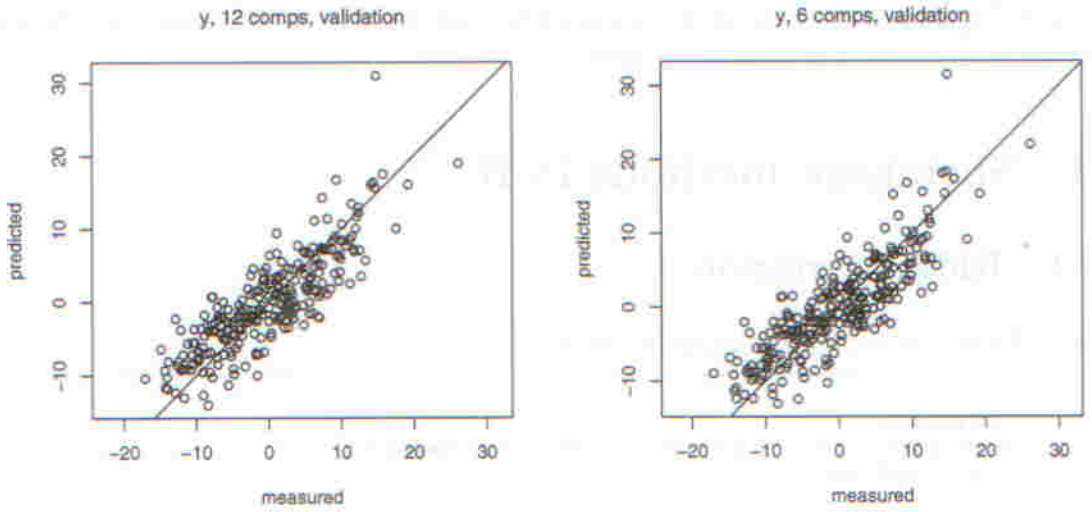


Figure 4.5: Determination of the number of components for PLS regression

Analyzing Figure 4.5 suggests that 6 components are sufficient. The RMSEP is 4.211.

Figure 4.6 compares the fit provided by PCR and PLSR. The predictions obtained by CV are plotted against the observed values. The left plot shows the $\hat{y}_i$ computed by the first 12 PCR components, the right plot uses the first 6 components of the PLSR.



(a) PCR (b) PLSR

Figure 4.6: Comparison of the fit provided by PCR and PLSR

4.3.4 Continuum regression

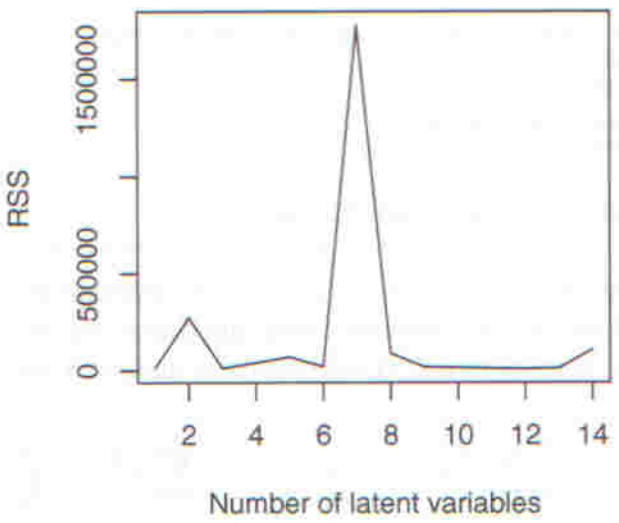


Figure 4.7: Continuum regression

```
>fat<-na.omit(fat)
>x<-scale(fat[,-1],T,T)
>y<-scale(fat[,1], T,F)
>source("cr.r")
>model.cont<-cr(x,y,lv=14, powers=0.75)
>y.hat<-x%*% model.cont
>resid<-(matrix(rep(y,14), nrow=length(y))-y.hat)
>RSS<-apply(resid^2,2,sum)
>plot(RSS, type = "l", xlab="Amount of latent variables", ylab="RSS")
```

The function `cr` will be provided in a library.

Figure 4.7 suggests the use of 6 components, just as PLSR. However, no checking with resampling methods has been done here.

4.4 Shrinkage methods in R

4.4.1 Ridge regression

- *RR without the special selection of λ*

```
> library(MASS)
> model.ridge <- lm.ridge(body.fat ~ ., data = fat, lambda = 1)
> model.ridge$Inter

[1] 1

> model.ridge$coef

       age      weight      height         BMI        neck       chest
0.79271367 -2.74668538  0.30184658  1.25208862 -1.06552555 -0.34544269
   abdomen         hip       thigh        knee       ankle       bicep
8.87964121 -1.43870319  1.06181696 -0.04751459  0.19309025  0.44022071
   forearm       wrist
0.79583229 -1.41917220
```

- *optimal selection of λ using GCV*

```
> model.ridge = lm.ridge(body.fat ~ ., data = fat, lambda = seq(0,
+     15, by = 0.2))
> select(model.ridge)

modified HKB estimator is 1.513289
modified L-W estimator is 4.41101
smallest value of GCV  at 1.2

> plot(model.ridge$lambda, model.ridge$GCV, type = "l")
```

Generalized cross validation (GCV) is a good approximation of the 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 $\boldsymbol{S}$ and this results in

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

with $\hat{f}^{-i}(\boldsymbol{x}_i)$ as fit without observation i and S_{ii} as ith diagonal element of S. The GCV approximation is

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

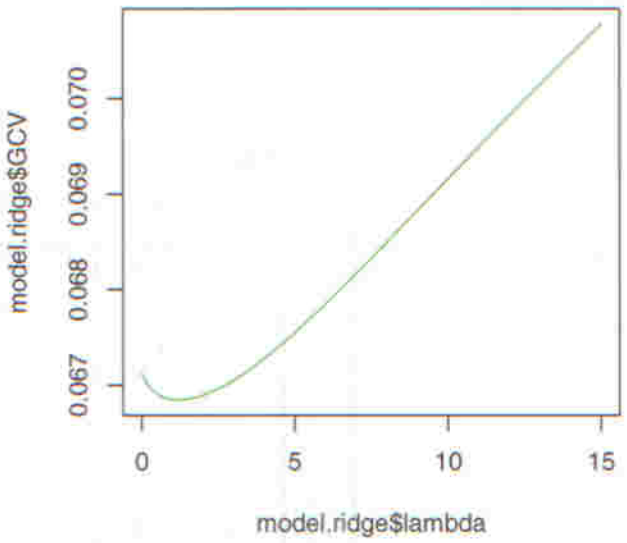


Figure 4.8: Optimal selection of λ with GCV

- *Plot of the ridge coefficients*

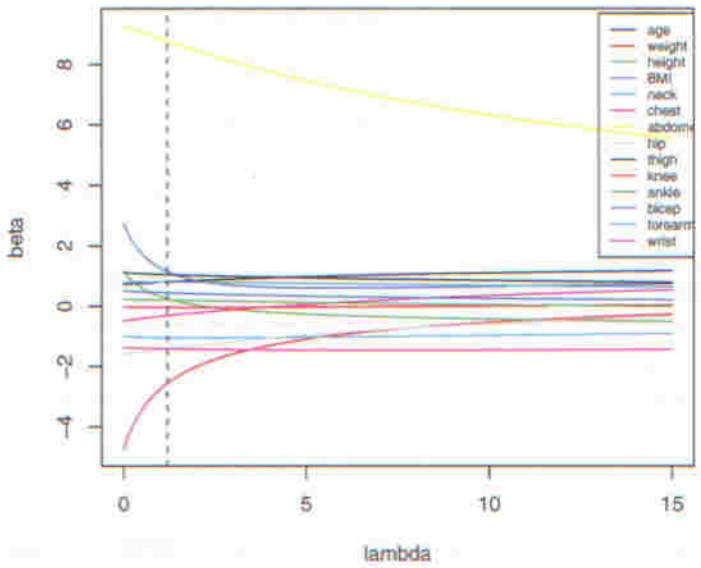


Figure 4.9: Ridge coefficients with varying λ

4.4.2 Lasso regression

- *Lasso regression*

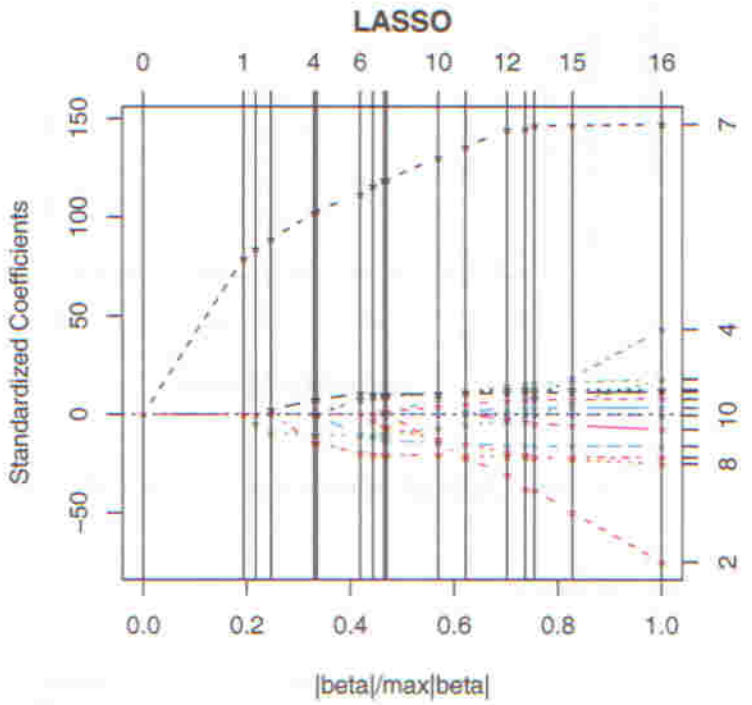


Figure 4.10: Lasso regression

```
>library(lars)
>model.lars<-lars(x,y)
>plot(model.lars)
```

- *optimal selection of s with CV*

```
> set.seed(123)
> cv.lasso <- cv.lars(x, y, K = 10, fraction = seq(0, 1, by = 0.05))
```

cv.lars() computes the 10-fold cross validation MSEP (MSE for test data set) of the lasso regression. We have set a random seed here in order to get a unique solution. For each considered fraction (standardized value of s) the average MSEP over all cv-segments is computed (points in the figure), as well as their standard error (drawn as intervals). The optimal value of the standardized s is the model to the smallest value of (standardized) s where the mean MSEP is no more than one standard error above the minimum mean MSEP. This rule selects a fraction of 0.3.

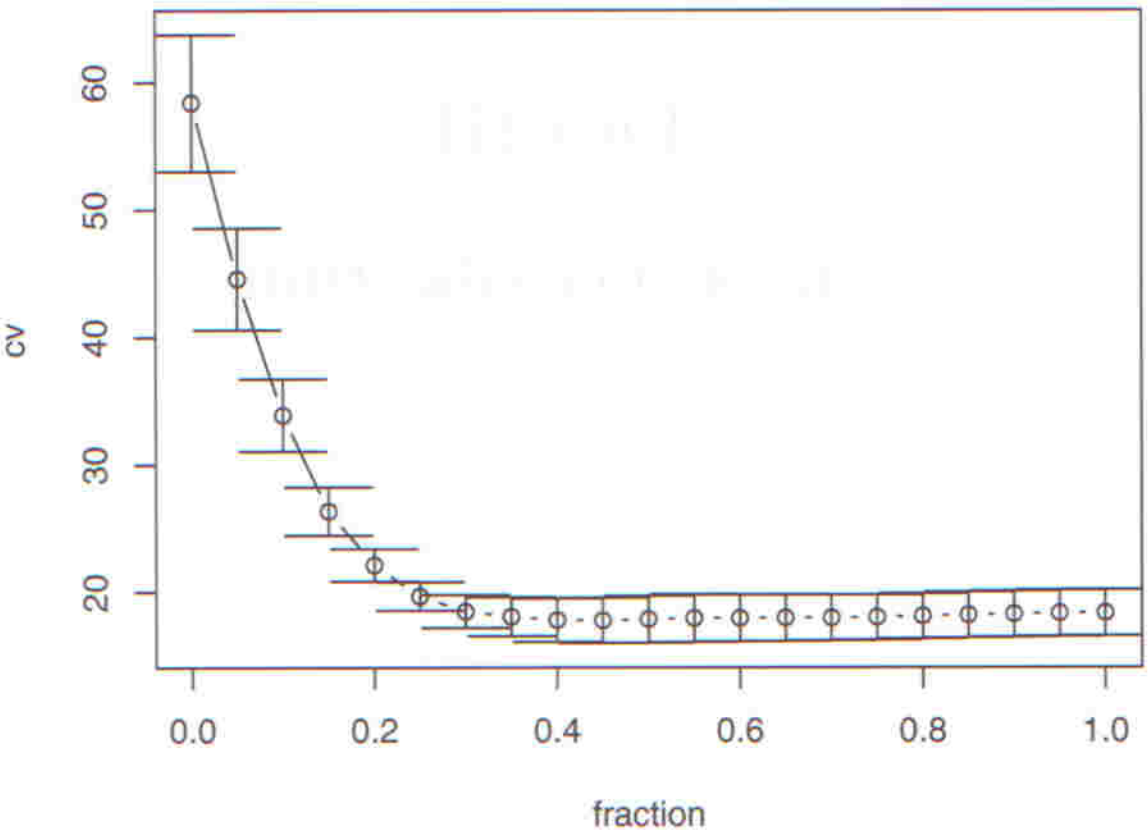


Figure 4.11: Optimal selection of s with CV

Part III

Linear classification

Chapter 5

Linear methods for classification

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

5.1 Linear regression of an indicator matrix

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

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

with

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

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

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

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

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

The estimated values are then

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

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

- Compute the K vector of the fitted values

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

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

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

Section 6.1, page 60

5.2 Linear discriminant analysis (LDA)

5.2.1 Classical LDA

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

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

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

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

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

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

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

The decision boundary between the classes k and l is

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

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

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

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

The linear discriminant function

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

provides an equivalent description of the decision rule

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

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

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

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

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

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

Comparison of LDA and LS

- When $\pi_1 = \pi_2$, both methods compute the same result, even if $\boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$ (Figure 6.7, Figure 6.8).

- Otherwise, LS computes a different intercept which is not optimal in terms of the minimization of the misclassification rate (Figure 6.9).
- In contrast to LS, LDA requires Gaussian distribution. LS could be used with non normally distributed data, though the intercept has to be optimized in terms of the minimal misclassification rate (Figure 6.10).

Section 6.2.1, page 64

5.2.2 Quadratic discriminant analysis (QDA)

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

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

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

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

Observation $\boldsymbol{x}$ is classified to that group which yields the maximal value of the discriminant function. $\boldsymbol{\Sigma}_k$, $\boldsymbol{\mu}_k$ and π_k can be estimated from the training data (see Figure 6.11).

Comparison QDA and LDA

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

Section 6.2.2, page 70

5.2.3 LDA with quadratic terms

LDA could be applied on a combination of the original variables with corresponding quadratic terms. For example, with 2 original variables we apply LDA on the variables x_1, x_2, $x_1 \cdot x_2$, x_1^2, x_2^2. This approach provides a quadratic decision boundary (see Figure 6.12).

Generally speaking, there is only a small difference between LDA with quadratic terms and QDA. Especially in higher dimension QDA should be preferred.

Section 6.2.3, page 71

5.2.4 Regularized discriminant analysis

This method (Figure 6.13) is a compromise between linear and quadratic discriminant analysis, that allows to shrink the separate covariances of QDA towards a common covariance as in LDA. These common (pooled) covariance matrices have the form

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

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

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

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

Section 6.2.4, page 72

5.3 Logistic regression

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

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

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

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

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

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

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

In the special case of $K=2$ classes the model is

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

or

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

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

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

where

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

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

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

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

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

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

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

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

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

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

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

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

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

and thus:

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

The Newton-Raphson algorithm has the form

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

with the adjusted response

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

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

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

Some remarks:

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

Comparison of logistic regression and LDA

Logistic regression uses

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

whereas LDA uses

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

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

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

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

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

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

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

Section 6.3, page 73

5.4 Separating hyperplanes

These procedure also constructs linear decision boundaries that explicitly try to separate the data into different classes. Classifiers that compute a linear combination of the input features and return the sign for the class membership are called "perceptrons" (this term is frequently used in "neural networks").

Mathematical background

A hyperplane (see Figure 5.1) is characterized by the set $\mathcal{L}$ which is defined by

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

with the following properties:

- For any 2 points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in $\mathcal{L}$,

  $$\boldsymbol{\beta}^\top (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0,$$

  thus $\boldsymbol{\beta}^* = \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|}$ is an orthogonal vector to $\mathcal{L}$.
- For any point $\boldsymbol{x}_0$ in $\mathcal{L}$ we have $\boldsymbol{\beta}^\top \boldsymbol{x}_0 = -\beta_0$.

- The signed distance of any point $\boldsymbol{x}$ to $\mathcal{L}$ is given by

$$\boldsymbol{\beta}^{*\top}(\boldsymbol{x} - \boldsymbol{x}_0) = \frac{1}{\|\boldsymbol{\beta}\|}(\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0) = \frac{1}{\|f'(\boldsymbol{x})\|} f(\boldsymbol{x})$$

  and thus, $f(\boldsymbol{x})$ is proportional to the distance from $\boldsymbol{x}$ to the hyperplane defined by $f(\boldsymbol{x}) = 0$.

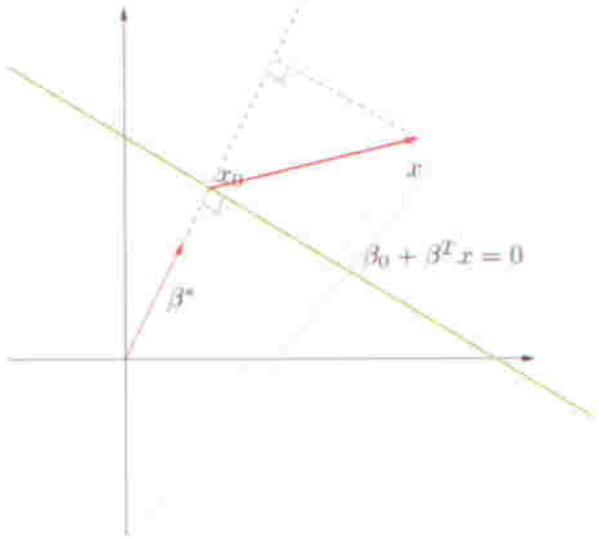


Figure 5.1: Hyperplane $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} = 0$

5.4.1 Rosenblatt's perceptron learning algorithm

The perceptron learning algorithm tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary. If a response $y_i = 1$ is misclassified, then $\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta} < 0$, else if $y_i = -1$ is misclassified, then $\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta} > 0$. The goal is to minimize

$$D(\beta_0, \boldsymbol{\beta}) = -\sum_{i \in \mathcal{M}} y_i(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta})$$

where $\mathcal{M}$ indexes the number of misclassified points. $D(\beta_0, \boldsymbol{\beta})$ is non-negative and proportional to the distance of the misclassified points to the decision boundary defined by $\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} = 0$. The gradient is

$$\begin{aligned} \frac{\partial D(\beta_0, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= -\sum_{i \in \mathcal{M}} y_i \boldsymbol{x}_i \\ \frac{\partial D(\beta_0, \boldsymbol{\beta})}{\partial \beta_0} &= -\sum_{i \in \mathcal{M}} y_i \end{aligned}$$

For finding the solution, a stochastic gradient algorithm is used. This algorithm does not compute the sum of the gradient, but the contributions of the observations followed by a step in the direction of the negative gradient. Hence each of the misclassified observations are visited in some sequence, and the parameters are updated via

$$\begin{pmatrix} \boldsymbol{\beta} \\ \beta_0 \end{pmatrix} \leftarrow \begin{pmatrix} \boldsymbol{\beta} \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} y_i \boldsymbol{x}_i \\ y_i \end{pmatrix}.$$

The learning rate ρ can be taken to be 1.

A disadvantage of the algorithm is: if the classes can be perfectly separated by hyperplanes, we have an infinite number of solutions (see Figure 5.2), depending on the starting value. "Support vector machines" or "optimal separating hyperplanes" perform better [see Hastie et al., 2001].

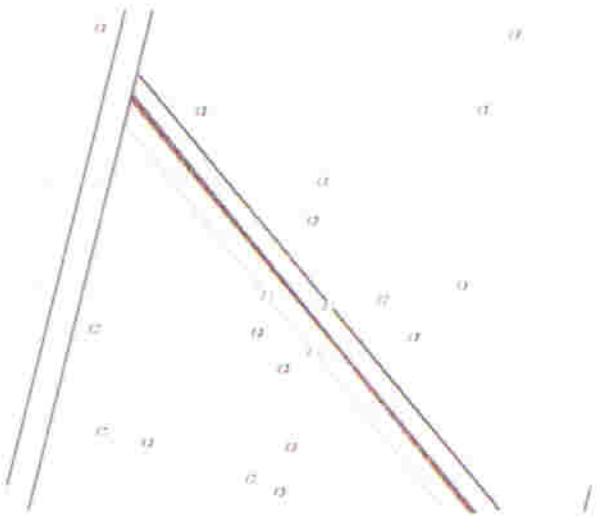


Figure 5.2: Perfect separation of 2 classes by a hyperplane.

Chapter 6

Linear methods for classification in R

6.1 Linear regression of an indicator matrix in R

- *Regression with 2 groups of the same covariance structure*

  Generate 2 classes with 100 observations each (Figure 6.1a) and the corresponding indicator variable y for each class.

```
> set.seed(1234)
> grp = c(rep(1, 100), rep(2, 100))
> g1 = matrix(rnorm(2 * 100, 0, 1), ncol = 2)
> g2 = matrix(rnorm(2 * 100, 4, 1), ncol = 2)
> x = rbind(g1, g2)
> plot(x, pch = grp, col = grp + 2, xlab = "X1", ylab = "Y1")
> y1 = c(rep(1, 100), rep(0, 100))
> y2 = c(rep(0, 100), rep(1, 100))
> y = cbind(y1, y2)
```

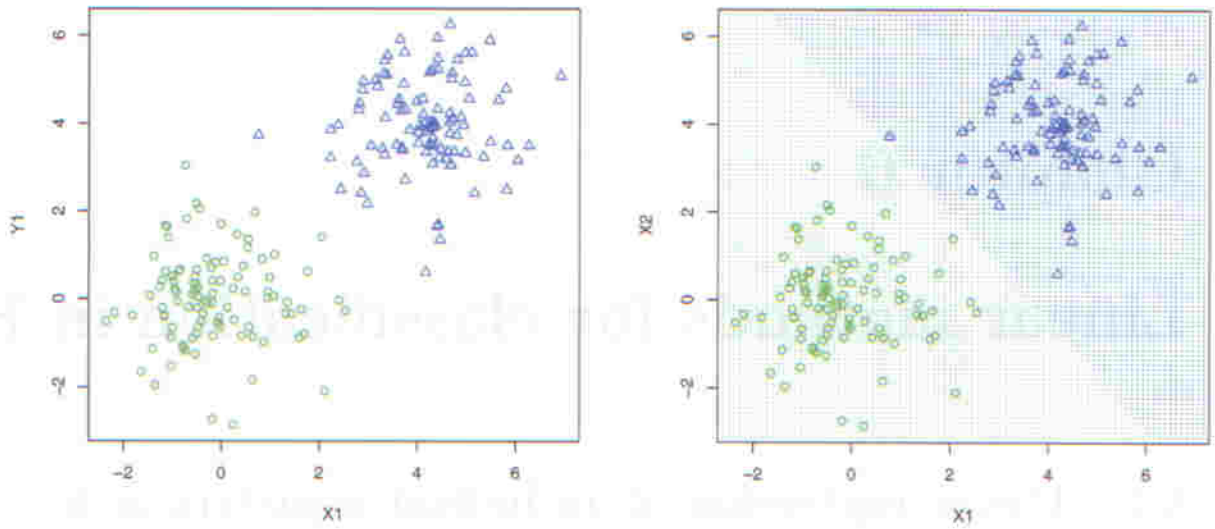
- *Regression with the indicator variable, and result shown in Figure 6.1b*

```
> res.lm <- lm(y ~ x)
```

- *Regression with 2 groups of different covariance structure*

  Generate 2 multivariate normally distributed classes with 100 observations each (Figure 6.2a) and the corresponding indicators y for each class.

```
> set.seed(1234)
> library(mvtnorm)
> grp = c(rep(1, 100), rep(2, 100))
> g1 = rmvnorm(100, mean = c(0, 0), sigma = matrix(c(1.2, 1, 1,
+     1.2), ncol = 2))
> g2 = rmvnorm(100, mean = c(2, 2), sigma = matrix(c(1, -1, -1,
+     2), ncol = 2))
> x = rbind(g1, g2)
> plot(x, pch = grp, col = grp + 2, xlab = "X1", ylab = "Y1")
> y1 = c(rep(1, 100), rep(0, 100))
> y2 = c(rep(0, 100), rep(1, 100))
> y = cbind(y1, y2)
```

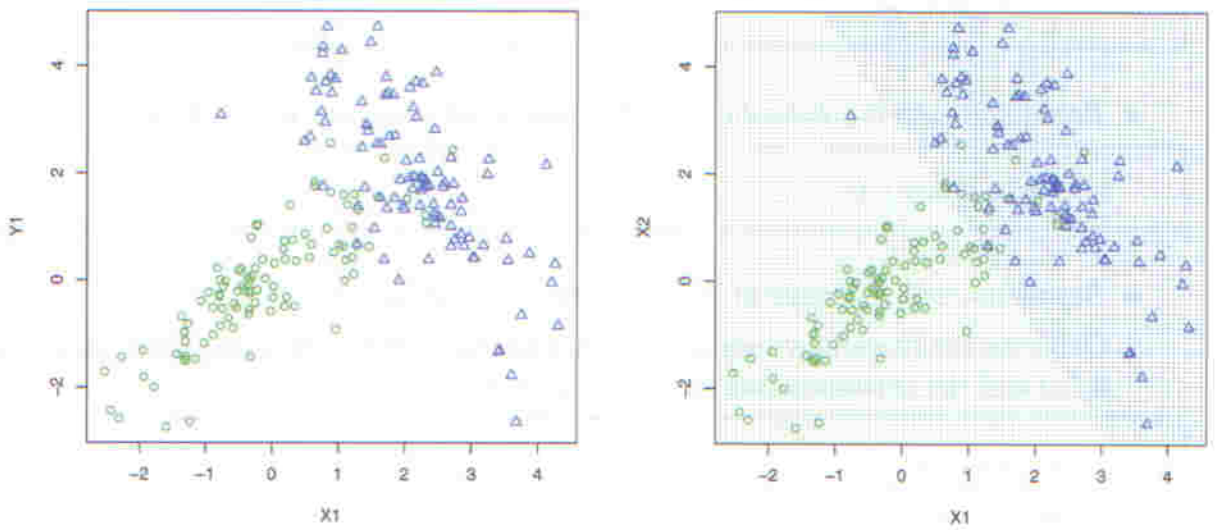


(a) generated data (b) result from regression

Figure 6.1: Classification of 2 groups with the same covariance structure

- *Regression with the indicator variable, and result shown in Figure 6.2b*

```
> res.lm <- lm(y ~ x)
```



(a) generated data (b) result from regression

Figure 6.2: Classification of 2 groups with different covariance structure

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

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

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

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

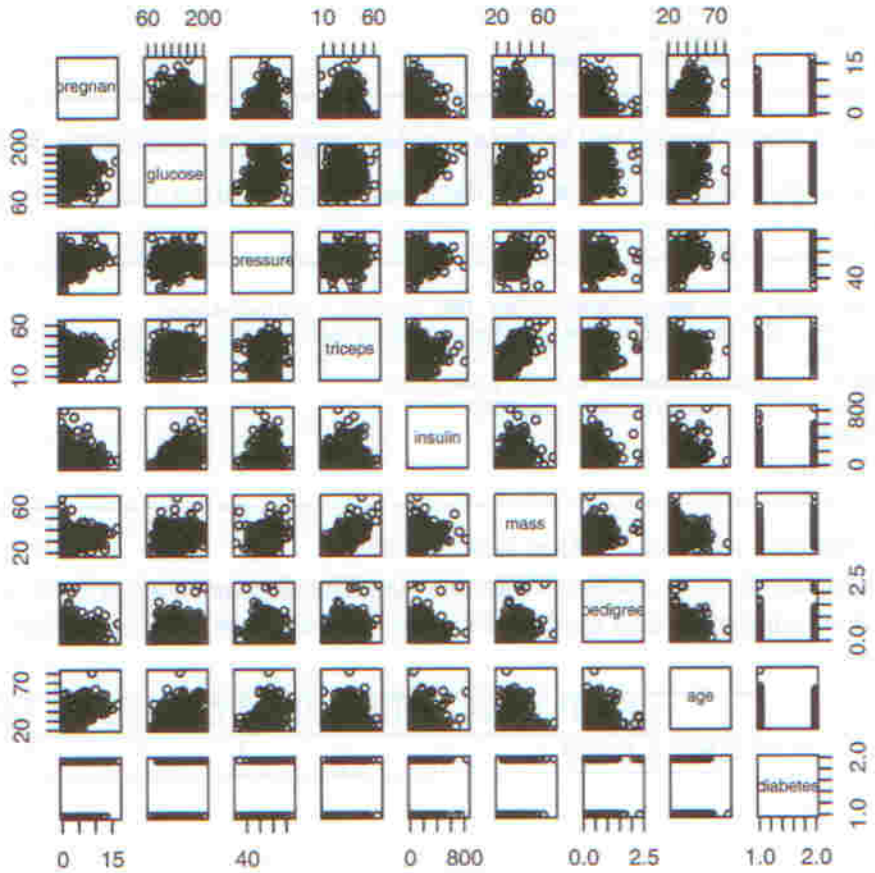


Figure 6.3: PimaIndiansDiabetes2-data

Generation of the indicator matrix for the regression:

```
> pidind <- pid
> pidindy <- array(rep(0), dim = c(392, 2))
> for (i in 1:392) {
+     if (pidind[i, 9] == "neg")
+         pidindy[i, 1] = 1
+     else pidindy[i, 2] = 1
+ }
> pidind[, 9] <- pidindy
```

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

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

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

Coefficients:
              [,1]         [,2]
(Intercept)  2.059e+00 -1.059e+00
pregnant    -1.292e-02  1.292e-02
glucose     -6.046e-03  6.046e-03
pressure     3.797e-04 -3.797e-04
triceps     -3.315e-03  3.315e-03
insulin      6.719e-05 -6.719e-05
mass        -7.820e-03  7.820e-03
pedigree    -8.852e-02  8.852e-02
age         -7.654e-03  7.654e-03
```

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

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

```
> predictind <- as.numeric(predict(mod.ind, newdata = pidind[-train,
+     ])[, 1] < predict(mod.ind, newdata = pidind[-train, ])[,
+     2])
> TAB <- table(pid$diabetes[-train], predictind)
> mklrate <- 1 - sum(diag(TAB))/sum(TAB)
> mklrate

[1] 0.2391304
```

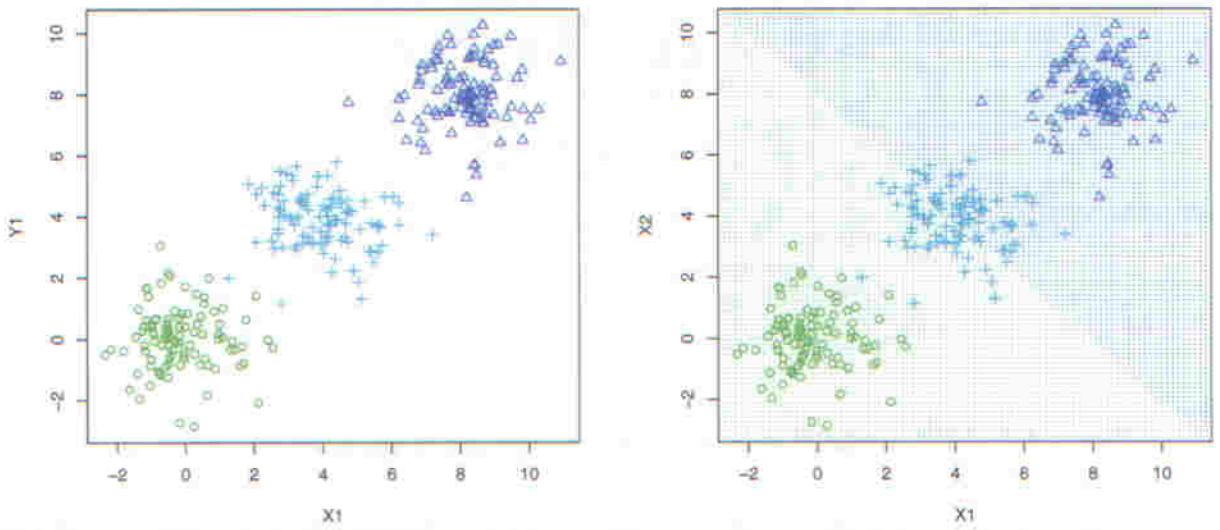
The resulting misclassification rate is 0.239.
The misclassification rates of different classification methods will be collected in a table and then compared and analyzed throughout the rest of this manuscript.

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239						

- *Regression with 3 classes having the same covariance structure*

  Generate 3 multivariate normally distributed classes (Figure 6.4a) with 100 observations each and an indicator y for each class.

```
> set.seed(1234)
> grp = c(rep(1, 100), rep(2, 100), rep(3, 100))
> g1 = matrix(rnorm(2 * 100, 0, 1), ncol = 2)
> g2 = matrix(rnorm(2 * 100, 8, 1), ncol = 2)
> g3 = matrix(rnorm(2 * 100, 4, 1), ncol = 2)
> x = rbind(g1, g2, g3)
> plot(x, pch = grp, col = grp + 4, xlab = "X1", ylab = "Y1")
> y1 = c(rep(1, 100), rep(0, 100), rep(0, 100))
> y2 = c(rep(0, 100), rep(1, 100), rep(0, 100))
> y3 = c(rep(0, 100), rep(0, 100), rep(1, 100))
> y = cbind(y1, y2, y3)
```



(a) Generated data

(b) Results from regression

Figure 6.4: Regression of data with 3 groups

- *Regression with the 3 indicator variables, and result in Figure 6.4b*

```
> res.lm <- lm(y ~ x)
```

As can be seen in Figure 6.4b, linear regression does not work well if $K > 2$. Some classes can be masked by others, this is the reason for the bad classification in this example. There are different methods to avoid such problems:

- use of other regression method, like quadratic regressions (quadratic terms $x_1 \cdot x_2, x_1^2, x_2^2$)
- use of other methods like LDA

6.2 Linear Discriminant Analysis in R

6.2.1 Classical LDA

- *LDA for 2-dimensional multivariate normally distributed data with parameters* μ_1, μ_2, Σ

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

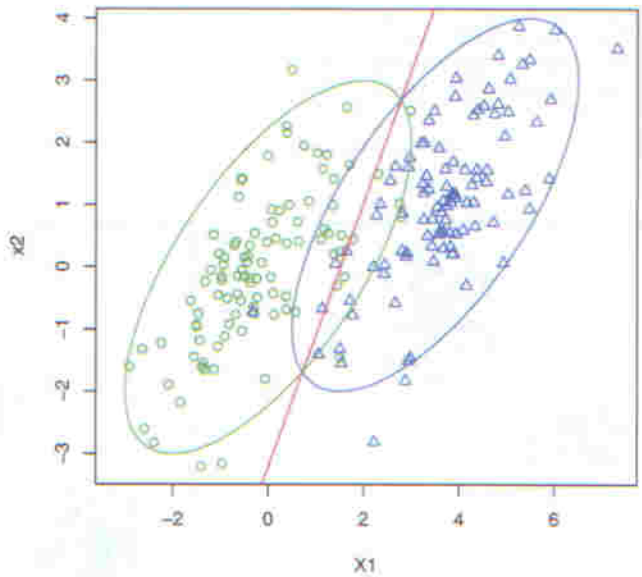


Figure 6.5: LDA separation line using the population parameters and $\pi_1 = \pi_2$

The ellipses in Figure 6.5 are so-called tolerance ellipses which (in case of multivariate normal distribution) include the inner 95% of the data of each group. The LDA separation line was determined using the known population parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$ and $\boldsymbol{\Sigma}$. The command **lda()** can be found in `library(MASS)`.

- *Change of the prior probabilities – see Figure 6.6*

  When changing the prior probabilities, the LDA separation line is moved towards the group with smaller prior probability (see Figure 6.6). This is done because LDA minimizes the probability of misclassification.

- *LDA boundary versus LS when* $\pi_1 = \pi_2$

  It can be seen from Figure 6.7 that LDA and LS regression find the same solution when $\pi_1 = \pi_2$.

- *LDA versus LS regression when* $\pi_1 = \pi_2, \boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$

  The LDA function (line) in Figure 6.8 has been computed from the known population parameters $\pi_1, \pi_2, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2$.

- *LDA versus LS regression when* $\pi_1 \neq \pi_2, \boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$

  LS regression gives separating line parallel to the LDA line (see Figure 6.9).

- *LDA versus LS regression when* $\pi_1 = \pi_2$, *no Gaussian distribution*

  The results of LS regression and LDA are the same (Figure 6.10), only because the prior probabilities are the same. However, both methods are not optimal with respect to minimizing the misclassification error (LDA needs normally distributed groups with equal covariances).

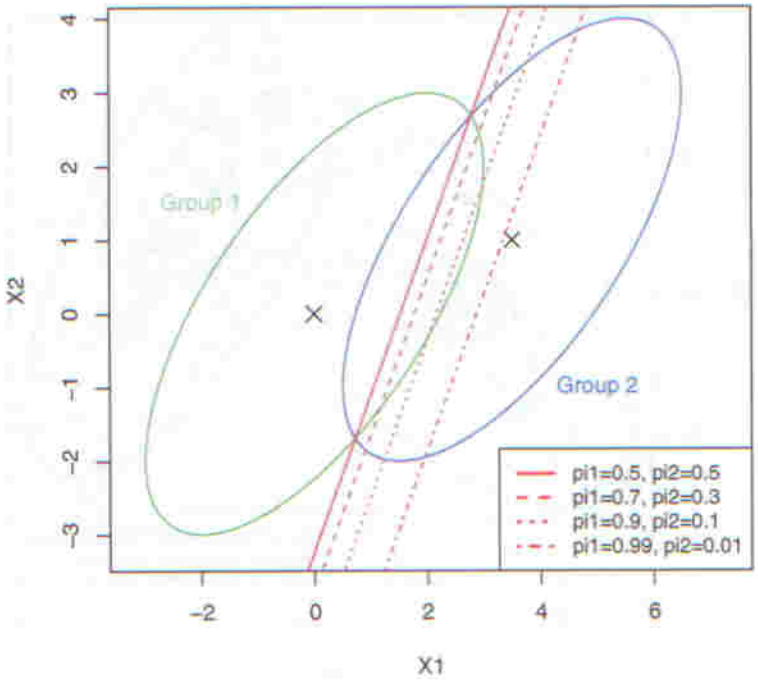


Figure 6.6: Change of the prior probabilities

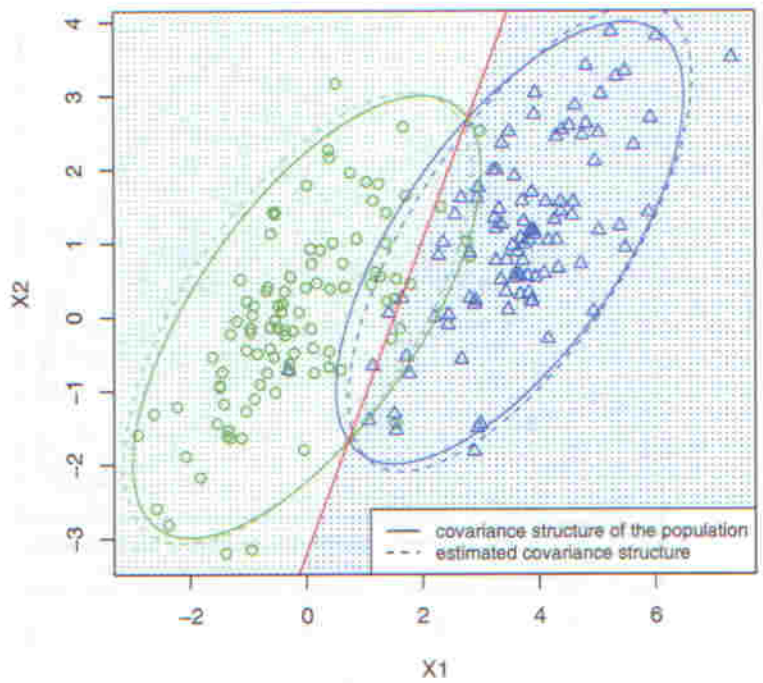


Figure 6.7: LDA (line) versus LS (dots) when $\pi_1 = \pi_2$

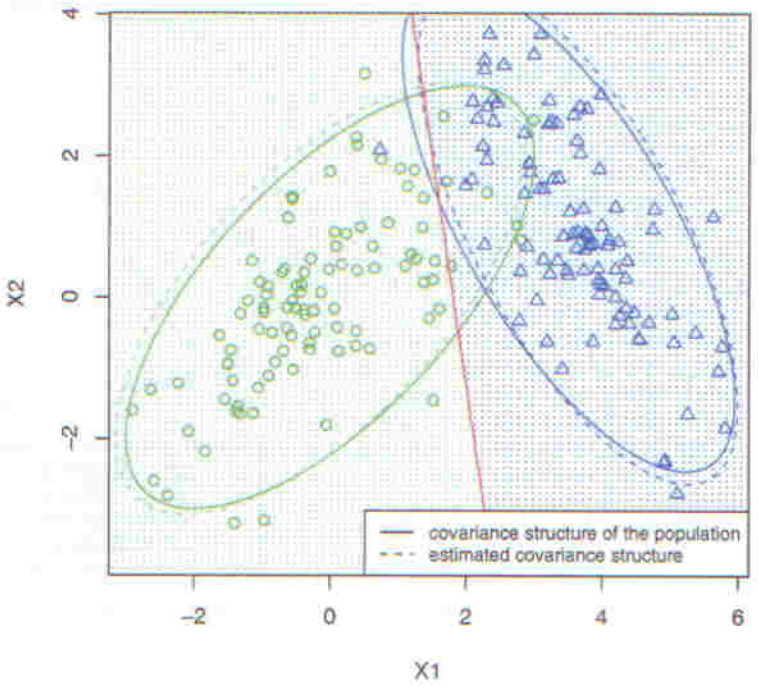


Figure 6.8: LDA versus LS regression when $\pi_1 = \pi_2, \boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$

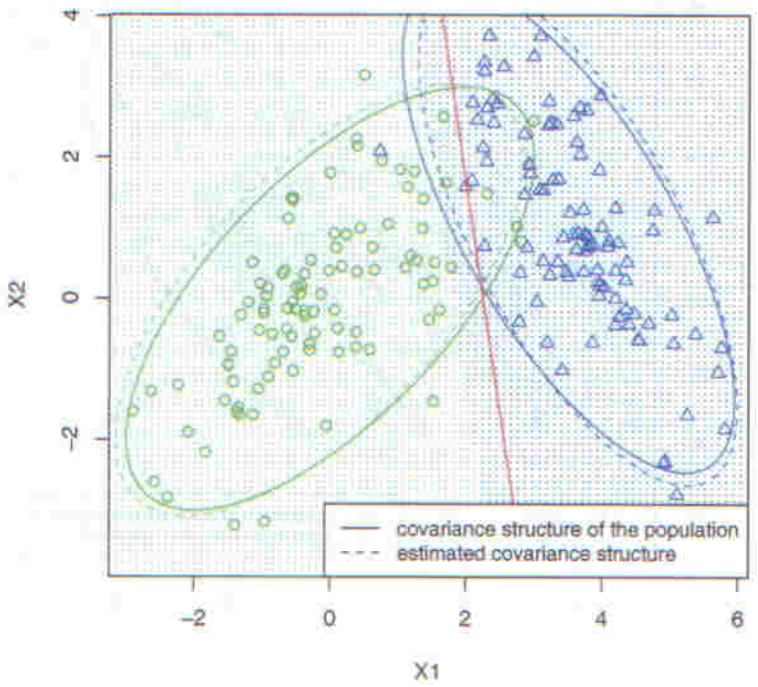


Figure 6.9: LDA (line) and LS (dots) when $\pi_1 \neq \pi_2, \boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$

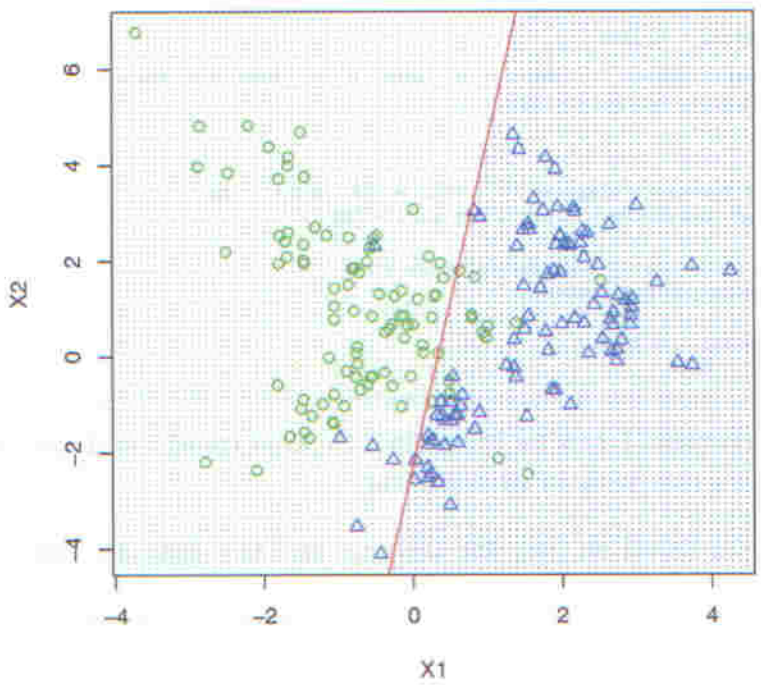


Figure 6.10: LDA (line) and LS (dots) $\pi_1 = \pi_2$, no Gaussian distribution

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

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

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

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

        5-fold cross-validation estimator of misclassification error

Misclassification error:  0.2168

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

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

        Bootstrap estimator of misclassification error
        with 10 bootstrap replications

Misclassification error:  0.2159
Standard deviation: 0.0066
```

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

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

```
> set.seed(101)
> train <- sample(1:nrow(pid), 300)
> mod.lda <- lda(diabetes ~ ., data = pid[train, ])
> TAB <- table(pid[-train, ]$diabetes, mypred(mod.lda, pid[-train,
+     ]))
> mkrlda <- 1 - sum(diag(TAB))/sum(TAB)
> mkrlda
[1] 0.2391304
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239					

6.2.2 QDA

- *Classification with* **qda()**

  Generate 3 classes with 100 two dimensional normal distributed values with the parameters:

$$\mu_1 = (0,0), \quad \mu_2 = (3,1), \quad \mu_3 = (-1,2)$$

$$\Sigma_1 = \begin{pmatrix} 1.2 & 1 \\ 1 & 1.2 \end{pmatrix}, \quad \Sigma_2 = \begin{pmatrix} 0.7 & -.1 \\ -.1 & 1.2 \end{pmatrix}, \quad \Sigma_3 = \begin{pmatrix} 1 & -0.1 \\ -0.1 & 0.5 \end{pmatrix}$$

```
>x1=rmvnorm(100,mu1,sig1)
>x2=rmvnorm(100,mu2,sig2)
>x3=rmvnorm(100,mu3,sig3)
>x=rbind(x1,x2,x3)
>grp=c(rep(1,100),rep(2,100),rep(3,100))
>res=qda(x,grp)
```

The result of the classification with QDA can be found in Figure 6.11.

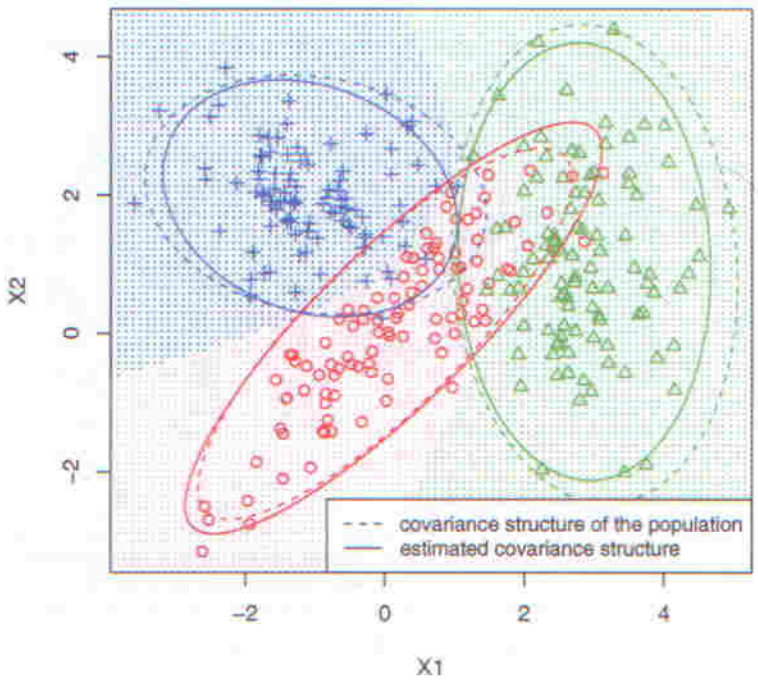


Figure 6.11: QDA with 3 normally distributed simulated data groups

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

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

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25				

6.2.3 LDA with quadratic terms

Generate the same data as in Figure 6.11 and apply LDA with quadratic terms included:

```
>res<-lda(cbind(x,x[,1]*x[,2],x[,1]^2,x[,2]^2),grp)
```

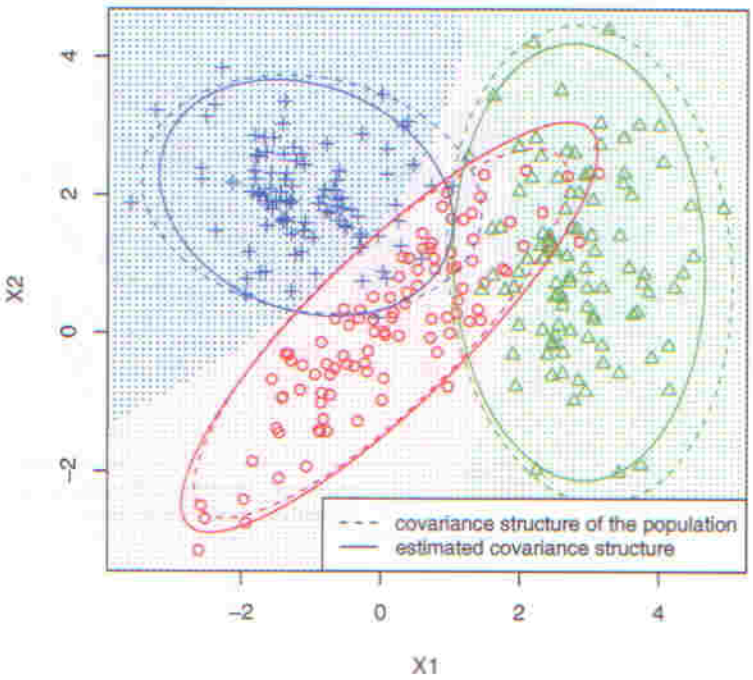


Figure 6.12: LDA with quadratic terms

6.2.4 Regularized discriminant analysis

- *Regularized discriminant analysis on generated data*

```
>library(klaR)
>res=rda(x,grp)
```

The function **rda()** can be found in `library(klaR)`. Here, the same data are used as with QDA in Figure 6.11.

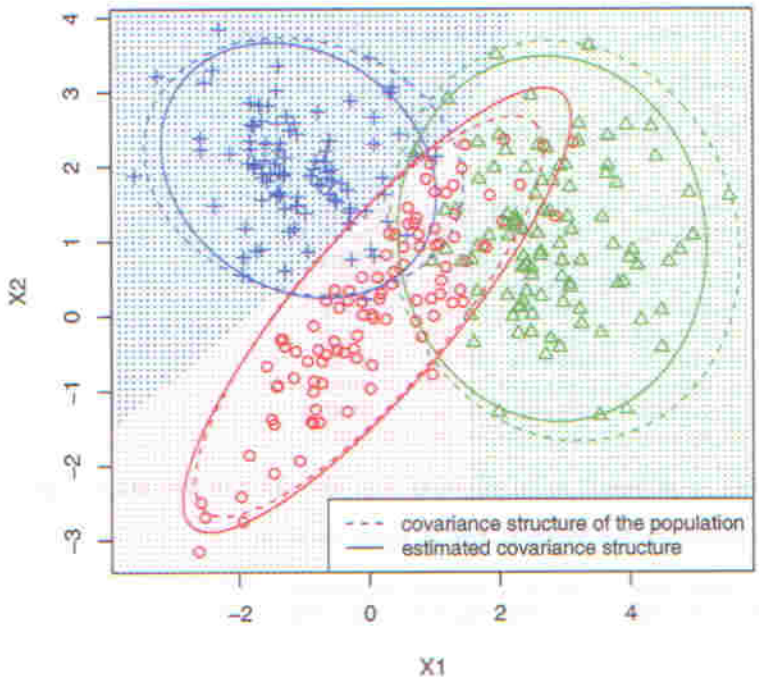


Figure 6.13: Regularized discriminant analysis on three simulated normally distributed data groups

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

```
> mod.rda <- rda(diabetes ~ ., data = pid)
> predictrda <- predict(mod.rda, pid[-train, ])
> TAB <- table(pid$diabetes[-train], predictrda$class)
> mkrrda <- 1 - sum(diag(TAB))/sum(TAB)
> mkrrda
[1] 0.2173913
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.217			

6.3 Logistic regression in R

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

  Logistic regression can be carried out with the function **glm()** under the model of the binomial family. Syntax and interpretation of the inference statistic are in analogy to **lm()**.

```
> attach(pid)
> modelglm <- glm(diabetes ~ ., data = pid, family = binomial)
> summary(modelglm)

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

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7823  -0.6603  -0.3642   0.6409   2.5612

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.004e+01  1.218e+00  -8.246  < 2e-16 ***
pregnant     8.216e-02  5.543e-02   1.482  0.13825
glucose      3.827e-02  5.768e-03   6.635 3.24e-11 ***
pressure    -1.420e-03  1.183e-02  -0.120  0.90446
triceps      1.122e-02  1.708e-02   0.657  0.51128
insulin     -8.253e-04  1.306e-03  -0.632  0.52757
mass         7.054e-02  2.734e-02   2.580  0.00989 **
pedigree     1.141e+00  4.274e-01   2.669  0.00760 **
age          3.395e-02  1.838e-02   1.847  0.06474 .
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 498.10  on 391  degrees of freedom
Residual deviance: 344.02  on 383  degrees of freedom
AIC: 362.02

Number of Fisher Scoring iterations: 5
```

- *Model selection with* **step()**

```
> mod.glm <- step(modelglm, direction = "both")
Start:  AIC=362.02
diabetes ~ pregnant + glucose + pressure + triceps + insulin +
    mass + pedigree + age

           Df Deviance    AIC
- pressure  1   344.04 360.04
- insulin   1   344.42 360.42
- triceps   1   344.45 360.45
<none>          344.02 362.02
- pregnant  1   346.24 362.24
- age       1   347.55 363.55
- mass      1   350.89 366.89
- pedigree  1   351.58 367.58
- glucose   1   396.95 412.95

              ⋮

Step:  AIC=356.89
diabetes ~ pregnant + glucose + mass + pedigree + age

           Df Deviance    AIC
<none>          344.89 356.89
- pregnant  1   347.23 357.23
```

```
+ triceps    1   344.42 358.42
+ insulin    1   344.46 358.46
- age        1   348.72 358.72
+ pressure   1   344.88 358.88
- pedigree   1   352.72 362.72
- mass       1   360.44 370.44
- glucose    1   411.85 421.85
Coefficients:
(Intercept)     pregnant      glucose         mass     pedigree          age
   -9.99208      0.08395      0.03646      0.07814      1.15091      0.03436

Degrees of Freedom: 391 Total (i.e. Null);  386 Residual
Null Deviance:      498.1
Residual Deviance: 344.9        AIC: 356.9
```

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

```
> summary(mod.glm)

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

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8827  -0.6535  -0.3694   0.6521   2.5814

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -9.992080   1.086866  -9.193  < 2e-16 ***
pregnant     0.083953   0.055031   1.526 0.127117
glucose      0.036458   0.004978   7.324 2.41e-13 ***
mass         0.078139   0.020605   3.792 0.000149 ***
pedigree     1.150913   0.424242   2.713 0.006670 **
age          0.034360   0.017810   1.929 0.053692 .
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 498.10  on 391  degrees of freedom
Residual deviance: 344.89  on 386  degrees of freedom
AIC: 356.89

Number of Fisher Scoring iterations: 5
```

- *Prediction of the class membership and presentation of the results, see Figure 6.14*

```
> predict(mod.glm, pid)
> plot(predict(mod.glm, pid), col = as.numeric(diabetes) + 2)
```

- *Comparison of LDA and logistic regression*
  - Logistic regression makes no assumption about the distribution. LDA assumes Gaussian distributions with equal covariances.
  - Comparison for the full model, see Figure 6.15

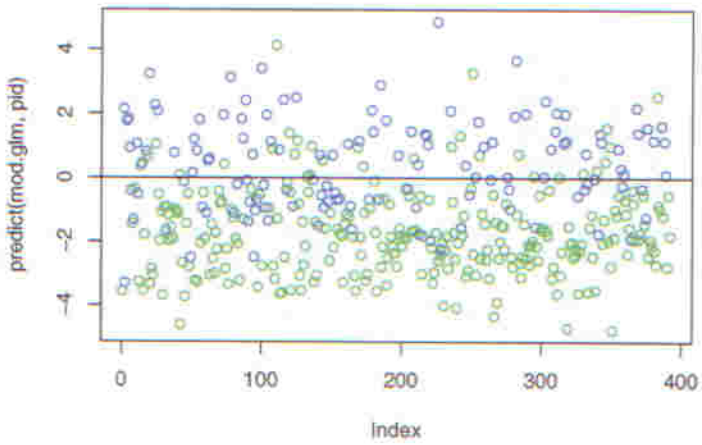


Figure 6.14: Prediction of the group memberships; the line is the separation from logistic regression, the color is the true group membership

```
> modful <- lda(diabetes ~ pregnant + glucose + pressure + insulin +
+     triceps + mass + pedigree + age, data = pid)
> plot(predict(modelglm, pid), col = as.numeric(diabetes) + 2,
+     pch = as.numeric(predict(modful, pid)$class))
```

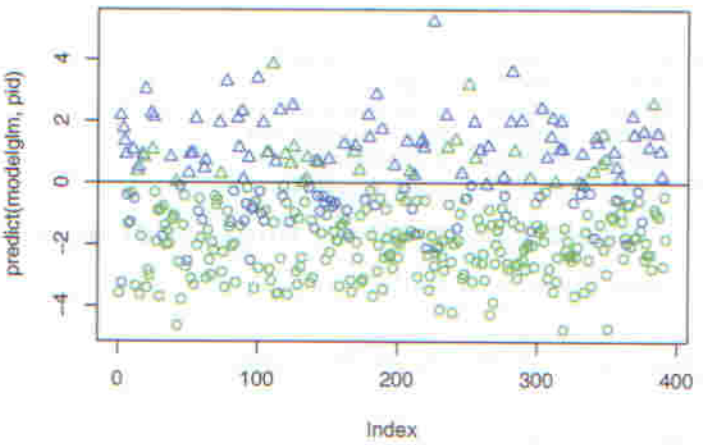


Figure 6.15: Full model: LDA (symbols) versus logistic regression (line); the color corresponds to the true classes

– Comparison for the reduced model, see Figure 6.16

```
> mod.lda <- lda(diabetes ~ pregnant + glucose + mass + pedigree +
+     age, data = pid)
> plot(predict(mod.glm, pid), col = as.numeric(diabetes) + 2, pch = as.numeric(predict(mod.lda,
+     pid)$class))
> abline(h = 0)
```

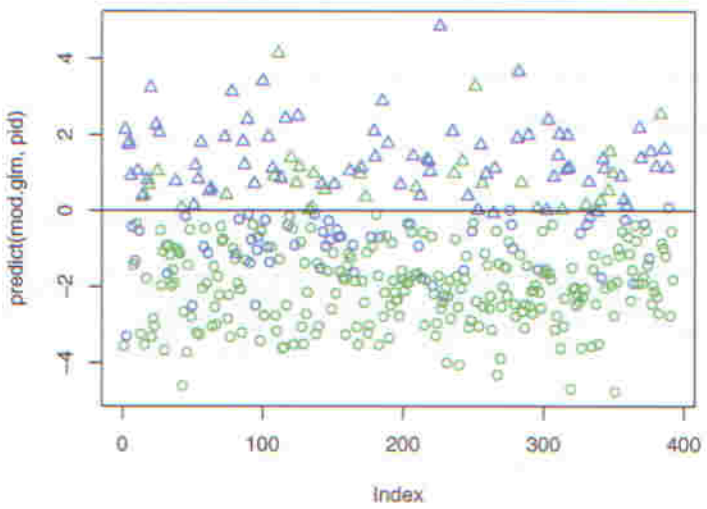


Figure 6.16: Reduced model: LDA (symbols) versus logistic regression (line); the color corresponds to the true classes

- *Prediction of the PimaIndianDiabetes data with logistic regression*

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

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

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

        5-fold cross-validation estimator of misclassification error

Misclassification error:  0.2194

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

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

        Bootstrap estimator of misclassification error
        with 10 bootstrap replications

Misclassification error:  0.2212
Standard deviation: 0.0065
```

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

```
> TAB <- table(pid$diabetes[-train], mypred(mod.glm, pid[-train,
+     ]))
> mkrlog <- 1 - sum(diag(TAB))/sum(TAB)
> mkrlog
[1] 0.2173913
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.217	0.217		

Part IV

Nonlinear methods

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

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

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

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

Chapter 7

Basis expansions

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

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

Some widely used examples are:

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

7.1 Interpolation with splines

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

Piecewise polynomials

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

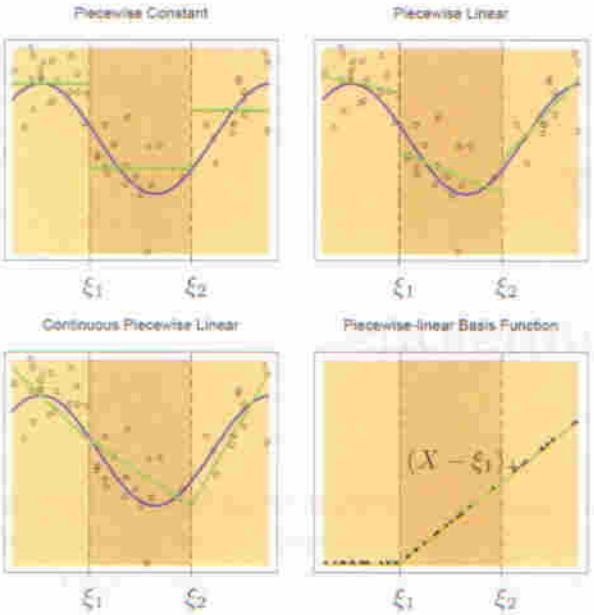


Figure 7.1: Piecewise constant and linear polynomials

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

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

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

Figure 7.1 upper right shows a piecewise fit by constant 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 constant functions but with the constraints of continuity at the knots. These constraints also lead to constraints on the parameters. For example, at the first know 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 left panel of Figure 7.1.

Splines

A spline of order M with knots ξ_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 = number of parameters (= df).

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

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

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

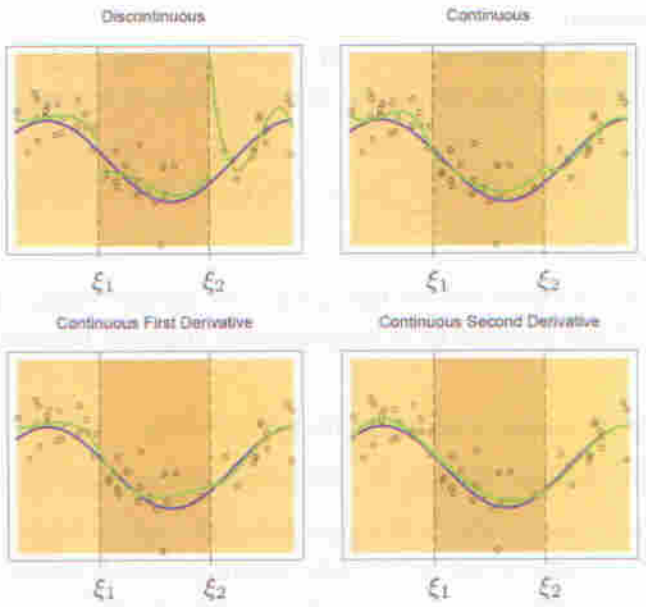


Figure 7.2: Piecewise cubic polynomials

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

The following parameters have to be chosen:

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

The function `bs(x,df=7)` generates a matrix of cubic spline bases functions which are evaluated at the n observations of `x`. Here, $7 - 3 = 4$ interior knots at the percentiles 25, 50, and 75% of `x` are determined. (A cubic spline with 4 knots is 8-dimensional. `bs()` omits the constant term since this can be included with other terms.) 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.

Natural cubic splines

Polynomials tend to be erratic near the lower and upper data range, which can result in poor approximations (Figure 8.6). This can be avoided by using natural cubic splines (Figure 8.8).

They have the additional constraint that the function has to be linear beyond the boundary knots. In this way we get back 4 degrees of freedom (two constraints each in both boundary regions), which can then be "invested" in a larger number of knots.

Section 8.1, page 85

7.2 Smoothing splines

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

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

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

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

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

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

where N_j is the n-dimensional set of basis functions that represent the natural splines. The criterion (7.1) thus reduces to

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

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

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

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

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

7.2.1 Choice of the degrees of freedom

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

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

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

The fit is linear in $\boldsymbol{y}$ and the finite operator $\boldsymbol{S}_\lambda$ 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. Let $\boldsymbol{B}_\xi$ denote a $n \times M$ matrix of M cubic spline basis functions, evaluated at the n training points x_i, with knot sequence ξ and $M << n$. Then

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

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

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

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

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

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

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

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

Section 8.2, page 92

Chapter 8

Basis expansions in R

8.1 Interpolation with splines in R

- *Fit of a linear model*

  Use of the dataset `wtloss` from `library(MASS)`, which consists of 2 variables with 52 observations each.

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

  The patients undergo a diet which lasts for 8 months.

- *Fit of a linear model*

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

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

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

Quadratic regression or, more generally, regression with polynomials allows a good approximation of the data (Figure 8.1). Most of the times, this only works in the range of the data and cannot be used for prediction (Figure 8.2):

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

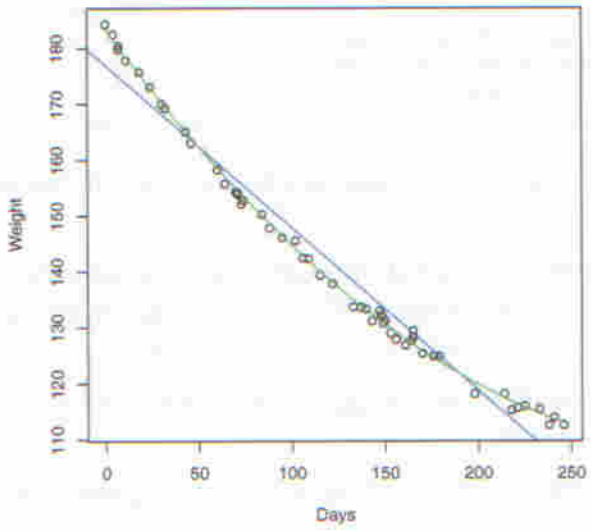


Figure 8.1: Graphical presentation of the fit of lm1 (blue) and lm2 (green)

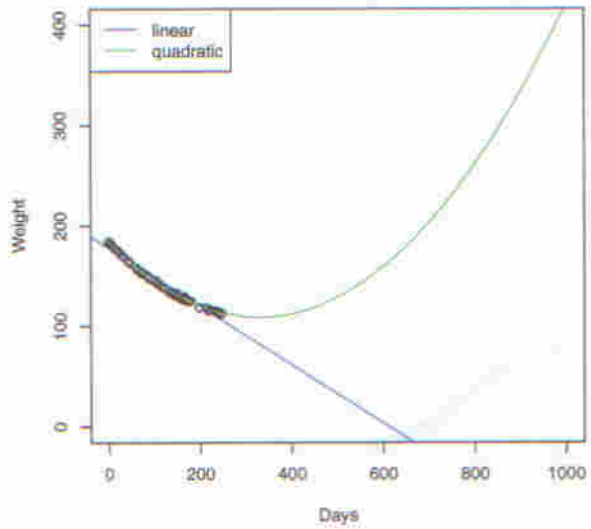


Figure 8.2: Prediction with lm1 and lm2

- *Nonlinear regression with* **nls()**

  The following nonlinear function computes a better fit, since weight loss is neither linear nor quadratic:

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

  where
  β_0 ... asymptotic final weight
  β_1 ... final weight loss
  θ ... time to divide the remaining weight into halves

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

162.2385 :  100  85 100
47.24374 :   86.38065  97.54973 131.36239
```

```
39.27375 :   81.67936 102.37229 141.32577
39.2447 :   81.37396 102.68386 141.91059
39.2447 :   81.37382 102.68411 141.91036
```

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

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

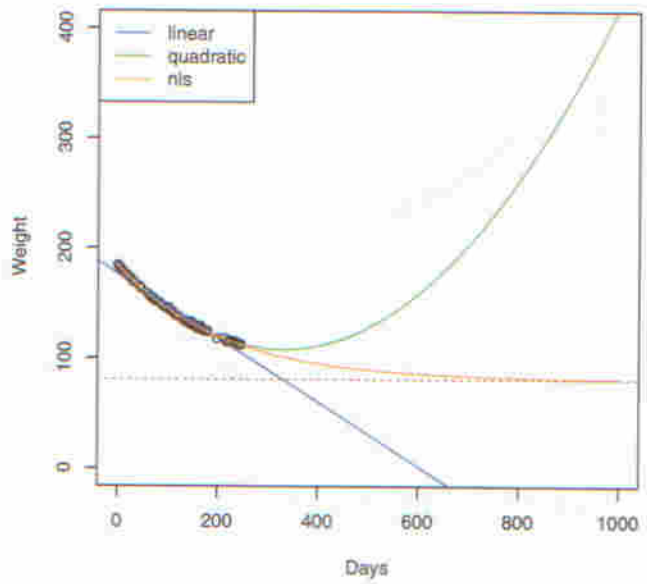


Figure 8.3: Prediction with lm1, lm2 and mod.nls (orange)

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

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

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

$value
[1] 39.2447

$counts
function gradient
     130       NA

$convergence
[1] 0

$message
NULL
```

Minimization of the sum of the absolute errors with **optim()**:

```
> funABSR <- function(p) {
+     sum(abs((wtloss$Weight - (p[1] + p[2] * 2^(-wtloss$Days/p[3])))))
+ }
> mod.opt2 <- optim(mod.start, funABSR)
> mod.opt2
$par
       b0        b1     theta
 83.65917 100.69083 136.76133

$value
[1] 33.97142

$counts
function gradient
     194       NA

$convergence
[1] 0

$message
NULL
```

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

```
> summary(mod.nls)

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

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

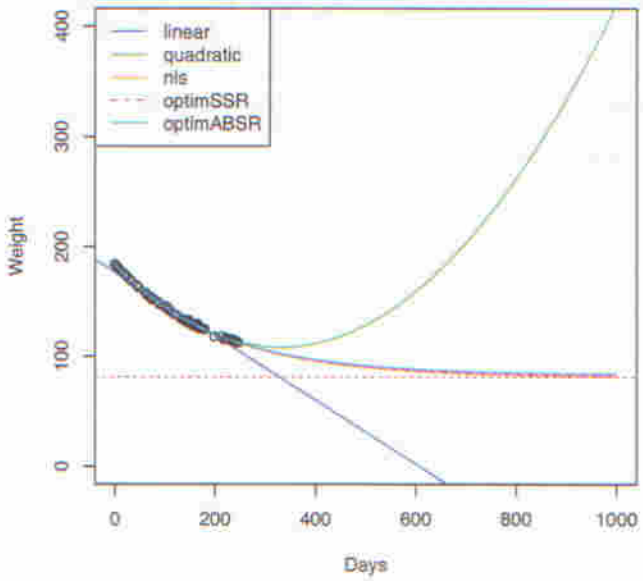


Figure 8.4: Prediction with lm1, lm2, mod.nls, mod.opt1, mod.opt2

```
Residual standard error: 0.8949 on 49 degrees of freedom

Number of iterations to convergence: 4
Achieved convergence tolerance: 1.271e-07
```

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

```
> summary(mod.opt1)
            Length Class  Mode
par         3      -none- numeric
value       1      -none- numeric
counts      2      -none- numeric
convergence 1      -none- numeric
message     0      -none- NULL
```

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

- *Model fit with B-splines*

  – Change of the order of the polynomials:

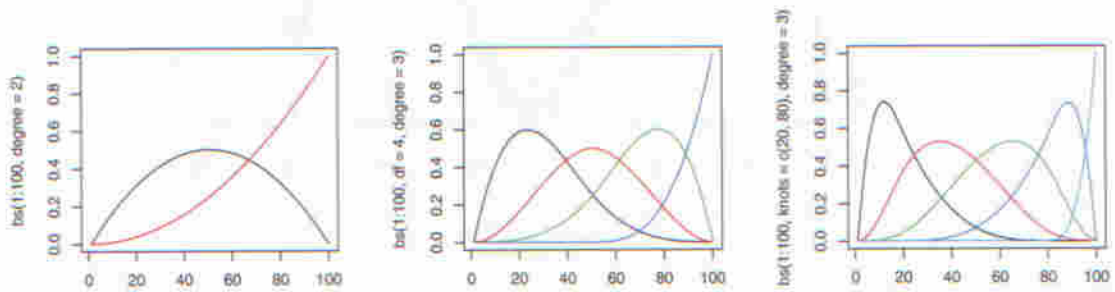
```
> library(splines)
> matplot(bs(1:100, degree = 2), type = "l", lty = 1)
```

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

```
> matplot(bs(1:100, df = 4, degree = 3), type = "l", lty = 1)
```

– Change of the knots:

```
> matplot(bs(1:100, knots = c(20, 80), degree = 3), type = "l",
+     lty = 1)
```



(a) Change of the order of the polynomials

(b) Change of the degrees of freedom

(c) Change of the number of knots

- *Estimation of the parameters*

```
> 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))
```

Generation of a sine function overlayed with noise (Figure 8.5).

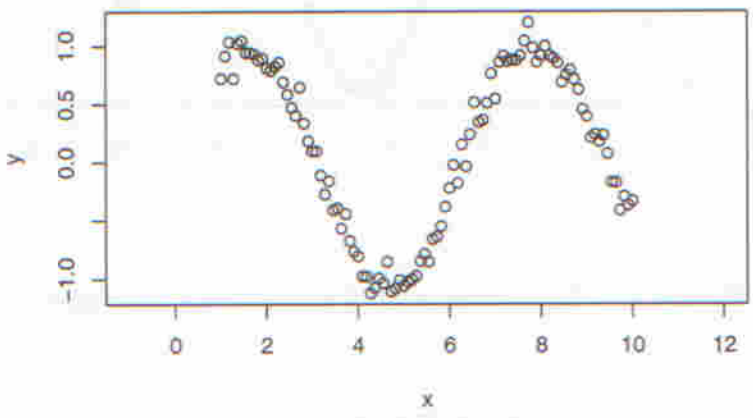


Figure 8.5: Sine function

Since we are looking for a linear relationship between the y variable and the spline bases, the function **lm()** can be used (Figure 8.6):

```
> lm1 = lm(y ~ bs(x, df = 4))
> lines(x1, predict.lm(lm1, list(x = x1)), col = "blue")
```

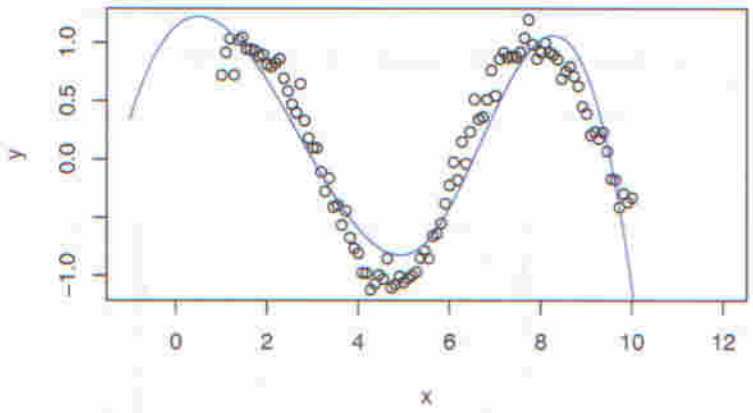


Figure 8.6: Prediction with 4 spline basis functions

An increase of the degrees of freedom provides a better fit (Figure 8.7):

```
> lm2 = lm(y ~ bs(x, df = 6))
> lines(x1, predict.lm(lm2, list(x = x1)), col = "green")
```

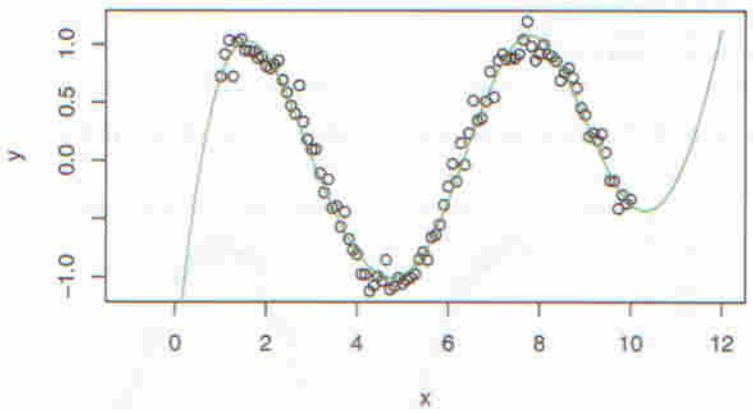


Figure 8.7: Prediction with 6 spline basis functions

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

```
> lm3 = lm(y ~ ns(x, df = 6))
> lines(x1, predict.lm(lm3, list(x = x1)), col = "orange")
```

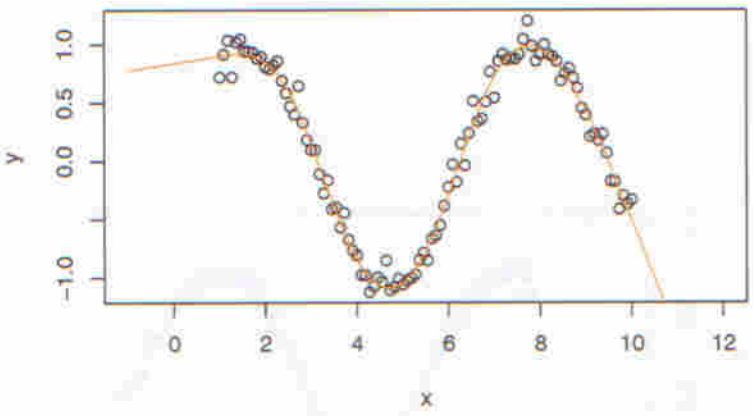


Figure 8.8: Prediction with cubic splines

8.2 Smoothing splines in R

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

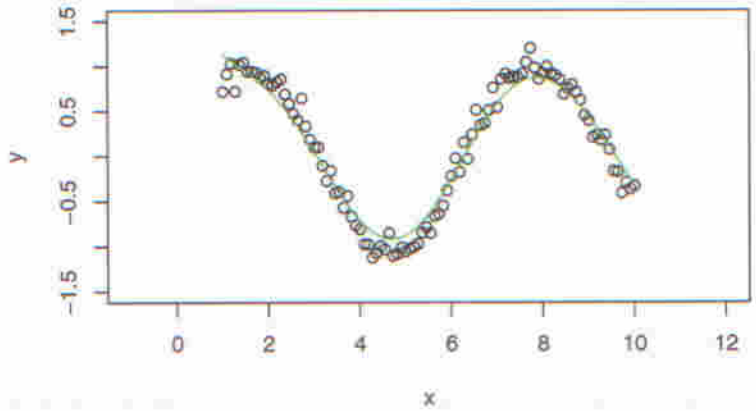


Figure 8.9: Fit with smoothing splines

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

- *Prediction at the boundaries*

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

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

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

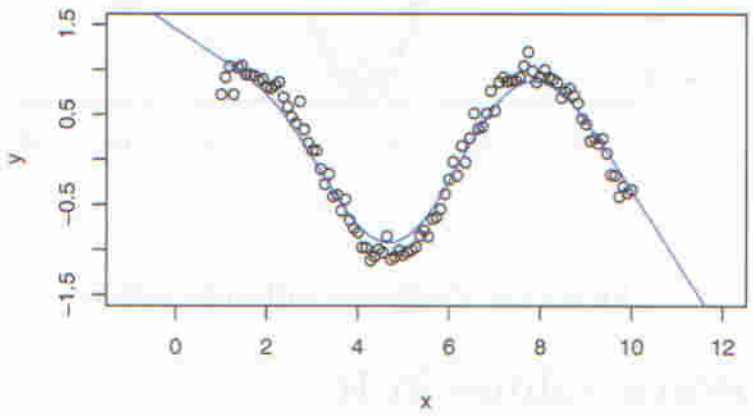


Figure 8.10: Prediction at the boundaries with smoothing splines

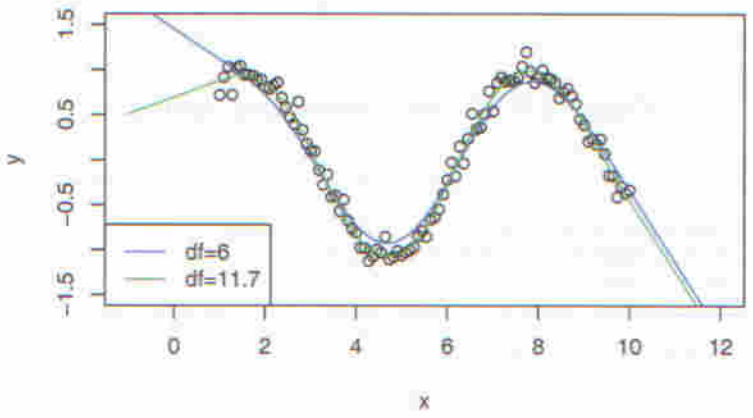


Figure 8.11: Choice of degrees of freedom

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

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

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

```
> plot(spnbmd ~ age, data = bone, col = ifelse(gender == "male",
+     "blue", "red2"), xlab = "Age", ylab = "Relative Change in Spinal BMD")
> bone.spline.male <- with(subset(bone, gender == "male"), smooth.spline(age,
+     spnbmd, df = 12))
> bone.spline.female <- with(subset(bone, gender == "female"),
+     smooth.spline(age, spnbmd, df = 12))
> lines(bone.spline.male, col = "blue")
> lines(bone.spline.female, col = "red2")
> legend(20, 0.2, legend = c("Male", "Female"), col = c("blue",
+     "red2"), lwd = 2)
```

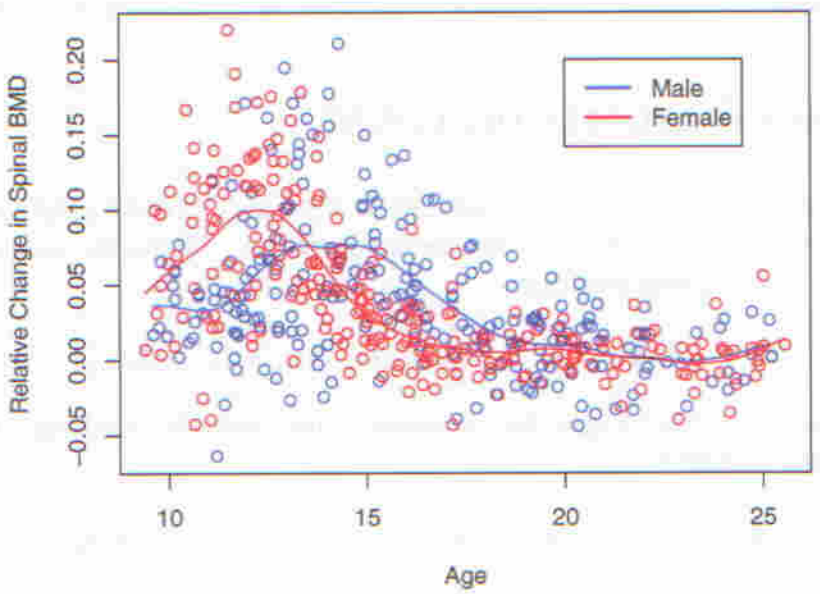


Figure 8.12: Bone density data

The regressor variable measures the relative change of the bone density. For each, the male and female measurements, a smoothing spline with a values of $\lambda \approx 0.0002$ was fitted. This corresponds to about 12 degrees of freedom.

Chapter 9

Generalized Additive Models (GAM)

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

9.1 General aspects on GAM

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

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

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

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

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

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

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

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

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

Generally we have for GAM:

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

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

Examples for classical link functions are:

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

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

9.2 Parameter estimation with GAM

The model has the form

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

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

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

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

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

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

Iterative algorithm for finding the solution:

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

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

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

Section 10, page 98

Chapter 10

Generalized additive models in R

- *Interpolation with GAM*

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

Family: gaussian
Link function: identity

Formula:
y ~ s(x)

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

Approximate significance of smooth terms:
      edf Est.rank     F p-value
s(x) 8.37        9 516.8  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

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

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

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

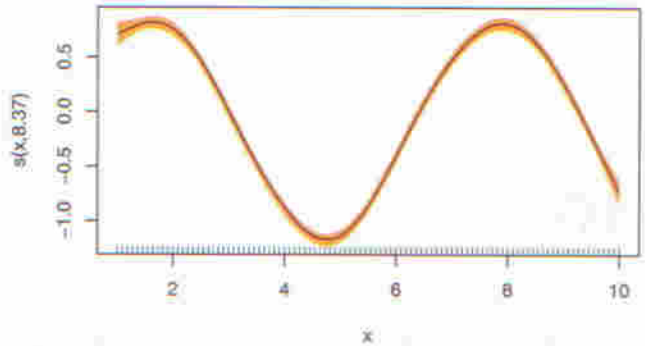


Figure 10.1: Fit with GAM

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

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

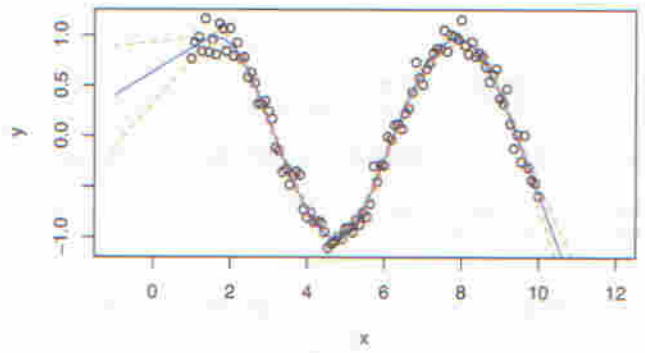


Figure 10.2: Prediction with GAM

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

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

Family: gaussian
Link function: identity

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

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.33000    0.02067   15.96   <2e-16 ***
---
Signif. codes:  0 ÃŸ***ÃŽ 0.001 ÃŸ**ÃŽ 0.01 ÃŸ*ÃŽ 0.05 ÃŸ.ÃŽ 0.1 ÃŸ ÃŽ 1
```

```
Approximate significance of smooth terms:
              edf Est.rank     F  p-value
s(pregnant) 3.043        7 2.209   0.0338 *
s(insulin)  2.095        5 2.207   0.0538 .
s(pressure) 1.000        1 0.548   0.4597
s(triceps)  1.000        1 1.890   0.1703
s(glucose)  7.322        9 6.708 1.11e-08 ***
s(age)      7.602        9 4.237 3.74e-05 ***
s(mass)     3.218        7 1.396   0.2070
s(pedigree) 1.269        3 1.043   0.3738
---
Signif. codes:  0 ãŸ***ãŹ 0.001 ãŸ**ãŹ 0.01 ãŸ*ãŹ 0.05 ãŸ.ãŹ 0.1 ãŸ ãŹ 1

R-sq.(adj) =  0.422   Deviance explained = 47.3%
GCV score = 0.14119   Scale est. = 0.12822   n = 300
```

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

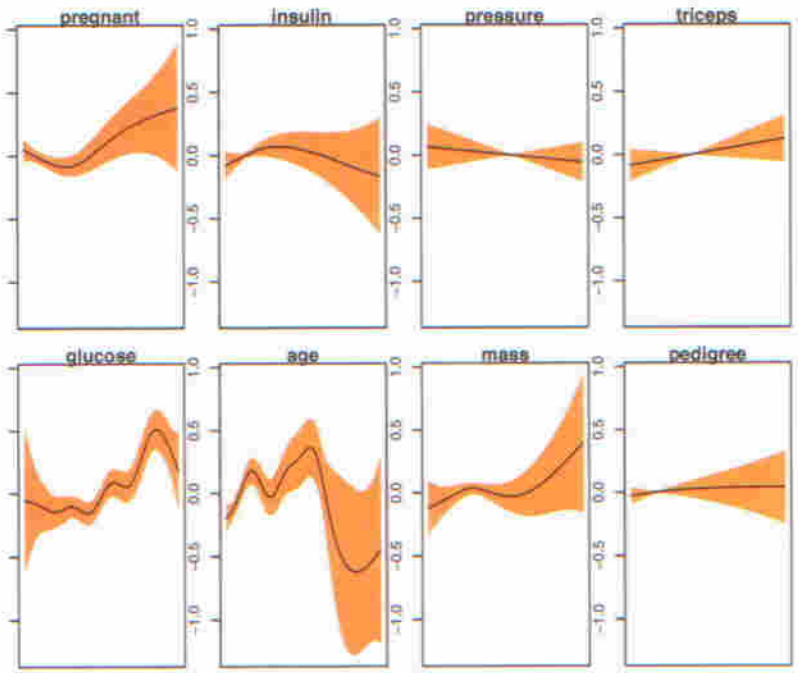


Figure 10.3: Fit of the different terms with additional 95% confidence region

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

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

    0  1
  0 61 10
  1 16 15
```

- *Misclassification rate for test set*

```
> mkrgam <- 1 - sum(diag(gam.TAB))/sum(gam.TAB)
> mkrgam
[1] 0.2826087
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.217	0.217	0.283	

Chapter 11

Tree-based methods

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

11.1 Regression trees

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

- Split variable
- Split point
- Form of the tree

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

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

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

Approximative solution:

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

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

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

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

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

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

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

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

and thus we define the cost complexity criterion

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

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 $\sum_m n_m Q_m(T)$. This is done as long as no node is left. It can be shown that this sequence of sub-trees must include T_α.

Section 12.1, page 105

11.2 Classification trees

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

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

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

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

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

1. Misclassification error: $\frac{1}{n_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$
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 number p of observations in the second class are (see Figure 11.1):

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

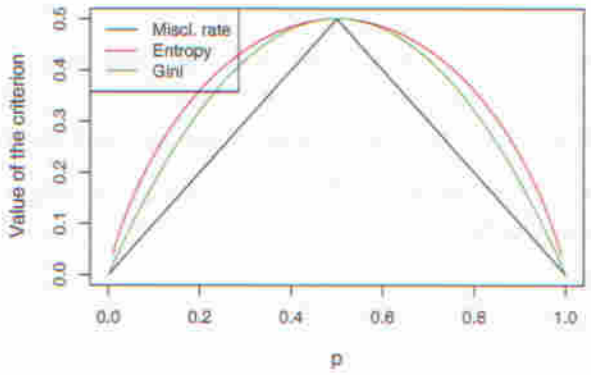


Figure 11.1: Impurity measures for two class classification

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

Section 12.2, page 107

Chapter 12

Tree based methods in R

12.1 Regression trees in R

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

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

  - `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=250 (1 observation deleted due to missingness)

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

   1) root 250 14557.340000 18.963200
     2) abdomen< 91.9 131  3834.820000 13.912210
       4) abdomen< 85.45 65  1027.789000 10.695380
         8) hip< 92.9 28   362.304300  9.264286
          16) ankle>=20.8 23   189.056500  8.143478
            32) height< 169.8625 3     5.846667  3.866667 *
            33) height>=169.8625 20   120.105500  8.785000
              66) neck>=36.75 2     0.320000  4.500000 *
              67) neck< 36.75 18    78.982780  9.261111
                134) height>=183.8325 2     0.500000  6.600000 *
                135) height< 183.8325 16    62.549380  9.593750

                ⋮

       15) abdomen>=112.3 11   220.900000 34.300000
         30) weight>=219.075 9    65.040000 32.666670
           60) hip< 111.85 3    26.000000 30.400000 *
           61) hip>=111.85 6    15.920000 33.800000 *
         31) weight< 219.075 2    23.805000 41.650000 *
```

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

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

- *Plot of the tree*

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

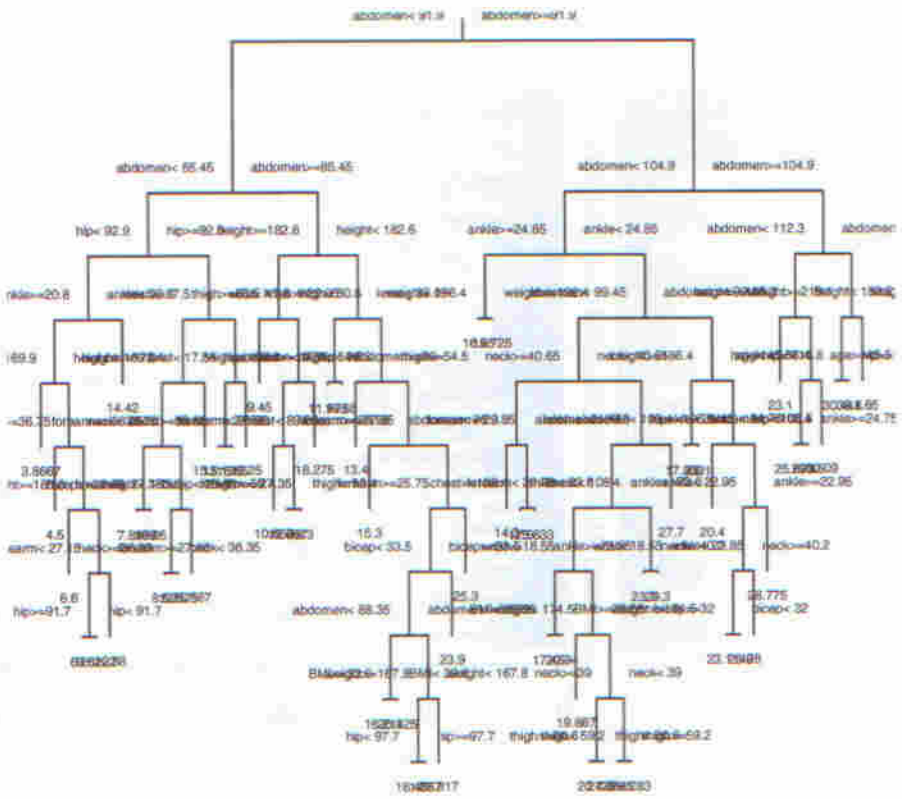


Figure 12.1: Regression tree of the body fat data

The procedure to prune the tree and the choice of the optimal complexity are shown in the next section (Section 12.2).

12.2 Classification trees in R

The data set `Spam` consists of 4601 observations and 58 variables. The goal is to divide the variable `Email` in "good" and "spam" emails with classification trees.

```
> load("Spam.RData")
> names(Spam)
 [1] "make"          "address"       "all"           "X3d"
 [5] "our"           "over"          "remove"        "internet"
 [9] "order"         "mail"          "receive"       "will"
[13] "people"        "report"        "addresses"     "free"
[17] "business"      "email"         "you"           "credit"
[21] "your"          "font"          "X000"          "money"
[25] "hp"            "hpl"           "george"        "X650"
[29] "lab"           "labs"          "telnet"        "X857"
[33] "data"          "X415"          "X85"           "technology"
[37] "X1999"         "parts"         "pm"            "direct"
[41] "cs"            "meeting"       "original"      "project"
[45] "re"            "edu"           "table"         "conference"
[49] "semicolon"     "parenthesis"   "bracket"       "exclamationMark"
[53] "dollarSign"    "hashSign"      "capitalAverage" "capitalLongest"
[57] "capitalTotal"  "class"
```

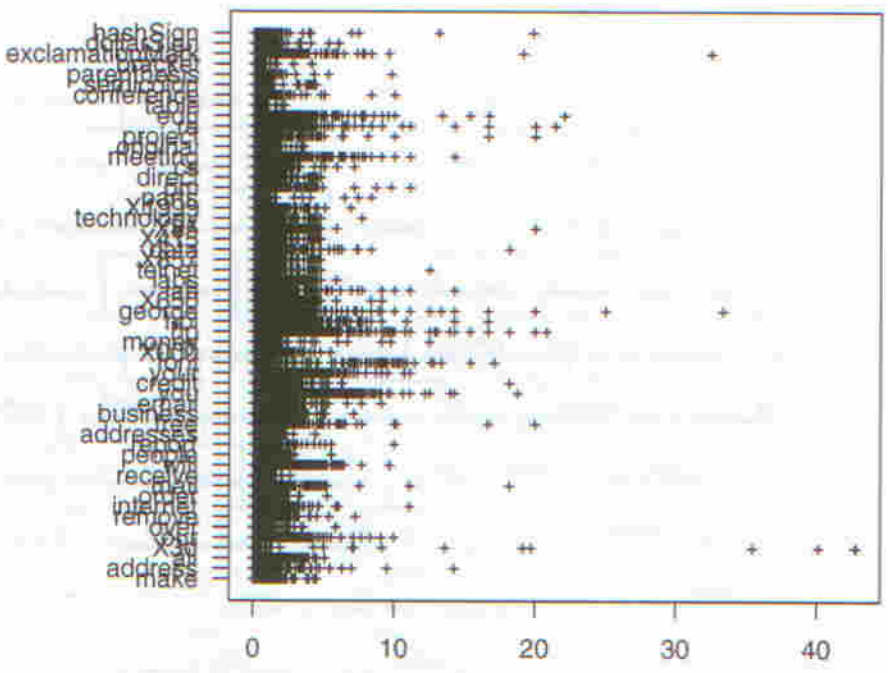


Figure 12.2: Spam data

```
> set.seed(100)
> train = sample(1:nrow(Spam), 3065)
> library(mvpart)
> tree1 = rpart(class ~ ., data = Spam, subset = train, method = "class",
+     cp = 0.001, xval = 20)
> plot(tree1)
> text(tree1)
```

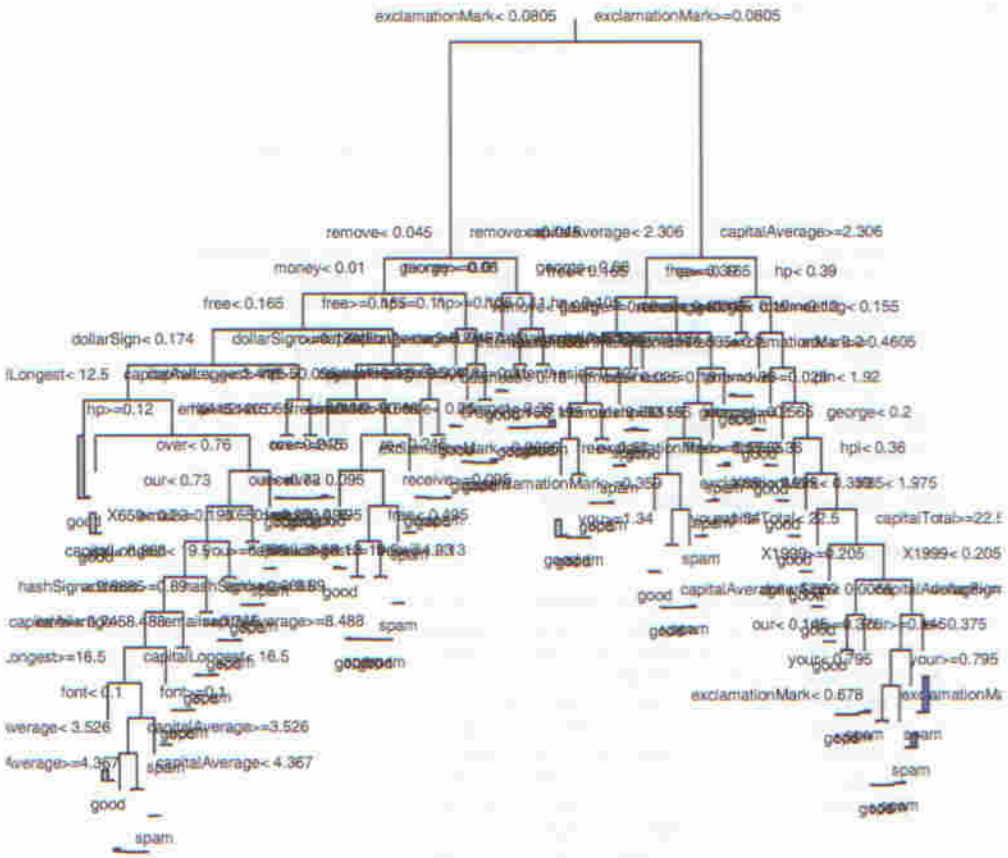


Figure 12.3: Classification tree of spam data

- *Prediction with the classification tree*

```
> tree1.pred <- predict(tree1, Spam[-train, ], type = "class")
> tree1.tab = table(Spam[-train, "class"], tree1.pred)
> tree1.tab
      tree1.pred
       good spam
  good  866   54
  spam   69  547
```

- *Misclassification rate for the test set*

```
> 1 - sum(diag(tree1.tab))/sum(tree1.tab)
[1] 0.08007812
```

- *Results of the cv*

```
> printcp(tree1)

Classification tree:
rpart(formula = class ~ ., data = Spam, subset = train, method = "class",
    cp = 0.001, xval = 20)

Variables actually used in tree construction:
 [1] address       business      capitalAverage capitalLongest
 [5] capitalTotal  dollarSign    edu            email
 [9] exclamationMark font        free           george
[13] hashSign      hp           hpl            internet
[17] mail          meeting      money          order
[21] our           over         parenthesis    pm
[25] re            receive      remove         will
[29] X1999         X415         X650           X85
[33] you           your

Root node error: 1197/3065 = 0.39054

n= 3065

          CP nsplit rel error  xerror     xstd
1  0.4745196      0  1.000000 1.00000 0.022565
2  0.0868839      1  0.525480 0.52882 0.018723
3  0.0593150      2  0.438596 0.44110 0.017465
4  0.0584795      3  0.379282 0.41855 0.017103
5  0.0225564      4  0.320802 0.35589 0.016000
6  0.0200501      5  0.298246 0.34336 0.015760
7  0.0192147      6  0.278195 0.32999 0.015497
8  0.0133668      7  0.258981 0.28237 0.014487
9  0.0100251      8  0.245614 0.26316 0.014045
10 0.0075188     11  0.215539 0.24478 0.013599
11 0.0066834     12  0.208020 0.24060 0.013495
12 0.0050125     13  0.201337 0.23141 0.013261
13 0.0045948     14  0.196324 0.23141 0.013261
14 0.0041771     16  0.187135 0.21888 0.012932
15 0.0037594     18  0.178780 0.21888 0.012932
16 0.0033417     20  0.171261 0.21470 0.012819
17 0.0025063     22  0.164578 0.20551 0.012566
18 0.0020886     23  0.162072 0.20635 0.012590
19 0.0018797     27  0.153718 0.20384 0.012520
20 0.0016708     32  0.143693 0.20468 0.012543
21 0.0012531     61  0.087719 0.20886 0.012659
22 0.0011139     63  0.085213 0.20718 0.012613
23 0.0010443     68  0.078530 0.20886 0.012659
24 0.0010000     72  0.074353 0.20718 0.012613
> plotcp(tree1, upper = "size")
```

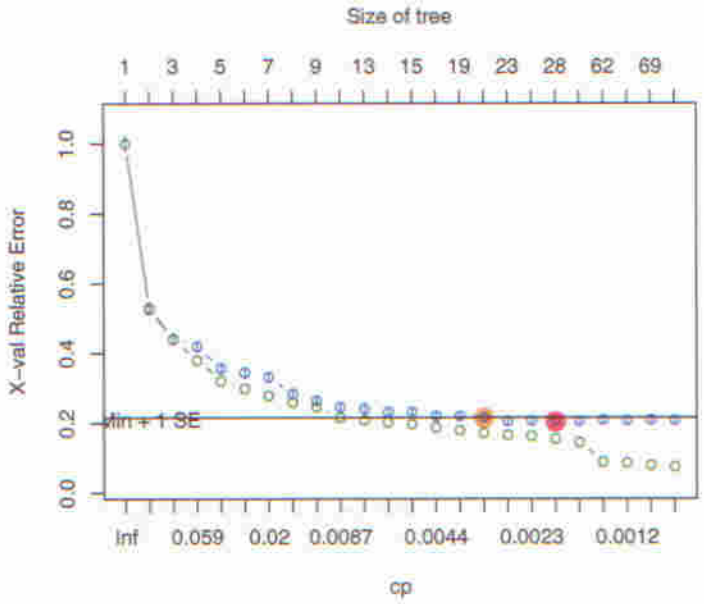


Figure 12.4: Cross-validation results of tree1: cv-error with standard error (blue) and error in test set (green)

Figure 12.4 shows the complexity of the tree and the cv estimate. The optimal, by cv computed value for α is 0.0035. The optimal size of the tree is approximately 20 nodes.

- *Pruning of the tree*

```
> tree2 <- prune(tree1, cp = 0.0035)
> plot(tree2)
> text(tree2)
```

Pruning of the tree with the optimal $\hat{\alpha}$.

- *Prediction with the new tree*

```
> tree2.pred = predict(tree2, Spam[-train, ], type = "class")
> tree2.tab = table(Spam[-train, "class"], tree2.pred)
> tree2.tab
      tree2.pred
       good spam
  good  868   52
  spam   80  536
```

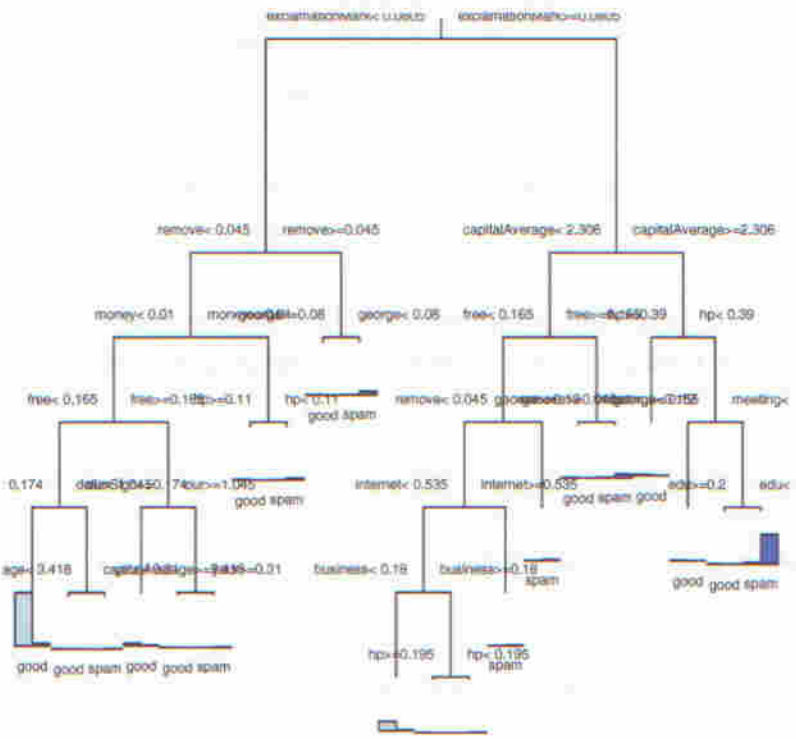


Figure 12.5: Result of the pruning of tree1

Chapter 13

Nearest neighbor methods

Our training data consists of n data pairs of the form $(\boldsymbol{x}_1, g_1), \ldots, (\boldsymbol{x}_n, g_n)$, where g_i is a class label taking values from the set $\{1, \ldots, K\}$. The goal is to partition the data pairs into K classes. For classification, K prototypes are chosen to represent the classes and then the remaining data are classified to the class of the closest prototype. "Closest" is usually defined by Euclidean distance $d_j = \|x_j - x_0\|$ in the feature space, after the data have been standardized to have overall mean 0 and variance 1. Because those methods are not model based they mostly do not help to understand the relationship between features and class membership. They can be very effective and represent irregular class boundaries when the prototypes are well positioned. The main task is to find out how many prototypes to use and where to place them.

13.1 K-means clustering

K-means clustering can be used to find clusters and cluster centers in data where the class membership is not known. The initial centers are R randomly chosen observations from the training data. The following algorithm reduces continuously the variance in the different clusters:

- for each center we identify the subset of training data points that is closer to it than any other center
- calculation of the new cluster centers

These two steps are iterated until convergence.

If we want to classify new data, and if the class membership to a training data set is available, the following K-means algorithm can be used:

- apply K-means for each class separately, using R prototypes (i.e. "R"-means clustering) per class
- assign a class label K to each of the $K \times R$ prototypes (cluster centers per class)
- classify a new feature to the class of the closest prototype

A disadvantage is the high misclassification rate for prototypes close to the boundaries. The reason for this disadvantage is that other classes have no influence on the positioning of the prototypes of the considered class. This drawback will be improved in Section 13.2 with a method called Learning Vector Quantization.

Section 14.1, page 119

13.2 Learning Vector Quantization (LVQ)

This algorithm is an online algorithm, where the observations are processed one at a time. The idea is that the training points attract prototypes of the correct class and push off others. Therefore, the prototypes should not lie near the boundaries.

The structure of the algorithm is as follows:

1. Select R initial prototypes for each class:

$$m_1(k), m_2(k), \ldots, m_R(k) \quad \text{for} \quad k = 1, \ldots, K$$

   for example by random selection of R training data points in each class.

2. Randomly select (with replacement) a point $\boldsymbol{x}_i$ of the training data. Let (j, k) be the indices of the prototype $m_j(k)$ next to $\boldsymbol{x}_i$.

   (a) If $g_i = k$ ($\boldsymbol{x}_i$ and prototype in the same class), move prototype to the training data point $\boldsymbol{x}_i$:

$$\boldsymbol{m}_j(k) \leftarrow \boldsymbol{m}_j(k) + \varepsilon(\boldsymbol{x}_i - \boldsymbol{m}_j(k))$$

   with learning rate ε.

   (b) If $g_i \neq k$ (different class), move prototype away from $\boldsymbol{x}_i$:

$$\boldsymbol{m}_j(k) \leftarrow \boldsymbol{m}_j(k) - \varepsilon(\boldsymbol{x}_i - \boldsymbol{m}_j(k))$$

3. Repeat step 2 and decrease ε in each iteration towards 0.

A disadvantage of LVQ is that this method is only defined by an algorithm and not by an optimization criterion. Therefore, it is difficult to understand the theoretical properties.

Figure 13.1 shows a LVQ solution with 5 prototypes, using the K-means solution as starting values.

Section 14.2, page 120

13.3 Expectation Maximization (EM) algorithm

This algorithm is used here for mixtures of normal distributions. Each group is represented by the density of the normal distribution, with own center and covariance matrix. The EM algorithm tries to estimate the centers and covariances, which results in a probability of group membership to class $k, k = 1, \ldots, K$ for each observation. The EM algorithm consists of two alternating steps, similar to K-means:

- **E-step:** Each observation receives a weight for each cluster membership, based on the likelihood function of the corresponding normal distribution.
- **M-step:** The weights of each observation are used to compute weighted mean and covariance for each cluster.

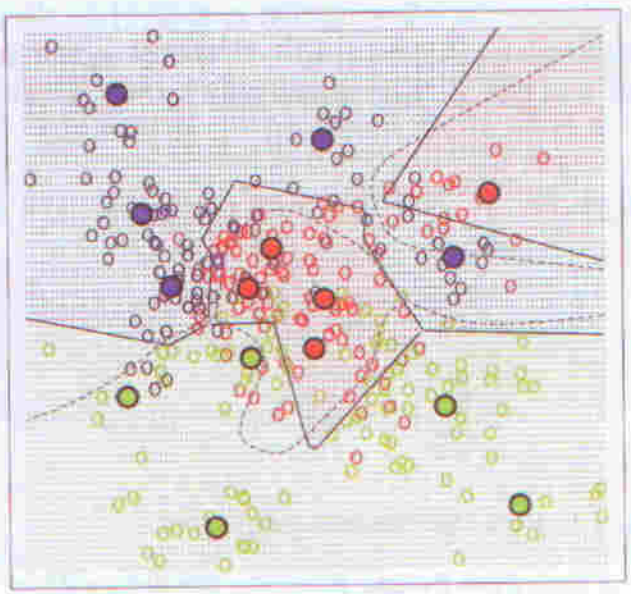


Figure 13.1: LVQ with 5 prototypes

This method results in *fuzzy* or *soft* clustering in contrast to *hard* clustering, because probabilities $\hat{p}_1(x), \ldots, \hat{p}_K(x)$ are used to assign each observation to each cluster. The "hard" assignment to the clusters can then be done by:

$$\hat{G}(x) = \underset{k}{\operatorname{argmax}}\, \hat{p}_k(x)$$

Example: Let us consider the simple case of a mixture of 2 univariate normal distributions. Thus we have the 2 populations

$$x_1 \sim N(\mu_1, \sigma_1^2) \quad \text{and} \quad x_2 \sim N(\mu_2, \sigma_2^2)$$

and their mixture

$$x = (1 - g) \cdot x_1 + g \cdot x_2 \quad \text{with} \quad g \in \{0, 1\} \tag{13.1}$$

and $P(g = 1) = p$ and $P(g = 0) = 1 - p$, respectively. The parameter p describes the a-priori probability of group membership. The density function of x is

$$f(x) = (1 - p) \cdot \phi(x; \mu_1, \sigma_1^2) + p \cdot \phi(x; \mu_2, \sigma_2^2).$$

The EM algorithm will now solve the task to estimate the unknown parameters $\boldsymbol{\theta} = (p, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2)$ at the basis of a sample with n observations, as well as the group memberships for given values $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$. For this purpose we use the maximum-likelihood estimation, and the log-likelihood function is given by

$$l(\boldsymbol{\theta}; \boldsymbol{x}) = \sum_{i=1}^{n} \log \left((1 - p) \cdot \phi_1(x_i) + p \cdot \phi_2(x_i) \right),$$

with the notation $\phi_1(x_i) = \phi(x_i, \mu_1, \sigma_1^2)$ and $\phi_2(x_i) = \phi(x_i, \mu_2, \sigma_2^2)$. A direct maximization of $l(\boldsymbol{\theta}; \boldsymbol{x})$ is numerically difficult because of the sum of terms within the logarithm.

A simpler procedure is the following: Let $g_i \in \{0,1\}$ be the group membership of each observation in the mixture model (13.1). Of course, g_i is unknown, but for the moment we assume that $\boldsymbol{g} = (g_1, \ldots, g_n)^\top$ are already known. Then the log-likelihood function is:

$$\begin{aligned} l_0(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{g}) &= \sum_{i=1}^{n} \{(1-g_i) \cdot \log\left((1-p) \cdot \phi_1(x_i)\right) + g_i \cdot \log\left(p \cdot \phi_2(x_i)\right)\} \\ &= \sum_{i=1}^{n} \{(1-g_i) \cdot \log \phi_1(x_i) + g_i \cdot \log \phi_2(x_i)\} \\ &\quad + \sum_{i=1}^{n} \{(1-g_i) \cdot \log(1-p) + g_i \cdot \log p\} \end{aligned}$$

Maximization of l_0 gives the classical estimators

$$\hat{\mu}_1 = \bar{x}_1 \text{ and } \hat{\sigma}_1^2 = s_1^2 \quad \text{for data with } g_i = 0,$$

$$\hat{\mu}_2 = \bar{x}_2 \text{ and } \hat{\sigma}_2^2 = s_2^2 \quad \text{for data with } g_i = 1.$$

So, if we would know the group memberships, the task would already be solved. However, since the g_i are unknown, we will plug in the (conditional) expected value of g_i into l_0. This expectation is given by (here for $g_i = 1$)

$$\gamma_i(\boldsymbol{\theta}) = \mathbb{E}(g_i | \boldsymbol{\theta}, \boldsymbol{x}) = \mathrm{P}(g_i = 1 | \boldsymbol{\theta}, \boldsymbol{x})$$

and can be estimated by

$$\hat{\gamma}_i = \frac{\hat{p} \cdot \hat{\phi}_2(x_i)}{(1-\hat{p}) \cdot \hat{\phi}_1(x_i) + \hat{p} \cdot \hat{\phi}_2(x_i)} = \frac{\hat{p} \cdot \phi(x_i, \hat{\mu}_2, \hat{\sigma}_2^2)}{(1-\hat{p}) \cdot \phi(x_i, \hat{\mu}_1, \hat{\sigma}_1^2) + \hat{p} \cdot \phi(x_i, \hat{\mu}_2, \hat{\sigma}_2^2)} \tag{13.2}$$

for $i = 1, \ldots, n$.

This estimation of the expected value forms the "Expectation" step in the **EM algorithm**, which now has the following structure:

1. Initialization of $\hat{\mu}_1$, $\hat{\sigma}_1^2$, $\hat{\mu}_2$, $\hat{\sigma}_2^2$, $\hat{p}$
   This can be done e.g. by:
   initialize $\hat{\mu}_1$, $\hat{\mu}_2$ by 2 randomly drawn data values x_i;
   initialize $\hat{\sigma}_1^2$, $\hat{\sigma}_2^2$ by s^2 which is computed from all data values x_i;
   set $\hat{p} = 0.5$.
2. Expectation step: compute $\hat{\gamma}_i$ according to (13.2) for $i = 1, \ldots, n$.
3. Maximization step: compute weighted means and variances:

$$\hat{\mu}_1 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i) x_i}{\sum_{i=1}^n (1-\hat{\gamma}_i)} \qquad \hat{\sigma}_1^2 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i)(x_i - \hat{\mu}_1)^2}{\sum_{i=1}^n (1-\hat{\gamma}_i)}$$

$$\hat{\mu}_2 = \frac{\sum_{i=1}^n \hat{\gamma}_i x_i}{\sum_{i=1}^n \hat{\gamma}_i} \qquad \hat{\sigma}_2^2 = \frac{\sum_{i=1}^n \hat{\gamma}_i (x_i - \hat{\mu}_2)^2}{\sum_{i=1}^n \hat{\gamma}_i}$$

$$\hat{p} = \sum_{i=1}^{n} \frac{\hat{\gamma}_i}{n}$$

4. Iterate steps 2 and 3 until convergence.

13.4 knn classification

This method uses the (training) data points that are in a local neighborhood of a point $\boldsymbol{x}_0$ for prediction. A new data point $\boldsymbol{x}_0$ thus will be assigned to the class which corresponds to the class of the majority of its k nearest neighbors (knn) $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$. Let y_i denote the group number of observation $\boldsymbol{x}_i$. The knn estimation of the group number for a new observation $\boldsymbol{x}_0$, the fit $\hat{y}(\boldsymbol{x}_0)$, is then

$$\hat{y}(\boldsymbol{x}_0) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x}_0)} y_i$$

where $N_k(\boldsymbol{x}_0)$ is the neighborhood of $\boldsymbol{x}_0$ which is defined by the k nearest neighbors of $\boldsymbol{x}_i$. This concept requires the definition of closeness, which is usually done by using a distance measure. As a distance measure again the Euclidean distance can be used.

Since the prediction at any point $\boldsymbol{x}_0$ corresponds to a "majority vote", the decision boundary is usually irregular and rough, and it strongly depends on k.

Section 14.3, page 121

13.5 Invariant metrics and tangent distance

Methods like knn save the training data together with the information of their group membership. Then they classify new observations by comparing them with the training data and assigning them to the same group as the group of the most similar training data points [compare Simard et al., 1996]. A determination of this similarity with the Euclidean distance can be inefficient because all possible changes of a group have to be available in the training data. For instance, for the recognition of hand written digits we would need all possible rotations, sizes, fonts, and line thicknesses of the digits in the training data (see Figure 13.2). This is usually impossible.

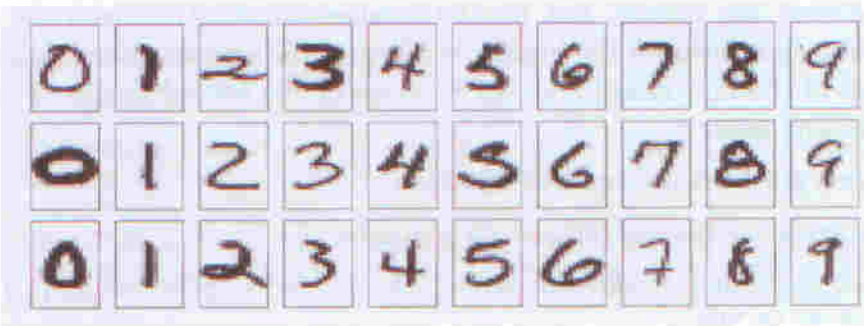


Figure 13.2: Examples of hand written digits

This problem can be avoided by using an "invariant metric". This is constructed such that the distance between prototype and observation is not influenced by irrelevant transformations. This allows a significant reduction of the number of required prototypes. Consider the example of hand written digits. A digit consists for example of $16 \times 16 = 256$ pixels which can be thought of the components of a 256-dimensional vector.

Figure 13.3 shows on top a few rotations of the digit "3". Of course, the human eye still is able to identify the digit correctly, but with respect to the pixel values the Euclidean distance in the 256-dimensional space can become quite large. Thus, this is not useful for a correct classification.

We can think of the "original" version of the digit "3" and all its rotations as a 1-dimensional curve in $\mathbb{R}^{256}$. This is also visualized by the green curve in Figure 13.3, and in a stylized version in Figure 13.4. The two points x_i and x_i' in Figure 13.4 can be thought of two different writings of the digit "3", and the corresponding green curves as their rotations. Such curves are also called *invariance manifolds*. Now we do not search for the Euclidean

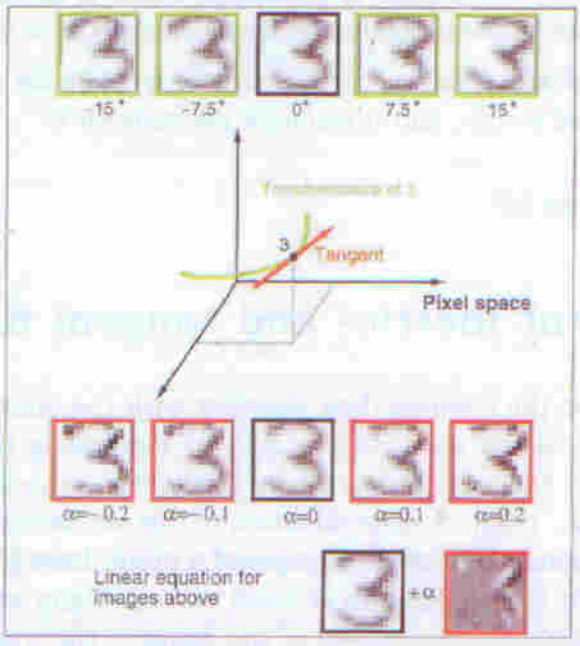


Figure 13.3: Rotations of "3"

distance between the images, but for the smallest distance between the curves. In other words, the distance between the images is taken as the smallest Euclidean distance between any rotation of the first image and any rotation of the second image. This distance is also called **invariant metric**.

Two problems arise when using this metric:

- The distance is usually difficult to compute.
- Large transformations are allowed which can lead to poor results (e.g. the transformation of "6" to "9").

The use of the "tangent distance" solves both problems. The "invariance manifold" is approximated by the tangent of the original image (see Figure 13.3). The tangent can be computed by estimating the direction vector from small rotations of the image. Thus, we compute the tangents for all images of the training data. For a new image also the tangent is computed, and classified to that image which has the smallest tangent distance (see Figure 13.4).

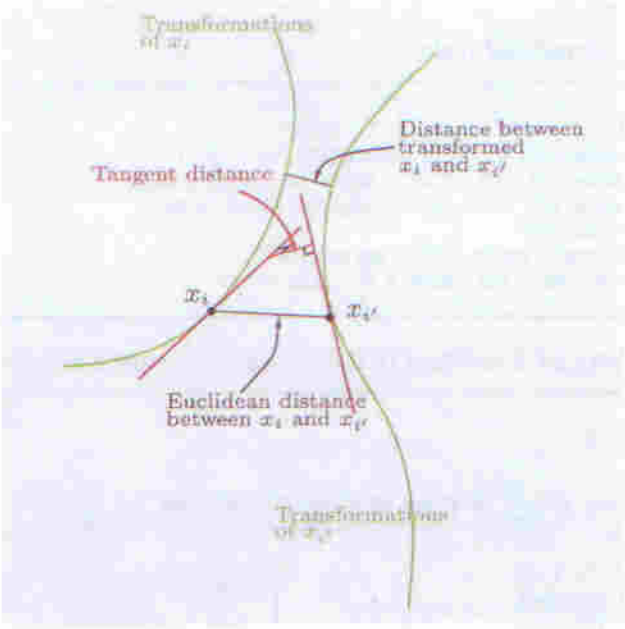


Figure 13.4: Computation of the tangent distance

Chapter 14

Nearest Neighbor methods in R

14.1 K-means clustering in R

- **kmeans()** *at simulated data*

```
> set.seed(1234)
> x1 <- cbind(rnorm(100, 0, 0.35), rnorm(100, 0, 0.35))
> x2 <- cbind(rnorm(100, 0.5, 0.25), rnorm(100, 0.5, 0.25))
> x3 <- cbind(rnorm(100, -0.5, 0.25), rnorm(100, 0, 0.25))
> x <- rbind(x1, x2, x3)
> col = c(rep(1, 100), rep(2, 100), rep(3, 100))
> plot(x, pch = 4, col = col, xlab = "", ylab = "")
```

Generate mixture of 3 normal distributions (Figure 14.1a).

```
> km = kmeans(x, 3)
> km
K-means clustering with 3 clusters of sizes 105, 86, 109

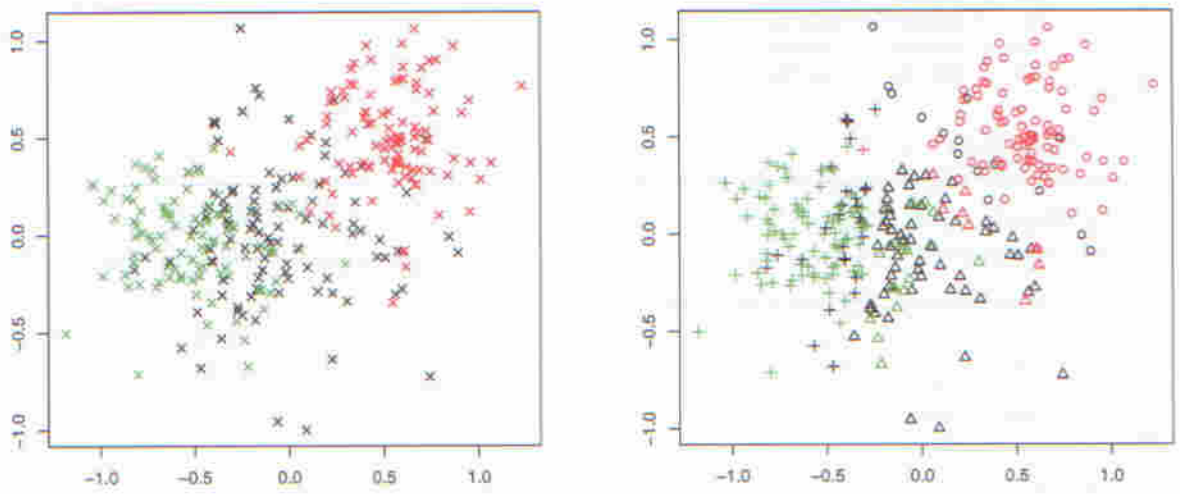
Cluster means:
        [,1]        [,2]
1  0.5259721  0.53282365
2  0.0402431 -0.11998836
3 -0.5582954  0.02010526

Clustering vector:
  [1] 3 2 2 3 2 2 2 2 2 3 2 3 2 2 2 2 2 3 3 1 2 2 1 2 3 3 2 2 2 3 1 2 2 2 3 3 3
 [38] 3 2 2 2 3 3 2 3 3 3 3 2 2 3 2 3 3 2 1 2 2 2 3 2 1 2 2 2 1 3 2 2 1 1 2 2 2
 [75] 1 2 3 1 2 2 2 2 3 2 1 1 1 2 2 3 2 2 2 1 1 2 3 2 2 2 1 1 1 1 1 1 1 1 1 1 1
[112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1 1 2
[149] 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1
[186] 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3
[223] 3 3 3 3 2 3 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 2 3 3 3 2 3 2 2 3 3 3 3 3 3
[260] 2 3 3 3 3 3 3 3 3 3 3 2 2 3 3 3 3 3 2 2 3 3 2 3 3 3 2 3 3 3 2 3 3 2 3 3 3
[297] 2 3 3 2

Within cluster sum of squares by cluster:
[1] 12.61988 12.01076 11.65476

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

Classification with 3 prototypes per class and plot of the results (Figure 14.1b).



(a) Mixture of 3 normal distributions

(b) Classification

Figure 14.1: Color = original classes, symbol = result of classification

- *Classification of the PimaIndiansDiabetes data with* **kmeans()**

```
> library(class)
> grp <- as.numeric(pid[, 9])
> x <- scale(pid[, 1:8])
> mod.kmeans <- kmeans(x, 2)
> table(mod.kmeans$cl, as.numeric(grp))

    1   2
1  61  89
2 201  41
```

14.2 Learning Vector Quantization (LVQ) in R

- *Initialisation of the prototypes*

```
> set.seed(9)
> train <- sample(1:nrow(pid), 300)
> test <- pid[-train, ]
> prototyp <- lvqinit(scale(pid[train, 1:8]), pid[train, 9], size = 5)
```

- *LVQ1 algorithm*

```
> mod.lvq1 <- lvq1(scale(pid[train, 1:8]), pid[train, 9], prototyp)
> mod.lvq1

$x
     pregnant    glucose    pressure    triceps    insulin       mass
123 -0.3336819 -0.9534282  0.54318434  0.1389217 -0.5485012  0.1924667
249  1.7278647 -0.1947944 -0.06696365  0.2704084  1.9016590  0.2722992
651 -0.4732706 -0.7597619 -0.92151492 -0.6310032 -0.5423977 -0.8728362
663  1.3183832  1.3162119  2.19723613  1.3724404  0.4808905  0.7124752
221 -1.0454227  1.9801461 -0.88538550  0.2108970  2.4655793  0.3290243
```

```
      pedigree        age
123 -0.6144417 -0.5403512
249 -0.8441459  0.2724831
651 -0.4469220 -0.6494000
663 -0.3982255  1.0792101
221  1.2591283 -0.8363823

$cl
[1] 0 0 0 1 1
```

- *Result for test data*

```
> lvqtest(mod.lvq1, scale(pid[-train, 1:8]))

 [1] 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
[39] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1
[77] 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1
Levels: 0 1
```

- *Other algorithms*

```
> mod.lvq2 <- lvq2(scale(pid[train, 1:8]), pid[train, 9], prototyp)
> lvqtest(mod.lvq2, scale(pid[-train, 1:8]))

 [1] 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 0
[39] 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1
[77] 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1
Levels: 0 1

> mod.lvq3 <- lvq3(scale(pid[train, 1:8]), pid[train, 9], prototyp)
> lvqtest(mod.lvq3, scale(pid[-train, 1:8]))

 [1] 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0
[39] 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1
[77] 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1
Levels: 0 1

> mod.lvq4 <- olvq1(scale(pid[train, 1:8]), pid[train, 9], prototyp)
> lvqtest(mod.lvq4, scale(pid[-train, 1:8]))

 [1] 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0 0 0
[39] 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 0 1
[77] 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1
Levels: 0 1
```

14.3 knn in R

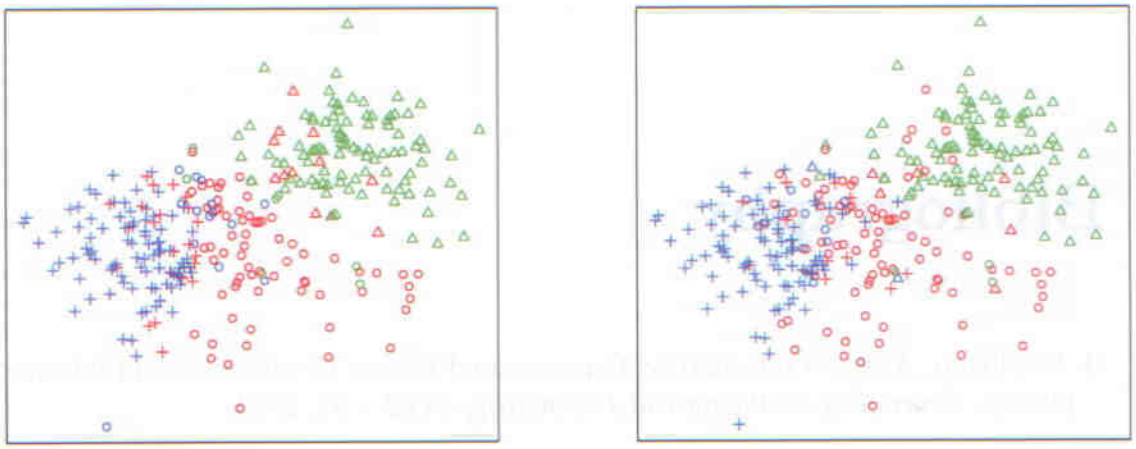
- Classification with **knn()** from `library(class)`

```
> mod.knn=knn(x.train, x.test,  k=15)
```

Classification using 15 Nearest Neighbors. The color represents the true group membership, the symbol shows the prediction (Figure 14.2a).

```
> mod.knn=knn(x.train, x.test,  k=2)
```

Classification using 15 Nearest Neighbors (Figure 14.2b).



(a) Classification with 15 nearest neighbors (b) Classification with 2 nearest neighbors

Figure 14.2: Classification with knn (Color: truth, symbol: prediction)

- Classification of the PimaIndianDiabetes data with knn

```
> set.seed(100)
> train = sample(1:nrow(pid), 300)
> mod.knn <- knn(pid[train, 1:8], pid[-train, 1:8], pid[train,
+     9], 2)
> mod.knn
 [1] 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0
[39] 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0
[77] 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1
Levels: 0 1
```

- Misclassification rate

```
> TAB1 = table(mod.knn, pid[-train, 9])
> mkrknn <- 1 - sum(diag(TAB1))/sum(TAB1)
> mkrknn
[1] 0.3260870
```

	INDR	LDA	QDA	RDA	GLM	GAM	knn
MKR	0.239	0.239	0.25	0.217	0.217	0.283	0.326

The comparison of the misclassification rates shows that RDA gives the best result. Regression with an indicator matrix, GAM and knn seem to perform rather poor for this data set. However, the evaluation was not done quite carefully because only one training and one test data set was used. More careful evaluation could give completely different results.

Bibliography

H. Bozdogan. Akaike's Information Criterion and Recent Developments in Information Complexity. *Journal of Mathematical Psychology*, 44:62 – 91, 2000.

K.P. Burnham and D.R. Anderson. Multimodel Inference: Understanding AIC and BIC in model selection. *Sociological Methods Research*, 33:261 – 304, 2004.

B. Fekedulegn, J.J. Colbert, R.R. Hicks, and Michael E. Schuckers Jr. Coping with multicollinearity: An example on application of principle component regression in dendroecology. Technical report, Unites States Department of Agriculture, 2002.

D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* A Bradford Book, The MIT Press, Cambridge, Massachusetts, 2001.

M. H. Hansen, J. Z. Huang, C. Kooperberg, C. Stone, and Y. K. Truong. Statistical modeling with spline functions: Methodology and theory. http://bear.fhcrc.org/ clk/monopdf/mono.html, January 2006.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer - Verlag, New York, 2001.

M.H. Kutner and C.J. Nachtsheim. *Applied Linear Regression Models.* McGraw-Hill / Irwin, Chicago, 2004.

P. Schönfeld. *Methoden der Ökonometrie, Band I.* Verlag Franz Vahlen GmbH, Berlin, 1969.

P. Simard, Y. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*, pages 239–27, 1996.