MSc in Bioinformatics: Statistical Data Mining

Wiesner Vos & Ludger Evers

31st August 2004

2

Contents

Chapter 1

Introduction

1.1 Background

> We are drowning in information and starving for knowledge.
>
> – Rutherford D. Roger

In the last two decades the amount of data available for (statistical) analysis has experienced an unpreceded growth and gave birth to the new field of *data mining*. Whether it was is in the context of biology and medicine (notably genomics) or in the context of marketing, much attention focused on data mining. Frawley *et al.* (1992) defined data mining as the "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data".

However much of the attention data mining received was due to the information technology hype of the last couple of years. Inferring information from data is not something new; for decades it has been and it still is the core domain of statistics. Or in more cynic words (Witten and Frank, 1999)

> What's the difference between machine learning and statistics? Cynics, looking wryly at the explosion of commercial interest (and hype) in this area equate data mining to statistics plus marketing.

But something has changed over the years. At the beginning of the 20th century statisticians mostly analysed systematically planned experiments to reply to a thoroughly formulated scientific question. These experiments lead to a small amount of high quality data. Under these controlled conditions one could often even derive an optimal way of collecting and analysing the data and (mathematically) prove this property.

At the beginning of the 21th century the situation is somewhat more chaotic. The scale of the datasets has changed. Due to the advance of information technology (computer based acquisition of data, cheap mass data storage solutions, etc.) the datasets have been growing in both dimensions: they do not only consist of more and more observations, they also contain more and more variables. Often these data are not directly sampled, but are merely a byproduct of other activities (credit card transactions, etc.). As such, they do not necessarily stem from a good experimental design and some variables might contain no interesting information. The data thus contains more and more "noise". And the questions to which statistical analysis has to respond got somehow more diffuse ("find something interesting").

The data analysed in genetics (especially microarray data) shares many of the above difficulties, although the data does result from systematic experiments. But a huge amount of genes is scanned in the hope of finding a "good" one.

1.2 "Data dredging"

Historically the term *data mining* however had a different meaning; it was used amongst statisticians to describe the improper analysis of data that leads to merely coincidental findings. This misuse of statistics is today referred to as *data dredging*.

When literally mining (for minerals) most of the material extracted from the pit is worthless. A good mining technology is characterised by efficiently extracting the small proportion of good material. A similar notion applies to data mining. We do *not* want to find patterns that are only characteristic to the dataset currently examined. We rather want to identify the structure behind the data. We want *generalisation*, i.e. we should be able to find the structure inferred from the dataset in future examples as well.

1.3 Tasks in statistical data mining

Data mining involves aspects of statistics, engineering and computer science. The tasks in statistical data mining can be roughly divided into two groups:

Supervised learning in which the examples are known to be grouped in advance and in which the objective is to infer how to classify future observations.[1]
Examples: Predicting from the genes whether a certain therapy will be effective or not. Classification of credit card transactions as fraudulent and non-fraudulent.

Unsupervised learning involves detecting previously unknown groups of "similar" cases in the data.
Example: Grouping predators with respect to their prey. Identifying genes with a similar biological function.

1.4 Notation

In data mining we usually analyse datasets with more than one variable (*multivariate* data). We generally use the letter "x" to designate variables. We denote then with x_{ij} the j-th measurement (variable) on the i-th individual (observation). We could imagine the data to be stored in a spreadsheet, where the columns correspond to the variables and the rows to the observations.

[1] Classification predicts a nominal variable whereas regression generally predicts a continuous variable, although there are regression techniques for nominal variables as well (see the chapter on logistic regression).

	Variable 1	Variable 2	...	Variable j	...	Variable p
Observation 1	x_{11}	x_{12}	...	x_{1j}	...	x_{1p}
Observation 2	x_{21}	x_{22}	...	x_{2j}	...	x_{2p}
...	...	...	...	...	...	...
Observation i	x_{i1}	x_{i2}	...	x_{ij}	...	x_{ip}
...	...	...	...	...	...	...
Observation n	x_{n1}	x_{n2}	...	x_{nj}	...	x_{np}

More mathematically speaking we can see our data as a *matrix* $\mathbf{X}$ with n rows and p columns, i.e.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1j} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2j} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \dots & x_{ij} & \dots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nj} & \dots & x_{np} \end{bmatrix}$$

Note that some statistical methods (especially clustering) can be carried out in the observation space as well as in the variable space. "In the variable space" means that the transposed data matrix is used, i.e. the data matrix is reflected: the observations are seen as variables and the variables are seen as observations. Microarray data is an example for it. We might want to group the samples (i.e. find clusters in the observation space), but we might as well want to group the genes (i.e. find clusters in the variable space). To that extent, the decision what is chosen to be "observation" and what is chosen to be "variable" can sometimes be arbitrary.

For microarray data often the transposed data matrix is used, i.e. the rows correspond to the genes (the variables) and the columns correspond to the samples (the observations). The reason for this is mainly that microarray data usually contains far more genes than samples and it is easier to visualise a table with more rows than columns.

Some statistical methods have been initially designed for *continuous* variables. Sometimes these methods can handle nominal variables if they are coded approriately. One possibility is *dummy coding*. It is nothing else than the bitwise representation of the discrete variable. A discrete variable with three levels (say "A", "B" and "C") for example is replaced by three "dummies". The first one is 1 if the discrete variable is "A", otherwise it is 0; the second one is 1 iff the discrete variable is "B"; and the third one is 1 iff the discrete variable is "C". The following table gives an example:

Discrete variable	Dummy A	Dummy B	Dummy C
A	1	0	0
B	0	1	0
C	0	0	1
B	0	1	0
C	0	0	1
...	...	...	...

Note that *Dummy C* can be computed from the first two by $1 - \text{Dummy A} - \text{Dummy B}$, the data matrix (including the intercept term) is thus not of full rank. This poses a problem for certain statistical methods. That's why the last column in the dummy coding is often omitted, the last level is then the *reference class*.

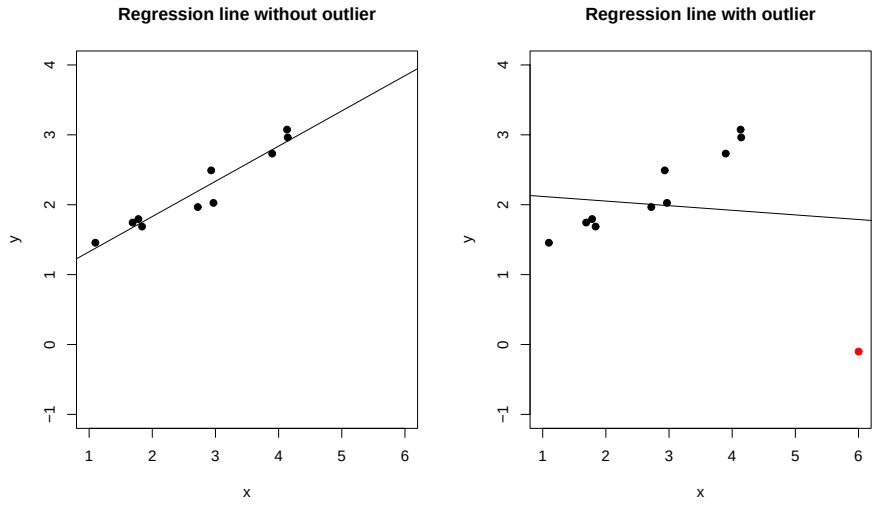


Figure 1.1: Least-squares regression line before and after adding an outlier (red dot)

1.5 Missing values

Sometimes some measurements for one observation or more are missing. Various techniques can be used to deal with this. Most of them use the information of all the other observations to find an appropriate value to fill in. The easiest, but clearly not the best approach would be just to use the mean of the other observations. These techniques however should not be applied blindly; in many cases data is missing for a certain reason and not just at random. Consider the case of a clinical study. The patients for whom the proposed treatment is not effective might decide to drop out of the study. Assuming that they were "normal" patients and filling in the average values would clearly bias the results of the study.

1.6 Outliers

Some datasets contain *outliers*. Outliers are "unusual" data values. Often outliers are just due to measurement errors or typing errors when entering the data into a database. In this case the only thing we have to do is removing them from the sample. Note that many statistical methods are very sensitive to outliers. Figure 1.1 gives an example. The left hand side shows a couple of data points with the corresponding regression line. Adding one outlier completely changes the regression line.

1.7 Datasets used in the lecture

This section presents datasets that are used more than one time. Datasets that are used only once are described when they are analysed.

1.7.1 The iris dataset

The famous iris dataset was collected by E. Anderson (Anderson, 1935) and analysed by R. A. Fisher (Fisher, 1936). It gives the measurements in centimetres of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of three species of iris. The species studied are *iris setosa*, *iris versicolor*, and *iris virginica*. In the chapters on classification we will however only use the measurements on the petals.

1.7.2 The European economies dataset

The European economies dataset contains information on the composition of the gross value added for 25 European countries (all EU countries apart from Ireland and Malta as well as Romania and Bulgaria). It gives for 14 industries the percentage of the gross value added by that industry. This dataset can be obtained online from Eurostat.

1.7.3 The quarter-circle dataset

This is a simulated dataset that contains a noisy quarter circle. The two covariates x_1 and x_2 were independently sampled from a uniform distribution on $[0, 1]$. The observations were classified into the first class if

$$\sqrt{x_{i1}^2 + x_{i2}^2} + \varepsilon_i < \sqrt{\frac{2}{\pi}},$$

where ε_i is a Gaussian error term with a standard deviation of 0.15.

1.7.4 The breast tumour dataset

This dataset van 't Veer *et al.* (2002) is from a study on primary breast tumours using DNA microarray data. The subjects were 78 sporadic lymph-node-negative female patients. The aim of the study was to search for a prognostic signature in their gene expression profiles. 44 patients remained free of disease after their initial diagnosis for an interval of at least 5 years (good prognosis group) and 34 patients developed distant metastases within 5 years (poor prognosis group). The aim is to discriminate between good and poor prognosis groups using the gene expression profiles of the patients. The original data includes expression profiles on $24,189$ genes. We will use two versions of this dataset. One will only contain the 30 "best" genes, the other one will contain the $4,000$ "best" genes. Both datasets are divided into a training set (40 patients) and a test set (36 patients).[2] [3]

WIESNER'S DATASETS

[2]The "best" genes were obtained by ordering the genes by the absolute value of the t-statistic for the comparison between the two groups.

[3]The original dataset contained some missing values that have been filled in using *KNNimpute*, i.e. the missing value was replaced by the mean of the nearest (complete) neighbours. Two observations have been deleted due to too many missing values.

Chapter 2

Exploratory data analysis

2.1 Visualising data

2.1.1 Visualising continuous data

Exploring the data visually is usually the first step of analysing data. It allows getting an idea of what the dataset is like and allows spotting outliers and other anomalies. A scatterplot matrix ("pairs plot"), in which every variable is plotted against every other variable, is usually a good way to visualise a dataset with a few continuous variables. Colours and the plotting symbol can be used to represent categorical variables. Figure 2.1 shows a pairs plot of the iris dataset. Sometimes it is useful not to plot the whole dataset but only a certain subset. One might for example want to look at one plot per level of a nominal variable. Instead of a nominal variable one can plot the data as well separately for different ranges of a continuous covariate. Figure 2.2 gives an example.

"Heatmaps" are another way of displaying multivariate datasets, exspecially microarray data. However they can be rather misleading, as the impression they generate depends highly on the ordering of the variables and observations. Unordered heatmaps (figure 2.3(a)) look rather chaotic and are thus difficult to interpret. Therefore one generally applies a clustering algorithm to the rows and columns and displays the variables and observations according to the dendrogram. This approach however yields the problem that the heatmap "inherits" the instability of clustering. Figure 2.3(b) and 2.3(c) show heatmaps for the same dataset however using two different clustering algorithms. Figure 2.3(b) suggests at first sight that there were two functional groups of genes, where as figure 2.3(c) suggests there were three.

Parallel coordinate plots are an alternative to heatmaps which do not suffer from this problem, however they might look less "fancy". In parallel plots each vertical line corresponds to a variable whereas the connecting lines correspond to the observations. Figure 2.4 shows a parallel coordinate plot of the European economies dataset.[1] Star plots (see figure 2.5) are another possibility to visualise such data.

[1] Sometimes, the variables are rescaled in order to use the full range of the plotting region — in our case however this does not seem very useful.

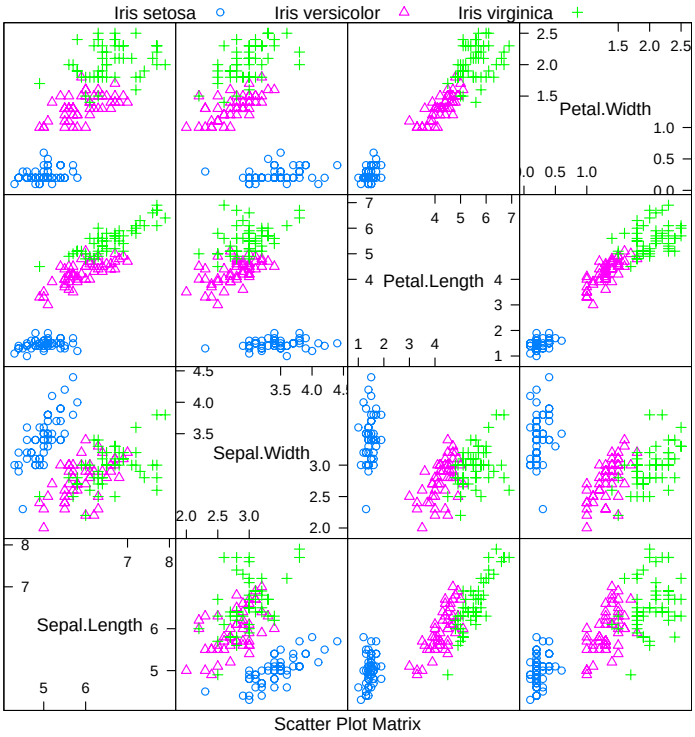


Figure 2.1: Pairs plot of the iris dataset

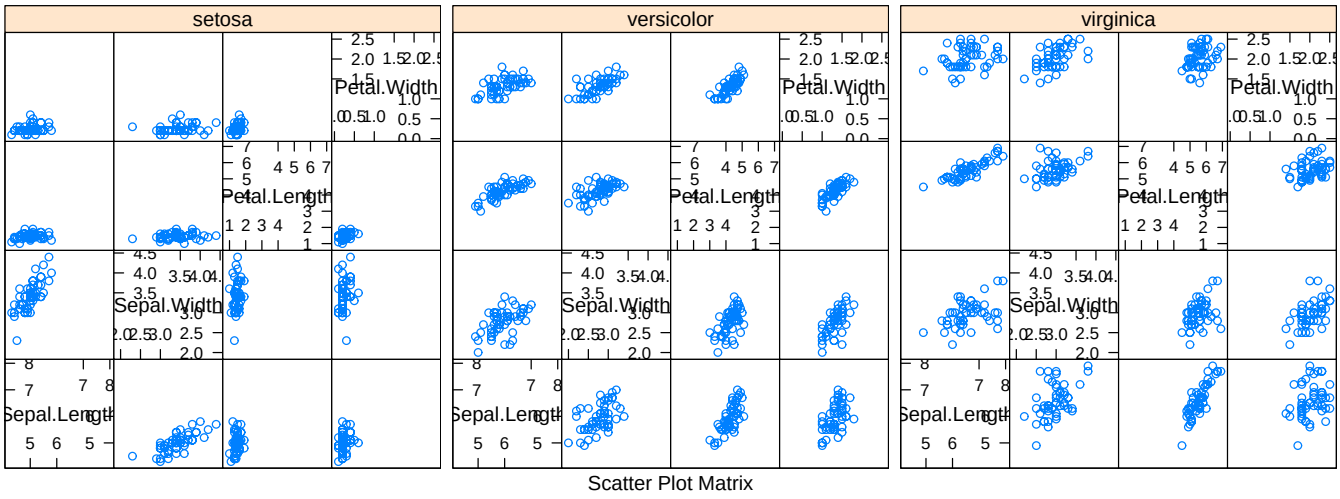
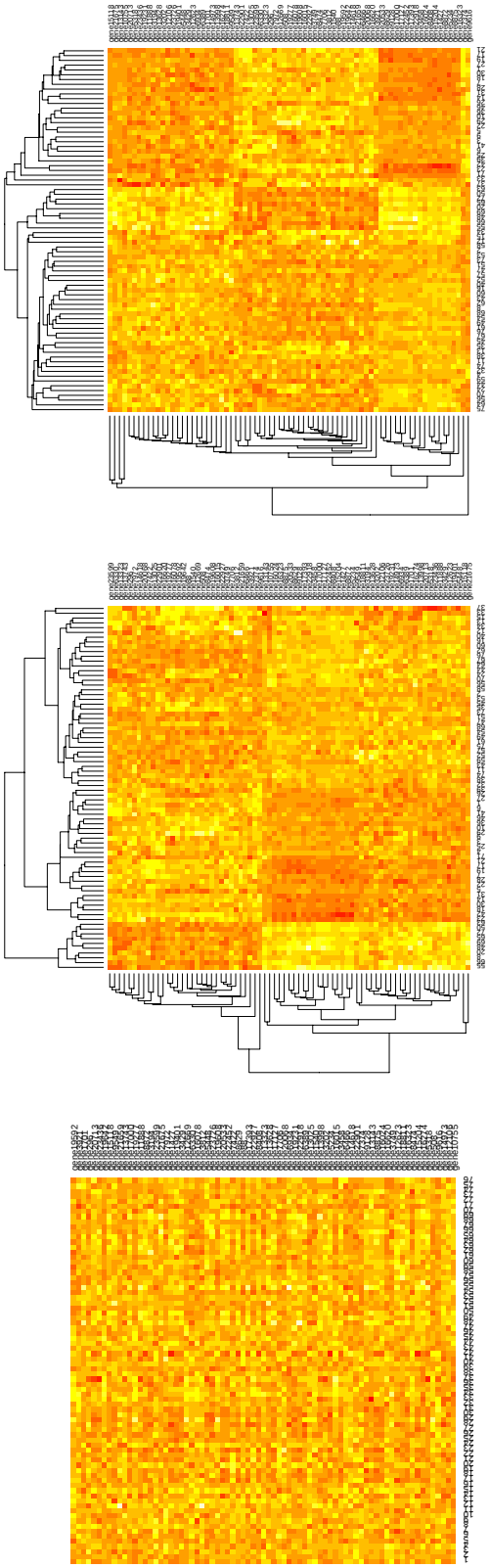


Figure 2.2: Conditional pairs plot of the iris dataset (by species)



(a) Unordered heatmap (b) Heatmap (using complete linkage clustering) (c) Heatmap (using average linkage clustering)

Figure 2.3: Heatmaps for the same observations and genes for the "best" 100 genes of the breast tumour data

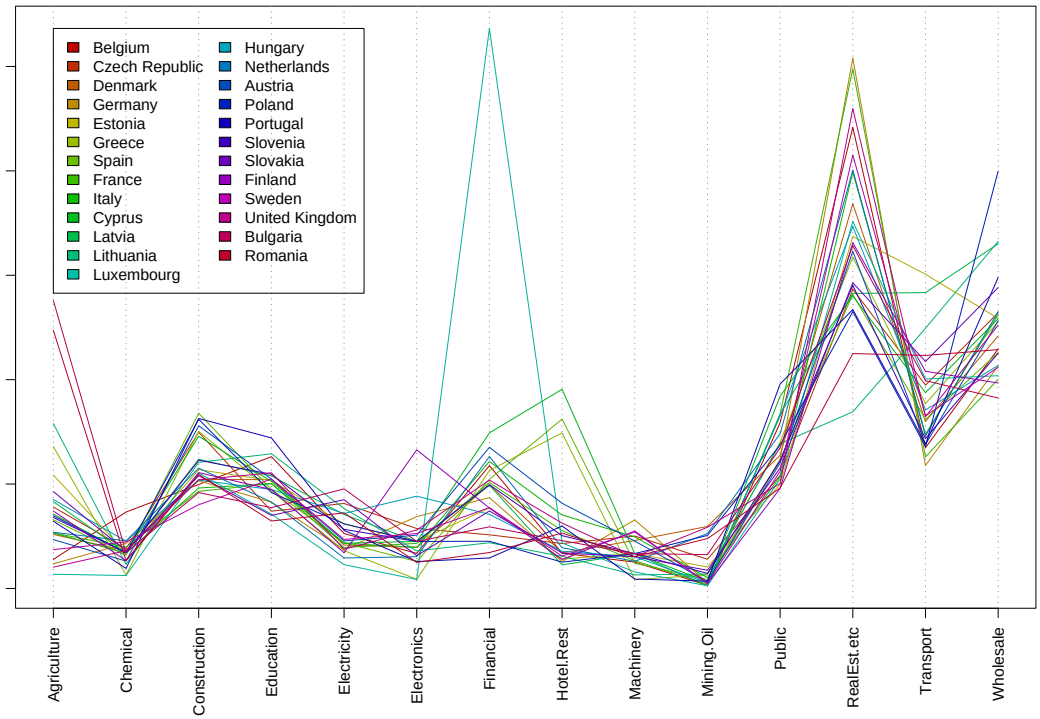


Figure 2.4: Parallel coordinate plot of the European economies dataset

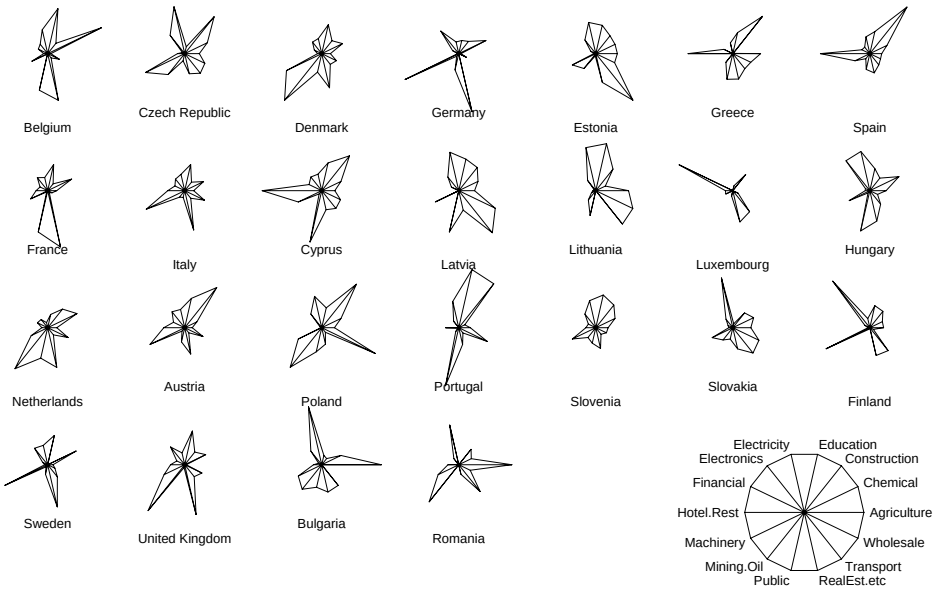


Figure 2.5: Star plot of the European economies dataset

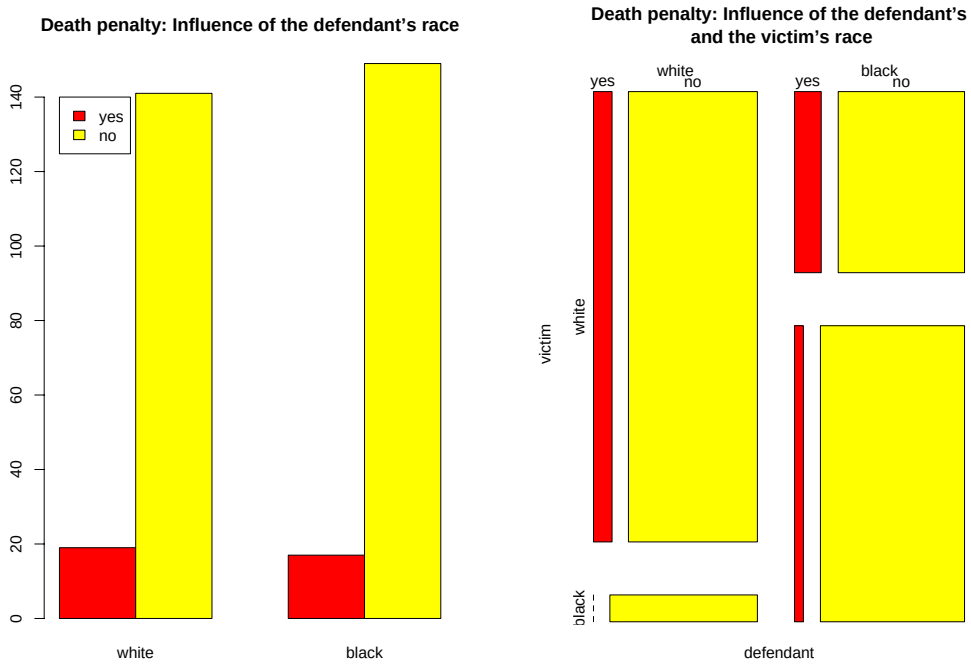


Figure 2.6: Bar plot and mosaic plot of the proportion of death penalty verdicts

2.1.2 Visualising nominal data

Nominal data is usually displayed using bar plots displaying the frequency of each class. The left hand side of figure 2.6 shows the frequency of death penalty verdicts for homicide in Florida in 1976 and 1977 — separately for white and black defendants (from Agresti, 1990). The plot suggests that there is no evidence for racism.

For displaying the joint distribution of several nominal variables mosaic plots are a good choice. In mosaic plots the surface of the rectangles is proportional to the probabilities. The right hand side of figure 2.6 gives an example. It shows the proportion of death penalty verdicts by the race of the defendant and by the race of the victim. This plot however leads to the opposite conclusion: If the victim is black the defendant is less likely to obtain the death penalty; black defendants generally seem to have a *higher* risk of being sentenced to death. So there *is* some evidence for racism amongst the jury members.

We couldn't see this in the bar plot, as we did not use the (important) information on the victim's race. The fact that whites are less likely to be sentenced to death was masked by fact that the defendant's and the victim's race are not independent.

2.1.3 Interactive tools

There are a couple of interactive tools to visualise high dimensional data like S-Plus's "brush and spin" are the open source tools XGobi and GGobi. They allow the user to change the axes and spin the data. Furthermore the data can be automatically spinned. Figure 2.7 shows a screenshot of XGobi.

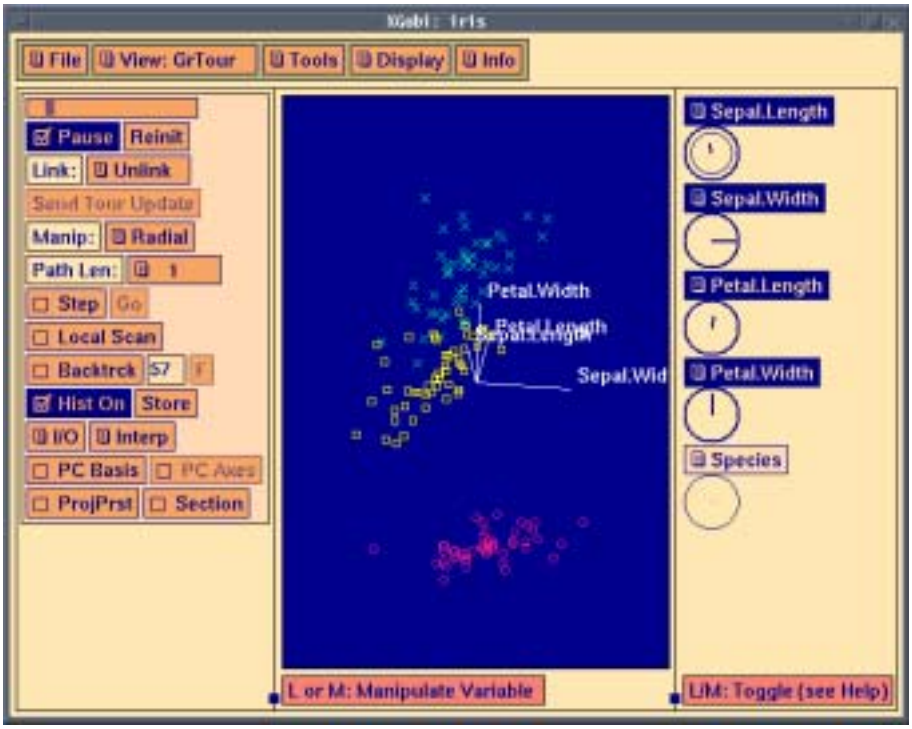


Figure 2.7: Screenshot of XGobi

2.2 Descriptive statistics

Much of the information contained in the data can be assessed by calculating certain summary numbers, known as *descriptive statistics*. For example, the sample mean, is a descriptive statistic that provides a measure of location; that is, a "central value" for a set of numbers. Another example is the variance that provides a measure of dispersion. In this course we encounter some descriptive statistics that measure location, variation and linear association. The formal definitions of these quantities follow.

Let $x_{11}, x_{21}, .., x_{N1}$ be the N measurements on the first variable. The *sample mean* of these measurements, denoted $\bar{x}_1$, is given by

$$\bar{x}_1 = \frac{1}{N}\sum_{i=1}^{N} x_{i1}$$

A measure of spread is provided by the *sample variance*, defined for N measurements on the first variable as

$$s_1^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i1} - \bar{x}_1)^2$$

In general, for p variables, we have

$$s_i^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)^2 \quad j = 1, 2, \ldots, p$$

The *sample covariance*

$$s_{ik}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \quad i, k = 1, 2, \ldots, p$$

measures the association between the ith and kth variables. Note that the covariance reduces to the sample variance when $i = k$. Moreover, $s_{ik} = s_{ki}$ for all i and k. The final descriptive statistic considered here is the *sample correlation coefficient* (sometimes referred to as *Pearson's correlation coefficient*). This measure of the linear association between two variables does not depend on the units of measurement. The sample correlation coefficient, for the ith and kth variables, is defined as

$$r_{ik} = \frac{s_{ik}}{\sqrt{s_i\sqrt{s_k}}} = \frac{\sum_{j=1} N(x_{ji} - \bar{x}_i)(x_{jk} - \bar{x}_k)}{\sqrt{\sum_{j=1} N(x_{ji} - \bar{x}_i)^2}\sqrt{\sum_{j=1} N(x_{jk} - \bar{x}_k)^2}}$$

for $i = 1, 2, \ldots, p; k = 1, 2, \ldots, p$. Covariance and correlation provide measures of *linear* association. Their values are less informative for other kinds of association. These quantities can also be very sensitive to *outliers*. Suspect observations must be accounted for by correcting obvious recording mistakes. An alternative is to make use of *robust* alternatives for the above descriptive statistics that are not affected by spurious observations. The descriptive statistics computed from n measurements

on p variables can also be organised into arrays.

$$\text{Sample mean vector} \quad \bar{\mathbf{x}} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_p \end{bmatrix}$$

$$\text{Sample covariance matrix} \quad \mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix}$$

$$\text{Sample correlation matrix} \quad \mathbf{R} = \begin{bmatrix} 1 & r_{12} & \cdots & r_{1p} \\ r_{21} & 1 & \cdots & r_{2p} \\ \vdots & \vdots & & \vdots \\ r_{p1} & r_{p2} & \cdots & 1 \end{bmatrix}$$

The above sample mean vector and sample covariance matrix can also expressed in matrix notation as follows:

$$\bar{\mathbf{x}} = \frac{1}{N}\mathbf{X}'\mathbf{1}$$

$$S = \frac{1}{N}\left(\mathbf{X}'\mathbf{X} - \frac{1}{N}\mathbf{X}'\mathbf{1}\mathbf{1}'\mathbf{X}\right) = \frac{1}{N}\mathbf{X}'\mathbf{X} - \bar{\mathbf{x}}\bar{\mathbf{x}}'$$

where $\mathbf{1}$ is a column vector of N ones.

2.3 Graphical Methods

2.3.1 Principal Component Analysis

Principal component analysis (PCA) is concerned with explaining the variance-covariance structure in data through a few *linear* combinations of the original variables. PCA is from a class of methods called projection methods. These methods choose one or more linear combinations of the original variables to maximize some measure of "interestingness". In the case of PCA this measure of "interestingness" is the variance. Its general objectives are (1) data reduction and (2) interpretation. PCA is particularly useful when one has a large, correlated set of variables and would like to reduce them to a manageable few. For example, in gene expression arrays we have many correlated genes being co-expressed (under or over), often in response to the same biological phenomenon. PCA can be used to identify small subsets of genes that behave very similarly, and account for large variances across samples. In the lectures we will consider an example of such an application.

In a full PCA, the set of response variables is re-expressed in terms of a set of an equal number of principal component variables. Whereas the response variables are intercorrelated, the principal component variables are not. Therefore, the variance-covariance structure of the principal components is described fully by their variances. The total variance – that is, the sum of their variances – is the same as the total variance of the original variances. For this reason, the variances of principal components are usually expressed as percentages of the total variation.

Deriving principal components

We describe principal components first for a set of real valued random variables $X_1, \ldots, X_p$. To obtain the first principal component, we seek a derived variable $Z_1 = a_{11}X_1 + a_{21}X_2 + \cdots + a_{p1}X_p$ such that var(Z_1) is maximized. Here the a_{i1} are real-valued coefficients. If left uncontrolled we can make the variance of Z_1 arbitrarily large. So we constrain $\mathbf{a}_1 = (a_{i1})$ to have unit Euclidean norm, $\mathbf{a}_1'\mathbf{a}_1 = \|\mathbf{a}_1\|^2 = 1$. This largest principal component attempts to capture the common variation in a set of variables via a single derived variable. The second principal component is then a derived variable, Z_2, uncorrelated with the first, with the largest variance, and so on.

Suppose that $\mathbf{S}$ is the $p \times p$ covariance matrix of the data. The eigendecomposition of $\mathbf{S}$ is given as

$$S = A\Lambda A'$$

where Λ is a diagonal matrix containing the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 0$ and A is a $p \times p$ orthogonal matrix whose columns consists of the p eigenvectors of S. The eigenvalues are real and non-negative since the covariance matrix $\mathbf{S}$ is symmetric non-negative definite. It is easy to show that the loading-vector for the first principal component $\mathbf{a}_1$ is the eigenvector of $\mathbf{S}$ corresponding to the largest eigenvalue, and so on. So the ith prinicpal component is then the ith linear combination picked by this procedure. It is customary to advocate using the eigendecomposition of the correlation matrix rather than the covariance matrix, which is the same procedure as rescaling the variables to unit variance before calculating the covariance matrix. Below we give a summary of the above results concerning principal components. We also show how we can establish which proportion of the total variance can be explained by the kth principal component. If most (say 80 to 90%) of the total population variance, for large p, can be explained by the first few components, then these components can "replace" the original p variables without much loss of information. For this reason, the variances of principal components are usually expressed as percentages of the total variation.

Note that another interpretation of principal components is that we seek new variables Z_i which are rotaions of the old variables to explain best the variation in the dataset. Suppose we consider the first k principal components. The subspace spanned by these new variables represent the "best" k-dimensional view of the data. It best approximates the originbal points in the sense of minimizing the sum of the squared distance from the points to their projections.

If $\mathbf{S} = (s_{ik})$ is the $p \times p$ sample covariance matrix with eigenvalue-eigenvector pairs $(\lambda_1, \mathbf{a}_1), (\lambda_2, \mathbf{a}_2), \ldots, (\lambda_p, \mathbf{a}_p)$, the ith sample principal component is given by

$$z_i = \mathbf{a}_1'\mathbf{x} = a_{1i}x_1 + a_{2i}x_2 + \cdots + a_{pi}x_p, \quad i = 1, 2, \ldots, p$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 0$ and $\mathbf{x}$ is any observation on the variables $X_1, X_2, \ldots, X_p$. Also

$$\text{Sample covariance}(z_i, z_k) = 0, \quad i \neq k$$

$$\text{Sample variance}(z_k) = \lambda_k, \quad k = 1, 2, \ldots, p$$

In addition,

$$\text{Total sample variance} = \sum_{i=1}^{p} s_{ii} = \lambda_1 + \ldots + \lambda_p,$$

The proportion of the total variance explained by the kth prinicipal component is

$$\frac{\lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_p} \quad k = 1, 2, \ldots, p$$

PCA usage

Note that the emphasis on variance with PCA creates some problems. The principal components depend on the units in which the variables are measured. PCA works best in situations where all variables are measurements of the same quantity, but at different times or locations. Unless we have good *a priori* reasons to regard the variances of the variables to be comparable, we would normally make them equal by rescaling all the variables to unit variance before calculating the covariance matrix. The fact that the extraction of principal components is based on variances, also means that the procedure is sensitive to the presence of outliers. This is particularly important for large noisy datasets such as gene expression data. It is not straight-forward to identify outliers in high-dimensional spaces, so it is desirable to use a method of extracting principal components that is less sensitive to outliers. This can be achieved by taking an eigendecomposition of a robustly estimated covariance or correlation matrix (Devlin *et al.*, 1981). An alternative approach is to to find a projection maximising a robust measure of variance in q dimensions as mentioned by Ripley (1996) on page 293. Although principal components are uncorrelated, their scatterplots sometimes reveal important structures in the data other than linear correlation. We will consider examples of such plots in the lectures and practicals. The first few principal components are often useful to reveal structure in the data. However, principal components corresponding to smaller eigenvalues can also be of interest. In the practical we will also consider "Gene Shaving" as an adaption of PCA that looks for small subsets of genes that behave similarly, and account for large variances across samples.

Example 1: Yeast Cell Cycle Data

Here we present examples of PCA applied to 3 real-life datasets. First we consider gene expression data on the cell cycle of yeast (Cho *et al.*, 1998). They identify a subset of genes that can be categorised into five different phases of the cell-cycle. Yeung and Ruzzo (2001) conduct a PCA on 384 of these genes. Changes in expression for the genes are measured over two cell cycles (17 time points). The data were normalised so that the expression values for each gene has mean zero and variance across the cell cycles. Figure 2.8 is a visualisation of the 384 genes used by Yeung and Ruzzo (2001) in the space of the first two prinicipal components. These two components account for 60% of the variation in the data (extracted from the correlation matrix). Genes corresponding to each of the five different phases of the cell cycle are plotted in different symbols. We see that despite some overlap the five classes are reasonably well separated.

Example 2: Morphometrics of Ordovician trilobites

Although distinctive and often common in Ordovician faunas, illaenid trilobites have relatively featureless, smooth exoskeletons; taxonomic discrimination within the group is therefore difficult. While a number of authors have attempted a qualitative approach to illaenid systematics. Bruton and Owen

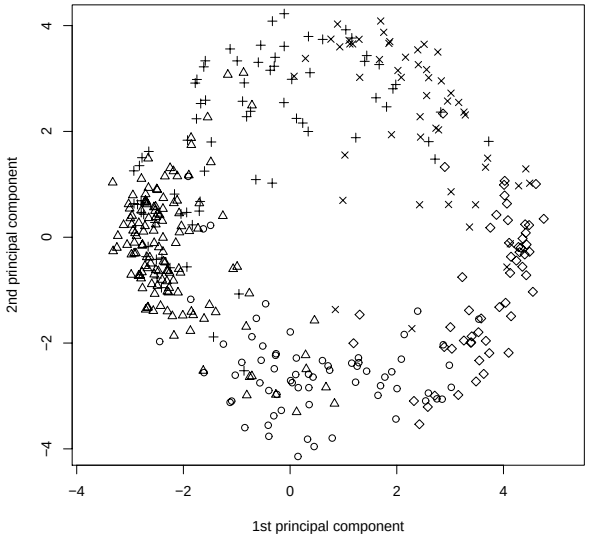


Figure 2.8: The yeast cell cycle data plotted in the subspace of the first two PCs

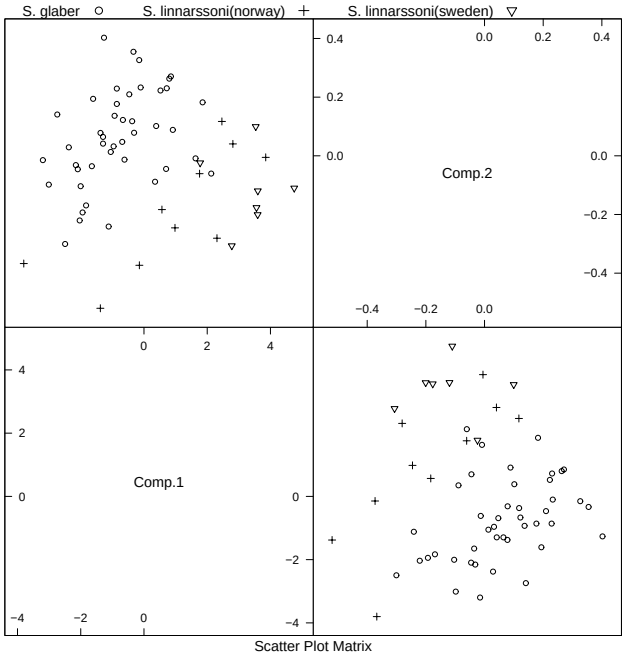


Figure 2.9: First two principal components for the paleontological data

(1988) described the Scandinavian Upper Ordovician members of the family in terms of measurements and statistics. This study seeks to discriminate between at least two different species of the illaenid trilobite *Stenopareia*. Four measurements were made on the cranidia of the Norwegian illaenid trilobites. The measurements were measured on *Stenopareia glaber* from the Ashgill of Norway together with *S. linnarssoni* from both Norway and Sweden,respectively. In figure 2.9 we plot the above data in the space of the first two principal components. We distinguish between the three groups of measurements in the plot. The two subspecies *Stenopareia glaber* and *Stenopareia linnarssoni* appear well-separated, while there also appear to be a reasonably clear separation between the Swedish and Norwegian observations for *Stenopareia*.

Example 3: Crabs Data

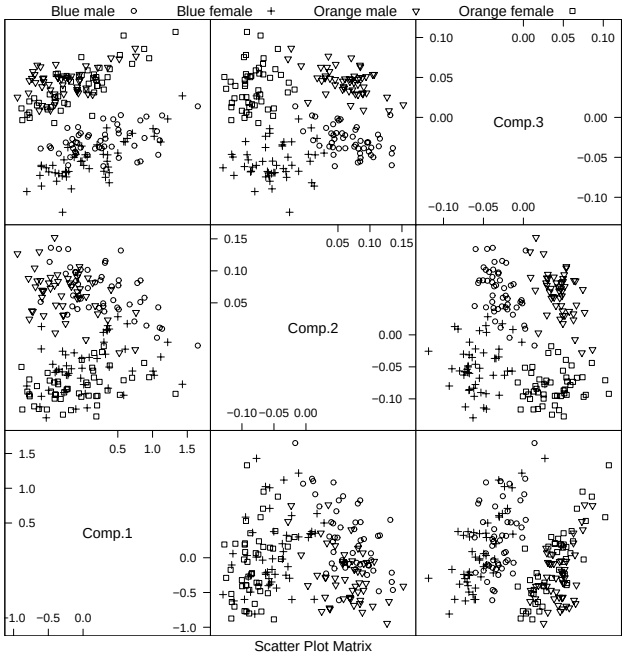


Figure 2.10: First three principal components for the crabs data

Campbell and Mahon (1974) studied rock crabs of the genus *leptograpsus*. One species, *L. variegatus*, had been split into two new species, previously grouped by colour form, orange and blue. Preserved specimens loose their colour, so it was hoped that morphological differences would enable museum material to be classified.

Data are available on 50 specimens of each sex of each species, collected on sight at Fremantle, Western Australia. Each specimen has measurements on the width of the frontal lip `FL`, the rear width `RW`, and length along the midline `CL` and the maximum width `CW` of the carapace, and the body depth `BD` in mm.

We plot the first three principal components in figure 2.10. What conclusions do you draw with regards to the structure revealed by each of the principal components? Do all the components in the plot reveal interesting structure? We discuss this in more detail in the lecture.

2.3.2 Biplots

Singular value decomposition of a real matrix: $\mathbf{X} = U\Lambda V'$, where Λ is a diagonal matrix with decreasing non-negative entries, U is a $n \times p$ matrix with orthonormal columns, and V is a $p \times p$ orthogonal matrix. The principal components are given by the columns of $\mathbf{X}V$.

Let $\tilde{\mathbf{X}} = U\Lambda_q V'$ be the matrix formed by setting all but the first q ($q \leq p$)singular values (diagonal elements of Λ) to zero. Then $\tilde{\mathbf{X}}$ is the best possible rank-q approximation to $\mathbf{X}$ in a least squares sense (i.e. the q-dimensional subspace has the smallest sum of squared distances between the original points and their projections.)

Consider n arbitrary cases (or items), each consiting of measurements on p variables summarised in a centered data matrix $\mathbf{X} : n \times p$ (the data matrix can be centered by subtracting the column means from each column). These cases have a geometrical representation in a p-dimensional space $\mathbb{R}^p$. The biplot (Gabriel, 1971) is a method for representing both the cases and variables. The biplot represents $\mathbf{X}$ by two sets of vectors of dimensions n and p producing a rank-2 approximation to $\mathbf{X}$. The best rank-q ($q < p$) approximation in a least squares sense can be obtained from the singular value decomposition (SVD) of $\mathbf{X}$ (see above). The best fitting q-dimensional subspace (in a least squares sense) is spanned by the columns of the matrix consiting of the first q columns of the matrix V in the SVD, say V_q. They define a natural set of orthogonal coordinate axes for the q-dimensional subspace. Therefore, $\mathbf{X}V_q$ are the linear projection of $\mathbf{X}$ into the "best"q-dimensional subspace.

So we can obtain the best 2-dimensional view of the data by setting $\lambda_3, \ldots, \lambda_p$ to zero in the SVD of $\mathbf{X}$, i.e.

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2\end{bmatrix} \begin{bmatrix}\lambda_1 & 0 \\ 0 & \lambda_2\end{bmatrix} \begin{bmatrix}\mathbf{v}_1' \\ \mathbf{v}_2'\end{bmatrix} = GH'$$

where the diagonal scaling factors can be absorbed into G or H in various ways. The biplot then consists of plotting the $n + p$ 2-dimensional vectors which form the rows of G and H. The rows of G specify the coordinates of the observations (items) in the 2-dimensional subspace, while the rows of H relate to the variables and are typically represented as arrows. The inner products between the rows of G represent the covariances between the variables. Therfore, the lengths of the arrows represent the standard deviations of the variables. In figure 2.11 we present an example for the trilobite data from example 2.

Gower and Hand (1996) devote a whole book to biplots and argue that these axes should just be considered as scaffolding, which are removed for display purposes. They recommend that representations of the original axes, representing the variables, in the q-dimensional subspace. These representations are referred to as biplot axes. These axes can be provided with suitable scales.

2.3.3 Independent Component Analysis

Independent component analysis (ICA) is becoming a very popular visualisation tool for uncovering structure in multivariate datasets. It is a method of decomposing a multi-dimensional dataset into a set of *statistically independent* non-gaussian variables, and is strongly connected to the technique of projection pursuit in the statistical literature (Friedman and Tukey, 1974). ICA assumes that the

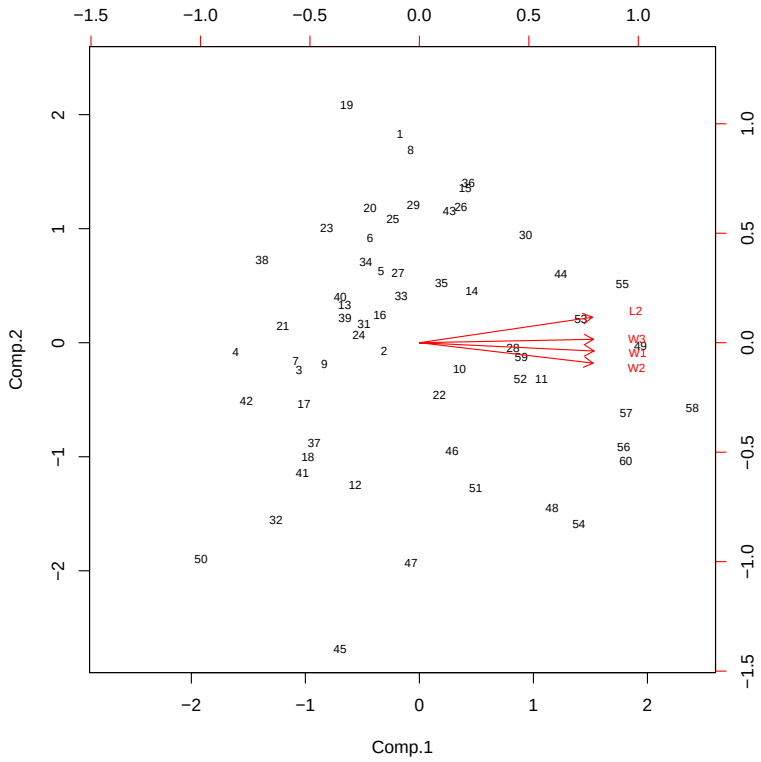


Figure 2.11: Principal component biplot of the trilobite data. Observations $1-43$ represent *Stenopareia glaber*, while observations $44-53$ and $54-60$ represent the two different groups of *S. linnarssoni* observations from Norway and Sweden, respectively

dataset is made up by mixing an unknown set of independent components or variables, and attempts to recover the hidden sources and unknown mixing process. Another way to consider its is that ICA looks for rotations of *sphered* data that have approximately independent coordinates. Suppose there are $k \leq p$ independent sources in a data matrix S, and we observe the p linear combinations $X = SA$ with mixing matrix S. The "unmixing" problem is to recover S. ICA attempts to unmix the data by estimating an unmixing matrix W, where $XW = S$. The ICA model will introduce a fundamental ambiguity in the scale and order of the recovered sources. In other words, the sources can be recovered up to an arbitrary permutation and scaling. Therfore, we may as well assume that the signals have unit variances. The above model assumes that all randomness come from the signals and is known as the "noise-free" ICA model. Below we identify both the "noise free" and "noisy" ICA models, but we will only consider the former. ICA is often very useful in uncovering structure hidden within high-dimensional datasets and is routinely used as an exploratory tool in areas such as medical imaging (Beckmann and Smith, 2002) and gene expression experiments (Liebermeister, 2002; Hori *et al.*, 2001; Lee and Batzoglou, 2003).

Definition 1 (Noise-free ICA model)
ICA of a $n \times p$ data matrix $\mathbf{X}$ consists of estimating the following generative model for the data:

$$X = SA$$

where the latent variables (components) are the columns of the matrix S and are assumed to be independent. The matrix A is the mixing matrix.

Definition 2 (Noisy ICA model)
ICA of a $n \times p$ data matrix $\mathbf{X}$ consists of estimating the following generative model for the data:

$$X = SA + N$$

where S and A are the same as above and N is a noise matrix.

The starting point for ICA is the simple assumption that the components are statistically independent. Under the above generative model, the measured signals in X will tend to be more Gaussian than the source components (in S) due to the Central Limit Theorem. Thus, in order to extract the independent components/sources we search for an un-mixing matrix W that maximises the non-gaussianity (independence) of the sources. The FastICA algorithm of Hyvärinen (1999) provides a very efficient method of maximisation suited for this task. It measures non-gaussianity using approximations to neg-entropy (J) (for speed) which are more robust than kurtosis based measures and fast to compute. This algorithm is implemented in the R package `fastICA` (`http://www.r-project.org`).

Example 1: Crabs Data

We illustrate ICA on the crabs data that we have considered before. In figure 2.12 we present a scatterplot matrix of the four independent components. It seems as though the 2nd and 3rd components picked out the sexes and colour forms, respectively. However, note that Venables and Ripley (2002) remark on the arbitrariness of ICA in terms of the number of components we select. In the crabs data we would perhaps anticipate two components, however, a two component model yield much less impressive results for the crabs data.

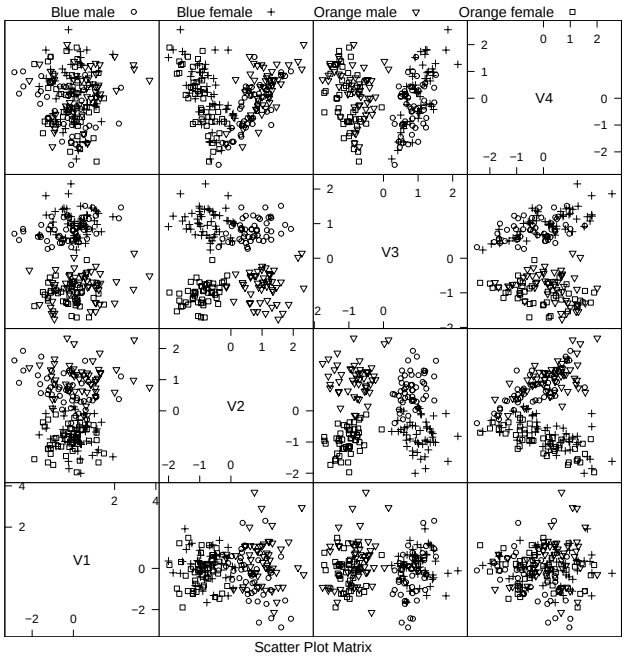


Figure 2.12: Scatterplot matrix of the four independent components for the crabs data

Example 2: Ordovician trilobite data

We also apply ICA to the trilobite data introduced earlier. In figure 2.13 we present boxplots of the recovered components for this data. It is clear that the first signal is reasonably successful in picking out the the three different subgroups of trilobites.

2.3.4 Exploratory Projection Pursuit

Friedman and Tukey (1974) proposed exploratory projection pursuit as a graphical technique for visualising high-dimensional data. Projection pursuit methods seek a q-dimensional projection of the data that maximises some measure of "interestingness" (unlike with PCA this measure would not be the variance, and it would be scale-free). The value of q is usually chosen to be 1 or 2 in order to facilitate visualisation. Diaconis and Freedman (1984) show that a randomly selected projection of a high-dimensional dataset will appear similar to a sample from a multivariate normal distribution. Therefore, most one or two dimensional projections of high-dimensional data will look Gaussian. Interesting structure, such as clusters or long tails, would be revealed by non-Gaussian projections. For this reason the above authors stress that "interestingness" has to mean departures from multivariate normality.

All so-called elliptically symmetric distributions (like the normal distribution) should be uninteresting. So projection pursuit is based on computing projection pursuit indices that measure the non-normality of a distribution by focussing on different types of departures from Gaussianity. A wide variety of indices have been proposed and there is disagreement over the merits of different indices. Two important considerations should be considered when deciding apon a specific index: (1) the computational time that is required to compute it (or some approximation of it); (2) its sensitivity to

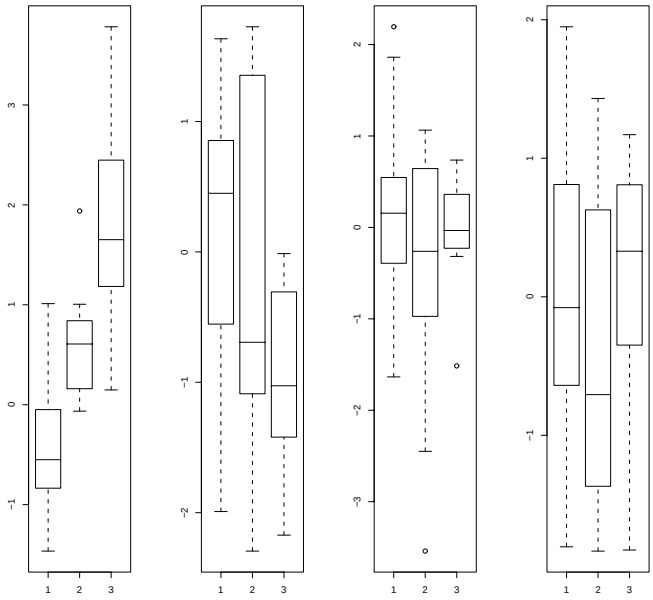


Figure 2.13: Boxplots of the "signals" recovered for the trilobite data split by the three subgroups

outliers. Once an index is chosen, a projection is chosen by numerically maximizing the index over the choice of projection. Most measures do not depend on correlations in the data (i.e. they are affine invariant). Affine invariance can be achieved by "sphering" the data before computing the projection index. A spherically symmetric point distribution has a covariance matrix proportional to the identity matrix. So data can be transformed to have an identity covariance matrix. A simple way to achieve this is to transform the data to the principal components and then rescaling each component to unit variance.

Ripley (1996) comments that once an interesting projection is found, it is important to remove the structure it reveals to allow other interesting views. Interested readers are referred to Ripley (1996) for a more detailed discussion on exploratory projection pursuit and projection indices. `XGobi` and `gGobi` are freely available visualization tools implementing projection pursuit methods. We will consider some applications in the lectures. Despite different origins, ICA and exploratory projection pursuit are quite similar.

Chapter 3

Clustering

3.1 Introduction

Clustering methods divide a collection of observations into subsets or "clusters" so that those within each cluster are more closely related to one another than objects assigned to different clusters. Grouping is done on the basis of similarities or distances (dissimilarities). The inputs required are similarity measures or data from which similarities can be computed. The number of groups may be pre-assigned, or it may be decided by the algorithm. Clustering can be used to discover interesting groupings in the data, or to verify suitability of predefined classes. There are various examples of both these applications in the microarray literature (Alon *et al.*, 1999; Eisen *et al.*, 1998; Alizadeh *et al.*, 2000; Yeung *et al.*, 2001a; Hamadeh *et al.*, 2002). Clustering is an *unsupervised* method, however, sometimes clustering is even used to "classify" (Sørlie *et al.*, 2001). Ripley (1996) warns of the dangers of using clustering to classify observations as the unsupervised classes may reflect a completely different classification of the data.

Even without the precise notion of a natural grouping, we are often able to cluster objects in two- or three-dimensional scatterplots by eye. To take advantage of the mind's ability to to group similar objects and to select outlying observations, we can make use of several visualisation methods such as multi-dimensional scaling and the projection pursuit methods. In fact, Venables and Ripley (2002) observe that we should not assume that 'clustering' methods are the best way to discovering interesting groupings in the data. They state that in their experience the visualisation methods are far more effective. Kaufman and Rousseeuw (1990) provide a good introduction to cluster analysis and their methods are available in the `cluster` package in R.

3.2 Dissimilarity

Clustering algorithms typically operate on either of two structures. The first of these structures is simply the $n \times p$ data matrix:

$$\begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}$$

Many clustering methods are based on a measure of similarity (or dissimilarity). A second structure which clustering algorithms may accept is a table consisting of pairwise proximities for the objects

$$\begin{bmatrix} d(1,1) & d(1,2) & d(1,3) & \cdots & d(1,n) \\ d(2,1) & d(2,2) & d(2,3) & \cdots & d(2,n) \\ \vdots & & & & \vdots \\ d(n,1) & d(n,2) & d(n,3) & \cdots & d(n,n) \end{bmatrix}$$

where $d(i,j)$ measures the dissimilarity between items i and j. The within-cluster dissimilarity can form the basis for a loss function that the clustering algorithm seeks to minimise. However, all of the clustering methods are just algorithms and are not guaranteed to find the global optimum. Dissimilarities are computed in several ways that are dependent on the type of the original variables. Dissimilarities often only satisfy the first three of the following axioms of a metric:

Dissimilarity metric axioms

1. $d(i,j=0)$
2. $d(i,j) \leq 0$
3. $d(i,j) = d(j,i)$
4. $d(i,j) \leq d(i,h) + d(h,j)$

There can be further distinguished between *metric* and *ultrametric* dissimilarities. The former satisfies the fourth axiom above, while ultrametric dissimilarities satifies $d(i,j) \leq \max\{d(i,k), d(j,k)\}$. Ultrametric dissimilarities have the appealing property that they can be represented by a *dendrogram*. Dendograms will be discussed when we look at hierarchical clustering algorithms.

Common measures for continuous data include Euclidean distance and the Manhattan (L_1) distance. Consider two observations $\mathbf{x}$ and $\mathbf{y}$ involving measurements on p variables. The above distance measures between these p-dimensional observations are defined as follows:

Euclidean distance

$$d(i,j) = \sqrt{\sum_{i=1}^{p} (x_{ik} - y_{jk})^2}$$

Manhattan distance

$$d(i,j) = \sum_{i=1}^{p} |x_{ik} - y_{jk}|$$

For continuous variables the issue of standardisation arises. The choice of measurement units has a strong effect on the clustering structure. According to Kaufman and Rousseeuw a change in the measurement units may even lead to a very different clustering structure. The variable with the

largest dispersion will have the largest effect on the clustering. Standardising the variables makes the clustering less dependent on the measurement units of the variables. Standardisation may not be beneficial in all clustering applications. The variables may for example sometimes have an absolute meaning that will be distorted by standardisation. Standardisation may also lead to an unwanted dampening of the clustering structure.

Standarised measures

$$z_{ik} = \frac{x_{ik} - m_k}{s_k}$$

In the above standardisation m_k and s_k represent measures of location (e.g. the sample mean) and dispersion (e.g. the sample standard deviation) for the kth variable. The disadvantage of using the standard deviation is its sensitivity to outliers. More robust measures such as the mean absolute deviation (MAD) $s_k = 1/n \sum i = 1^n \|x_{ik} - m_k\|$ may be more suitable for noisy datasets. The choice of dissimilarity is best motivated by the subject matter considerations. Chipman *et al.* (2003) propose the "1-correlation" distance as a popular choice for microarray data. This dissimilarity can be related to the more familiar Euclidean distance. Changes in the average measurement level or range of measurement from one sample to the next are effectively removed by this dissimilarity.

"1-correlation" distance

$$d(i, j) = 1 - r_{xy}$$

where r_{xy} is the sample correlation between the observations $\mathbf{x}$ and $\mathbf{y}$. If

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sum_i (\mathbf{x}_i - \bar{\mathbf{x}})^2 / p} \quad \text{and} \tilde{\mathbf{y}} = \frac{\mathbf{y} - \bar{\mathbf{y}}}{\sum_i (\mathbf{y}_i - \bar{\mathbf{y}})^2 / p}$$

then

$$\|\tilde{\mathbf{x}} - \tilde{\mathbf{y}}\| = 2p(1 - r_{xy})$$

That is, squared Euclidean distance for the above standardised objects is proportional to the correlation of the original objects.

In this course we only consider continuous variables. Dissimilarities for other types of variables are discussed in Kaufman and Rousseeuw. In the following section we will discuss a few well-known clustering methods.

3.3 Clustering Methods

3.3.1 Hierarchical Methods

Hierarchical clustering techniques proceed by either a series of successive mergers or a series of successive divisions. Hierachical methods result in a nested sequence of clusters which can be graphically represented with a tree, called a *dendrogram*. We can distinguish between two main types of hierarchical clustering algorithms: *agglomerative* algorithms and *divisive* algorithms.

Agglomerative algorithms (Bottom-up methods)

Agglomerative methods start with the individual objects. This means that they initially consider each object as belonging to a different cluster. The most similar objects are first grouped, and these initial groups are then merged according to their similarities. Eventually, all subgroupes are fused into a single cluster. In order to merge clusters we need to define the similarity between our merged cluster and all other clusters. There are several methods to measure the similarity between clusters. These methods are sometimes referred to as *linkage methods*. There does not seem to be consensus as to which is the best method. Below we define four widely used rules for joining clusters. In the lecture we will explain each method at the hand of a simple example.

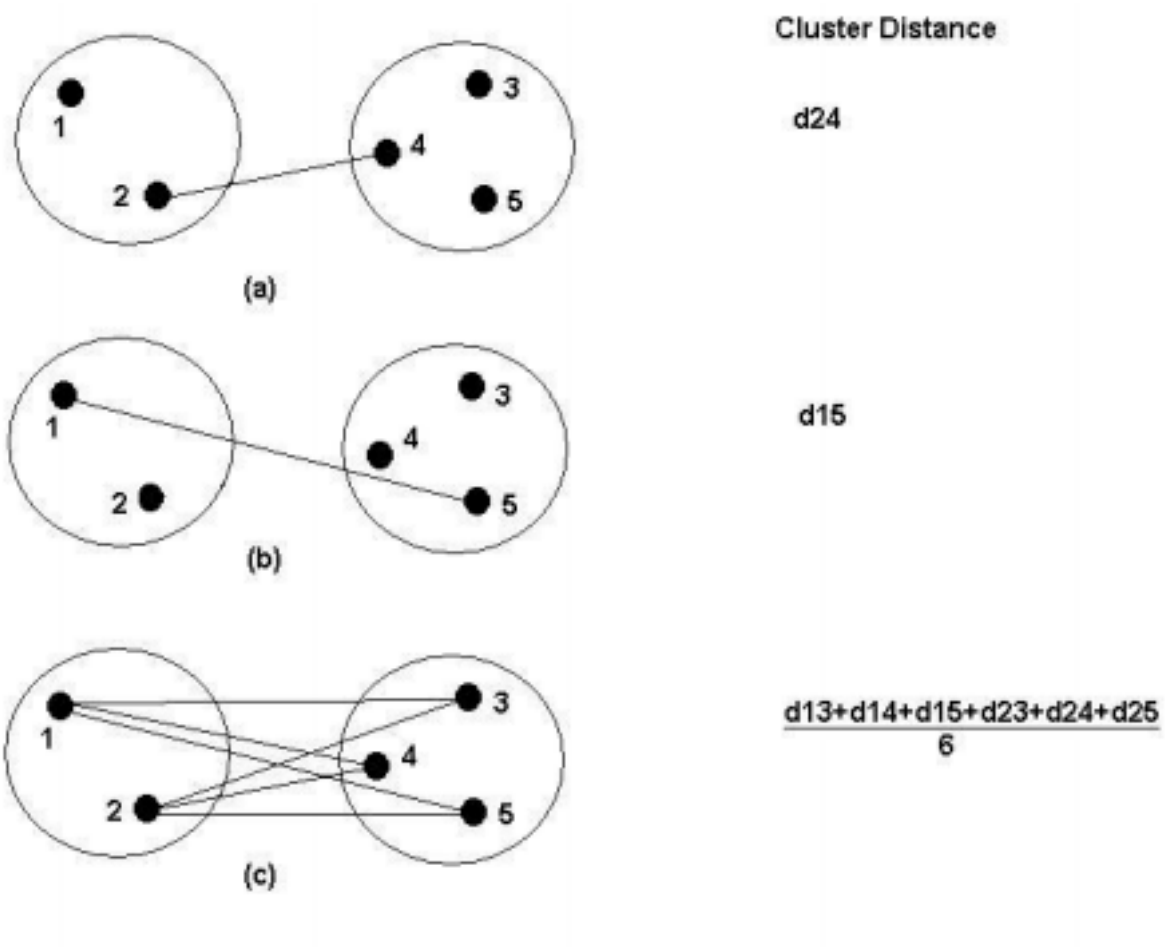
The function `hclust` in R implements the above approaches through its `method` argument. The function `agnes` in the R package `cluster` implements the agglomerative hierarchical algorithm of Kaufman and Rousseeuw with the above methods. Eisen *et al.* (1998) is a well-known application of mean linakge agglomerative hierarchical clustering applied to gene expression data. Their `Cluster` software implements their algorithm.

Divisive algorithms (Top-down methods)

Divisive hierarchical methods work in the opposite direction. They are not as well known and not used as much as agglomerative methods. An advantage of these methods over the previous ones is that they can give better reflections of the structure near the top of the tree. Agglomerative methods can perform poorly in reflecting this part of the tree because many joins have been made at this stage. Each join depends on all previous joins, so some questionable early joins cannot be later undone. When division into a few large clusters is of interest the divisive methods can be a more attractive option, however, these methods are less likely to give good groupings after many splits have been made.

There are several variations on divisive hierarchical clustering, each offering a different approach to the combinatorial problem of subdividing a group of objects into two subgroups. Unlike the agglomerative methods, where the best join can be identified at each step, the best partition cannot usually be found and methods attempt to find a local optimum. One such method is the algorithm developed by Macnaughton-Smith *et al.* (1964). In this algorithm we select a single example whose average dissimilarity to the remaining ones is greatest, and transfer it to a second cluster (say B). For all the remaining examples of cluster A we compare the average dissimilarity to B with that to the remainder of A. If any examples in A are on average nearer to B, we transfer to B that for which the difference in average similarity is greatest. If there are no such examples we stop. This process splits a single cluster. We can subsequently split each cluster that is created, and repeat the process as far as required. The R function `diana` from the `cluster` package perform divisive clustering.

Linkage methods



a) *Single linkage*. Clusters are joined based on the minimum distance between items in different clusters. This method will tend to produce long and loosely connected clusters.

b) *Complete linkage*. Clusters are joined based on the maximum distance between items in different clusters. This means that two clusters will be joined only if all members in one cluster are close to the other cluster. This method will tend to produce "compact" clusters so that relatively similar objects can remain separated up to quite high levels in the dendogram.

c) *Average linkage*. Clusters are joined based on the average of all distances between the points in the two clusters.

Example 1: Agglomerative hierarchical clustering of prostate cancer expression data

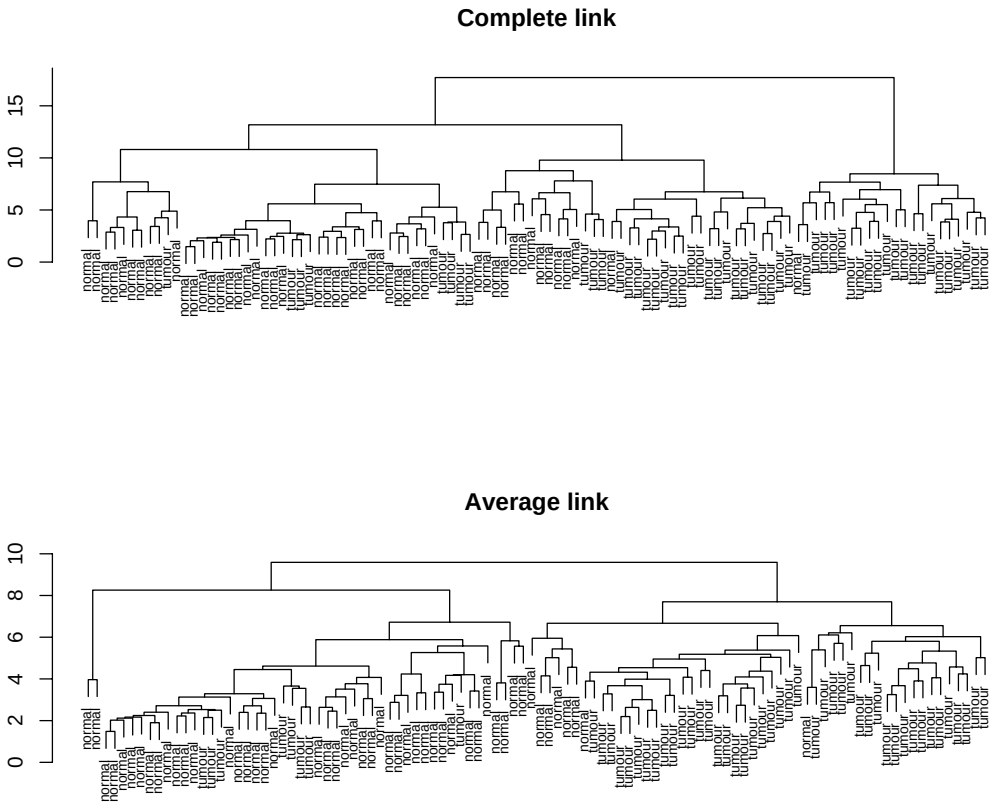


Figure 3.1: Agglomerative hierarchical clustering of the prostate cancer gene expression data of Singh *et al.* (2002) using different links

Singh *et al.* (2002) make use of the Affymetrix U95Av2 chip set and derive expression profiles on 52 prostate tumours and 50 expression profiles on nontumour prostate samples (normal). Each array contains approximately 12 600 genes and ESTs. The aim is to predict the clinical behaviour of prostate cancer based upon gene expression analysis of primary tumours. Their aim is to discriminate between the normal and tumour samples based on the expression profiles of the various samples. In figure 3.1 we produce the dendrogram of an agglomerative hierarchical clustering that we perform using the AGNES algorithm proposed by Kaufman and Rousseeuw (1990). This algorithm is implemented in the function `agnes` in the R package `cluster`. We perform the clustering using both complete and average link functions. Typically, clustering is performed after a filtering step that aims to identify the most discriminating genes between the different classes. In the above application we use the the 100 genes with the largest absolute t-values between the two classes. However, it must be noted that the filtering method and the number of genes included in the clustering will all impact on the performance of the clustering algorithm and potentially bias the final result. In figure 3.2 we repeat the clustering

based on the top 100 genes and all genes, respectively. We see that there is a reasonable difference in the separation achieved by these two sets of genes. Note that we can also perform a agglomerative hierachical clustering in R using the function `hclust`.

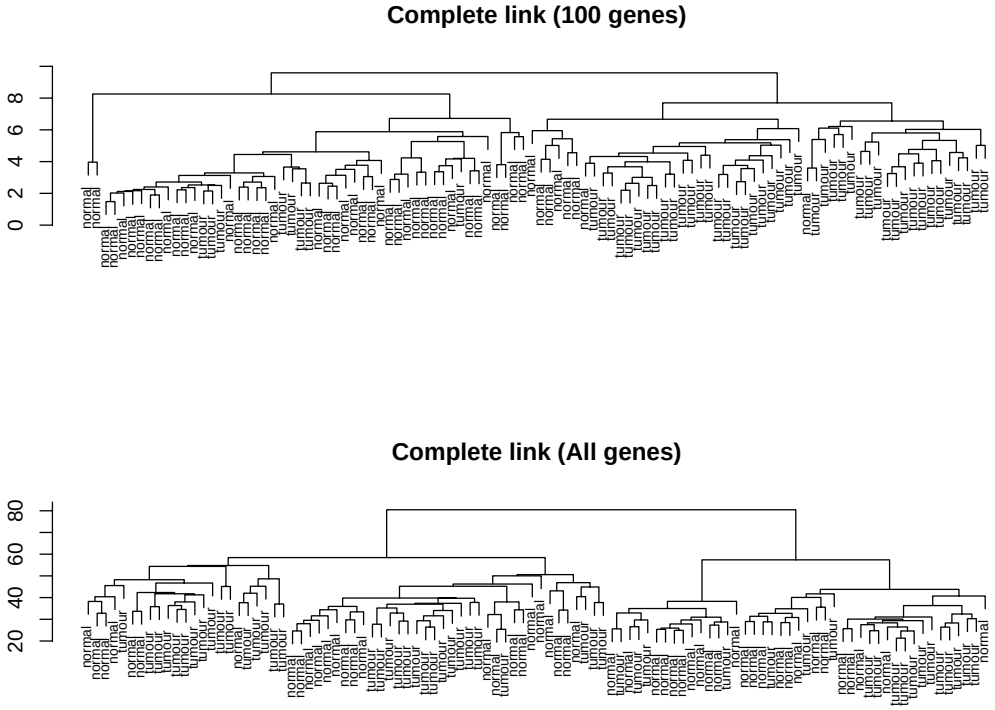


Figure 3.2: Agglomerative hierarchical clustering of the prostate cancer gene expression data of Singh *et al.* (2002) using the same link (average) and different numbers of genes

Example 3: Heat maps of two gene expression datasets

Heat maps are graphical tools that have become popular in the gene expression literature. Similarities between the expression levels of genes and samples are of interest. These representations are most effective if the rows and columns are ordered. Hierarchical clustering is typically used to order the genes and samples according to similarities in their expression patterns. Figure 3.3 is such a representation for the prostate cancer data of Singh *et al.* (2002) that we clustered in the previous example. This heatmap was constructed in R using the function `heatmap`. There is a clear separation between the normal and tumour samples, and there appear to be a small number of groups of co-regulated genes that appear to be responsible for the good discrimination between the groups. For illustartion we only select a subset of the 50 most descriminating genes in terms of the absolute values of their t-statistics here.

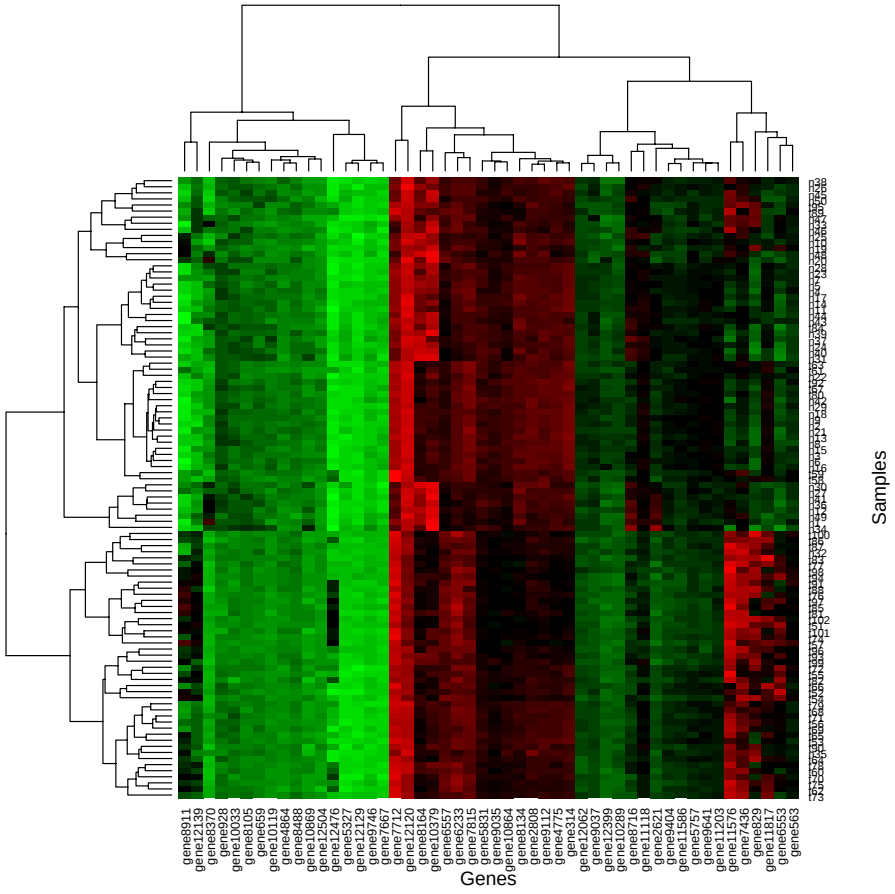


Figure 3.3: Heat map of prostate cancer gene expression data (Singh *et al.*, 2002)

Example 4: Divisive hierarchical clustering of trilobite data

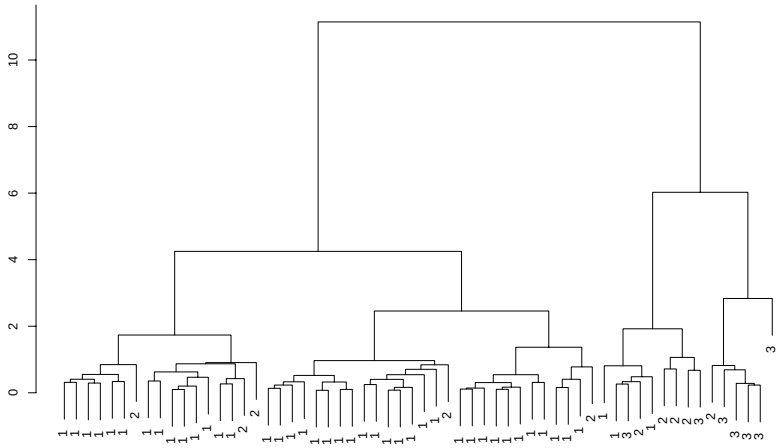


Figure 3.4: Divisive hierarchical clustering of the trilobite data

In figure 3.4 we present the dendrogram of of a divisive hierarchical clustering performed using the algorithm of Kaufman and Rousseeuw (1990) implemented in the function `diana` included in the `cluster` package. We see that the clustering show a reasonable separation between groups 1 and 3, but there appear to be some overlap between groups 1 and 2.

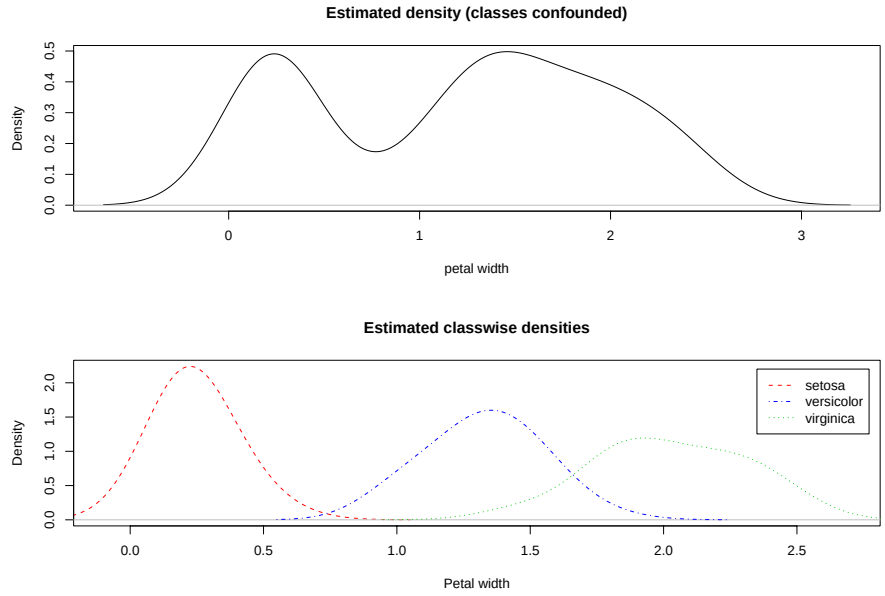


Figure 3.5: Estimated density of the petal width for all species confounded (above) and by species (below)

3.3.2 Gaussian mixture models

The clustering algorithms presented so far were all based on heuristic concepts of proximity. In this chapter we will examine *Gaussian mixture models*, a model-based approach to clustering. The idea of model-based clustering is that the data is an independent sample from different group populations, but the group labels have been "lost".

An introductory example

Consider the measurements of the petal width in the iris dataset. Figure 3.5 shows a plot of the (estimated) densities of the petal width for the three different species and a plot of the density for the whole dataset. The plots suggest that it is reasonable to assume that the petal width in each group is from a Gaussian distribution. Denote with X_i the petal width and with S_i the species name of the i-th flower. Then X_i is (assumed to be) Gaussian in each group and the mean and the variance can be estimated by the empirical mean and variance in the different populations:

$(X_i|S_i = se) \sim N(\mu_{se}, \sigma^2_{se})$ for *iris setosa* (estimates: $\hat{\mu}_{se} = 0.246$, $\hat{\sigma}^2_{se} = 0.01133$)
$(X_i|S_i = ve) \sim N(\mu_{ve}, \sigma^2_{ve})$ for *iris versicolor* (estimates: $\hat{\mu}_{ve} = 1.326$, $\hat{\sigma}^2_{ve} = 0.03990$)
$(X_i|S_i = vi) \sim N(\mu_{vi}, \sigma^2_{vi})$ for *iris virginica* (estimates: $\hat{\mu}_{vi} = 2.026$, $\hat{\sigma}^2_{vi} = 0.07697$)

The density of X (species confounded) is then

$$f(x) = \pi_{se} f_{\mu_{se},\sigma^2_{se}}(x) + \pi_{ve} f_{\mu_{ve},\sigma^2_{ve}}(x) + \pi_{vi} f_{\mu_{vi},\sigma^2_{vi}}(x)$$

The density of X is thus a *mixture* of three normal distributions. The π_{se}, π_{ve} and π_{vi} denote the probability that a randomly picked iris is of the corresponding species. These probabilities are

sometimes referred to as *mixing weights*.
Our objective is however clustering, not classification. Thus we assume the species labels were lost and we only know the petal widths. Nonetheless, we can stick to our idea that the density of X is a mixture of Gaussians. Thus we assume

$$f(x) = \pi_1 f_{\mu_1,\sigma_1^2}(x) + \pi_2 f_{\mu_2,\sigma_2^2}(x) + \pi_3 f_{\mu_3,\sigma_3^2}(x).$$

Note that we had to replace the species names with indexes, because we don't possess the species information any more. This makes the estimation of the means μ_1, μ_2, μ_3 and variances $\sigma_1^2, \sigma_2^2, \sigma_3^2$ far more difficult. The species name S_i is now a *latent* variable, as we are unable to observe it.

The idea behind Gaussian mixture models is like the situation in the example. We assume that the data comes from K clusters.[1] The probability that an observation is from cluster k is π_k. We assume that the data from cluster k is from a (multivariate) Gaussian distribution with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. Thus the density of $\mathbf{x}$ in cluster k is

$$f_{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p \cdot |\boldsymbol{\Sigma}_k|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)'\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)}$$

Often one assumes that all clusters have the same orientation and spread, i.e. $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$ or that all clusters are spherical, i.e. $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$. Another popular assumption is that the covariance matrices are diagonal, i.e. $\boldsymbol{\Sigma}_k = \begin{bmatrix} {\sigma_1^{(k)}}^2 & \cdots 0 & \\ \vdots & \ddots & \vdots \\ 0 & \cdots & {\sigma_p^{(k)}}^2 \end{bmatrix}$. In this case the clusters are ellipses with axes parallel to to coordinate axes. Figure 3.6 visualises these different assumptions.

The density of $\mathbf{x}$ (all clusters confounded) is the *mixture* of the distributions in the clusters:

$$f(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f_{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}(\mathbf{x}).$$

This yields the likelihood

$$\ell(\pi_1,\ldots,\pi_K,\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma}_1,\ldots,\boldsymbol{\Sigma}_K) = \prod_{i=1}^{n}\left(\sum_{k=1}^{K}\pi_k f_{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}(\mathbf{x}_i)\right). \tag{3.1}$$

Gaussian mixture models are an elegant model for clustering, however they suffer from one drawback. Fitting Gaussian mixture models is rather difficult. The density of the mixture distribution involves a sum and taking the derivative of its logarithm of the likelihood (3.1) does not lead to the usual simplifications. The most popular method for fitting Gaussian mixture models is the expectation-maximisation (EM) algorithm of Dempster *et al.* (1977). The basic idea of the EM algorithm is to introduce a *latent* (hidden) variable S_i, such that its knowledge would simplify the maximisation. In our case S_i indicates — like in the example — to which cluster the i-th observation belongs.

The EM algorithm now consists of the following iterative process. For $h = 1, 2, \ldots$ iterate the E-Step and the M-Step:

[1] This parameter needs to be fixed in advance.

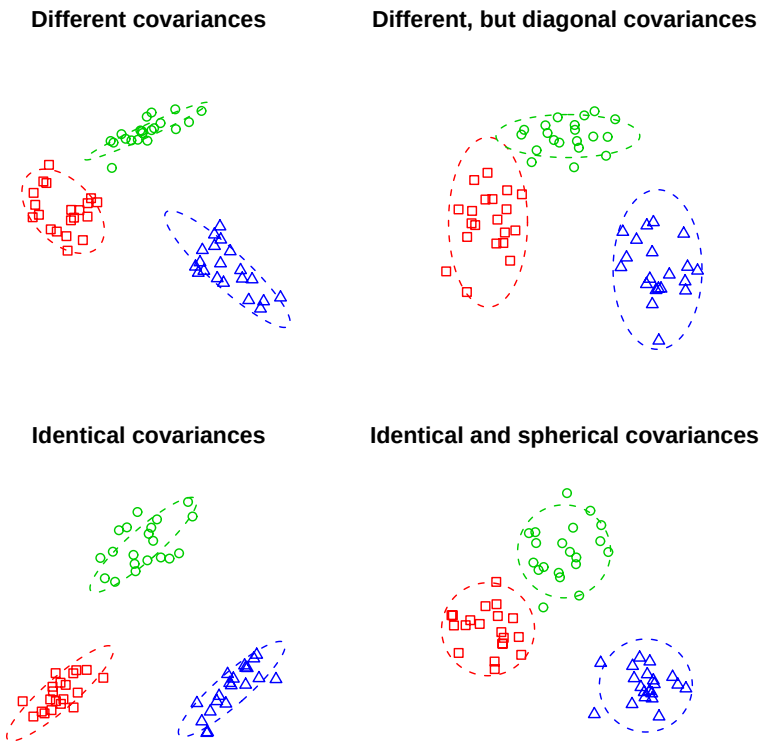


Figure 3.6: Examples for the different assumptions on the covariance structure

Recall: Multivariate Gaussian distribution

A p-dimensional random variable

$$\mathbf{x} = (x_1, \ldots, x_p)$$

is from a *multivariate Gaussian distribution* with mean vector

$$\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)$$

and covariance matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{12} & \sigma_2^2 & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1p} & \sigma_{2p} & \cdots & \sigma_p^2 \end{bmatrix}$$

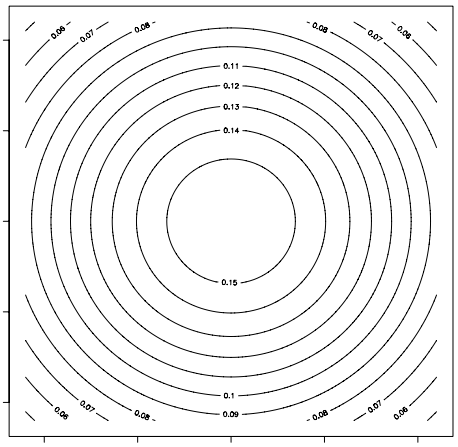
iff it has the density

$$f_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p \cdot |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}.$$

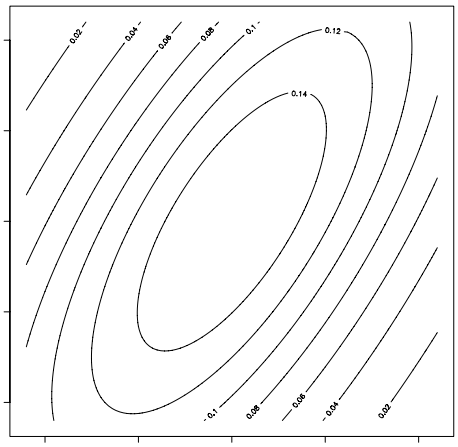
Note that the marginal distribution of the x_j is Gaussian with mean μ_j and variance σ_j^2. The correlation between X_j and X_k is $\rho_{jk} = \frac{\sigma_{jk}}{\sqrt{\sigma_j^2 \sigma_k^2}}$.

If $\boldsymbol{\mu} = (\mu, \ldots, \mu)$ and $\boldsymbol{\Sigma} = \sigma^2 \cdot \mathbf{I} = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{bmatrix}$, then $\mathbf{x}$ consists of p *independent* realisations of a $N(\mu, \sigma^2)$ distribution.

Density of a Gaussian with mean $\mu = (0, 0)$ and covariance matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$:



Density of a Gaussian with mean $\mu = (0, 0)$ and covariance matrix $\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$:



E-Step Estimate the distribution of S_i given the data and the previous values of the parameters $\hat{\pi}_k^{(h-1)}$, $\hat{\boldsymbol{\mu}}_k^{(h-1)}$ and $\hat{\boldsymbol{\Sigma}}_k^{(h-1)}$ $(k = 1, \ldots, K)$. This yields the "responsibilities" $w_{ik}^{(h)}$:

$$w_{ik}^{(h)} := \frac{\pi_k^{(h-1)} f_{\hat{\boldsymbol{\mu}}_k^{(h-1)}, \hat{\boldsymbol{\Sigma}}_k^{(h-1)}}(\mathbf{x}_i)}{\sum_{\zeta=1}^{K} \pi_\zeta^{(h-1)} f_{\hat{\boldsymbol{\mu}}_\zeta^{(h-1)}, \hat{\boldsymbol{\Sigma}}_\zeta^{(h-1)}}(\mathbf{x}_i)}$$

M-Step Compute the updated $\hat{\pi}_k^{(h)}$, $\hat{\boldsymbol{\mu}}_k^{(h)}$, and $\hat{\boldsymbol{\Sigma}}_k^{(h)}$ $(k = 1, \ldots, K)$ by estimating them from the data using the vector $\mathbf{w}_k^{(h)} = (w_{1k}^{(h)}, \ldots, w_{nk}^{(h)})$ of responsibilities as weights: This leads to the updated estimators

$$\hat{\pi}_k^{(h)} = \frac{1}{n} \sum_{i=1}^{n} w_{ik}^{(h)},$$

$$\hat{\boldsymbol{\mu}}_k^{(h)} := \frac{1}{\sum_{i=1}^{n} w_{ik}^{(h)}} \sum_{i=1}^{n} w_{ik}^{(h)} \mathbf{x}_i,$$

and

$$\hat{\boldsymbol{\Sigma}}_k^{(h)} := \frac{1}{\sum_{i=1}^{n} w_{ik}^{(h)}} \sum_{i=1}^{n} w_{ik}^{(h)} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k^{(h)})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k^{(h)})'$$

for the covariances $\boldsymbol{\Sigma}_k$ (assuming different covariances in the classes).

Some mathematical details

The EM algorithm focuses on the likelihood that is obtained when assuming the class labels S_i are known. In this case the likelihood is

$$\prod_{i=1}^{n} \pi_{S_i} f_{\boldsymbol{\mu}_{S_i}, \boldsymbol{\Sigma}_{S_i}}(\mathbf{x})$$

Note that the sum has now disappeared. This is due to the fact that we know the class labels now, so we just multiply the densities of the distribution in the corresponding clusters instead of multiplying the unpleasant mixture density. For simplification of the notation denote with $\boldsymbol{\theta}$ the whole parameter vector, i.e.

$$\boldsymbol{\theta} := (\pi_1, \ldots, \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K)$$

The log-likelihood is thus

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \pi_{S_i} + \log f_{\boldsymbol{\mu}_{S_i}, \boldsymbol{\Sigma}_{S_i}}(\mathbf{x}) \tag{3.2}$$

In reality however, the S_i are unknown. So the EM-algorithm uses the trick of taking the (conditional) expectation of the likelihood.

In the E-step we have to compute the conditional expectation of the log-likelihood (3.2) given the data using the previous values of the parameters $\hat{\boldsymbol{\theta}}^{(h-1)}$. Thus we have to compute the conditional

distribution of S_i (using the Bayes formula) yielding the responsibilities

$$w_{ik}^{(h)} := \mathbb{P}_{\hat{\boldsymbol{\theta}}^{(h-1)}}(S_i = k) = \frac{\hat{\pi}_k^{(h-1)} f_{\hat{\boldsymbol{\mu}}_k^{(h-1)}, \hat{\boldsymbol{\Sigma}}_k^{(h-1)}}(\mathbf{x}_i)}{\sum_{\zeta=1}^{K} \hat{\pi}_\zeta^{(h-1)} f_{\hat{\boldsymbol{\mu}}_\zeta^{(h-1)}, \hat{\boldsymbol{\Sigma}}_\zeta^{(h-1)}}(\mathbf{x}_i)}$$

The conditional expectation is thus

$$\mathbb{E}_{\hat{\boldsymbol{\theta}}^{(h-1)}}\left(L(\theta)|\mathbf{x}_1, \ldots, \mathbf{x}_n\right) = \sum_{i=1}^{n}\sum_{k=1}^{K} w_{ik}^{(h)} \log \pi_k + w_{ik}^{(h)} \log f_{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k}(\mathbf{x}). \tag{3.3}$$

In the M-step we have to find $\hat{\boldsymbol{\theta}}^{(h)}$ by maximising the conditional expectation (3.3) ober $\boldsymbol{\theta}$. One can easily see that the updated parameter estimates $\hat{\pi}_k^{(h)}$, $\hat{\boldsymbol{\mu}}_k^{(h)}$, and $\boldsymbol{\Sigma}_k^{(h)}$ maximise (3.3).

Figure 3.7 visualises the flow of the algorithm for an easy toy example. Despite being theoretically elegant, the EM algorithm has major drawbacks: It is very slow; the convergence is only linear. Furthermore it does only converge to a stationary point of the log-likelihood (practically speaking a *local* maximum). The EM algorithm is very sensitive to the inital values $\hat{\pi}_k^{(0)}$, $\hat{\boldsymbol{\mu}}_k^{(0)}$ and $\hat{\boldsymbol{\Sigma}}_k^{(0)}$ $(k = 1, \ldots, K)$; it is thus essential to choose appropriate initial values.

There is another class of models which is often confused with mixture models: Likelihood-based partitioning methods assume as well that the data comes from different clusters, in which the data are Gaussian with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$.[2] The partitioning $\Omega = \Omega_1 \cup \ldots \cup \Omega_K$ is however not modelled using a latent variable, but being seen a model parameter. If one assumes that all clusters have identical and spherical covariance matrices this approach leads to the k-means algorithm presented in the next section.

3.3.3 k-Means

In Gaussian mixture models the means $\hat{\boldsymbol{\mu}}_k$ were updated by weighted averages of the $\mathbf{x}_i$. The weights w_{ik} were the conditional probabilities that the i-th observation was from cluster k ("responsibilities"). This probability is typically high for observations that are close to the cluster centre $\hat{\boldsymbol{\mu}}_k$. This idea can be used to construct a simple, yet powerful clustering algorithm. The k-means algorithm (McQueen, 1967) can be seen as a simplification of the EM-algorithm for fitting Gaussian mixture models. Instead of calculating the responsibilities, it simply assigns all observations to the closest cluster, i.e. the cluster with the closest centre $\hat{\boldsymbol{\mu}}_k$, which is then updated by the taking the arithmetic mean of all observations in the cluster.

Using suitably chosen initial values $\hat{\boldsymbol{\mu}}_1^{(0)}, \ldots, \boldsymbol{\mu}_K^{(0)}$ the k-means-algorithm thus iterates until convergence the following two steps $(h = 1, 2, \ldots)$:

i. Determine for all observations $\mathbf{x}_i$ the cluster centre $\hat{\boldsymbol{\mu}}_k^{(h-1)}$ that is closest and allocate the i-th observation to this cluster.

[2] Like for mixture models it is possible to use other probability distribution as well.

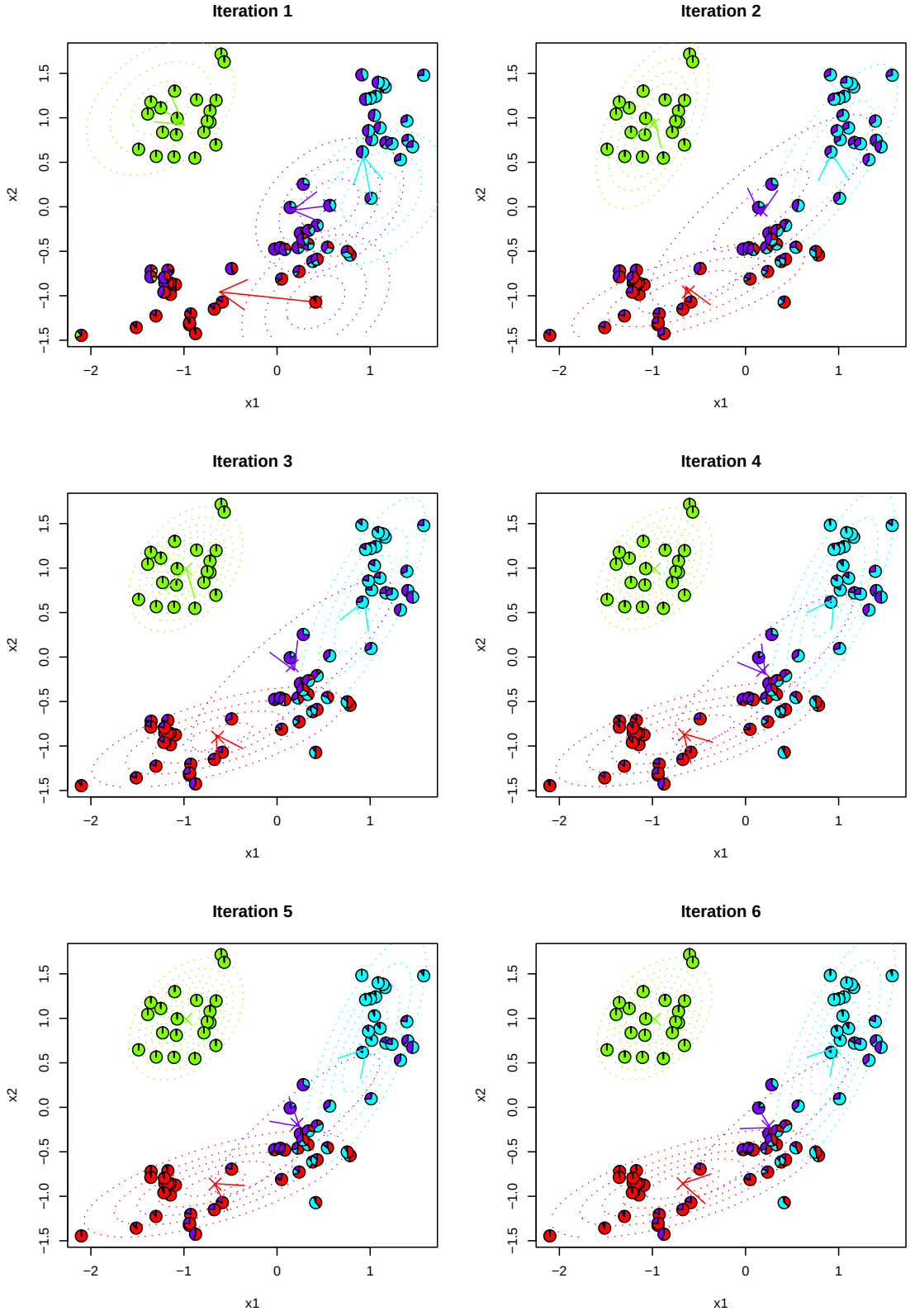


Figure 3.7: First 6 iterations of the EM algorithm for fitting a Gaussian mixture model to a toy example. The colouring of the data points corresponds to the responsibilities, the dotted lines show the isodensites of the Gaussians, and the arrow indicates where the means of the clusters are moved.

ii. Update the class centres by computing the mean in each cluster:

$$\hat{\boldsymbol{\mu}}_k^{(h)} := \frac{1}{\text{size of cluster } k} \sum_{\text{observations } \mathbf{x}_i \text{ in cluster } k} \mathbf{x}_i$$

Figure 3.8 visualises the flow of the k-means algorithm for an easy example.

The k-means algorithm determines the clusters and updates the means in a "hard" way. The EM algorithm computed the (conditional) probabilities that one observation is from one class ("soft partitioning");[3] the k-means algorithm simply assigns one class to the closest cluster ("hard partitioning"). The partitions created by the k-means algorithm are called *Voronoi regions*.

The k-means algorithm minimises the (trace of the) within-class variance. Thus the sum of squares, which is n times the trace of the within-class variance,

$$\sum_{k=1}^{K} \sum_{\text{observations } \mathbf{x}_i \text{ in cluster } k} \|\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k\|^2 \tag{3.4}$$

can be used as a criterion to determine the goodness of a partitioning.

Bottou and Bengio (1995) show that the k-means algorithm is a gradient descent algorithm and that the update of the cluster centres (step ii.) is equivalent to a Newton step. So one can expect the k-means algorithm to converge a lot faster than the EM-algorithm for mixture models. Thus it is often used to find the starting values for the latter.

As the k-means algorithm is based on Euclidean distance, it is essential that the data is appropriately scaled before the algorithm is used.

The k-means algorithm depends highly on the initial values chosen. Thus it is essential to run the algorithm using different starting values and to compare the results. Many implementations use randomly picked observations are starting values, so all you have to do is running the k-means clustering several times (without resetting the random number generator). If the algorithm always leads to (clearly) different partitionings, then you have to assume the the dataset is *not* clustered *at all*! This is even more the case if the different partitionings always have approximately the same goodness criterion (3.4).[4] The dependence of the k-means algorithm on the initial values makes it very easy to manipulate its results. When being started often enough with different starting values that are close to the desired solution, the algorithm will eventually converge to the desired solution and thus "prove" the desired hypothesis.

One way of analysing the solution of the k-means algorithm is looking at a couple of plots. One possibility is plotting the cluster centres in the plain of the first two principal components and colouring the observations according the their clusters. A parallel coordinate plot comparing the cluster centres and clusterwise parallel coordinate plots can be helpful as well. Figure 3.9 to 3.11 show the corresponding plots for the result obtained by applying the k-means algorithm to the European economies dataset ($K = 5$ clusters). The clustering corresponds to what one might have expected: Luxembourg's banking sector makes it outstanding, and Romania and Bulgaria, which are not yet part of the EU, have an economy still highly depending on agriculture. The composition of the other three clusters seems

[3] Thus the EM algorithm for fitting Gaussian mixture models is sometimes referred to as "soft k-means".

[4] It is possible that the algorithm sometimes gets "stuck" in a suboptimal solution, so even if there is a clear clustering it might happen that the k-means algorithm does not find it. In this case the sum of squares of the "bad" solution should be clearly less than the sum of squares of the "good" solution.

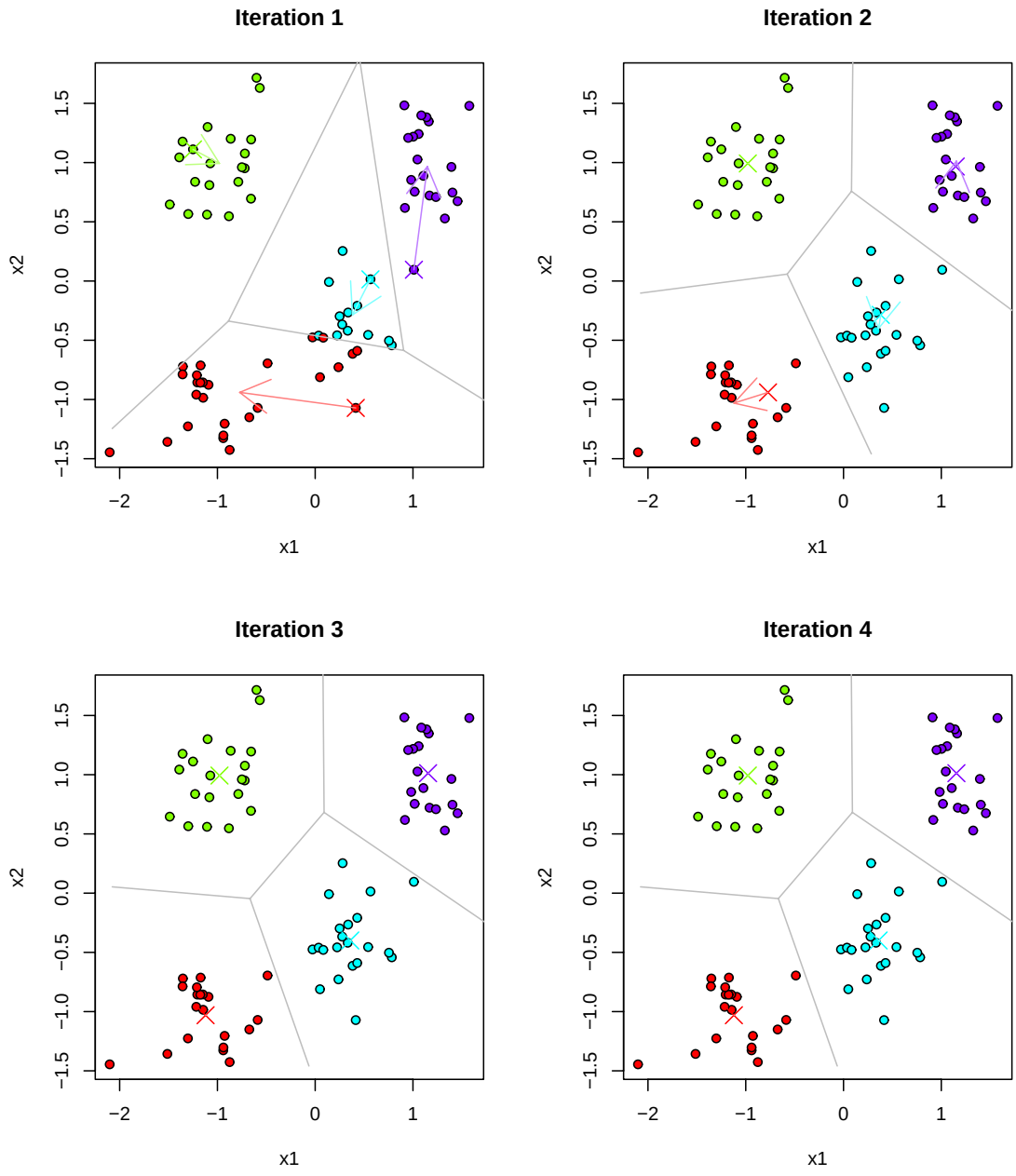


Figure 3.8: First 4 iterations of the k-means algorithm applied to a toy example. The colouring of the data points corresponds to the cluster allocation, the "x" designates the cluster centres and the arrow indicates where they are moved. The grey lines are the borders of the corresponding Voronoi regions

at first sight reasonable as well: Western European industrial countries, Eastern Eurpean Countries and the remaining countries. Even though the clusters have a nice interpretation, there is not enough evidence for them in the data — the three latter clusters are not different enough that they could be clearly distinguished. When rerunning the k-means algorithm with another initial configuration we would see that the composition of these three clusters changes all the time.

A "silhouette" plot of the distances between the observations and the corresponding cluster centres can give some evidence on the goodness of the partitioning. Figure 3.12 shows a silhouette plot for the European economies dataset.

3.3.4 k-Medoids (Partitioning around medoids)

As stated in the preceding chapter the k-means algorithm is based on the squared Euclidean dissimilarity. This makes the k-means algorithm rather prone to outliers. The k-medoid algorithm (used with appropriate dissimilarity measures) does not have this flaw. Furthermore, like the hierarchical clustering algorithms, it can be carried out using any dissimilarity measure d_{ij}, where d_{ij} is the dissimilarity between the i-th and the j-th observation. The k-medoids algorithm can thus be seen as a generalisation of the k-means algorithm. However one often chooses $d_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$. One usually restricts the cluster centres to be one of the observations in each cluster, thus $\hat{\boldsymbol{\mu}}_k = \mathbf{x}_{c_k}$ for some c_k.

The k-medoids algorithm then consists of the iteration of the following two steps until the cluster assignments do not change any more.

i. For all observations determine the cluster centre that is closest, i.e. assign $\mathbf{x}_i$ to the class k whose centre $\hat{\boldsymbol{\mu}}_k = \mathbf{x}_{c_k}$ is closest, i.e. has the lowest dissimilarity d_{ic_k}.

ii. For the k-th cluster ($k = 1, \ldots, K$) determine the observation (from cluster k) that minimises the sum of dissimilaries to the other observations, i.e. choose the index c_k such that

$$\sum_{\text{observations } \mathbf{x}_i \text{ in cluster } k} d_{ic_k}$$

is minimal.

The k-medoids algorithm is usually initialised at random, i.e. the indexes of the cluster centres $i_1, \ldots, i_K$ are chosen at random from $\{1, \ldots, n\}$ (without replacement).

In analogy to k-means algorithm the above k-medoids algorithms seeks to minimise

$$\sum_{k=1}^{K} \sum_{\text{observations } \mathbf{x}_i \text{ in cluster } k} d_{ic_k} \tag{3.5}$$

However, the above ad-hoc algorithm gives no guarantee of converging to this minimum (nor does the k-means algorithm0. Kaufman and Rousseeuw (1990) and Massart *et al.* (1983) propose alternative and less heuristic algorithms to minimise (3.5). The k-means algorithm is a more efficient implementation of this algorithm using the dissimilarity function $d_{ij} = |\mathbf{x}_i - \mathbf{x}_j|^2$ and without the restriction on the cluster centres.

The solutions of the k-medoid algorithm should be handled with the same care (and sceptic) as the solutions of the k-means algorithm. Thus it is advisable to run the algorithm using different starting configurations and to compare the results obtained.

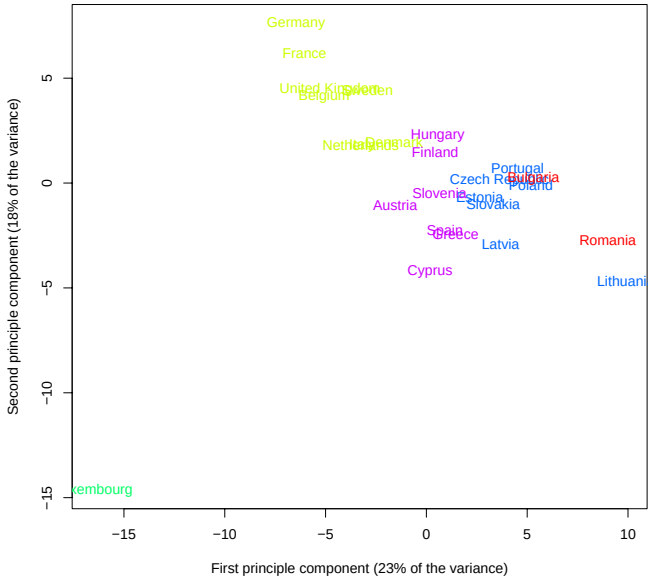


Figure 3.9: Plot of the cluster structure in the first two principle components of the European economies dataset. Note that the first two principal components only account for 41% of the variance.

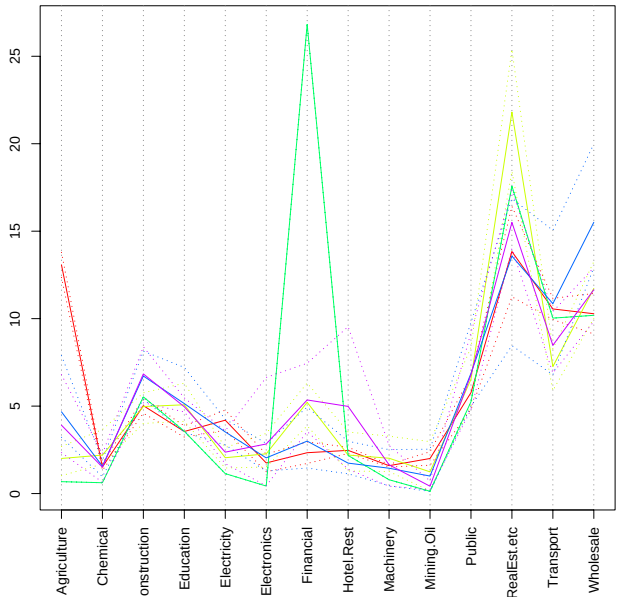


Figure 3.10: Parallel coordinate plot of the clusters found in the European economies dataset. The soluid lines sorrespond to the cluster centres, whereas the dotted lines correspond to the minimal/maximal values is the clusters.

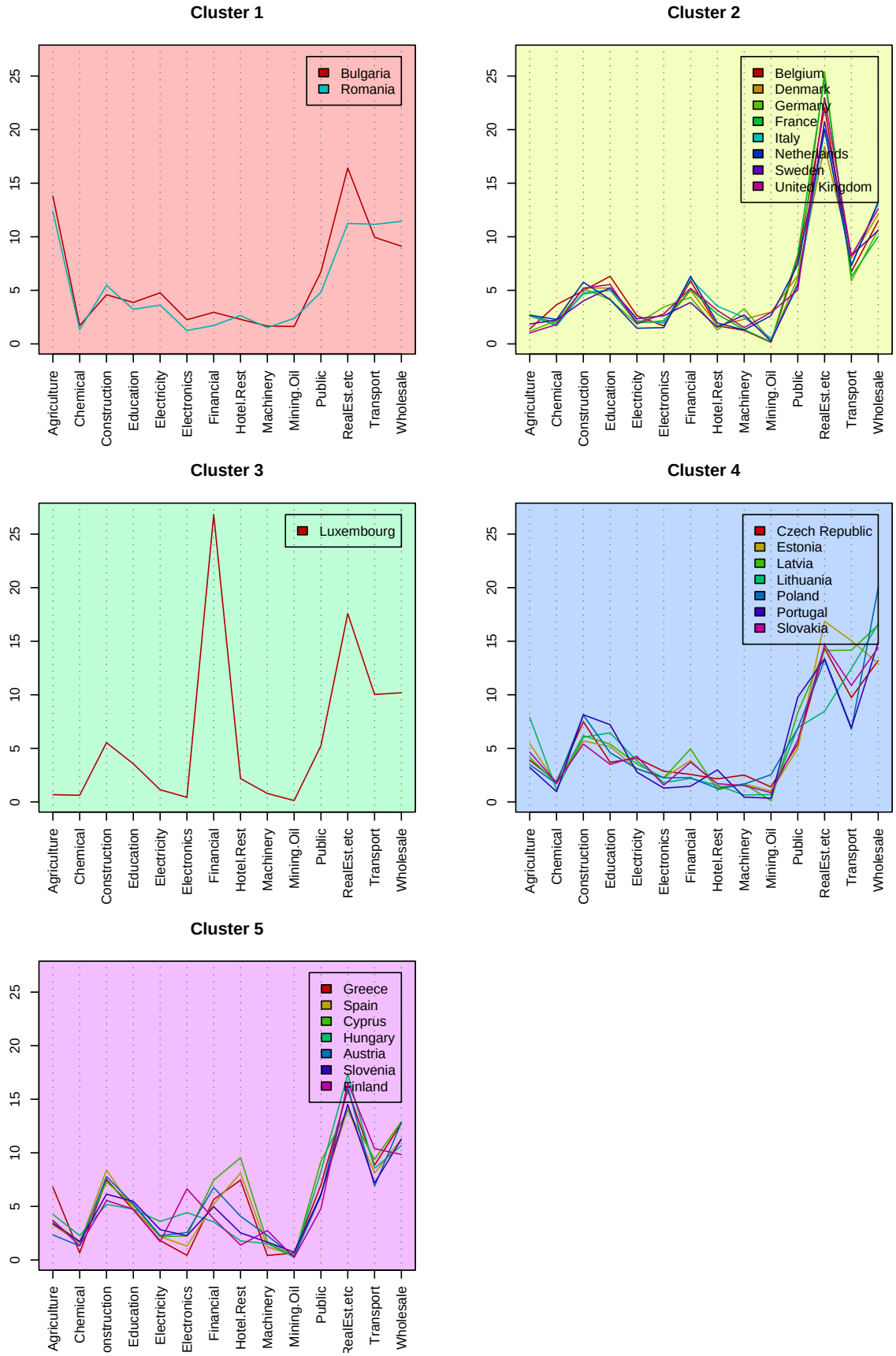


Figure 3.11: Clusterwise parallel coordinate plot for the European economies dataset.

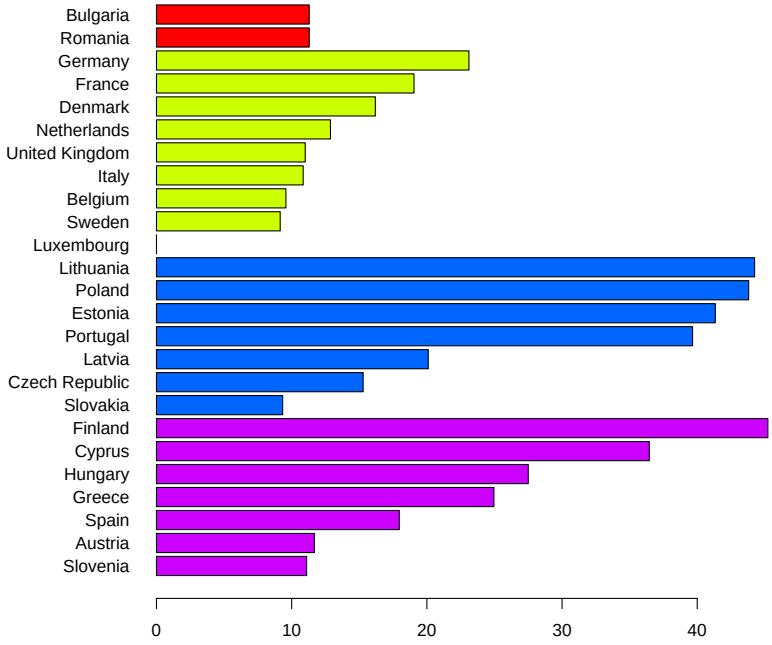


Figure 3.12: Silhouette plot of the distances to the cluster centres for the European economies dataset

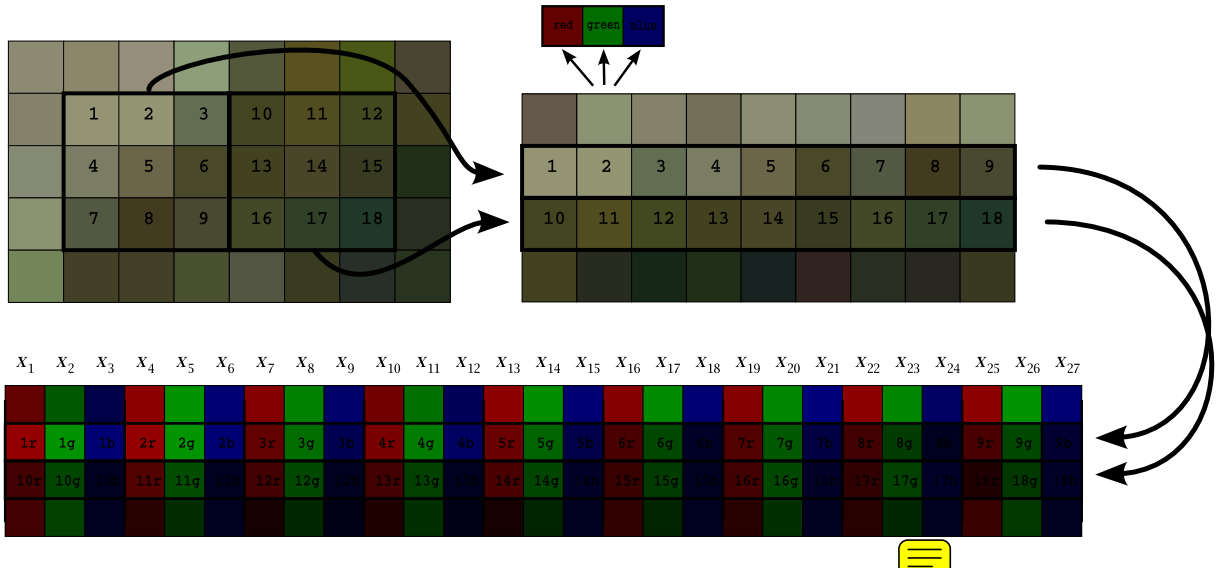


Figure 3.13: Illustration of block vector quantisation: Each block of $3times3$ pixels is regarded as one observation with 27 variables (3 rows $\times$ 3 columns $\times$ 3 colour channels (red, green, blue)

3.3.5 Vector quantisation

Vector quantisation is a tool in (lossy) signal compression. A sophisticated version of vector quantisation is e.g. part of the MPEG-4 audio and video compression standard. The idea of vector quantisation is to replace all signals by a number of "representative signals", the prototypes (*codewords*). So instead of saving the multidimensional signal (e.g. a block of pixels in an image) itself we just have to store the index of the corresponding prototypes in the *codebook*. This requires far less space. Note that this is a lossy compression technique, as we are not able to reconstruct the original signal. However we hope that the prototypes are representative and thus the quality does not degrade too much.

The k-means algorithm[5] can now be used to find the codebook. One simply uses the cluster centres $\hat{\boldsymbol{\mu}}_k$ as the prototypes. The idea is that the cluster centres are good representatives of the clusters. So all signals (i.e. data points) in the Voronoi region corresponding to one cluster are represented by the same prototype. The number of clusters controls how much the signals are compressed as we need $\log_2(k)$ bits to save the index of the prototype.

We will illustrate vector quantisation with an image from Yann-Arthus Bertrand's *La terre vue du ciel*[6]. We will apply a 3×3 block vector quantisation, i.e. we see each block of 3×3 pixels as one observation, leading to 27 variables; figure 3.13 illustrates this. Figure 3.14(a) shows the original images; figures 3.14(b) to 3.14(d) show the results for different codebook lengths. Already the first compressed image leads to file size of $61,932$ bytes (including the codebook), thus only 5.9% of the size of the original image.

[5]The k-means algorithm is sometimes called Lloyd's algorithm is this context.
[6]The photographs are available from `www.yannarthusbertrand.com`.

(b) Codebook length 2048 (1.22 bits/pixel)

(d) Codebook length 8 (0.44 bits/pixel)

(a) Original image (24 bits/pixel)

(c) Codebook length 128 (0.78 bits/pixel)

Figure 3.14: Original image and results of 3×3 block vector quantisation for different codebook lengths

3.3.6 Self-organising maps (SOM)

The clustering techniques studied so far lead to clusters that are topologically unrelated; clusters that are for example close to each other in the dendrogram, are not necessarily more similar than other clusters. Self-organising maps (Kohonen, 1982) is a clustering algorithm that leads to cluster that have a topological similarity structure. It assumes that the clusters lie on a rectangular (or hexgonal) grid; on this grid, neighbouring clusters are more similar than distant ones.

Like in k-means clustering an observation is assigned to the cluster whose "centre" is closest to it. In k-means clustering an observation only influences the closest cluster centre. In self-organising maps an observation influences neighbouring clusters as well, however the influence on more distant clusters is less. Kohonen proposed an *online* algorithm, in which the observations $\mathbf{x}_i$ are presented repeatedly and one after another. For each observation $\mathbf{x}_i$ presented the algorithm updates all cluster centres $\hat{\boldsymbol{\mu}}_k$ ($k = 1, \dots K$), which are called *representatives* in this context, by setting

$$\hat{\boldsymbol{\mu}}_k \leftarrow \hat{\boldsymbol{\mu}}_k + \alpha_t \cdot h(d_{ik})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k).$$

d_{ik} is hereby the distance (on the grid, *not* in the observation space) between the cluster containing the i-th observation and the k-th cluster. $h : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is a decreasing function like $e^{-1/2d^2}$ ("Gaussian neighbourhood"). If one sets $h(d) := 1_{\{0\}}(d)$, then only the cluster containing the i-th observation (and no other cluster) gets updated; in this case the SOM algorithm is very inefficient variant of the k-means algorithm.

The SOM algorithm is highly dependent on the initial configuration — to an even greater extent than the k-means algorithm. When running the SOM algorithm with randomly picked inital cluster centres, one gereally never gets the same result. Therefore, the SOM algorithm is usually initliased by putting the cluster centres on a grid in the plain of the first two principal components. So the SOM solution is highly dependent on the MDS solution (see section 3.5)

Note that we have to choose the number of clusters and the topology beforehand. One usually chooses a higher number of clusters than one would choose for the k-means algorithm. The reason for this is the topological constraint on the clusters, that makes them less flexible. It is even common that the SOM algorithm leads to one ore more empty clusters.

Figure 3.15 and 3.16 show the results obtained when applying the SOM algorithm to the *European economies* dataset. Figure 3.16 shows the grid of clusters in the principal component plain after the initialisation, and after 1, 10 and 100 iterations. Figure 3.15 gives the resulting self-organising map.

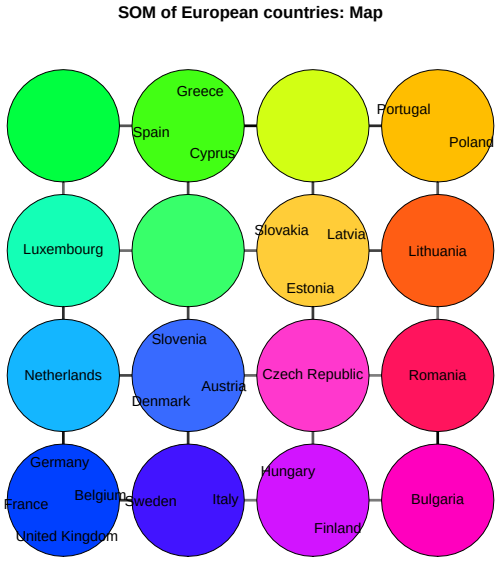


Figure 3.15: SOM map of the European economies dataset

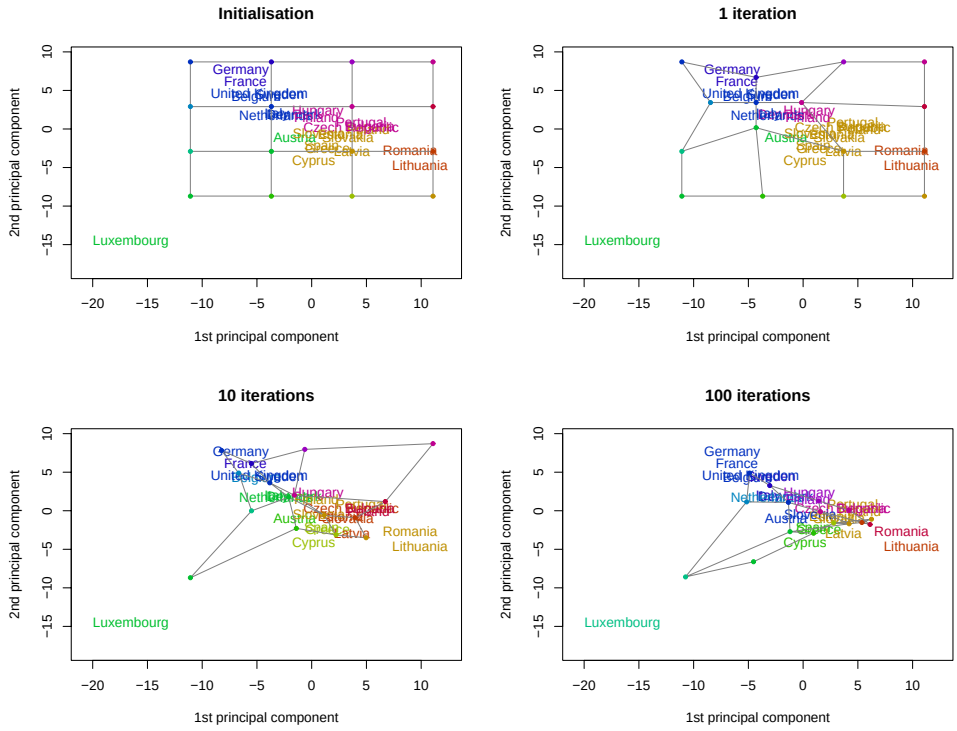


Figure 3.16: Plot of the grid of the SOM for the European economies dataset after initialisation and after 1, 10, and 100 iterations.

3.4 Comments on clustering in a microarray context

3.4.1 Two-way clustering

In microarray studies samples can be clustered independently of genes, and vice versa. Some clustering applications use both samples and genes simultaneously to extract joint information about both of them. This apporach has been motivated by the fact that the overall clustering of samples may be dominated by a set of genes involved in a particular cellular process, which may mask another interesting group supported by another group of genes. For this reason, two-way clustering aims to extract joint information on the samples and genes.

One approach to the above clustering problem is to systematically uncover multiple sample clusterings based on different sets of genes. Getz *et al.* (2000) propose such an approach. Their method aims to find subsets of genes that result in stable clusterings of a subset of samples. They find pairs $(O_k, F_k), k = 1, 2, \ldots, K$ where O_k is a subset of genes and F_k is a subset of the samples, so that when genes O_k are used to cluster samples F_k, the clustering yields stable and significant partitions. This will be useful when the overall clustering of samples based on all genes is dominated by some subset of genes, for example genes related to metabolism. The above authors do appear to be reasonably successful in their stated goal – to find stable clustering pairs (O_k, F_k) – however, it is difficult to assess the biological relevance of their results and to compare them to more standard hierarchical clustering approaches.

Block clustering due to Hartigan (1972) and named by Duffy and Quiroz (1991) is another approach for obtaining a divisive hierachical row-and-column clustering of a data matrix. This approach reorders the rows (genes) and columns (samples) to produce homogeneous blocks of gene expression. Block clustering only provides one reorganisation of the samples and genes, and cannot discover multiple groupings. In some sense, the plaid model (Lazzeroni and Owen, 1992) is a further extension of the ideas behind block clustering and gene shaving.

3.4.2 Discussion: Clustering gene expression data

Clustering as an exploratory tool

Cluster analysis has caught on as technique for revealing common patterns of gene expression across different samples in the microarray literature. The idea is that genes with similar profiles of activity will have related functions. However, clustering is often misused and the results over-interpreted. Some statisticians argue that clustering is best suited to determining relationships between a small number of variables, rather than deriving patterns of involving thousands of genes across a very large dataset. As we have mentioned before, certain hierarchical methods can be unreliable for obtaining good high-level clusters for example. Some people do not even regard clustering as statistics, but rather regard it as an arbitrary manupilation of data that is not principled.

Biologists can be sent down blind alleys if they see clusters that reinforce their own assumptions about the relationships between individual genes. F. H. C Marriott made the following comment in the context of cluster analysis on p.89 of Marriott (1974):

> "If the results disagree with informed opinion, do not admit a simple logical interpretation, and do not show up clearly in a graphical presentation, they are probably wrong. There is no magic about numerical methods, and many ways in which they can break down.

> They are a valuable aid to the interpretation of data, not sausage machines automatically transforming bodies of numbers into packages of scientific fact."

Caution should be exercised when intrepreting and assessing clustering results. As an example we consider a study by Alizadeh *et al.* (2000). They used hierarchical clustering to group B-cell lymphomas into two distinct classes. These correlated with inferences in patient survival, and seemed from their gene expression profiles to be related to the stage of development of the cell from which the cancer originated. However, last year another group, using a different clustering method, failed to find this association (Shipp *et al.*, 2002). Choosing a clustering method is still very much dependent on the data, the goals, and the investigator's personal preferences. This often leads to a lot of subjective judgement when applying clustering in a microarray context. In many cases where you have a good idea what you are looking for you will probably find it!

The primary goals of clustering should be data exploration, pattern-discovery and dimension reduction. Clustering should not be used to classify. If the intent is to distinguish between currently defined sample classes (e.g. tumour and normal tissues), the supervised methods that we will discuss in later lectures should be used. Unsupervised groupings may reflect a completely different classification of the data as mentioned on page 288 of Ripley (1996).

Evaluating clustering performance

A major problem encountered when using cluster analysis is the difficulty in interpreting and evaluating the reliability of the clustering results and various clustering algorithms. The boundries between clusters are often very noisy and many clusters can arise by chance. Most clustering algorithms do not formally account for sources of error and variation. This means that these clustering methods will be sensitive to outliers. Therefore, assessing the clustering results and interpreting the clusters found are as important as generating the clusters. In the absence of a well-defined statistical model, it is often difficult to define what a *good* clustering algorithm or the *correct* number of clusters is.

In hierarchical clustering there is no provision for a reallocation of objects that may have been "incorrectly" grouped at an early stage. If clustering is chosen as the method of choice it is advisable to examine the final configuration of clusters to see if it is sensible. For a particular application, it is a good idea to try several clustering methods, and within a given method, a couple of different ways of assigning similarities. If the outcomes from several methods are roughly consistent with one another it perhaps strenghens the case for a "natural" grouping.

The *stability* of a hierarchical solution can sometimes be checked by applying the clustering algorithm before and after *small* errors (pertubations) have been added to the data. If the groups are fairly well distinguished, the clusterings before and after pertubation should agree. Kerr and Churchill (2001) for example, propose the use of bootstrapping to assess the stability of results from cluster results in a microarray context. This involve sampling the noise distribution (with replacement). They emphasise that irrespective of the clustering algorithm that is chosen, it is imperative to assess whether the results are statistically reliable relative to the level of noise in the data. Yeung *et al.* (2001b) also propose an approach for validating clustering results from microarray experiments. Their approach involves leaving out one experiment at at a time and then comparing various clustering algorithms using the left-out experiment. McShane *et al.* (2002) propose a noise model to assess the reproducibility of clustering results in the analysis of microarray data.

In most gene expression studies there will typically only be a reasonably small percentage of the total number of genes (typically thousands) that will show any significant change across treatment

levels or experimental conditions. Many genes will not exhibit interesting changes in expression and will typically only add noise to the data. A gene-filtering step to extract the "interesting genes" is often advisable before applying a clustering algorithm, as they may lead to a reduction in noise and a subsequent improvement in the performance of the clustering algorithm. However, extreme caution must be exercised, as the results can depend strongly on the set of genes used. For example, use of all available genes or genes that were chosen by some filtering method can produce very different hierarchical clusterings of the samples for example.

Choosing the number of clusters

Choosing the number of clusters is crucial for many hierarchical and partioning methods, especially in gene expression studies. Some approaches to address this problem have been suggested. Ben-Hur *et al.* (2002) use a sampling approach to determine the number of clusters that provides the most stable solution. Another approach introduces the so-called *Gap-statistic* (Tibshirani *et al.*, 2001). This approach compares a measure of within-cluster distances from the cluster centroids to its expectation under an appropriate reference distribution, and chooses the number of clusters maximizing the difference.

Some studies propose a mixture model approach (i.e. model-based clustering) for the analysis of gene expression data (Ghosh and Chinnaiyan, 2002; Yeung *et al.*, 2001a; Barash and Friedman, 2001; McLachlan *et al.*, 2002). An attractive feature of the mixture-model approach is that it is based on a statistical model. This provides one with a probabilistic framework for selecting the best clustering structure and the correct number of clusters, by reducing it to a model selection problem. The above studies focus on Gaussian mixtures. Yeung *et al.* (2001a) propose various data transformations in order to transform the data to be nearly Gaussian. However, it is notoriously difficult to test for multivariate Gaussian distributions in higher dimensions. Furthermore, the problem of providing protection against outliers in multivariate data is a very difficult one that increases with difficulty with the dimension of the data.

The fitting of tivariate t distributions, as proposed in McLachlan McLachlan and Peel (1998) and McLachlan McLachlan and Peel (2000) provides a longer-tailed alternative to the normal distribution and hence added robustness. For many applied problems (including gene expression studies), the tails of the normal distribution are often shorter than required. The strongest criticism against the use of model-based clsutering methods for gene expression studies is the strong parametric assumptions they make about the distribution of gene expression data, which may not always be realistic.

3.4.3 Alternatives to clustering

PCA and ICA are often used as alternatives to clustering when the aim is to identify small subsets of "interesting" genes. As we have discussed in earlier lectures, PCA and ICA differ in the measure of "interestingness" they use. Gene shaving is an adaptation of PCA that looks for small subsets of genes that behave very similarly, and account for large variance across samples. Yeung and Ruzzo (2001) also propose a PCA-based approach to clustering.

ICA is emerging as another strong candidate to visualise and and find structure in expression data. Liebermeister (2002), Lee and Batzoglou (2003) and Hori *et al.* (2001) apply ICA to gene expression data and find that ICA is successful in revealing biologically relevant groups in the data. Liebermeister motivates the application of ICA to gene expression data, as ICA relies on the idea of combinatorial control, by describing the expression levels of genes as linear functions of common hidden variables.

The author refers to these hidden variables as "expression modes". Expression modes may be related to distinct biological causes of variation, such as responses to experimental treatments. ICA differs from clustering in that clustering identifies groups of genes responding to different variables, but it does not represent the combinatorial structure itself. A small number of "effective" regulators, each acting on a large set of genes and varying between distinct biological situations may describe the co-regulation of genes. Liebermeister proposes a method for ordering independent components according to their ability to discriminate between different groups of genes. However, the above study does not address the important issue of choosing the number of components. Similar to the problem of choosing the number of clusters in cluster analysis, ICA also suffers from determining the number of independent components that are "hidden" within the data. Venables and Ripley (2002) make the following remark on page 314:

> "There is a lot of arbitrariness in the use of ICA, in particular in choosing the number of signals."

Lee (2003) proposes the SONIC statistic for choosing the number of independent components. SONIC is based on the idea behind the Gap statistic for estimating the number of clusters in cluster analysis. SONIC compares the ordered negentropy values of estimated independent components to its expectation under a null reference distribution of the data.

3.5 Multidimensional scaling

Multidimensional scaling (MDS) is part of a wider class of methods sometimes referred to as *distance methods*. These methods attempt to represent a high-dimensional cloud of points in a lower dimensional space (typically 2 or 3 dimensional) by preserving the inter-point distances as well as possible.

Suppose that $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are observations in $\mathbb{R}^p$. The first step is to define a measure of dissimilarity d_{ij} between observations i and j, for example the Euclidean distance $\|x_i - x_j\|$. The function `daisy` in the R package `cluster` provides a general way to compute dissimilarities that extends to variables that are not measured on interval scale (e.g. ordinal and binary variables). Multidimensional scaling seeks a representation $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$ so that the distances between the points best match those in the dissimilarity matrix. Here the best representation is that one which minimises the *stress function*. There are different forms of MDS that use different stress functions. Two popular choices of the stress function are given below.

Stress functions

1. *Classical MDS*:

$$S_{\texttt{classical}}(d_{ij}, \tilde{d}_{ij}) = \sum_{i \neq j} \left[d_{ij}^2 - \tilde{d}_{ij}^2 \right] / \sum_{i \neq j} d_{ij}^2$$

2. *Sammon's nonlinear mapping*:

$$S_{\texttt{Sammon}}(d_{ij}, \tilde{d}_{ij}) = 1 / \sum_{i \neq j} d_{ij} \sum_{i \neq j} \left(d_{ij} - \tilde{d}_{ij} \right)^2 / d_{ij}$$

3. *Kruskal-Shepard nonmetric scaling*:

$$\texttt{STRESS}^2 = \sum_{i \neq j} \left(\theta(d_{ij}) - \tilde{d}_{ij}\right)^2 / \sum_{i \neq j} \tilde{d}_{ij}$$

where d represents dissimilarity in the original p-dimensional space and $\tilde{d}$ represents disimilarity in the reduced k-dimensional space.

Note that using classical MDS with Euclidean distance is equivalent to plotting the first k prinicipal components (without rescaling to correlations). *Sammon mapping* places more emphasis representing smaller pairwise distances accurately. In R classic MDS, Kruskal-Shepard nometric MDS and classical MDS can be performed using the functions `cmdscale`, `sammon` and `isoMDS`. The last two functions are available in the `MASS` package.

3.5.1 Example 1 : MDS and prostate cancer gene expression data

In figure 3.17 we present three distance-based representations of the prostate cancer data that we have considered before. The tumour (t) amd normal (n) samples seem to be well separated in all three plots. There appears to be one tumour sample that is more similar to the normal samples and one normal sample that is more similar to the tumour samples (based on the 200 genes we used). In later lectures we shall return to this data to see how well some classifiers fare in distinguishing between these groups. The MDS plots suggest that there is a clear separation between the two classes and we expect the classifiers to be reasonably successful in distinguishing between the groups.

3.5.2 Example 2 : MDS and virus data

In this example we consider a dataset on 61 viruses with rod-shaped particles affecting various crops (tobacco, tomato, cucumber and others) described by Fauquet *et al.* (1988) and analysed by Eslava-Gómez (1989). There are 18 measurements on each virus, the number of amino acid residues per molecule of coat protein; the data come from a total of 26 sources. There is an existing classification by the number of RNA molecules and mode of transmission, into 39 *Tobamoviruses* with monopartite genomes spread by contact, 6 *Tobraviruses* with bipartite genomes spread by nematodes, 3 *Hordeiviruses* with tripartite genomes, transmission mode unknown and 13 *Furoviruses* , 12 of which are known to be spread fungally. The question of interest to Fauquet et al. was whether the furoviruses form a distinct group, and they performed various multivariate analyses. Here we only consider the *Tobamoviruses* to investigate if there are further subgroups within this group of viruses. In figure 3.18 we use the same distance-based representations as for the previous example to visualise the data. There are two viruses with identical scores, of which only one is included in the analyses. (No analysis of these data could differentiate between the two).

Figure 3.18 reveals some clear subgroups within the *Tobamoviruses*, which are perhaps most clearly visible in the Kruskal MDS plot. Viruses 1 (frangipani mosaic virus), 7 (cucumber green mottle mosaic virus) and 21 (pepper mild mottle virus) have been clearly separated from the other viruses in the Kruskal MDS plot, which is not the case in the other two plots. Ripley (1996) states that the non-metric MDS plot shows interpretable groupings. The upper right is the cucumber green mottle virus,

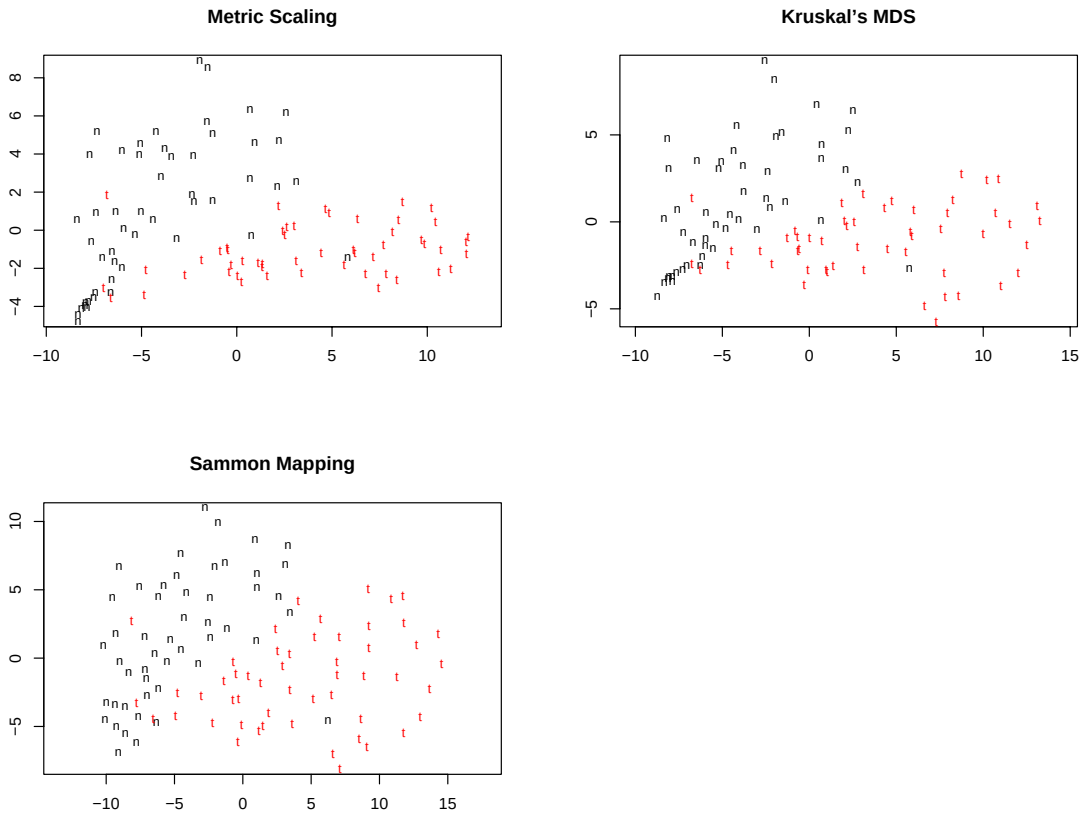


Figure 3.17: Distance based representations of of the prostate cancer data (200 genes). The top left plot is by classic multidimensional scaling, the top right by Kruskal's isotonic multidimensional scaling, and the bottom left by Sammon's nonlinear mapping

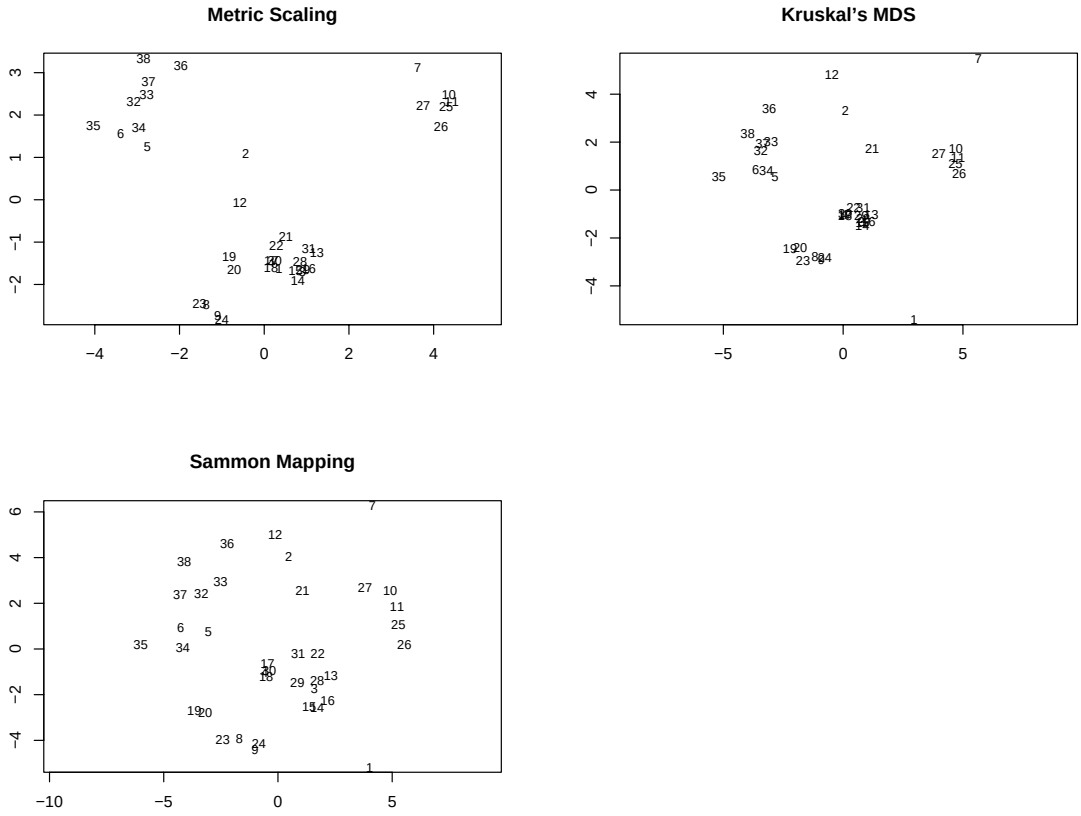


Figure 3.18: Distance-based representations of the *Tobamovirus* group of viruses. The variables were scaled before Euclidean distance was used. The point are labelled by the index number of the virus

the upper left is the ribgrass mosaic virus and the two others. The one group of viruses at the bottom, namely 8,9,19,20,23,24, are the tobacco mild green mosaic and odontoglossum ringspot viruses.

Chapter 4

Classification: Theoretical backgrounds and discriminant analysis

4.1 Classification as *supervised learning*

The objective of the clustering methods presented so far was to infer how the data might be partitioned into different classes (*unsupervised learning*). The task in *supervised learning* is different. We *know* to which classes the observations $\mathbf{x}_i$ belong. The objective is to "learn" from these examples so that we are able to predict the class label for future observations. Examples for tasks in supervised learning are diagnosing diseases, identifying "good" customers, reading handwritten digits *etc.*

4.2 Some decision theory

This chapter gives a very brief overview over the statistical decision theory for classification.

Assume the observations come from L different classes and let the random variable C be the class label. For each object we posses measurements $\mathbf{x}_i$, the *feature vector*. We assume that the feature vectors $\mathbf{x}_i$ are from a *feature space* $\mathcal{X}$, typically a subset of $\mathbb{R}^p$.

Our task is to find a classification rule

$$\hat{c}: \ \mathcal{X} \to \{1, \ldots, L\}, \ \mathbf{x} \mapsto \hat{c}(\mathbf{x}).$$

The interpretation of the map is as follows: If we observe the feature vector $\mathbf{x}$, then we classify the observation into the class $\hat{c}(\mathbf{x})$.[1]

For now, we make the unrealistic assumption that the classwise densities $f(\mathbf{x}|l)$ and the prior probabilities π_l are known. For an observation from class l, the probability of misclassification of the decision rule $\hat{c}(\cdot)$ is then

$$\text{pmc}(l) = \mathbb{P}(\hat{c}(\mathbf{x}) \neq l | C = l)$$

For assessing the overall performance of a classifier, we need to specify how "bad" wrong decisions are. We usually assume that all wrong decisions are equally bad, but this needs not to be the case

[1] Sometimes additional decisions like "outlier" or "doubt" are considered. See e.g. Ripley (1996).

under all circumstances: In medicine the cost of failing to detect a disease is much higher than the cost of a false positive result. We use a *loss function* to model these costs:

$$L(l, m) = \text{Loss incurred when classifying an observation from class } l \text{ into class } m$$

If we make the above assumption that all wrong decisions are equally bad, we obtain

$$L(l, m) = \begin{cases} 0 & \text{if } l = m \\ 1 & \text{if } l \neq m \end{cases}. \tag{4.1}$$

Using the loss function we now can compute the expected loss when using the classification rule $\hat{c}(\cdot)$. This expected loss is called *total risk*.

$$R(\hat{c}) = \mathbb{E}\left(L(C, \hat{c}(\mathbf{X}))\right)$$

When using the usual loss function (4.1) one obtains

$$R(\hat{c}) = \sum_{l=1}^{L} \pi_l \cdot \text{pmc}(x). \tag{4.2}$$

Assuming we know the prior probabilities π_l and the classwise densities $f(x|l)$, we can give an best decision rule: On can show that the classification rule

"Classify $\mathbf{x}$ into the class with highest posterior probability $p(l|\mathbf{x})$"

minimises the total risk, when using the loss function (4.1). We can compute the posterior probability using the Bayes formula

$$p(l|\mathbf{x}) = \frac{\pi_l f(\mathbf{x}|l)}{\sum_{\lambda=1}^{L} \pi_\lambda f(\mathbf{x}|\lambda)}. \tag{4.3}$$

Therefore this best decision rule is often called *Bayes rule* and its total risk is called *Bayes risk*. As the Bayes rule is the best decision rule, the Bayes risk is a lower bound for the total risk for all possible classification rules. Thus it is often used as a benchmark.

Figure 4.1 illustrates the Bayes rule for an easy two-class example, where the prior probabilities are $\pi_1 = 0.4$ and $\pi_2 = 0.6$ and the classwise distributions are Gaussians with means $\mu_1 = -1$ and $\mu_2 = 1$ and standard deviation $\sigma = 1$. For all x on the left hand side $\pi_1 f(x|1)$ is greater than $\pi_2 f(x|2)$, so for these x we have $\hat{c}(x) = 1$.

In reality however, the classwise densities $f(\mathbf{x}|l)$, the classwise probabilities π_l, and thus the posterior probabilities $p(l|\mathbf{x})$ are unknown; we have to estimate them from the data.

Logistic regression (see chapter 5) estimates the posterior probability $p(l|\mathbf{x})$ — which is nothing else than the conditional distribution of the class labels given the feature vector $\mathbf{x}$ — directly from the data.

Alternatively, we can estimate the classwise densities $f(\mathbf{x}|l)$ (and the prior probabilities π_l) from the data and plug them into (4.3) yielding

$$\hat{p}(l|\mathbf{x}) = \frac{\hat{\pi}_l \hat{f}(\mathbf{x}|l)}{\sum_{\lambda=1}^{L} \hat{\pi}_\lambda \hat{f}(\mathbf{x}|\lambda)}.$$

The resulting classifier called the *plug-in* classifier. For estimating $f(\mathbf{x}|l)$ we have two possibilities. We can make a parametric assumption about the classwise distribution of $\mathbf{x}$ and only estimate the

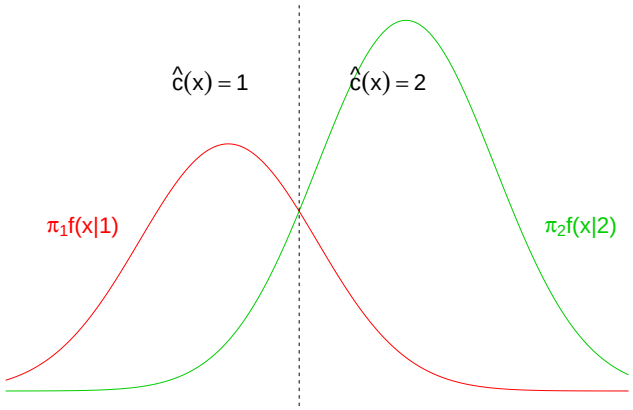


Figure 4.1: Example for the decision boundary generated by the Bayes rule

parameters. We could for example assume that the classwise distributions are all Gaussians. This leads to linear and quadratic discriminant analysis (see sections 4.3 and 4.4). We could as well estimate the densities $f(\mathbf{x}|l)$ non-parametrically (see section 4.6).

One problem of the plug-in approach is that it does not account for the uncertainty in $\hat{f}(\mathbf{x}|l)$. The more Bayesian approach of *predictive classification* takes this into account. However it needs an assumption on the prior distributions of the model parameters of $f(\mathbf{x}|l)$. (see e.g. Ripley, 1996, ch. 2.4)

4.3 Linear discriminant analysis (LDA)

Linear discriminant analysis assumes that the classwise densities are multivariate Gaussian distributions with means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_L$ and *common* covariance matrix $\boldsymbol{\Sigma}$, i.e.

$$f(\mathbf{x}|l) = f_{\boldsymbol{\mu}_l, \boldsymbol{\Sigma}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p \cdot |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_l)' \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}_l)}.$$

The (log) posterior probability then becomes

$$\begin{aligned} \log p(l|\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_l)' \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}_l) + \log \pi_l + \text{const} \\ &= \mathbf{x}' \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_l - \frac{1}{2} \boldsymbol{\mu}_l' \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_l + \log \pi_l + \text{const} = \mathbf{x}' \mathbf{a}_l + b_l, \end{aligned} \tag{4.4}$$

with appropriately defined $\mathbf{a}_l$ and b_l. As this expression in *linear* in $\mathbf{x}$, the decision boundaries will be (linear) hyperplanes.[2]

[2]The decision boundary between the l-th and the m-th class is the set of points for which $p(l|\mathbf{x}) = p(m|\mathbf{x})$. We can take the logarithm on both sides yielding $\log p(l|\mathbf{x}) = \log p(m|\mathbf{x})$, using (4.4) we obtain $\mathbf{x}'\mathbf{a}_l + b_l = \mathbf{x}'\mathbf{a}_m + b_m$, an equation linear in $\mathbf{x}$.

In practice the means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_L$ and covariance $\boldsymbol{\Sigma}$ are unknown. But using the plug-in approach all we have to do is replacing them by their estimator. The maximum likelihood estimators for the parameters are

$$\hat{\boldsymbol{\mu}}_l := \bar{\mathbf{x}}_l = \frac{1}{n_l} \sum_{\mathbf{x}_i \text{ in class } l} \mathbf{x}_i$$

for the mean and for the covariance

$$\hat{\boldsymbol{\Sigma}} := \frac{1}{n} \sum_{l=1}^{L} \sum_{\mathbf{x}_i \text{ in class } l} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_l)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_l)'.$$

However, $\boldsymbol{\Sigma}$ is often estimated by the *within-class* covariance matrix

$$\mathbf{W} := \frac{1}{n-L} \sum_{l=1}^{L} \sum_{\mathbf{x}_i \text{ in class } l} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_l)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_l)', \tag{4.5}$$

which has a different denominator.

In order to be able to compute the value of the Gaussian density we need to calculate the inverse of the covariance matrix. This is only possible if it is of full rank. This can be only the case if the dataset has at least as many rows as columns[3] — especially in genetics this might not be the case.

Figure 4.2 shows the decision boundary obtained by applying linear discriminant analysis to the iris dataset (only using the covariates `petal width` and `petal length`). Note that all estimated classwise densities have the same orientation.

4.4 Quadratic discriminant analysis (QDA)

Unlike linear discriminant analysis, quadratic discriminant analysis assumes that the Gaussians in each class have *different* covariance matrices $\boldsymbol{\Sigma}_l$. In this case the (log) posteriori becomes

$$\begin{aligned} \log p(l|\mathbf{x}) &= -\frac{1}{2}\log|\boldsymbol{\Sigma}_l| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_l)'\boldsymbol{\Sigma}_l^{-1}(\mathbf{x}-\boldsymbol{\mu}_l) + \log \pi_l + \text{const} \\ &= -\frac{1}{2}\mathbf{x}'\boldsymbol{\Sigma}_l^{-1}\mathbf{x} + \mathbf{x}'\boldsymbol{\Sigma}\boldsymbol{\mu}_l - \frac{1}{2}\boldsymbol{\mu}_l'\boldsymbol{\Sigma}_l^{-1}\boldsymbol{\mu}_l - \frac{1}{2}\log|\boldsymbol{\Sigma}_l| + \log \pi_l + \text{const} \\ &= \mathbf{x}'\mathbf{A}_l\mathbf{x} + \mathbf{x}'\mathbf{b}_l + c_l \end{aligned} \tag{4.6}$$

with appropriately defined $\mathbf{A}_l$, $\mathbf{b}_l$, and c_l. As (4.6) is quadratic in $\mathbf{x}$ the decision boundaries are (parts of) ellipses.

Using the plug-in approach we estimate $\boldsymbol{\mu}_l$ by the mean $\bar{\mathbf{x}}_l$ in class l and $\boldsymbol{\Sigma}_l$ by the covariance in class l. Note that each class must have more observations than covariates, otherwise one of the classwise covariance matrices is not of full rank.

Figure 4.3 shows the decision boundaries obtained by quadratic discriminant analysis for the iris data. Note the the classwise Gaussian distributions now have different orientations and spread.

[3] Note that this is just a necessary condition, not a sufficient one.

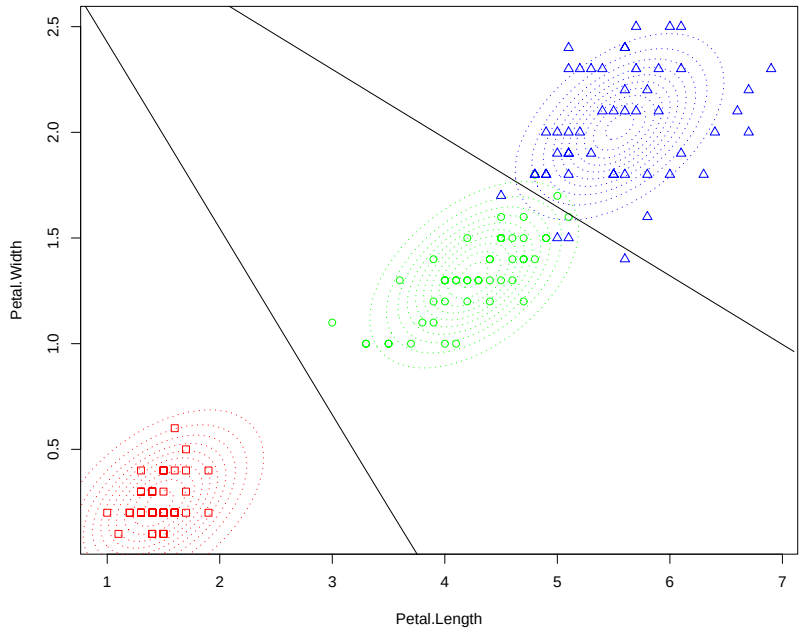


Figure 4.2: Decision boundaries and classwise Gaussian distributions for an LDA fit to the iris dataset

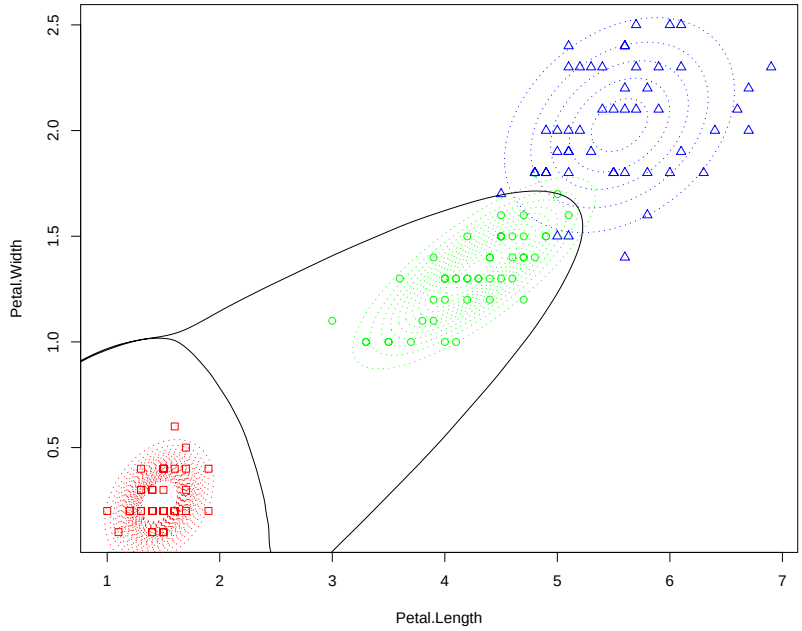


Figure 4.3: Decision boundaries and classwise Gaussian distributions for a QDA fit to the iris dataset

Sometimes however, quadratic discriminant analysis is too flexible and yields an overfit to the data. In this case one can use *regularised* discriminant analysis (Friedman, 1989), a "compromise" between LDA and QDA. Regularised discriminant analysis replaces the estimator for $\mathbf{\Sigma}_l$ by

$$\hat{\mathbf{\Sigma}}_l^{RDA}(\alpha) = \alpha\hat{\mathbf{\Sigma}}_l + (1-\alpha)\hat{\mathbf{\Sigma}};$$

α is hereby from $[0, 1]$. For $\alpha = 0$ we obtain LDA, for $\alpha = 1$ we obtain QDA.

Couldn't we perform regression as well?

The log posteriori (4.4) in LDA is a linear function in $\mathbf{x}$. So one might ask why it would not be easier to estimate the parameters $\mathbf{a}_l$ and b_l directly using some sort of regression model. This would mean that we have to estimate less parameters.

Let's consider only the two-class case. It is enough to know the ratio $\frac{p(1|\mathbf{x})}{p(2|\mathbf{x})}$, as we classify an observation to class 1, if the ratio is greater than 1. Now using (4.4) we obtain

$$\begin{aligned}\log\frac{p(1|\mathbf{x})}{1-p(1|\mathbf{x})} &= \log\frac{p(1|\mathbf{x})}{p(2|\mathbf{x})} = \log p(1|\mathbf{x}) - \log p(2|\mathbf{x}) = \mathbf{x}'\mathbf{a}_1 + b_1 - \mathbf{x}'\mathbf{a}_2 - b_2 = \\ &= \mathbf{x}'\underbrace{(\mathbf{a}_1 - \mathbf{a}_2)}_{=:\boldsymbol{\beta}} + \underbrace{(b_1 - b_2)}_{=:\alpha} = \mathbf{x}'\boldsymbol{\beta} + \alpha\end{aligned}$$

This is the assumption of *logistic regression* (see chapter 5). We can even see quadratic discriminant analysis as a logistic model, however we must include as well all interactions and all squared covariates.

The difference between the two approaches is that *logistic regression* makes no assumption about the classwise distribution of $\mathbf{x}_i$ as it directly models the conditional likelihood given the $\mathbf{x}_i$.

4.5 Fisher's approach to LDA

Fisher's approach to LDA is not at all based on the decision theory from section 4.2; however he obtains the same results as in section 4.3 by looking for projections that separate the data best.

Consider first the case of two classes. We want to project the data onto a line, such that the classes are as separated as possible. A projection onto a line is mathematically defined over a vector $\mathbf{a}$ and the projected values are proportional to the linear combination $\mathbf{x}'\mathbf{a}$.

But how to characterise a good projection? Fisher proposed to use the projection that maximises the (squared) difference between the projected class means, however relative to the variances in the projected classes. Figure 4.4 visualises this idea.

We thus want to maximise

$$\frac{\text{Distance between the projected class means}}{\text{Variance of the projected data}} = \frac{\mathbf{a}'\mathbf{B}\mathbf{a}}{\mathbf{a}'\mathbf{W}\mathbf{a}}, \tag{4.7}$$

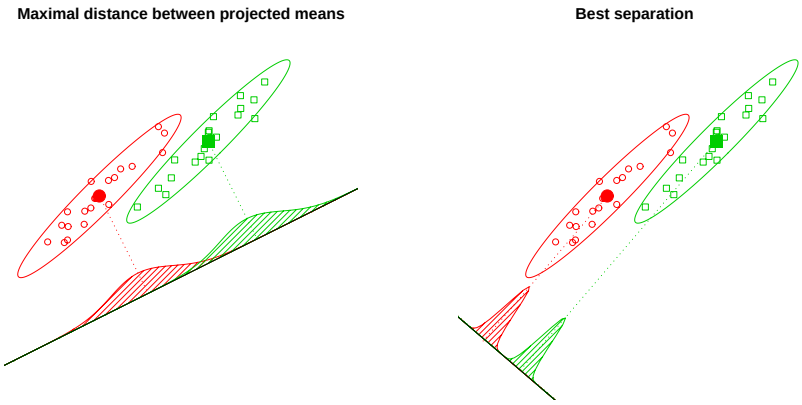


Figure 4.4: Maximising the difference between the projected means is not always the best way to seperate the data, the covariance structure has to be considered as well.

where

$$\mathbf{B} := \frac{1}{L-1}\sum_{l=1}^{L}(\bar{\mathbf{x}}_l - \bar{\mathbf{x}})(\bar{\mathbf{x}}_l - \bar{\mathbf{x}})'$$

is the between-class variance matrix, which measures the dispersion of the class means. When the data is spherical (i.e. the covariance matrix is proportional to the identity matrix), then we can omit the denominator. This leads to an equivalent approach to LDA: We first rescale the data that the within-class covariance matrix is spherical, i.e. the data of each class forms a ball. Then we project the data such that the projected class means have as much spread as possible.[4] Once we found the projection direction $\mathbf{a}$, we can give the LDA classification rule. Classify a new observation $\mathbf{x}_{new}$ into the class whose mean is closest to $\mathbf{x}_{new}$ in the *projected* space. We thus calculate the distance between $\mathbf{x}'_{new}\mathbf{a}$ and the $\bar{\mathbf{x}}'_l\mathbf{a}$.

For problems with more than two classes, only one projection might not be enough to separate the data. We can define additional projection directions $\mathbf{a}_2, \ldots, \mathbf{a}_{L-1}$ by maximising (4.7) over all $\mathbf{a}$ that are orthogonal in the metric implied by $\mathbf{W}$ to the previous projection directions. One can show that the approach of section 4.3 based on the normal distribution (and equal priors) is equivalent Fisher's LDA if all $L-1$ projection directions are used.[5] However, it can be useful in practice *not* to use all projection directions, as the projection directions of higher order can contain more noise than useful information about the classes. This type of LDA is sometimes referred to as *reduced rank LDA*. Figure 4.5 shows the decision boundaries obtained by fiiting a reduced rank LDA with rank 1 to the iris dataset.

One can show that Fisher's approach is equivalent to canonical correlation analysis and optimal scoring (see e.g. Mardia *et al.*, 1979). In optimal scoring one performs a regression from $\mathbf{X}$ to an artificial variable y, where y_i is set to a value θ_l determined by the class l the i-th observations belongs to. So not only the regression parameter $\boldsymbol{\beta}$, but also the class parameters $\theta_1, \ldots \theta_L$ have to be estimated. This view allows two generalisations of linear discriminant analysis:

[4]Note that LDA requires the within-class covariance matrix $\mathbf{W}$ to be of full rank.

[5]If the number of covariates is less than $L-1$, then it is of course enough to consider as many projections as covariates.

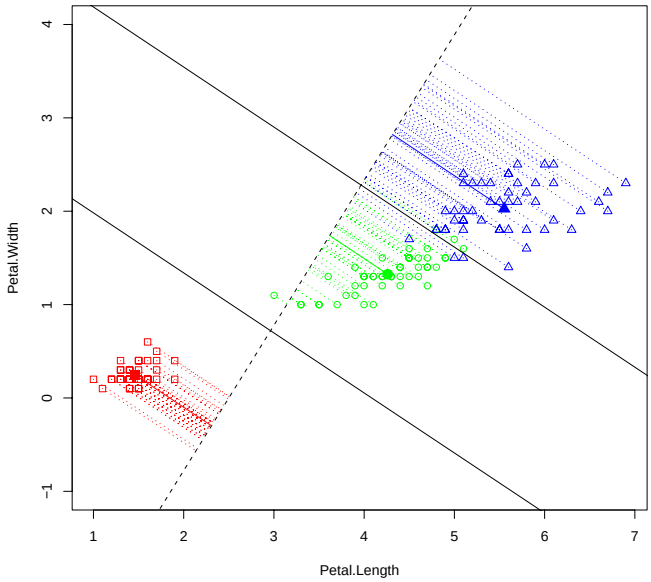


Figure 4.5: Reduced rank LDA (rank 1) fitted to the iris dataset. The dashed line is the projection line. The coloured solid lines are the projections of the class mean, and the solid black lines are the decision boundaries.

- When using penalised (ridge) regression[6] instead of linear regression one obtains *penalised discriminant analysis* (PDA) (Hastie *et al.*, 1995). PDA corresponds to using a regualised within-class covariance matrix

$$\mathbf{W}^{PDA} = \mathbf{W} + \lambda\mathbf{\Omega},$$

  where $\mathbf{\Omega}$ is a suitable penalisation matrix. The within-class covariance matrix thus does not need to be of full rank any more. This version of discriminant analysis is very useful when dealing with a huge amount of (highly correlated) covariates. PDA has been successfully applied to digit recignition (the initial problem of Hastie *et al.*), medicial imaging (see e.g. Kustra and Strother, 2001) and microarray analysis (see e.g. Kari *et al.*, 2003).

  Note that "plain" LDA is rather unsuited for microarray analysis.

- When using a non-parametric regression technique[7] one obtains *flexible discriminant analysis* (FDA) (Hastie *et al.*, 1994). FDA can be used when linear decision boundaries are too rigid and a more flexible discrimination rule is needed.

Despite being a quite simple and maybe old-fashioned technique, linear discriminant analysis is yet very powerful. In the STATLOG project (Michie *et al.*, 1994), which compared different classification techniques, LDA was amongst the top three classifiers for 7 of the 22 studied datasets.

[6] Ridge regression adds a penalisation term to the least-squares criterion and obtains thus a shrunken solution. It is mainly used when the predictors and (nearly) collinear.

[7] Nonparametric regression fits a smooth curve through the data, whereas linear regression fits a straight line.

4.6 Non-parametric estimation of the classwise densities

We have seen in section 4.2 that the (Bayes) optimal decision rule classifies $\mathbf{x}_i$ into the class l that maximises the posterior probability

$$p(l|\mathbf{x}_i) = \frac{\pi_l f(\mathbf{x}_i|l)}{\sum_{\lambda=-1}^{L} \pi_\lambda f(\mathbf{x}_i|\lambda)}. \tag{4.8}$$

For linear and quadratic discriminant analysis we assumed that these class-wise densities $f(\mathbf{x}_i|l)$ were Gaussians, once with common covariance matrix, once with different covariance matrices, and we estimated their means and covariances. However we can also estimate the densities $f(\mathbf{x}_i|l)$ non-parametrically, i.e. without assuming a certain type of distribution.

Non-parametric density estimation can be carried out in a variety of ways. Here, we will only consider *kernel density estimation*. Kernel density estimation estimates the density at $\mathbf{x} \in \mathbb{R}^p$ by

$$\hat{f}(\mathbf{x}|l) = \frac{1}{n_l h^p} \sum_{\mathbf{x}_i \text{ in class } l} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

K is hereby any bounded function with integral 1 and a peak at 0. A sensible choice for K is the density of the p-variate standard normal distribution. The *bandwidth* h controls how smooth the estimated density is: the bigger the chosen h is, the wigglier the estimated density gets.

Now we can estimate $p(l|\mathbf{x}_i)$ by plugging the estimates $\hat{f}(\mathbf{x}|\lambda)$ into (4.8):

$$\hat{p}(l|\mathbf{x}) = \frac{\pi_l \hat{f}(\mathbf{x}_i|l)}{\sum_{\lambda=-1}^{L} \pi_\lambda \hat{f}(\mathbf{x}_i|\lambda)} = \frac{\frac{\pi_l}{n_l} \sum_{\mathbf{x}_i \text{ in class } l} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{\lambda=1}^{L} \frac{\pi_\lambda}{n_\lambda} \sum_{\mathbf{x}_i \text{ in class } \lambda} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}. \tag{4.9}$$

When estimating the prior probabilities by $\hat{\pi}_l = n_l/n$ expression (4.9) simplifies to

$$\hat{p}(l|\mathbf{x}) = \frac{\sum_{\mathbf{x}_i \text{ in class } l} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}.$$

We classify a new observation $\mathbf{x}_{new}$ into the class l with highest posterior probability $\hat{p}(l|\mathbf{x}_{new})$.

Figure 4.6 shows the decision boundaries when estimating the classwise densities in the iris dataset non-parametrically.

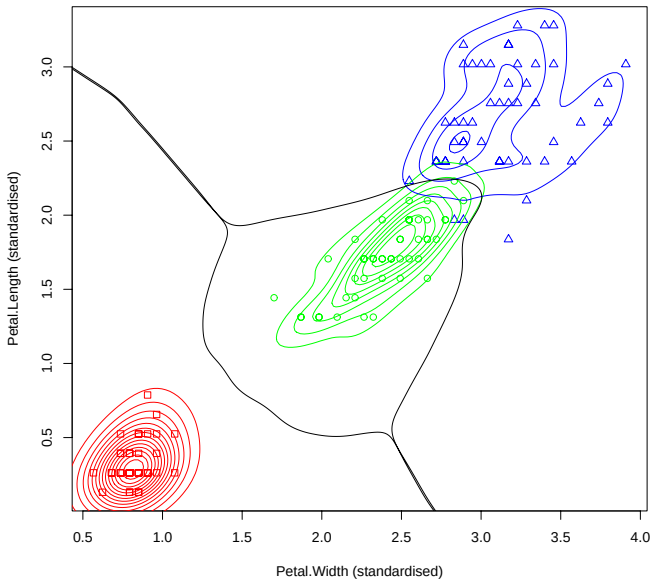


Figure 4.6: Classwise densities and decision boundaries obtained when extimating the classwise densities nonparametrically

Chapter 5

Logistic Regression

5.1 Introduction

Statistical regression models for binary response variables are of great importance in many fields of application. The categories for binary response variables are typically coded zero or one. For example, let Y be a response that indicates if subjects developed lungcancer or not. Subjects who developed lung cancer will be coded with a one and the remainder with a zero. The mean of Y is a probability in this case that measures the probability of developing lung cancer. Regression models for binary responses are used to describe probabilities as functions of explanatory variables.

Linear regression models are not suited to binary responses as they allow responses outside the inteval $[0, 1]$, which is not permitted for probabilities. Logistic regression provides a satisfactory alternative for such problems. It describes a function of mean (which is a probability here) as a function of the explanatory variables. The fuction of mean it uses is the logit function or the logarithm of the odds.

The estimated coefficients from a logistic regression fit are interpreted in terms of odds and odds ratios, where these coefficients are estimated using maximum likelihood methods.

A logistic regression model can be extended to problems where we wish to model the posterior probabilities of L classes via linear functions in the explanatory variables. We will define the model for both the binary and L class setup. However, we will mainly focus on binary responses here and only consider the L class setup as a direct extention.

5.2 The logistic regression model

The logistic regression model is part of a wider class of models called generalised linear models (GLMs). A generalised linear model is a probability model in which the mean of a response variable is related to explanatory variables through a regression equation. Let $\mu = \mu\{Y|X_1, \ldots, X_p\}$ be the mean response. Then some specified function of the mean, μ, is related to a linear funtion of the explanory variables (where we mean a function that is linear in the parameters $\beta_1, \ldots, \beta_p$):

$$g(\mu) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

The function g is called the *link function*. The choice of link function depends on the distribution of the response variable. For example, in the case of a normal response variable the identity link,$g(\mu) =$

μ, is appropriate, leading to ordinary multiple regression. The natural link for a binary response variable is the *logit* (or log-odds) function. In this case it is conventional to use π to signify the mean mean response, so that it is emphasised that it is a probability. The logit link is

$$g(\pi) = \text{logit}(\pi) = \log\left(\frac{\pi}{1-\pi}\right)$$

so that the logistic regression model is

$$\text{logit}(\pi) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

The inverse of this function is called the *logistic function*. So if $\text{logit}(\pi) = \eta$, then

$$\pi = \exp(\eta)/[1 + \exp(\eta)]$$

Furthermore, note that in ordinary regression the variance, $\texttt{Var}\{Y|X_1, \ldots, X_p\} = \sigma^2$, is constant and does not depend on the values of the explanatory variables. However, the variance of a population of binary response variables with mean π is $\pi(1-\pi)$.

Logistic Regression Model

$$\mu = \mu\{Y|X_1, \ldots, X_p\} = \pi; \quad \text{Var}\{Y|X_1, \ldots, X_p\} = \pi(1-\pi)$$
$$\text{logit}(\pi) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

5.3 The logistic regression model for L classes

When we have L classes we can specify the logistic regression model in terms of $L-1$ logit transformations. Although the model uses the last class as the denominator or baseline in the definition below, the choice of denominator is arbitrary in that the estimates are equivariant under this choice.

Logistic Regression Model for L classes

$$\log\frac{G=1|X_1,\ldots,X_p}{G=L|X_1,\ldots,X_p} = \beta_{10} + \beta_{11}X_1 + \cdots + \beta_{1p}X_p$$
$$\log\frac{G=2|X_1,\ldots,X_p}{G=L|X_1,\ldots,X_p} = \beta_{20} + \beta_{21}X_1 + \cdots + \beta_{2p}X_p$$
$$\vdots$$
$$\log\frac{G=L-1|X_1,\ldots,X_p}{G=L|X_1,\ldots,X_p} = \beta_{L-1,0} + \beta_{L-1,1}X_1 + \cdots + \beta_{L-1,p}X_p$$

When $L = 2$ we have the simple binary logistic regresion model defined before.

5.4 Interpretation and estimation of coefficients

In the logistic regression model the odds for a positive response (i.e. $Y = 1$) given $X_1, \ldots, X_p$ are

Interpreting the coefficients

$$\text{Odds that } Y = 1 : \omega = \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p)$$

Odds ratio

The ratio of the odds at $X_1 = A$ relative to the odds at $X_1 = B$, for fixed values of the other X's, is

$$\omega_A/\omega_B = \exp[\beta_1(A - B)]$$

Thus, if X_1 increases by 1 unit, the odds that $Y = 1$ will change by a multiplicative factor of $\exp(\beta_1)$, other variables being the same.

In generalised linear models the method of *maximum likelihood* is used to estimate the parameters. When the values of the parameters are given, the logistic regression model specifies how to calculate the probability that a specific outcome will occur (e.g. $y_1 = 1$, $y_2 = 0$, $y_3 = 1$ etc.). Suppose that there are n independent responses, with π_i the mean response for observation i then

$$P(Y_1 = y_1, \ldots, Y_n = y_n) = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi)^{1 - y_i}$$

for the outcomes $y_1, \ldots, y_n$. This depends on the parameters through π, so we can compute the probability of the observed outcome under various possible choices for the β's. This is known as the *likelihood function*. Below we give a simple description of maximum likelihood estimation without going into technical details.

Likelihood

The *likelihood* of any specified values for the parameters is the probability of the actual outcome, calculated with those parameter values.

Maximum Likelihood Estimates

The estimates of the parameters that assign the highest probability to the observed outcome (which maximise the likelihood) are the *maximum likelihood estimates*.

Properties of Maximum Likelihood Estimates

If the model is correct and the sample size is large enough, then:

1. The maximum likelihood estimators are essentially unbiased

2. Standard deviations of the sampling distributions of the estimators can easily be obtained
3. The estimators are about as precise as any nearly unbiased estimators that can be obtained
4. The sampling distributions are nearly normal.

The properties of the estimates imply that each estimated coefficient β_j in the logistic regression has an approximately normal sampling distribution.

$$\text{Z-ratio} = (\hat{\beta}_j - \beta_j)/\text{SE}(\hat{\beta}_j) \sim \text{Standard Normal (approx)}$$

This can be used to construct confidence intervals or to obtain p-values for tests about the individual coefficients. A test based on the above approximate normality is referred to as *Wald's test*.

5.5 The Drop-in-Deviance Test

It is often of interest to judge the adequacy of a *reduced model* to the *full model*. The reduced model is obtained from the full model by excluding some of the explanatory variables from the full set of explanatory variables. This corresponds to the hypothesis that several of the coefficients in the full model equal zero. We now need a measure for comparing such a reduced model to the full model inculding all the explanatory variables. We can use a likelihood ratio test sometimes known as the drop-in-deviance test for this purpose.

Above we mentioned that the estimated maximum likelihood parameter estimates for a logistic regression model maximse the corresponding likelihood for the model. If we now represent the maximum of the likelihood for the full model by $LMAX_{\text{full}}$ and the maximum of the likelihood of the reduced model by $LMAX_{\text{reduced}}$, then the general form of the *likelihood ratio test statistic* is

$$LRT = 2\log(LMAX_{\text{full}}) - 2\log(LMAX_{\text{reduced}})$$

Under the null hypothesis (i.e. if the reduced model is correct), the LRT has approximately a chi-square distribution with degrees of freedom equal to the difference between the numbers of parameters in the full and reduced models. Inadequacies of the reduced model tend to inflate the LRT statistic, so that we will reject the reduced model. Note that the LRT depends on the mathematical form of the likelihood function, which we will not present here. A quantity called the *deviance* is often presented in software packages, rather than the maximum likelihood, where

$$\text{deviance} = \text{constant} - 2\log(LMAX)$$

and the constant is the same for the full and reduced models. It follows that

$$LRT = deviance_{\text{reduced}} - deviance_{\text{full}}$$

This represent the drop in the deviance due to the additional parameters included in the full model. For this reason the likelihood ratio test is often referred to as the *drop-in-deviance test*. This test can for example be used to test whether all coefficients apart from the constant term are zero taking the reduced model as the one with no explanatory variables. Similarly, we can test for the significance of individual coefficients by taking as the reduced model the full model minus the single term that we are testing for (note that this is not the same as Wald's test).

5.6 Some General Comments

5.6.1 Graphical methods

One can construct preliminary logistic regression models that include polynomial terms and/or interaction terms to judge whether the desired model needs expansion to compensate for very specific problems.

5.6.2 Model selection

The range of possible models can often be very large. An Information Criterion (AIC) is a model selection device that emphasises parsimony by penalising models for having large numbers of parameters. This can be used as a guide for choosing an appropriate model.

5.7 Discriminant Analysis and Logistic Regression

Logistic regression might be used to predict future binary responses, by fitting a logistic rgression model to a training set. This model can then be applied to a set of explanatory varibles for which the binary response is unknown. The estimated probability π obtained from the logistic regression model might be used as an estimate of the probability for a "positive" response.

Logistic regression prediction problems are similar in structure to problems in discriminant analysis. In fact, it can be shown that they can be written in exactly the same form (see Hastie *et al.* (2001), $p.103-104$). The difference between the approaches lies in the way the coefficients are estimated. The logistic regression model is more general in that it makes less asssumptions. In discriminant analysis a parametric form is assumed for the marginal density of X. Observations in each group are assumed to follow a gaussian distribution. In the case of logistic regression we can think of this marginal density as being estimated in a fully nonparametrical and unrestricted fashion, using the empirical distribution function which places a mass of $1/N$ at each observation.

What difference does the additional assumptions make between the approaches? The additional model assumptions for discriminant analysis has the advantage that it gives us more information about the parameters, which means that we estimate them more efficiently (lower variance). However, discriminant analysis can also be badly affected by outliers and the distributional assumptions may not hold. Logistic regression prediction problems are perhaps preferable to discriminant analysis when the explanatory variables have nonnormal distributions (e.g. when they are categorical). There is

no general answer to which method is to be preferred. Logistic regression requires less parameters (is thus less likely to be unstable) and the coefficients α and β are nicely interpretable (see chapter 5). However, if the assumption that the classwise densities are Gaussians is true, then discriminant analysis is more efficient, i.e. less observations are necessary to get the same precision as compared to logistic regression (Efron, 1975). Furthermore discriminant analysis can handle completely separated datasets. Logistic regression is nearly as efficient discriminant analysis when the explanatory variables do follow normal distributions. Therefore, when discrimination between two groups is the goal, logistic regression should always be considered as an appropriate tool.

Hastie *et al.* (2001) report that it is generally felt that logistic regression is a safer, more robust approach than discriminant analysis. In their experience the two models give very similar results though, even when discriminant analysis is used inappropriately (e.g. with categorical explanatory variables). For a more detailed discussion on logistic discriminantion refer to Ripley (1996) and Hastie *et al.* (2001).

5.8 Example: Birdkeeping and lung cancer

5.8.1 Background

Ramsey and Schafer (2002) introduce data based on that of Holst *et al.* (1988). This data are from a study that investigate birdkeeping as a risk factor for developing lung cancer. The study is based on a case-control study of patients in 1985 at four hospitals in The Hague, Netherlands. They identified 49 cases of lung cancer among patients who were registered with a general practice, who were age 65 or younger, and who had resided in the city since 1965. They also selected 98 controls from a population of residents having the same general age structure. Data were gathered on the following variables for each subject:

```
FM = Sex (1=F,0=M)
AG = Age, in years
SS = Sosioeconomic status (1=High,0=Low), based on wages
     of the principal wage earner in the household
YR = Years of smoking prior to diagnosis or examination
CD = Average rate of smoking, in cigarettes per day
BK = Indicator of birdkeeping (caged birds in the home for more than
     6 consecutive months from 5 to 14 years before diagnosis (cases) or
     examination (controls)
```

It is known that smoking and age are both associated with lung cancer incidence. The research question of interest is the following: after age, socioeconomic status and smoking have been controlled for, is an additional risk associated with birdkeeping?

5.8.2 Data Exploration

In figure 5.1 we present a coded scatterplot of of the number of years that a subject smoked, versus the individual's age. From the plot it appears as though for individuals of similar ages, the persons with lung cancer are more likely to have been relatively long-time smokers. To investigate the effect

of birdkeeping we can compare subjects with similar smoking history and ask whether more of the birdkeepers amongst them tend to have cancer. Do you observe any trends here?

We construct a further exploratory plot by plotting the sample logits, $\log[\hat{\pi}/(1-\hat{\pi}))]$, against the number of years of smoking. In this plot $\hat{\pi}$ is the proportion of lung cancer patients among subjects grouped into similar years of smoking $(0, 1-20, 21-30, 31-40, 41-50)$. The midpoint of the interval was used to represent the years of smoking for the members of the group. The resulting plot is given in figure 5.2. Does the logit increase linearly with years of smoking? How can we test this more formally?

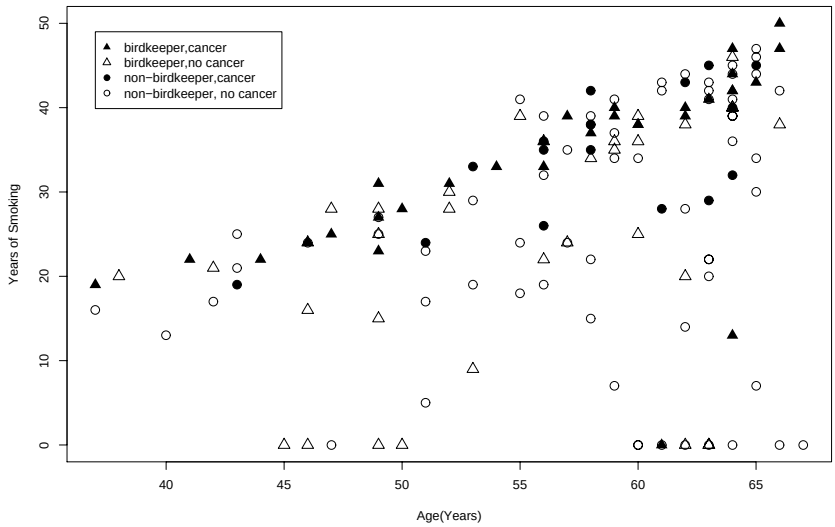


Figure 5.1: Coded scatterplot of years of smoking versus age of subject

5.8.3 Finding a suitable model

We approach the problem by first fitting a fairly rich model to the data. We include years of smoking, birdkeeping, cigarettes per day, and squares and interaction of these in the model. One approach to obtain the simplest possible model that fits the data well is to perform a backward elimination of terms based on the AIC. Below we present the output from the full model.

```
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.2776698  2.3858069   0.116 0.907348
SEXMALE      0.6155883  0.5450581   1.129 0.258729
SSLOW        0.1865263  0.4746561   0.393 0.694341
AG           0.0381615  0.0398243   0.958 0.337938
YR          -0.0283392  0.0961924  -0.295 0.768292
CD          -0.1500281  0.1058075  -1.418 0.156210
BKNOBIRD     1.3795187  0.4123760   3.345 0.000822
I(YR^2)     -0.0009586  0.0019860  -0.483 0.629310
```

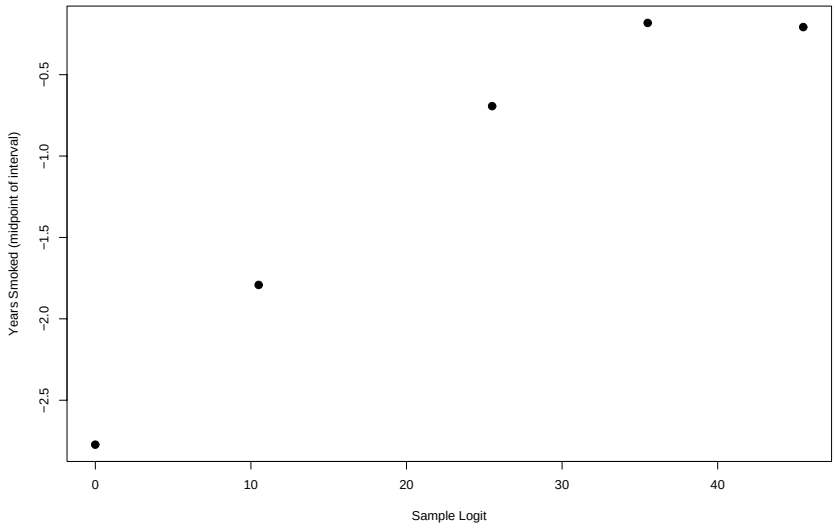


Figure 5.2: Plot of sample logits versus years smoking for data grouped by intervals (see text) of years of smoking

```
I(CD^2)      0.0020067  0.0021790   0.921 0.357094
YR:CD        0.0011864  0.0024376   0.487 0.626455
```

After a stepwise model selection step based on AIC we obtain a much simplified model. The R output for this model selection procedure are given below. We see that all terms cans be dropped except for the YR and BK terms.

```
Stepwise Model Path
Analysis of Deviance Table

Initial Model:
LC ~ SEX + SS + AG + YR + CD + BK + I(YR^2) + I(CD^2) + YR *
    CD

Final Model:
LC ~ YR + BK


       Step Df  Deviance Resid. Df Resid. Dev      AIC
1                              137   152.5932 172.5932
2      - SS  1 0.1540821       138   152.7473 170.7473
3   - YR:CD  1 0.2366595       139   152.9839 168.9839
4 - I(YR^2)  1 0.0724717       140   153.0564 167.0564
5      - AG  1 0.7160968       141   153.7725 165.7725
6     - SEX  1 1.0987324       142   154.8712 164.8712
```

```
7 - I(CD^2)  1 1.8761359       143   156.7474 164.7474
8      - CD  1 1.3670333       144   158.1144 164.1144

Coefficients:
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  1.70460392 0.56266766  3.029504 0.0024495567
YR          -0.05824972 0.01684527 -3.457927 0.0005443496
BKNOBIRD     1.47555151 0.39587626  3.727305 0.0001935383
```

5.8.4 Interpretation

The estimate for the coefficient of birdkeeping is 1.4756, so the odds of lung cancer for birdkeepers are estimated to be $\exp(1.4756) = 4.37$ times as large as the corresponding odds for non-birdkeepers. This is after the other explanatory variables are accounted for. A 95% confidence interval for the odds of lung cancer for birdkeepers relative to the odds of lung cancer for non-birdkeepers is $[2.01, 9.50]$.

Chapter 6

Non-parametric classification tools

6.1 k-nearest neighbours

A very simple, but rather powerful approach to classification is the *nearest-neighbour* rule. It classifies a new observation $\mathbf{x}_{new}$ into the class of its nearest neighbour. This rule is however very sensitive to outliers. However it can be easily generalised to the *k-nearest neighbour (k-nn) rule*. The k-nn rule determines the k nearest neighbours[1] [2] and performs a majority vote amongst the k nearest neighbours. Sometimes it is useful to include a "doubt" option, if the proportion of the winning class is less then a specified threshold; in this case no prediction is possible at that location.

There is another way of seeing the k-nn algorithm. For every point we estimate the posterior probability $p(l|\mathbf{x})$ that $\mathbf{x}_i$ is in class l by the proportion of the observations of class l amongst the neighbours of $\mathbf{x}_i$. We then classify $\mathbf{x}_i$ into the class with the highest posterior probability.

The idea behind this is that the neighbours are close to $\mathbf{x}_i$ and thus have similar posterior probabilities. Increasing the value of k thus reduces the variance of the classifier, as we use more observations to calculate the posterior probability (law of large numbers!). Increasing the value of k however increases the bias as we use examples that are more and more distant from $\mathbf{x}_i$ and thus might have different posterior probabilities. We have to make a *bias-variance trade-off*. Figure 6.1 shows the decision boundaries obtained by applying the k-nn algorithm with different values of k to the quarter-circle dataset.

Normally, the Euclidean distance is used to determine the nearest neighbours. Note that this means that the variables have to be rescaled appropriately before the k-nn algorithm is applied. However, k-nn works well with other metrics as well. For digit recognition *invariant metrics* are used. Rotation invariant metrics remove the effect of rotation (handwriting can be more or less slanted) before distances are calculated. (see e.g. Simard et al. 1993).

Note that the k-nn rule is susceptible to the *curse of dimensionality*: for high-dimensional data, the "nearest" neighbours can be rather far away. This causes the bias to increase, as we average over wide areas. One way to deal with this problem is to use adaptive metrics like the *discriminant adaptive nearest-neighbour* (DANN) (Hastie & Tibshirani, 1996) rule. The basic idea behind this approach is to stretch out the neighbourhood into directions for which the class probabilities don't change much.

Note that k-nn requires the *whole* training dataset to be stored in order to be able to classify new

[1] The k nearest neighbours are the k training points $\mathbf{x}_{i_j}$ with the lowest distance $\|\mathbf{x}_{i_j} - \mathbf{x}_{new}\|$

[2] In case of a tie in the distance, one can either pick k points at random or use all points not further away than the k-th nearest neighbour.

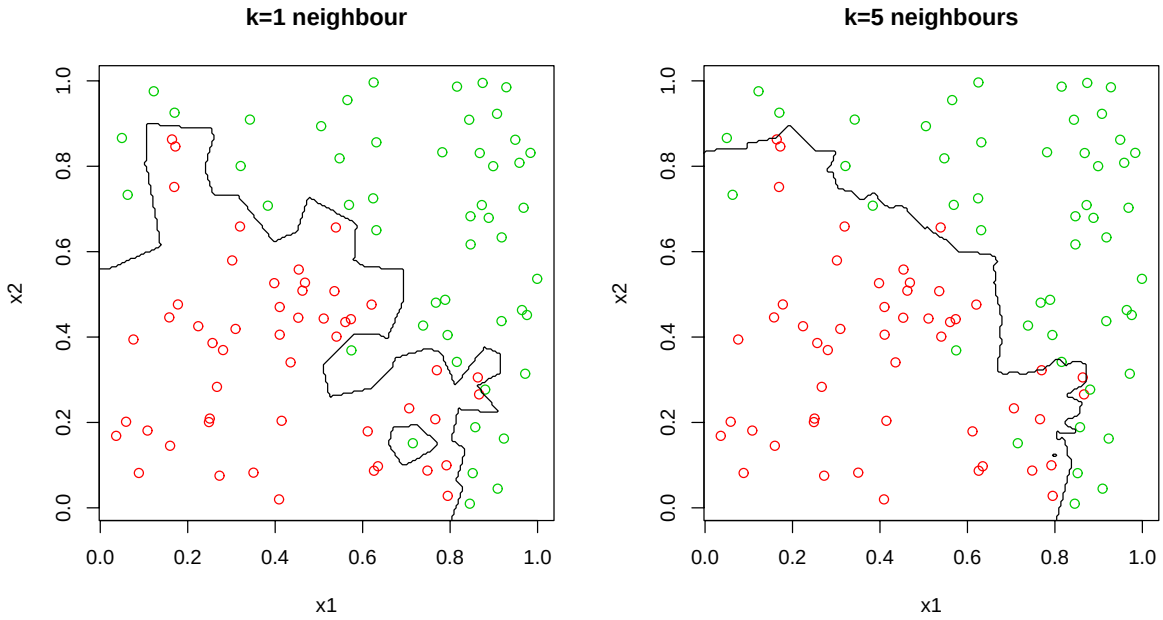


Figure 6.1: Decision boundaries obtained using the k-nn rule once with $k = 1$ neighbour and once with $k = 5$ neighbours

observations. This can pose quite a serious problem, as training datasets can exceed the size of several giga byte. Various *data editing* techniques have been proposed to deal with this matter. Their aim is to find a subset of training examples that suffices for prediction and to discard the remaining part of the training set. (see e.g. Devijver and Kittler, 1982)

Table 6.1 gives the error rates of the k-nn algorihtm applied to the breast tumour dataset, once using just the "best" 30 genes, once using the "best" $4,000$ genes.

	Test set error rate
$k = 1$ neighbour	0.33
$k = 10$ neighbours	0.22
$k = 20$ neighbours	0.25

(a) 30 genes

	Test set error rate
$k = 1$ neighbour	0.33
$k = 20$ neighbours	0.17
$k = 30$ neighbours	0.44

(b) $4,000$ genes

Table 6.1: Test error rate for the k-nn algorithm with different numbers of neighbours lengths applied to the breast tumour dataset

6.2 Learning vector quantisation

For the k-nn classifier we need to store the whole training dataset in order to be able to predict (unless we use data editing techniques). The idea of learning vector quantisation (LVQ) (Kohonen, 1988) is to store only "typical examples" of each class. Like in vector quantisation we look for an optimal codebook, but now we look for optimal prototypes for each class.

Consider once again a problem with L classes. We want to find R prototypes per class. Let $\boldsymbol{\xi}_{lr}$ denote the r-th prototype of the l-th class.

The LVQ algorithm is an *on-line algorithm*, i.e. the training data is presented repeatedly on a one by once basis.[3] Every time a training point $\mathbf{x}_i$ is presented, one determines the closest prototype $\boldsymbol{\xi}_{lr}$. If this prototype is of the same class as $\mathbf{x}_i$ then it is moved closer to $\mathbf{x}_i$ by updating

$$\boldsymbol{\xi}_{lr} \to \boldsymbol{\xi}_{lr} - \alpha \cdot (\boldsymbol{\xi}_{lr} - \mathbf{x}_i).$$

Is the prototype $\boldsymbol{\xi}_{lr}$ however not of the same class as $\mathbf{x}_i$, then it is moved away from $\mathbf{x}_i$ by updating

$$\boldsymbol{\xi}_{lr} \to \boldsymbol{\xi}_{lr} + \alpha \cdot (\boldsymbol{\xi}_{lr} - \mathbf{x}_i).$$

The idea of the LVQ algorithm is that prototypes of the correct class are "attracted", whereas the wrong prototypes are "repelled". α is hereby the learning rate and governs how much the prototypes are moved. During the iteration process the learning rate α is decreased to 0. Figure 6.2 illustrates the run of the algorithm.

In practice it is useful to use different and adaptive learning rates for each prototype and compute the current learning rate $\alpha_{lr}^{(h)}$ for the r-th prototype of the l-th class from its last value $\alpha_{lr}(h-1)$ by setting[4]

$$\alpha_{lr}^{(t)} = \begin{cases} \frac{\alpha_{lr}^{(h-1)}}{1+\alpha_{lr}^{(h-1)}} & \text{if the current training point is correctly classified} \\ \frac{\alpha_{lr}^{(h-1)}}{1-\alpha_{lr}^{(h-1)}} & \text{if the current training point is incorrectly classified.} \end{cases} \tag{6.1}$$

The idea behind this approach is that the learning rate should be *increased* if there are still data points of the wrong class close to the prototype. This causes the algorithm to converge faster away from these "wrong" points. On the other hand, if the example is correctly classified once can assume that the prototype is in a "good" neighbourhood, so the learning rate can be decreased.

Table 6.2 gives the results of applying the learning vector quantisation algorithm to the breast tumour dataset. Adaptive learnings rates as in (6.1) were used.

6.3 Neural networks

Neural networks were initially proposed in the 1960s (Widrow and Hoff, 1960; Rosenblatt, 1962) and received a lot of attention in the mid-1980s (see e.g. Rumelhart and McClelland, 1986). Statistically speaking, neural networks are two-level nonlinear models. In this section we will only cover feed-forward neural networks with one hidden layer, the most popular "vanilla" neural network.

[3] The LVQ algorithm described here is the basic LVQ1 algorithm from Kohonen. For more sophisticated versions like LVQ2.1 and LVQ3 see Ripley (1996).

[4] This variant of the LVQ1 algorithm is often called OLVQ1.

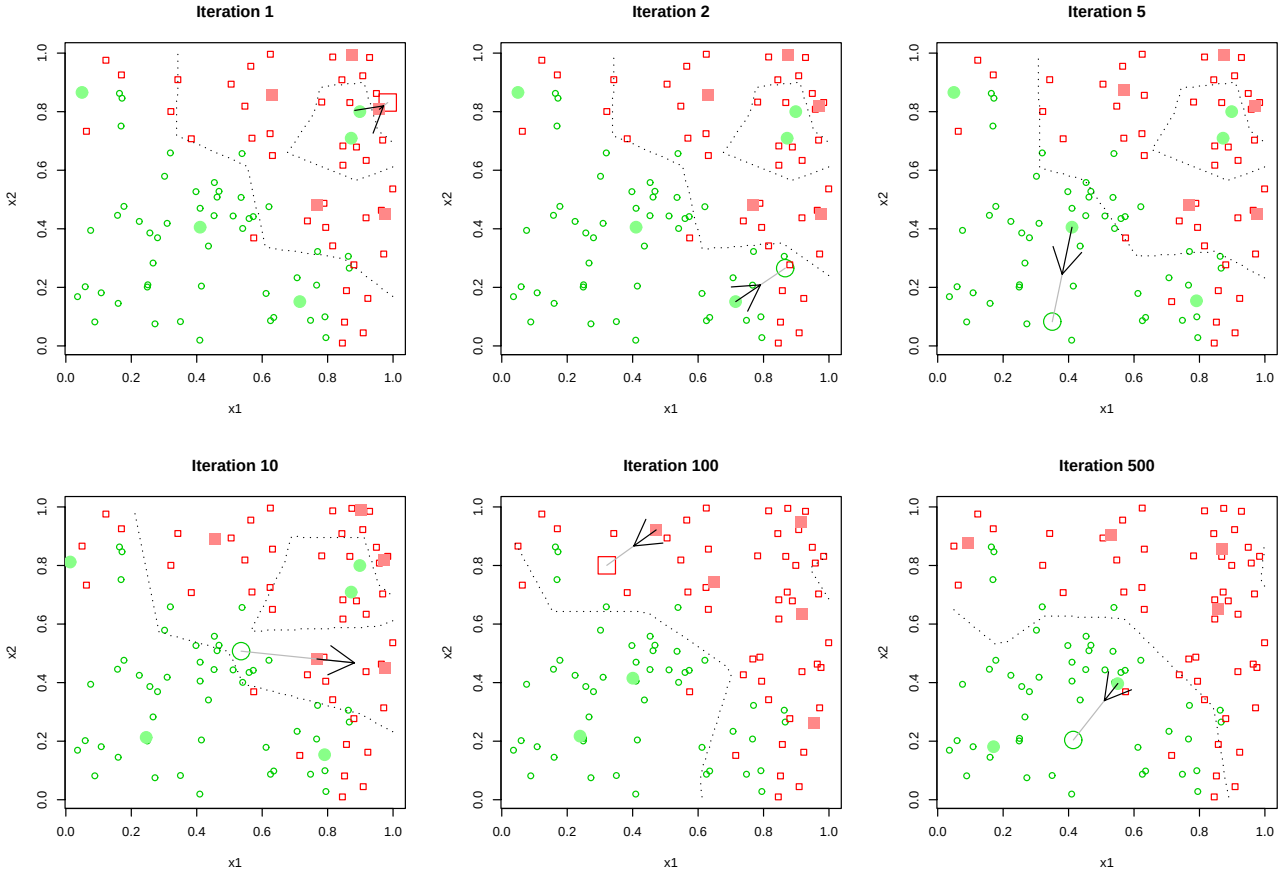


Figure 6.2: Iterations of the LVQ algorithm for the quarter-circle dataset. The empty points represent the data, the big empty point the data point currently presented. The filled dots correspond to the codebook. The closest code vector is linked to the data point presented by a grey line and an arrow shows how this prototype is changed. The dotted line is the decision boundary

	Error rate	
	Training set	Test set
5 prototypes per class	0.075	0.25
10 prototypes per class	0.075	0.17
15 prototypes per class	0.075	0.25

(a) 30 genes

	Error rate	
	Training set	Test set
5 prototypes per class	0.35	0.42
10 prototypes per class	0.125	0.19
15 prototypes per class	0.1	0.25

(b) $4,000$ genes

Table 6.2: Training and test error rate for the LVQ algorithm with different codebook lengths applied to the breast tumour dataset

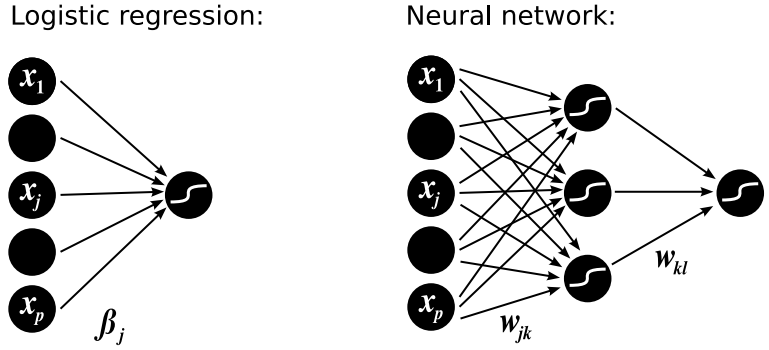


Figure 6.3: Network diagram of a neural network (right) and corresponding diagram for logistic regression (left)

Recall the logistic regression model from chapter 5: a new observation was classified to one group if

$$\phi(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) > \frac{1}{2},$$

where ϕ was the logistic function. The left hand side of figure 6.3 gives a visualisation of this model. Neural networks introduce an additional layer (see the right hand side of that figure) that allows to model more complex decision functions. Logistic regression leads to a decision boundary that is a hyperplane: neural networks can lead to more sophisticated decision boundaries (see figure 6.4).

Neural networks as typically represented by a *network diagram* like in in figure 6.3. In feed-forward neural networks "signals" are only passed towards the output layer and not backwards. The input layer corresponds to the covariates in the regression context. These input signals are then "transmitted" to the units in the hidden layer. At each of these units all incoming signals are summed up using weights w_{jk} and after adding a constant α_k they are transformed by a function ϕ_k ("activation function"). The output of the hidden layer units is then used as input for the output layer, leading to the transformation

$$\hat{y}_{il} = \phi_l \left(\alpha_l + \sum_{k \to l} w_{kl} \phi_k \left(\alpha_k + \sum_{j \to k} w_{jk} x_{ij} \right) \right) \tag{6.2}$$

The most common activation function for the hidden layer is the logistic function $\frac{e^t}{1+e^t}$. For the output units mostly either no transformation ("linear unit"), the logistic function, the threshold function[5] or the "softmax" function (see (6.5) below) is used.

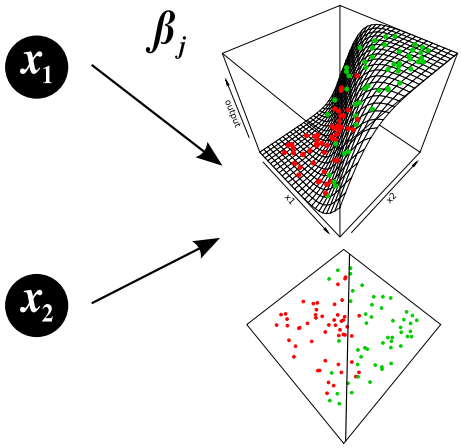
Note that neural networks can have more than one output node. For two-class classification problems however one output node is sufficient. In all other cases an output unit per class is used. A *formal* definition of a neural network can be found in the glossary of Ripley (1996).

The weights w_{jk} (and w_{kl})[6] are chosen such that the output $\hat{y}_{il}$ from (6.2) is "closest" to the observed value y_{il}. For a two-class classification problem $y_i = 1$ if the i-th observation is in one of the classes, and $y_i = 0$ otherwise.[7] If there are more than two classes one has to use one output unit per class, i.e. $\mathbf{y}_i = (y_{i1}, \ldots, y_{iL})$, where $y_{il} = 1$ if the i-th observation is in class l and 0 otherwise.

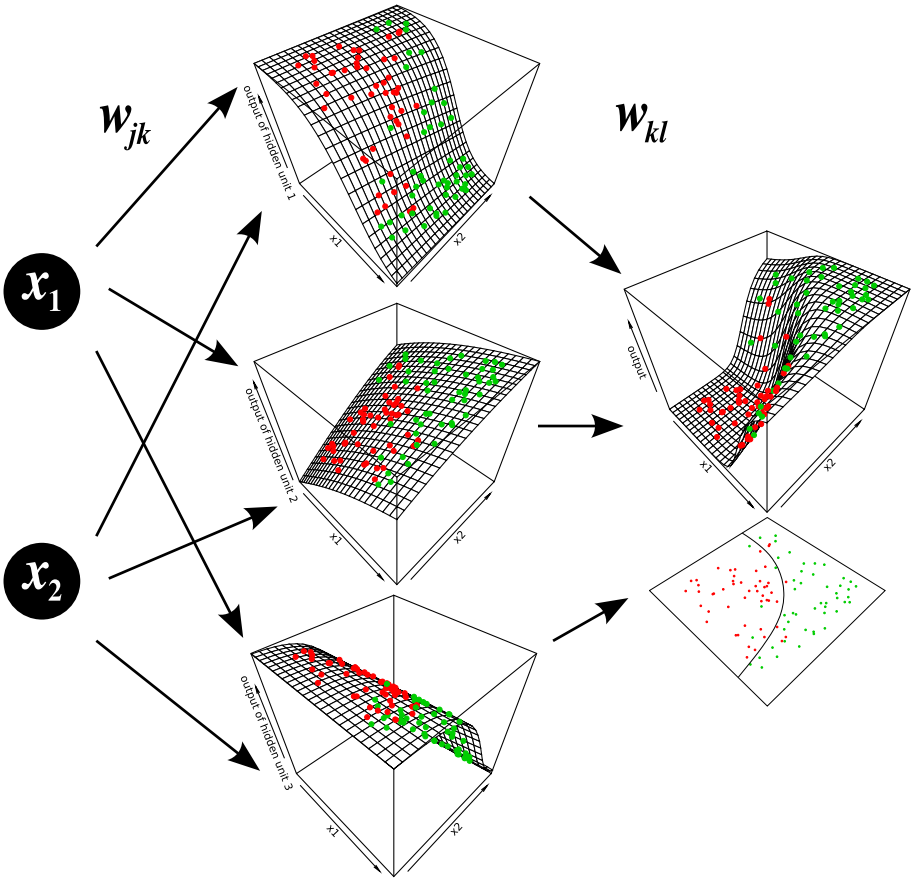
[5] The threshold function is 1 if the argument is positive, otherwise it is 0.

[6] In the following "w_{jk}" refers to all weights.

[7] The second index l of y_{il} can be dropped as there is only one output unit (always $l = 1$).



(a) Fitted values and decision boundary for a logistic regression fit to the quarter-circle data.



(b) Output of the hidden layer, (transformed) output of the output layer(fitted values) and decision boundary for a neural network fit to the quarter-circle data

Figure 6.4: Logistic regression (above) and neural network (below) compared

The weights w_{jk} are thus chosen to minimise a fitting criterion R like

$$R(\mathbf{w}) = \sum_{i=1}^{n} \sum_{l=1}^{L} (y_{il} - \hat{y}_{il})^2, \tag{6.3}$$

which corresponds to fitting the weights by least squares, a fitting technique mainly used for regression settings.[8]

For a two-class problem one can use the (conditional) log-likelihood

$$R(\mathbf{w}) = \sum_{i=1}^{n} \left(y_i \log \left(\frac{y_i}{\hat{y}_i} \right) + (1 - y_i) \log \left(\frac{1 - y_i}{1 - \hat{y}_i} \right) \right); \tag{6.4}$$

in this case the model is a logistic regression model in the hidden units and the weights are estimated by maximum-likelihood. For more than two classes one mostly uses the "softmax" activation function, which is given by

$$\frac{e^{t_l}}{\sum_{\lambda=1}^{L} e^{t_\lambda}}, \tag{6.5}$$

where t_l is the input to the current output unit, and t_λ ($\lambda = 1, \ldots, L$) is the input to the other output units. As a fitting criterion one then uses the cross-entropy

$$R(\mathbf{w}) = \sum_{i=1}^{n} \sum_{l=1}^{L} -y_{il} \log(\hat{y}_{il}). \tag{6.6}$$

Once again this leads to a model which is a multiple logistic regression model in the hidden units.[9] When using the fitting criteria (6.4) or (6.6) one can interpret the output $\hat{y}_{il}$ as the predicted probability that the i-th observation is from class l.

The fitting criterion is usually minimised by gradient descent, i.e.

$$w_{jk} \rightarrow w_{jk} - \eta \frac{\partial R}{\partial w_{jk}} \tag{6.7}$$

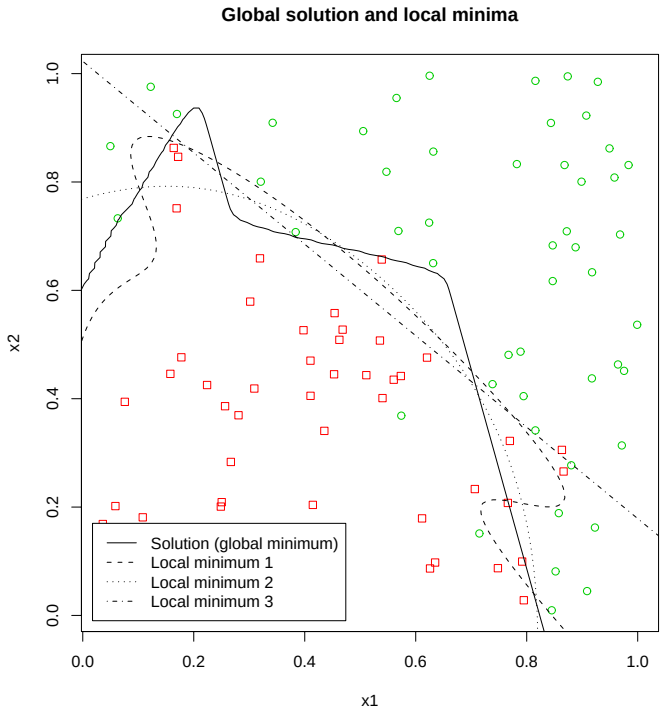
These derivatives are calculated from the output nodes to the input nodes across the network (chain rule!). This gave the algorithm the name of *back-propagation* (details can be found in Ripley, 1996). Contrary to logistic regression, we are *not* guaranteed that the algorithm converges to a *global* minimum. Depending on the initial choice of the weights the algorithm can get "stuck" in a local minimum. Figure 6.5 shows the results of fitting a neural network to the artificial quarter-circle dataset with different starting values. One can see that neural networks are very flexible tools[10], in fact they are *too* flexible in many cases. The "optimal" solution in our example corresponds to an overfit to the data; compared to the other solutions it leads to a *higher* error rate in the test set (well despite leading to the *lowest* error rate in the training set).

The solution which is optimal with respect to the fitting criteria (6.3), (6.4) or (6.6) is often *not* what we are looking for, as it can lead to a tremendous overfit to the data. This can be solved by

[8] Note that this is a sum both over observations and over output unit.

[9] For a two-class problem this approach (using *two* output nodes) is equivalent to the criterion (6.4).

[10] In fact, any continuous decision boundary can be modelled by a neural network, if enough hidden units are used.



	Training data		Test data
	Fitting criterion	Error rate	Error rate
Global solution	16.3520	0.05	0.14
Local minimum 1	18.2518	0.08	0.13
Local minimum 2	23.1583	0.10	0.10
Local minimum 3	27.2862	0.12	0.12

Figure 6.5: Optimal solution and local minima (decision boundaries) for the quarter-circle data (3 hidden units, no weight decay)

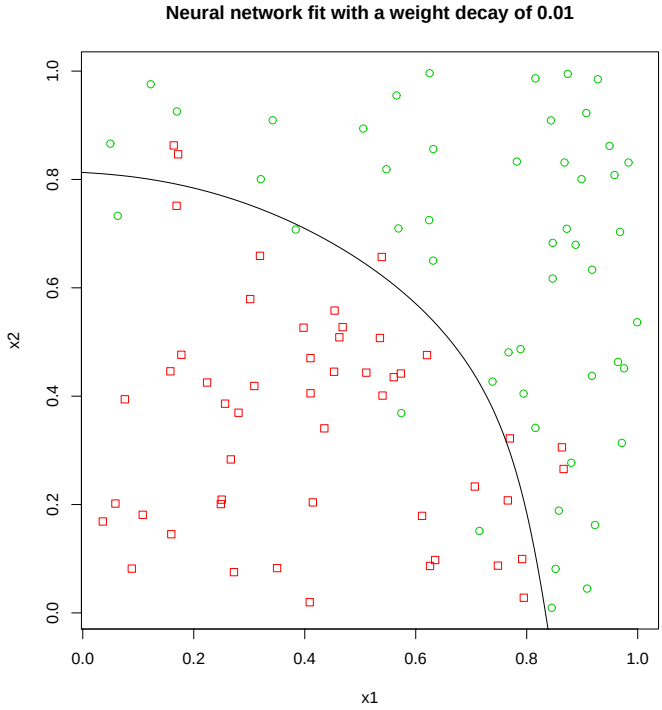


Figure 6.6: Solution (decision boundary) for the quarter-circle data using a weight decay of 0.01

minimising a *regularised* version of the fitting criteria by adding a penalisation term like in ridge regression:

$$R^{reg}(\mathbf{w}) = R(\mathbf{w}) + \lambda \sum_{j,k} w_{jk}^2$$

This changes the update rule (6.7) to

$$w_{jk} \rightarrow w_{jk} - \eta \frac{\partial R}{\partial w_{jk}} - 2\eta\lambda w_{jk}$$

Figure 6.6 shows the result when using a weight decay of 0.01 for the quarter-circle data. Apart from avoiding an overfit, using the weight decay leads to several other benefits: The regularised fitting criteria has generally less *local* minima, making it easier to find the *global* solution. Furthermore it generally speeds up the convergence of the fitting algorithm (Ripley, 1996).

Table 6.3 gives the results of fitting different neural networks to the breast tumour data. ...

6.4 Support vector machines (SVM)

The history of support vector machines reaches back to the 1960s. The "Generalised Portrait" algorithm, which constructs a separating hyperplane with maximal margin, was originally proposed by Vapnik (Vapnik and Lerner, 1963; Vapnik and Chervonenkis, 1964). In the last few years support vector machines have become one of the most popular le ng algorithms.

		Error rate	
		Training set	Test set
1 hidden unit,	no weight decay	0.025	0.22
2 hidden units,	no weight decay	0.025	0.33
2 hidden units,	weight decay $\lambda = 0.001$	0.025	0.17

(a) 30 genes

		Error rate	
		Training set	Test set
1 hidden unit,	no weight decay	0.025	0.28
1 hidden unit,	weight decay $\lambda = 0.006$	0.025	0.14
2 hidden units,	weight decay $\lambda = 0.006$	0.025	0.25

(b) $4,000$ genes

Table 6.3: Training and test error rate for neural networks fits to the breast tumour data

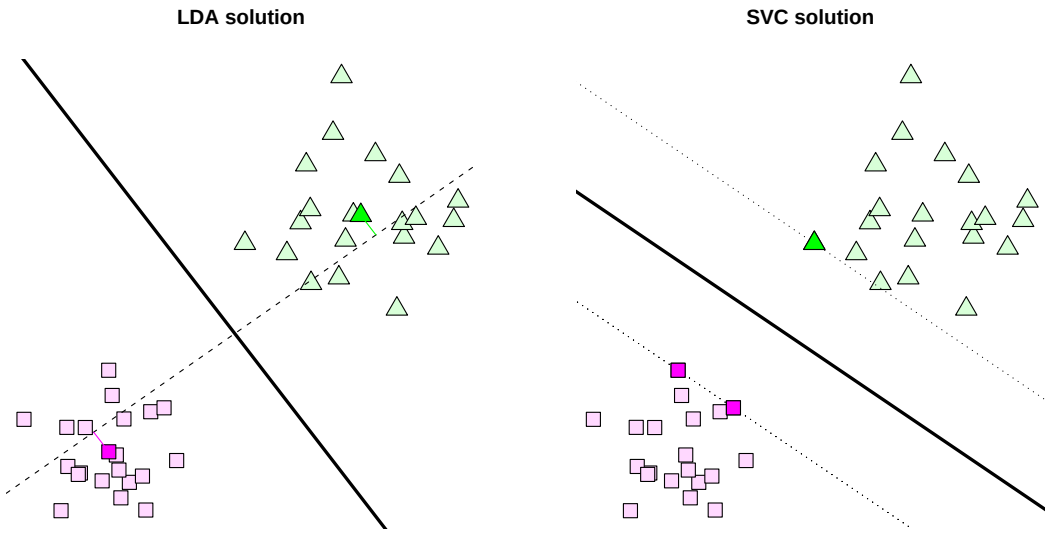


Figure 6.7: Decision boundary for linear discriminant analysis (left) and support vector classification (right) for a two-dimensional toy example: LDA maximises (after rescaling the data) the distance between the projected class means (filled symbols), whereas SVC maximises the margin between the two classes.

There are support vector machines for classification and for regression (Vapnik, 1995). We will only cover support vector *classification* machines. The vanilla type of support vector machine can only deal with two-class classification problems.[11].

The basic idea of support vector classification is to maximise the margin between the two classes. Figure 6.7 compares this to the idea of linear discriminant analysis (LDA). As one can see from the figure, the solution depends only on the observations that lie on the margin. This makes the solution vector of the support vector machine extremely sparse.

Thus we look for separating hyperplane $\{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle = -b\}$ maximising the margin.[12]. For the sake of uniqueness we have to restrict the norm of $\mathbf{w}$ to 1. Mathematically it is however easier to fix the margin to 1 and look for the hyperplane with minimal $\|\mathbf{w}\|^2/2$. Graphically this corresponds to zooming the data by a factor of $1/\rho$. We thus want to minimise

$$\frac{1}{2}\|\mathbf{w}\|^2$$

Obviously maximising the margin in the above sense is only possible if the dataset is separable. In practice however, most datasets are not separable.[13] In this case, slack variables ξ_i must be introduced so that the constraints can be met. The value of these slack variables indicates how far the data point lies outside the margin ("error"). Figure 6.8 visualises this idea. These "errors" are considered with a cost factor $C > 0$ in the objective function, which then becomes

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i.$$

The solution is thus a trade off between maximising the margin and not having too many points dropping outside the "margin". This optimisation problem can be solved using the standard tools of semi-definite quadratic programming (e.g. interior point algorithms).

So far, we can only generate separating hyperplanes. Many problems in real life are however *not* linearly separable. Consider the situation at the left hand side of figure 6.9: Observations from one class (green circles) are on both sides whereas observations from the other class (red boxes) are in between. We cannot separate the data with one hyperplane. But instead of looking only at the x_i, we can consider the pair (x_i, x_i^2). In the (x, x^2)-space, we are now able to separate the two classes. Not only in this toy example it is often necessary to map the data into a higher-dimensional *feature space* before trying to separate the data. Thus we transform the data using a feature map

$$\boldsymbol{\phi} : \mathbb{R}^p \mapsto \mathcal{H},\ \mathbf{x} \mapsto \boldsymbol{\phi}(\mathbf{x})$$

that transforms our data into a high-dimensional space $\mathcal{H}$, that can even be of infinite dimension. Support vector machines have a very important mathematical property: They depend on the data only through dot products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ or in the case of a feature map $\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle$. This allows the so called "kernel trick": Instead of specifying the feature map, we just specify the *kernel* $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ (i.e. the dot product in the induced feature space). Examples of kernel functions are the usual dot product in the $\mathbb{R}^p$

$$k : (\mathbf{x}, \mathbf{x}') \mapsto k_1(\mathbf{x}, \mathbf{x}') := \mathbf{x}^T\mathbf{x}',$$

[11] Most implementations deal with $L > 2$ classes by fitting L support vector machines: The l-th machine is trained on the dataset "class l vs. the rest".

[12] $\mathbf{w}$ is its normal vector and governs its orientation, whereas b controls the offset of the hyperplane

[13] Another way of dealing with non-separable data is mapping it into a high-dimensional feature space (see below).

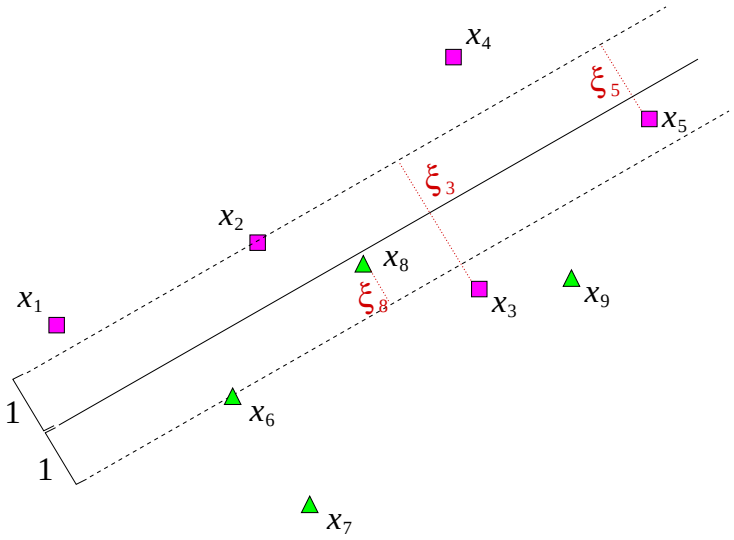


Figure 6.8: Slack variables ξ_i used in support vector classification for non-separable data

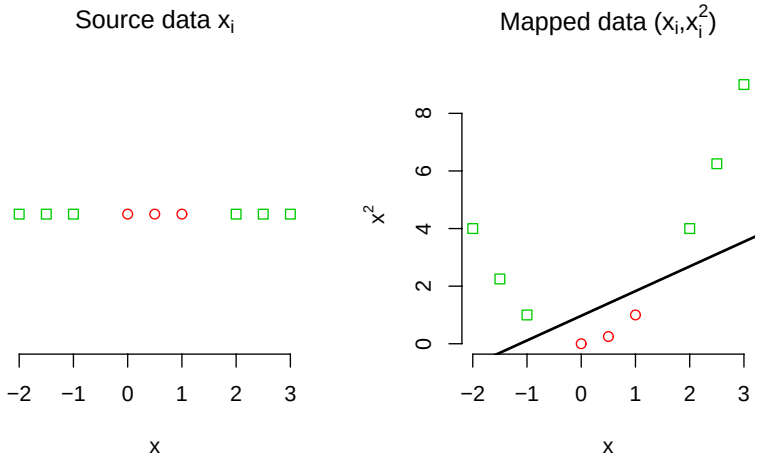


Figure 6.9: One-dimensional data x_i and mapped data (x_i, x_i^2). Contrary to the (unmapped) source data, the mapped data is separable by a hyperplane.

Mathematical background of support vector machines

We seek the hyperplane $\{\mathbf{x} : \langle \mathbf{x}, \mathbf{w}^\circ \rangle = -b^\circ\}$ ($\|\mathbf{w}^\circ\| = 1$) which maximises the margin ρ. The margin is the smallest number satisfying

$$\langle \mathbf{x}_i, \mathbf{w}^\circ \rangle + b^\circ \geq \rho \text{ ror } y_i = 1, \qquad \langle \mathbf{x}_i, \mathbf{w}^\circ \rangle + b^\circ \leq -\rho \text{ for } y_i = -1$$

Equivalently (divide $\mathbf{w}^\circ$ and b° by ρ), we can fix the margin to 1 and minimise the norm of $\mathbf{w}$ with respect to

$$\langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w} \rangle + b \geq 1 \text{ for } y_i = 1, \qquad \langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w} \rangle + b \leq -1 \text{ for } y_i = -1 \tag{6.8}$$

The corresponding Lagrangian is

$$L(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \left(\langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w} \rangle + b - 1 \right)$$

with Lagrangian multipliers $\alpha_i \geq 0$. The dual problem is to maximise

$$D(\boldsymbol{\phi}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle \tag{6.9}$$

over all $\alpha_i \geq 0$ with $\sum_{i=1}^{n} \alpha_i y_i = 0$. $\mathbf{w}$ can be obtained from the α_i via $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x}_i)$. b can be computed using any observation $x_{i_{sv}}$ with $\alpha_{i_{sv}} > 0$ via $y_{i_{sv}} \left(\langle \boldsymbol{\phi}(\mathbf{x}_{i_{sv}}), \mathbf{w} \rangle + b \right) = 1$.

For "soft margin" support vector machines (not separable datasets) we must change (6.8) to

$$\langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w} \rangle + b \geq 1 - \xi_i \text{ for } y_i = 1, \qquad \langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w} \rangle + b \leq 1 + \xi_i \text{ for } y_i = -1 \tag{6.10}$$

with $\xi_i \geq 0$. The objective function is now

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i.$$

The corresponding dual problem is once again (6.9), now to be maximised over all $0 \leq \alpha_i \leq C$ with $\sum_{i=1}^{n} \alpha_i y_i = 0$.

We can compute $\mathbf{w}$ and b from $\boldsymbol{\phi}$: $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x}_i)$ and b can be computed using any observation $x_{i_{sv}}$ with $0 < \alpha_{i_{sv}} < C$ using equations (6.10) with a "=" sign instead of the $\leq$ and $\geq$. Note that $\xi_i > 0$, iff $\alpha_i = C$.

The prediction formula for a new observation with covariate $\mathbf{x}_{new}$ is

$$\hat{y}_{new} = \text{sign} \left(\langle \boldsymbol{\phi}(\mathbf{x}_{new}), \mathbf{w} \rangle + b \right) = \text{sign} \left(\sum_{i=1}^{r} \alpha_i y_i \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_{new}) \rangle + b \right)$$

the homogeneous polynomial kernel ($q \in \mathbb{N}$)

$$k\dot{}(\mathbf{x}, \mathbf{x}') \mapsto k_1(\mathbf{x}, \mathbf{x}') := (\mathbf{x}^T \mathbf{x}')^q$$

as well as the Gaussian kernel ($\gamma > 0$)

$$k : (\mathbf{x}, \mathbf{x}') \mapsto k_2(\mathbf{x}, \mathbf{x}') := \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right).$$

Note that the polynomial kernel and the Gaussian kernel lead to non-linear decision boundaries in the (unmapped) *data space*. The decision boundary in the *feature space* is still linear. The feature space however is a black box[14] which we can airily ignore.

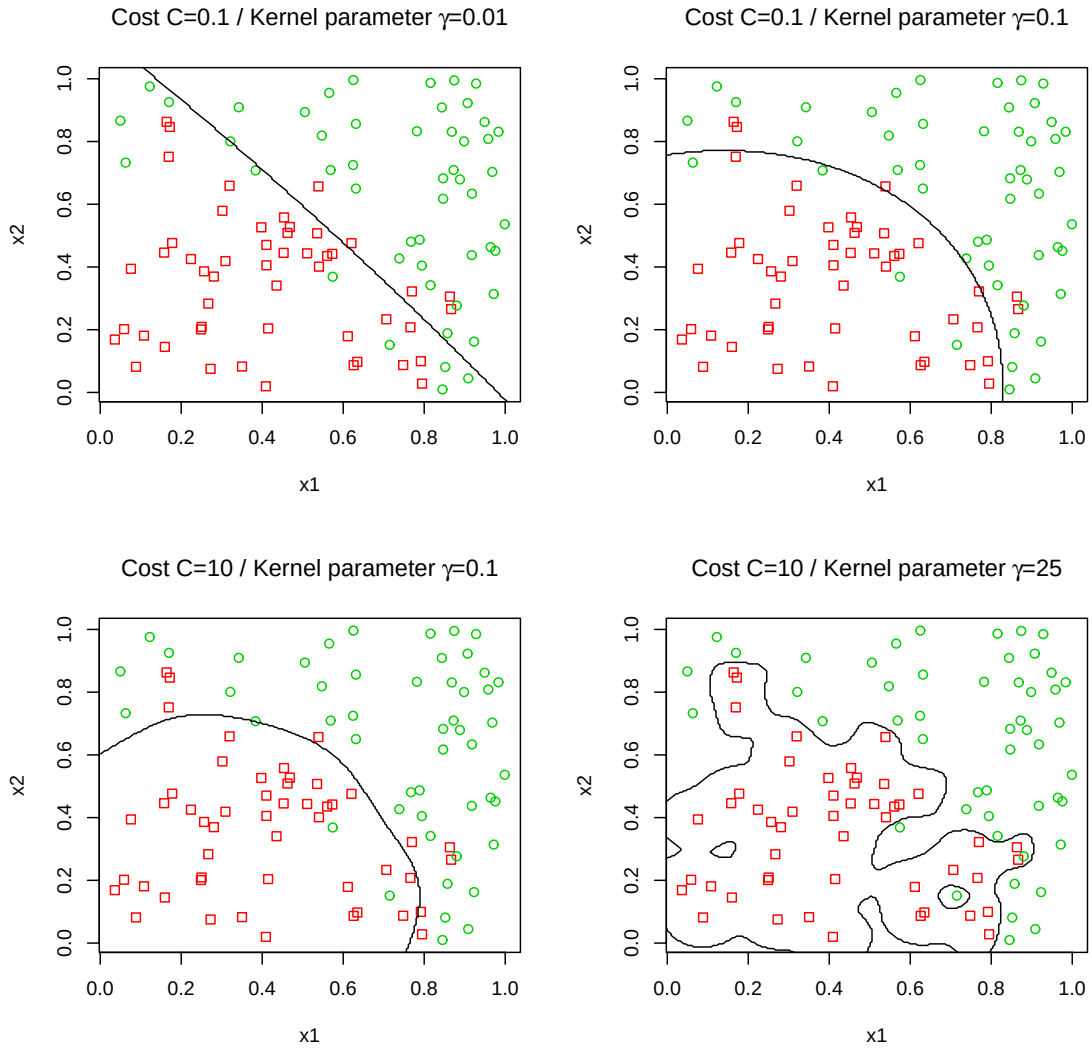
The performance of a support vector machines depends crucially on the hyperparameters chosen: Big values of C generally yield an overfit to the data. The bigger the degree q of the polynomial or the smaller the "width" γ of the Gaussian kernel is selected, the more wigglier the decision boundary will be and the more likely the fit will result in an overfit. Figure 6.10 visualises the effect of the hyperparameters. Historically the Vapnik-Chervonenkis-(VC)-bound has been used to determine optimal values for the hyperparameters (Vapnik and Chervonenkis, 1968, 1971). The VC bound is a (loose) upper bound for the actual risk. One just picks the value of the hyperparameter that leads to the lowest VC bound. The idea of minimising the VC bound is the central idea of the *structural risk minimisation* (SRM) principle. However, there seems to be some empirical evidence that the VC bound is *not* suitable for choosing optimal hyperparameters (see e.g. Duan *et al.*, 2003). Simulation studies suggest that it is best to determine the hyperparameters using either a validation set or cross-validation (see section 7.2.1 below).

Support vector machines are not scale-invariant, so it is necessary to scale the data beforehand. However most implementations of SVM (like the one used in `R`) perform the standardisation automatically.

Support vector machines have been successfully trained classifiers with huge amounts of data (like speech or image (digit) recognition). This applies to the number of observations as well as to the number of covariates.[15]. Support vector machines are one of the most competitive methods is the area of two-class classification (see e.g. Schölkopf and Smola, 2002). The result obtained from fitting a support vector machine to the `breast tumour` data is given in table 6.4.

[14] For the Gaussian kernel it is even completely unknown how the feature space looks like.

[15] One can even fit support vector machines to datasets with more covariates than observations — conditions under which other algorithms (e.g. logistic regression) fail.



		Error rate	
		Training set	Test set
Cost $C = 0.1$,	kernel parameter $\gamma = 0.01$	0.15	0.15
Cost $C = 0.1$,	kernel parameter $\gamma = 0.1$	0.09	0.12
Cost $C = 10$,	kernel parameter $\gamma = 0.1$	0.11	0.19
Cost $C = 10$,	kernel parameter $\gamma = 25$	0.01	0.25

Figure 6.10: SVM fits (decision boundary) and error rates for different values of the cost C and the parameter γ of the Gaussian kernel

			Error rate	
			Training set	Test set
Linear kernel	Cost $C = 0.001$		0.40	0.54
	Cost $C = 0.01$		0	0.17
	Cost $C = 0.1$		0	0.17
Gaussian kernel	Cost $C = 0.35$,	kernel parameter $\gamma = 0.04$	0.05	0.17
	Cost $C = 1$,	kernel parameter $\gamma = 0.01$	0.025	0.27

(a) 30 genes

		Error rate	
		Training set	Test set
Linear kernel	Cost $C = 0.0001$	0	0.20
	Cost $C = 0.0009$	0	0.17
	Cost $C = 0.005$	0	0.20

(b) $4,000$ genes

Table 6.4: Training and test error rate for SVM fits to the breast tumour data

Chapter 7

Classifier assessment

7.1 Introduction

Different classifiers can have very different accuracies (i.e. different misclassification rates). Therefore, it is important to assess the performance of different learning methods. The perfomance assessment can be split into determining if we have a good estimate of the posterior probabilities of class membership ($p(c|\mathbf{x})$), and direct assessment of performance.

Reliable estimates of the classification error or other measures of performance are important for (i) training the classifier (e.g. selecting the appropriate predictor variables and parameters), and (ii) estimating the generalisation error of the classifier. The *generalisation* performance of a learning method relates to its prediction capability on independent test data. Hastie *et al.* (2001) refer to the above two goals as *model selection* and *model assessment*.

7.2 Bias, Variance and Error Rate Estimation

In Figure 7.1 we illustrate the ability of a learning method to generalise graphically. We can explain before-mentioned figure by distinguishing between *test error* or *generalisation error* and *training error* as in Hastie *et al.* (2001). Test error is the expected prediction error over an independent test sample, while training error is the average loss over the training sample.

First we consider the test error for our estimated model. As the model becomes more and more complex, it is able to adapt to more complicated structures in the data (decrease in bias), but the estimation error increases as we have to estimate more parameters (increase in variance). There is typically a trade-off between bias and variance that gives minimum test error.

We see that the training error shows very different behaviour to the test error, and is therefor not a good estimate of the test error. The training error consistently decreases with model complexity. We can make the training error arbitrarily small by just increasing the model complexity enough. However, such a model will overfit the training data and will generalise poorly.

We will consider a few methods for estimating the test error for a specific model. If large amounts of data are available a popular approach to model selection and model assessment is to divide the dataset into three parts: a training set, a validation set and a test set. The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalisation error of the final chosen model. Note that the test set should not

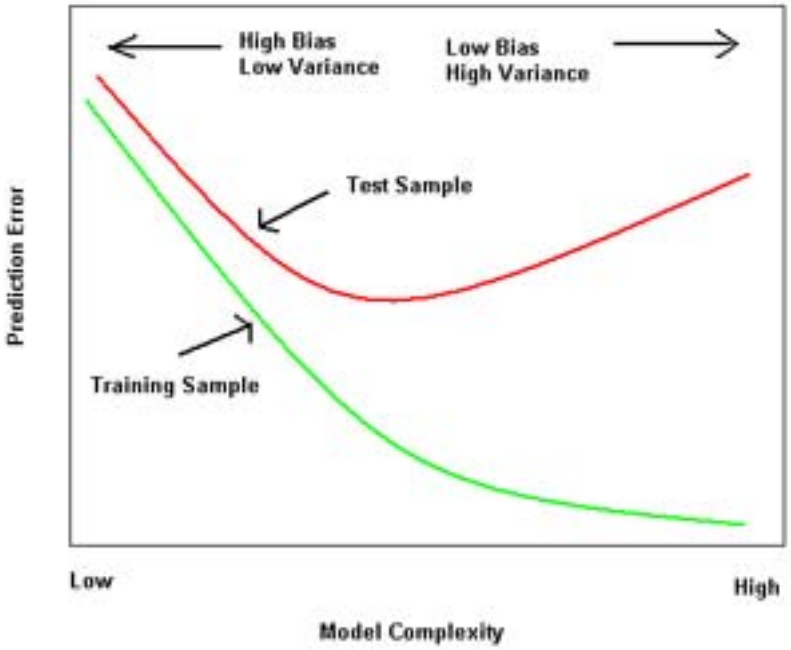


Figure 7.1: Changes in test sample and training sample error as the model comlexity is varied

be available until we assess the final model. If we use the test set repeatedly to find the model with the lowest test set error, we risk underestimating the true test error.

There is no general rule on how to choose the number of observations in each of the three parts. This will typically depend on the signal-to-noise ration and the size of the training sample. Hastie *et al.* (2001) suggest that a typical split might be 50% for training, and 25% each for validation and testing. A limitation of this approach is that it reduces the effective size of the training set. This is a problem for microarray datasets that have a very limited number of observations. Below we discuss methods that are designed for situations where there is insufficient data to split it into three parts. The reader is referred to Section 2.7 in Ripley (1996) and Chapter 7 in Hastie *et al.* (2001) for more detailed discussions.

Ripley (1996) makes the following remarks on page 76 in the context of studies for comparing error rates:

- It is an experiment and should be designed and analysed as such.
- Comparisons between classifiers will normally be more precise than assessment of the true risk.
- We often need a very large test set to measure subtle differences in performance, as standard errors of estimates decrease slowly with an increase in sample size

7.2.1 Cross-validation

Cross-validation is one approach to get around the wasteful approach based on separate training, validation and test sets. In V-fold cross-validation (CV) (Breiman *et al.*, 1984b), the data set $\mathcal{L}$ is randomly divided into V sets $\mathcal{L}_\nu$, $\nu = 1, \ldots, V$, of as nearly equal size as possible. Classifiers are built on the training sets $\mathcal{L} - \mathcal{L}_\nu$, error rates are computed from the validation sets $\mathcal{L}_\nu$, and averaged over ν.

A question that arises is how large V should be. A common form of CV is leave-one-out cross-validation (LOOCV)(Stone, 1974), where $V = n$. LOOCV can result in smaller bias but high variance estimators of classification. Therefore, it is clear that there is a bias-variance trade-off in the selection of V. LOOCV can also be expensive in terms of computational time. Breiman (1996a) shows that LOOCV provides good estimators of the generalisation error for stable (low variance) classifiers such as k-NN. Choices of $V = 5$ or $V = 10$ are often reasonable.

Cross-validation is commonly used to choose a suitable model by estimating a measure of performance (e.g. the error rate) or a measure of model fit (e.g. deviance). However, we can also use cross-validation for parameter estimation, by choosing the parameter value which minimises the cross-validated measure of performance. If we then want to assess the performance of the resulting classifier, we have to do so by a double layer of cross-validation.

7.2.2 The Bootstrap

Cross-validation is used to estimate error rates on a test set by constructing a pseudo test set from the training set. A different approach is to accept that the training set error is biased, and then to try and estimate that bias, and correct for it by using this estimate.

Suppose we take a sample of size n by sampling *with replacement* from our original sample (known as a bootstrap sample), and then calculate an estimate θ^* from this sample. Suppose that $\hat{\theta}$ is the estimate computed from the original data and θ is the true value of the quantity that we want to estimate. Then the idea is that the variability of $\theta^* - \hat{\theta}$ should mimic that of $\hat{\theta} - \theta$. In particular, the mean of of $\theta^* - \hat{\theta}$ should estimate the bias of $\hat{\theta}$. This can be estimated by resampling B times and averaging.

We can use the above ideas of the bootstrap to estimate the difference (bias) between a classifier's apparent error and true error. We can obtain a bootstrap estimate of the bias by computing the average (over B bootstrap samples) of the error rate on the bootstrap training set $\mathcal{L}^*$ and subtracting the estimate computed on the larger set $\mathcal{L}$. ($\mathcal{L}^*$ contains some of the members of $\mathcal{L}$ more than once, and usually some not at all, so as a set $\mathcal{L}^*$ is smaller). A problem is that the example being predicted may be in the bootstrap training set. In this case we may sometimes predict it too well, leading to biased estimates. Efron (1983) propose the ".632" bootsrap estimate, which considers only the predictions of those members of $\mathcal{L}$ not in $\mathcal{L}^*$. By using a similar approach as above, we can also use the boostrap to directly estimate the precision of the apparent error rate.

The *jackknife* (Quenouille, 1949) is another approach to estimate the bias and is loosely related to the bootstrap. These methods are sometimes considered to be improvements on cross-validation, however, Ripley (1996) warns on page 75 that this is perhaps optimistic, since the full power of the boostrap seems not to have been fully tested.

7.2.3 Out-of-bag Estimation

An out-of-bag estimate of the error rate can be obtained as a byproduct of *bootstrap aggregating* or *bagging* of unstable classifiers such as tree classifiers (see next lecture). For each bootstrap sample, about one-third of the cases are left out and not used in the construction of the tree. These could be used as test set observations for performance assessment purposes. For the bth bootstrap sample, put the *out-of-bag* cases down the bth tree to get a test set classification. For each observation in the learning set, let the final classification be the class having the most votes for the bootstrap samples in which that observation was out-of-bag. Then we can obtain an ubiased estimator of the error rate by comparing the above classification with the class labels.

7.2.4 Log-probability scoring

The counting of errors on a test set is one approach to assess the performance of a classifier. However, this approach often has high variance. Sometime other approaches may perform better. One good approach is *log-probability scoring*:

$$-\sum_{\text{test set}} \log \hat{p}(c_i|\mathbf{x}_i).$$

This has lower variance than error counting. Note that the above measure is reliant on the fact that $\hat{p}(c|\mathbf{x})$ is a good estimate of $p(c|\mathbf{x})$. We can use calibration plots to evaluate this. If a fraction p of the events we predict with probability p using our model for $p(c|\mathbf{x})$ actually occur, the predicted probabilities are set to be *well-calibrated*. A calibration plot plots the predicted probabilities against the corresponding fractions. A relatively straight line indicates that the predicted probabilities are well-calibrated. However, often extreme probabilities are estimated to be too extreme. This is the

result of the fact that we are not using $p(c|\mathbf{x})$ but $p(c|\mathbf{x}, \hat{\theta})$, a *plug-in* estimate in which we assume that the fitted parameters $\hat{\theta}$ represent the true parameters. So the uncertainty in $\hat{\theta}$ is ignored.

Calibration plots can be used to re-calibrate our probabilities, but most often deviations from a straight line in the plot are indicative of more fundamental problems with the model.

7.2.5 ROC curves

Receiver operator characteristic (ROC) curves is another summary of classifier performance (see Hand (1997), Chapter7). Consider a two class medical problem where the outcomes are "normal" or "diseased". The specificity is defined as

$$\text{specificity} = p(\text{predict normal}|\text{is normal})$$

and sensitifity is defined as

$$\text{sensitivity} = p(\text{predict diseased}|\text{is diseased}).$$

If our classifier provides us with estimates of $p(\text{diseased}|\mathbf{x})$. We can now declare a subject as diseased if $p(\text{diseased}|\mathbf{x}) > \alpha$. By varying $\alpha \in [0, 1]$ we get a family of classifiers. An ROC curve plots sensitivity against 1-specificity. For equal losses we want to minimize the sum of sensitivity and 1-specificity, or sensitivity-specificity. The best choice of α corresponds to the point where the ROC curve has slope 1.

7.3 Example: Toxicogenomic Gene Expression Data

In the lecture we will consider an example involving toxicogenomic gene expression data.

Chapter 8

Tree-based classifiers, bagging, and boosting

8.1 Tree-based classifiers

Decision trees are hierarchical classifiers based on "If ... then ... else ..." rules. This makes them extremely easy to interpret and made decision trees very popular tools in medical diagnosis and other fields. Decision trees have been commercialised successfully under the acronyms CART (Breiman *et al.*, 1984a) and C4.5 (Quinlan, 1993). Figure 8.1 gives an example of a decision tree; it is taken from the NHS Direct self-help guide. The left hand side shows the original page from the NHS guide and the right hand side shows the decision tree in a more usual away. At every node there is one question that has to be answered either with "yes" or "no"; thus the tree is binary. The top node is called *root*, the terminal nodes are called *leafs*.[1]

Mathematically speaking decision trees do nothing else than cutting the feature space into smaller and smaller hypercubes. Each hypercube (a rectangle in the case of $\mathbb{R}^2$) corresponds to a terminal node of the tree and *vice versa*. Figure 8.2 visualises this for a decision tree for the *iris* dataset. The splits are hereby carried out in a way that the partitions get as pure as possible, ideally there should be only observations of one class in a partition (terminal node).

The trees are grown from the root note using the "divide and conquer" device: First the root node is split, then the two just created nodes are split etc.[2]. When carrying out a split, we have to make two decisions. What variable to use? and: Which values to assign to the left branch and which to assign to the right one? For nominal covariates we have to try all possible combinations of attributes.[3] For ordinal and metrical variables we have to find a cut-off point.

As mentioned above we want to reduce the *impurity* of the partitions, which is assessed by a *measure of impurity*. Common measures of impurity are the *Gini coefficient*

$$i(\mathbf{p}) = 1 \sum_{l \neq m} p_l p_m = 1 - \sum_l p_l^2$$

[1] Note that decision trees are grown "upside-down".

[2] We will only consider *binary* trees in this chapter

[3] For two-class trees one can order the attributes by the proportion of one of the classes, and then one just has to find a cut-off point.

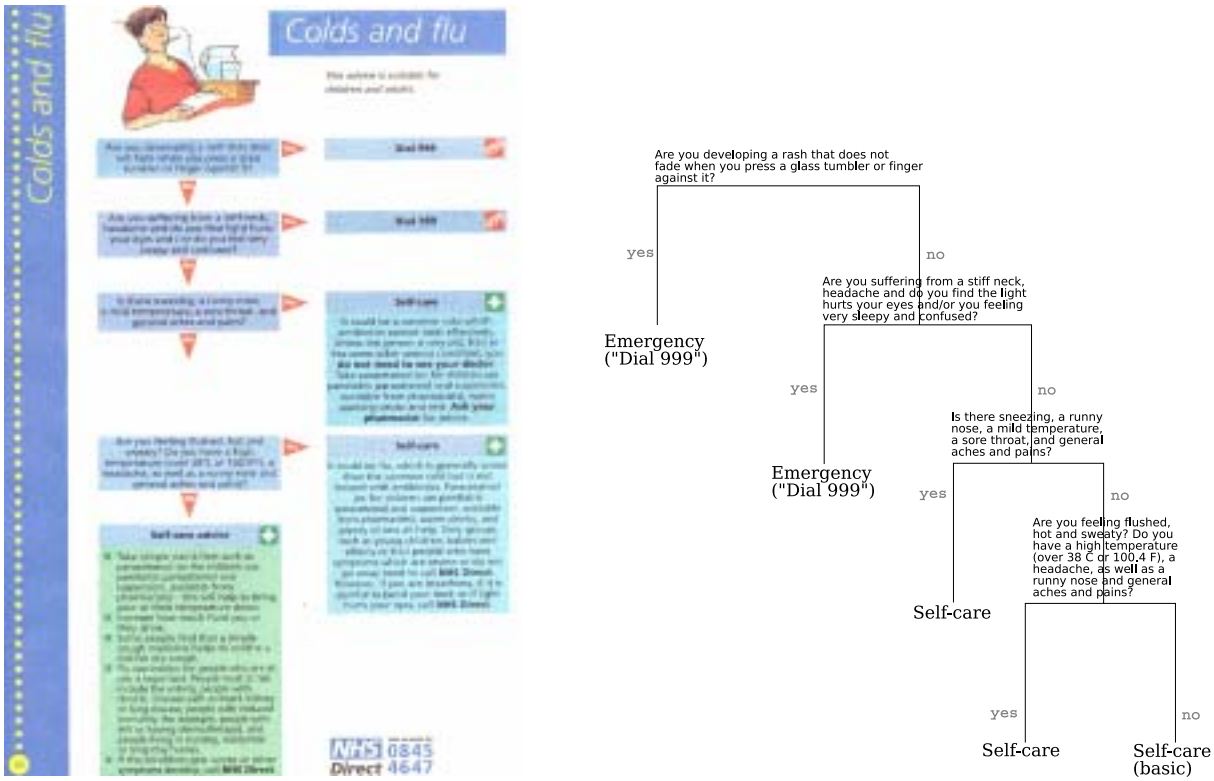


Figure 8.1: Page taken from the NHS Direct self-help guide (left) and corresponding decision tree (right)

and the *entropy*

$$i(\mathbf{p}) = -\sum_l p_l \log p_l,$$

where $\mathbf{p} = (p_1, \ldots, p_L)$ denotes the empirical distribution of the class labels in the partition.[4] Figure 8.3 displays the Gini coefficient and the entropy for the two-class case. If the partition consists of only one class (frequency p_1 either 0 or 1), the impurity is 0. Are both classes equally present (frequency $p_1 = 0.5$), then both impurity measures are maximal.

When splitting a node with empirical distribution $\mathbf{p}$ into two nodes with empirical distributions $\mathbf{p}_l$ (left node) and $\mathbf{p}_r$ (right node) the decrease in impurity is

$$i(\mathbf{p}) - (\pi_l i(\mathbf{p}_l) + \pi_r i(\mathbf{p}_r)),$$

where π_l is the proportion of observations that is allocated to the left node and $\pi_r = 1 - \pi_l$ is the proportion of observations allocated to the right node.

We now can use the decrease in impurity to "grow" the tree. Starting with one partition (i.e. the root node), we repeatedly split all terminal nodes such that each time th decrease in impurity is maximal. We can repeat this until no more decrease is possible. Figure 8.4 shows the decision tree for the Pima Indians dataset. The Pima Indians dataset was collected by the US National Institute of

[4]It might occur that $p_l = 0$, in this case we define $0 \log 0 := 0$.

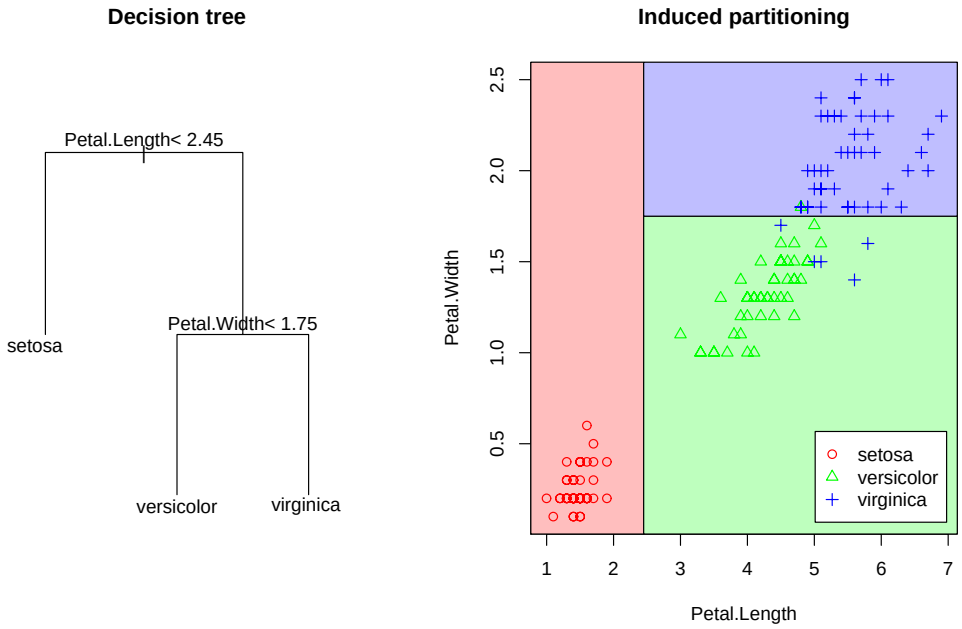


Figure 8.2: Decision tree for the iris dataset and corresponding partitioning of the feature space

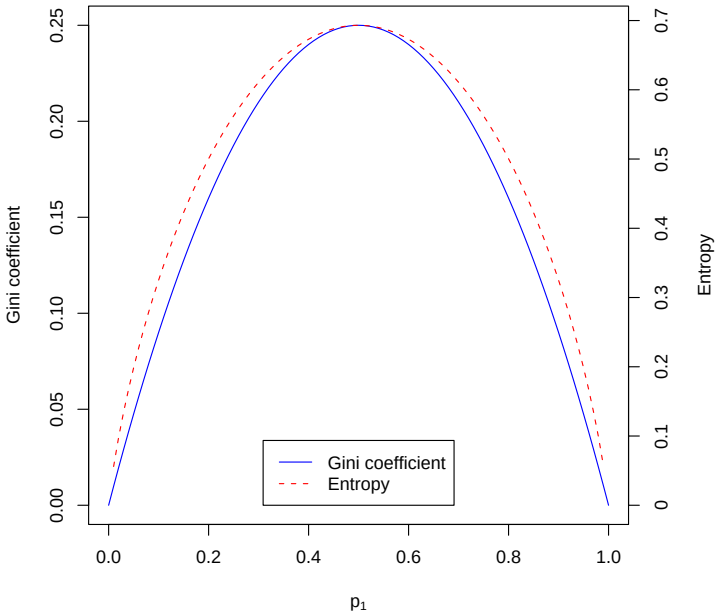


Figure 8.3: Gini coefficient and entropy for a two-class problem

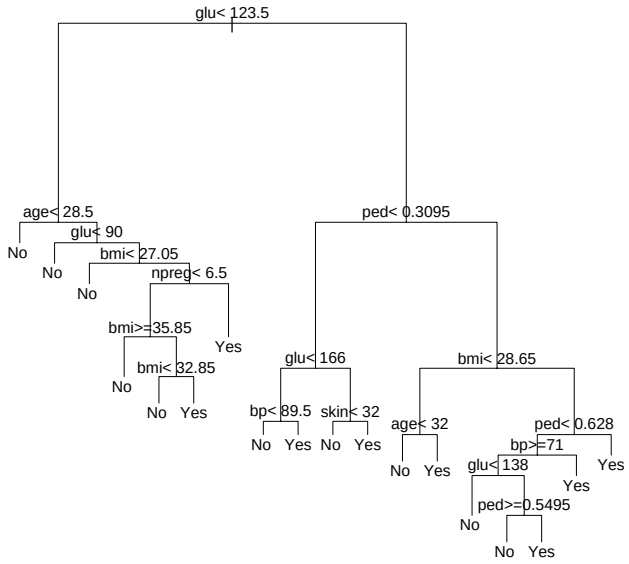


Figure 8.4: Unpruned decision tree for the Pima Indians dataset

Diabetes and Digestive and Kidney Diseases. The subjects were women who were at least 21 years old, of Pima Indian heritage and living near Phoenix, Arizona. They were tested for diabetes according to World Health Organisation criteria. The variables measured were the number of pregnancies (`npreg`), the plasma glucose concentration in an oral glucose tolerance test (`glu`), the diastolic blood pressure in mm Hg (`bp`), the triceps skin fold thickness in mm (`skin`), the body mass index (`bbi`), the diabetes pedigree function (`ped`), and the age (`age`).

Growing the tree until no more decrease in impurity is possible often leads to an overfit to the training data. We thus have to *prune* the tree. The most popular pruning approach is the one proposed by Breiman *et al.* (1984a). The idea behind this approach is that too big trees yield an overfit. Thus too big trees must be penalised. Denote with $R(T)$ a measure of fit for the tree; this can be the misclassification rate on the training set or the entropy of the partitioning. Instead of minimising the fit criterion $R(T)$ itself, we now minimise the penalised fitting criterion

$$R(T) + \alpha \cdot \text{size}(T),$$

where size(T) is the number of leafs and α controls the amount of penalisation. If we choose $\alpha = 0$, there will be no pruning; if we choose $\alpha = +\infty$ all nodes but the root node are removed. Breiman *et al.* (1984a) showed that there is a nested sequence of subtrees of the fitted tree such that each is optimal for a range of α. So all we have to do is to pick one of the trees of this sequence.

If we have a validation set at hand, we can pick the subtree yielding the lowest error rate in the validation set. Otherwise one generally uses cross-validation to pick the optimal subtree. Figure 8.5 shows the error (relative to a tree with the root node only) for the different subtrees for the Pima Indians data. The error rate was estimated using cross validation and the vertical lines indicate the standard error of the estimates. One can now pick the subtree leading to the lowest error in cross-validation. Breiman *et al.* (1984a) propose to choose the smallest tree that is not more than one standard error worse that the best tree. ("One SE rule"). The idea behind this is that smaller trees

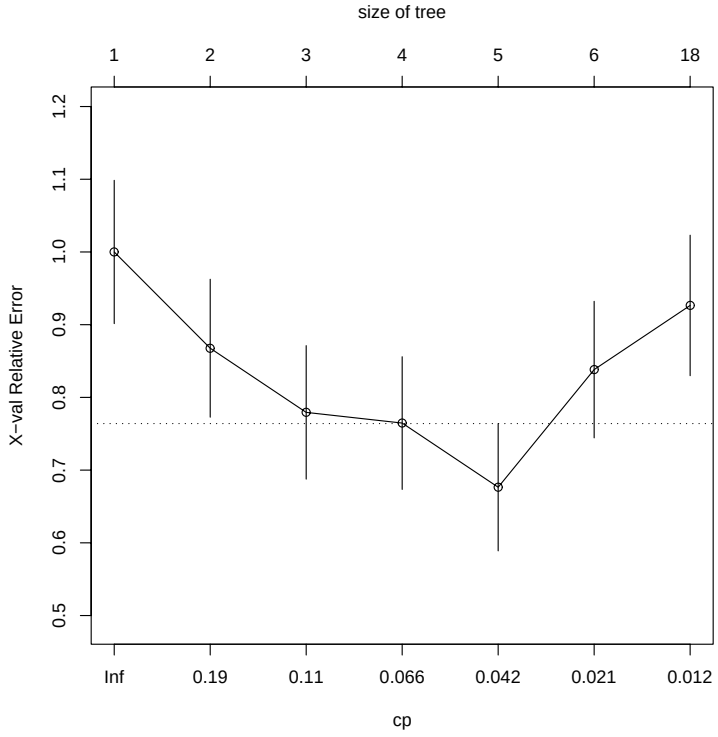


Figure 8.5: Error rate estimated by cross-validation for the different subtrees. The vertical lines show the standard error, and the horizontal dashed line in one standard error worse than the best subtree. Using the "one SE rule" we would pick the first value (from the left) of the tree size for which the curve is below the dashed line.

are generally preferable and the tree picked that way is only "insignificantly" worse than be best one. In our example the "one Se rule" would pick a tree of size 5. Figure 8.6 shows the decision tree after pruning (i.e. the optimal subtree) for the *Pima Indians* dataset.

Decision trees are classifiers that are easy to interpret. Decision trees are however often outperformed by other methods when it come to the the accuracy of the prediction; they have a rather high variance and are thus unstable classifiers. For this reason one should not over-interpret which splits come first or later. A slight change in the dataset sometimes causes the whole tree to "topple over."

8.2 Bagging

Decision trees are *unstable* predictors with high variance. Bagging is a technique to reduce the variance of an unstable predictor. Although bagging is usually presented in the context of decision trees, but bagging can be applied to other predictors (neural nets *etc.*) as well.

Imagine we had B training datasets each of size n: $\mathcal{L}_1, \ldots, \mathcal{L}_B$. Using this data we could obtain B classifiers $\hat{\mathbf{p}}^{(b)}(\mathbf{x}) = \left(\hat{p}_1^{(b)}(\mathbf{x}), \ldots, \hat{p}_L^{(b)}(\mathbf{x})\right)$. $\hat{p}_l^{(b)}(\mathbf{x})$ denotes hereby the probability predicted by the b-th classifier that a new observation with covariates $\mathbf{x}$ belongs to the l-th class. To form a final classifier,

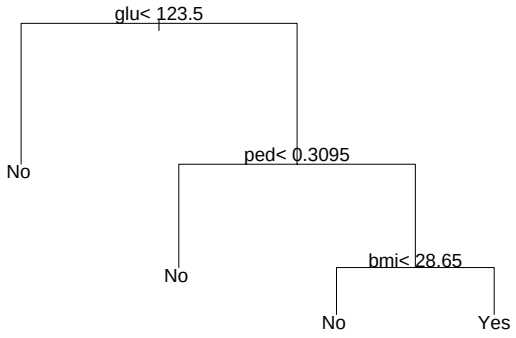


Figure 8.6: Pruned decision tree for the Pima Indians dataset.

we could aggregate the classifiers by taking the arithmetic mean of the predicted probabilities

$$\hat{\mathbf{p}}^{average}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{\mathbf{p}}^{(b)}(\mathbf{x}) \tag{8.1}$$

We then classify $\mathbf{x}$ then into the class l with the highest predicted probability $\hat{p}_l^{average}$.[5]

Taking the arithmetic mean over several training datasets should reduce the variance significantly.[6]

In practice however we can seldom afford the luxury of obtaining more than one learning dataset of size n. We now use the technique of *bootstrapping*; we draw B *bootstrap samples* of size n from our training set (with replacement). This technique is called *bagging*, the acronym for bootstrap aggregating (Breiman, 1996b). The classifier defined in (8.1) is called *bagging classifier*.[7]

Note that bagging only works with unstable predictors (like decision trees or regressions after variable selections). When using stable predictors, bagging generally deteriorates the variance. Or in Breiman's words: "Bagging goes a ways toward making a silk purse out of a sow's ear, especially if the ear is twitchy." (Breiman, 1996b).

As a byproduct bagging furthermore leads to an unbiased estimate of the error rate. As we use a randomly drawn sample for constructing the trees, there are (nearly) always a few observation that were not used; we can use them to get an unbiased estimate of the error rate ("out of bag" estimation).

The idea of bagging can be generalised. Random forests (Breiman, 1999) do not only use a random subset of the observations, they also just use only a small random subset of the variables in each iteration. This does not only offer theoretical advantages, it makes fitting the decision tree computationally a lot easier, as we only have to consider a possible small subset of covariates.

Table 8.1 gives the training set and test set error rate for a plain decision tree, bagged decision trees and random forests for the breast tumour data set. Note that decision trees are generally unsuited for microarray data, as they concentrate on a very small number of genes. The out-of-bag estimates are rather close to the actual test set performance.

[5]Instead of using the predicted probabilities, we could as well use the predictions. We would then classify a new observation into the class that gets the most "votes".

[6]Recall that the variance of the arithmetic mean of a sample $Z_1, \ldots, Z_n$ is $\frac{\sigma^2}{n}$, if the Z_i are uncorrelated random variables with variance σ^2.

[7]Strictly speaking, (8.1) is *not* the bagging classifier, but an Monte-Carlo approximation to it.

	Error rate		
	Training set	Test set	(out of bag estimate)
Decision tree	0.025	0.36	
Bagged decision tree	0	0.19	(0.20)
Random forest	0	0.22	(0.25)

(a) 30 genes

	Error rate		
	Training set	Test set	(out of bag estimate)
Decision tree	0.025	0.36	
Bagged decision trees	0	0.28	(0.275)
Random forest	0	0.25	(0.23)

(b) 2,000 genes

Table 8.1: Training and test error rate obtained when fitting a decision tree, bagged decision trees and a random forest to the breast tumour dataset

8.3 Boosting

Bagging's cousin, boosting has been developed to deal with the converse situation; it aggregates stable classifiers with a poor performance (high bias), so called *weak learners*. An example for this type of classifier are *stumps*. Stumps are decision trees with only two terminal nodes. The idea behind boosting is to re-weight the observations in every iteration, such that the weak learner is forced to concentrate on the "difficult" observations. The weak learners can then be seen as an "committee", which "votes" by majority to get the final prediction. Statistically speaking, boosting is a greedy gradient-descent algorithm for fitting an additive model, although "committee" and "majority vote" sound more glamorous.

The Adaboost algorithm of Freund and Schapire (1995)

We will only consider the two-class case. The two classes will be coded with ± 1, i.e. either $y_i = 1$ or $y_i = -1$. Denote with $\hat{f}^{(h)}$ the predictions of the h-th weak learner. We start with a weighting vector of $\mathbf{w}^{(1)} = (1/n, \ldots, 1/n)$.[8]

Now iterate for $h = 1, 2, \ldots, H$:

i. Train the weak learner $\hat{f}^{(h)}$ using the weights $\mathbf{w}^{(h)}$.

ii. Compute the weighted training error $\epsilon^{(h)} = \sum_{i:\ \hat{f}^{(h)}(\mathbf{x}_i) \neq y_i} w_i$ and set $\alpha^{(h)} = \frac{1}{2} \log \left(\frac{1 - \epsilon^{(h)}}{\epsilon^{(h)}} \right)$.

iii. Update the weights ($i = 1, \ldots, n$)

$$w_i^{(h+1)} = w_i^{(h)} \cdot \begin{cases} e^{-\alpha^{(h)}} & \text{if } \hat{f}^{(h)}(\mathbf{x}_i) = y_i \\ e^{\alpha^{(h)}} & \text{if } \hat{f}^{(h)}(\mathbf{x}_i) \neq y_i \end{cases}$$

and normalise them, so that they sum to 1.

The final predictor is then $\hat{F}^{(H)}(\mathbf{x}) = \sum_{h=1}^{H} \alpha^{(h)} \hat{f}^{(h)}(\mathbf{x})$. We predict the first class for a new observation $\mathbf{x}_{new}$ if $\hat{F}^{(H)}(\mathbf{x}_{new}) > 0$, otherwise we predict the second.

Step iii. of the Adaboost algorithm is responsible for "focusing" on the difficult examples: if an observation is misclassified by the current weak learner $\hat{f}^{(h)}$ then the weight of this observation is increased, otherwise it is decreased. We can rewrite $w_i^{(h+1)}$ as

$$w_i^{(h+1)} = e^{-y_i \sum_{\eta=1}^{h} \alpha^{(\eta)} \hat{f}^{(\eta)}(\mathbf{x}_i)} = e^{-y_i \hat{F}^{(h)}(\mathbf{x}_i)}.$$

The weights are thus a function of $y_i \hat{F}^{(h)}(\mathbf{x}_i)$, which is nothing else than the "margin" of the final predictor up to the h-th step. The margin $y_i \hat{F}^{(h)}(\mathbf{x}_i)$ indicates how far the predictions are away from the decision boundary, i.e. how "confident" the final predictor is. Boosting can thus be seen as a technique that maximises the margin.

Friedman *et al.* (1998) showed that boosting is a greedy gradient descent algorithm for fitting an additive logistic model. However contrary to logistic regression (cf. section 5) Adaboost does not maximise the likelihood, but minimises

$$\mathbb{E}e^{-yf(x)} = \frac{|\tilde{y} - p(x)|}{\sqrt{p(x)(1-p(x))}},$$

where $\tilde{y} := \begin{cases} 1 & \text{if } y = 1 \\ 0 & \text{if } y = -1 \end{cases}$ (thus just y in $0/1$ coding instead of $-1/+1$ coding) and $p(x) := \mathbb{P}(y = 1|\mathbf{x})$.

Freund and Schapire (1995) motivated this expression as a differentiable upper bound to the misclassification error. However one can see it as a χ^2-type approximation to the log-likelihood. However one can use other loss functions as well. Friedman *et al.* (1998) propose using the log-likelihood as a criterion ("Logit-Boost"); Friedman (1999) proposed to use the squared loss function ("L_2-Boost").

A main theoretical property of boosting is its ability to reduce the training error, it uses the weak learners to form a *strong learner*. Already if the weak learner performs slightly better than random guessing (i.e. has a training error rate of slightly less than $1/2$), the training error of the Adaboost algorithm drops exponentially fast. This is however not the most important *practical* property of boosting. Boosting is rather resistant to overfitting. This is the main reason for the popularity of boosting. This is mainly an empirical finding and the theory behind it is not completely understood. Freund and Schapire (1995) argue that boosting leads to larger margins and thus reduces the generalisation error in the test set. They thus connect boosting to support vector machines, which have a similar property. Bühlmann and Yu (2003) argue that boosting leads to a more favourable bias-variance trade-off: "When the iteration b increases by 1, one more term is added in the fitted procedure, but due to the

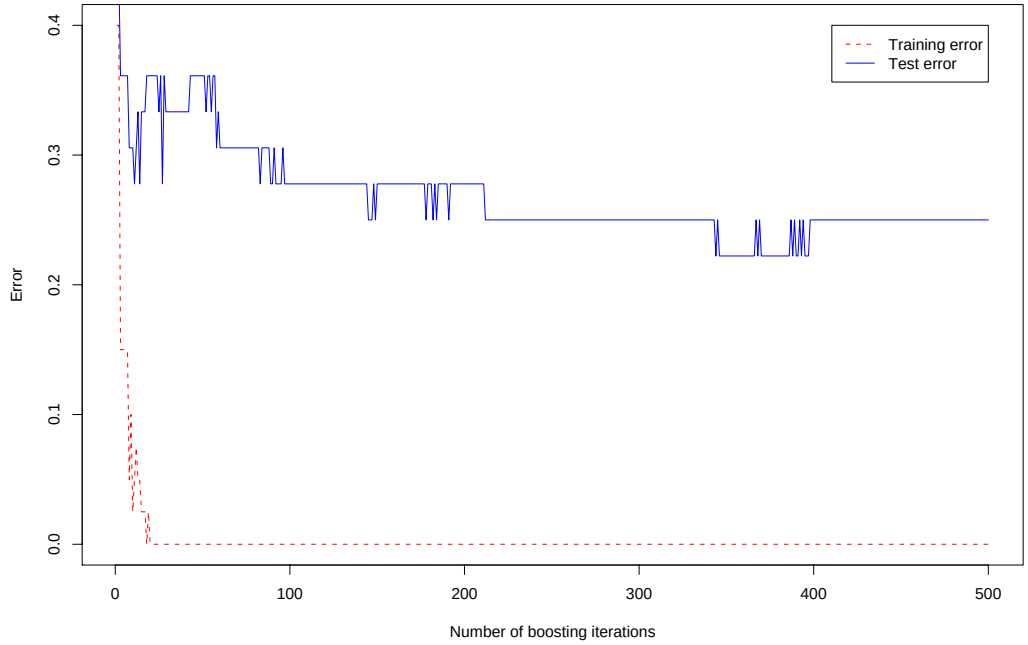


Figure 8.7: Error rates for the Adaboost fit to the breast tumour dataset)

dependence of this new term on the previous terms, the 'complexity' of the fitted procedure is not increased by a constant amount as we got used to in linear regression, but an exponentially diminishing amount as b gets large." Breiman (1999) however argues that only reason why boosting performs that well is that it emulates the behaviour of random forests.

Boosting reduces the bias of a classifier, whereas bagging reduces its variance. Thus they are based on two extremely different types of base classifiers. Unstable predictors (like e.g. "full" decision trees) are unsuitable for boosting. Sometimes even stumps possess a too high variance, in this case *shrunken stumps (trees)* are used. Shrunken stumps use the prediction of the tree, but multiply it by a shrinkage factor $\nu \in (0; 1)$.

When fitting boosted stumps (with a shrinkage factor of $\nu = 0.5$) to the breast tumour data (30 genes), one obtains a training error of 0 and a test set error of 0.22. Figure 8.7 shows a plot of the error rates during the fitting process. It shows two important characteristics of boosting. There is virtually no overfit even when using all 500 weak learners. Although the training error reaches 0 quite early, the test set error is still reduced further. The reason for this is that boosting does not only minimise the training error, but aims to maximise the margin (see figure 8.8).

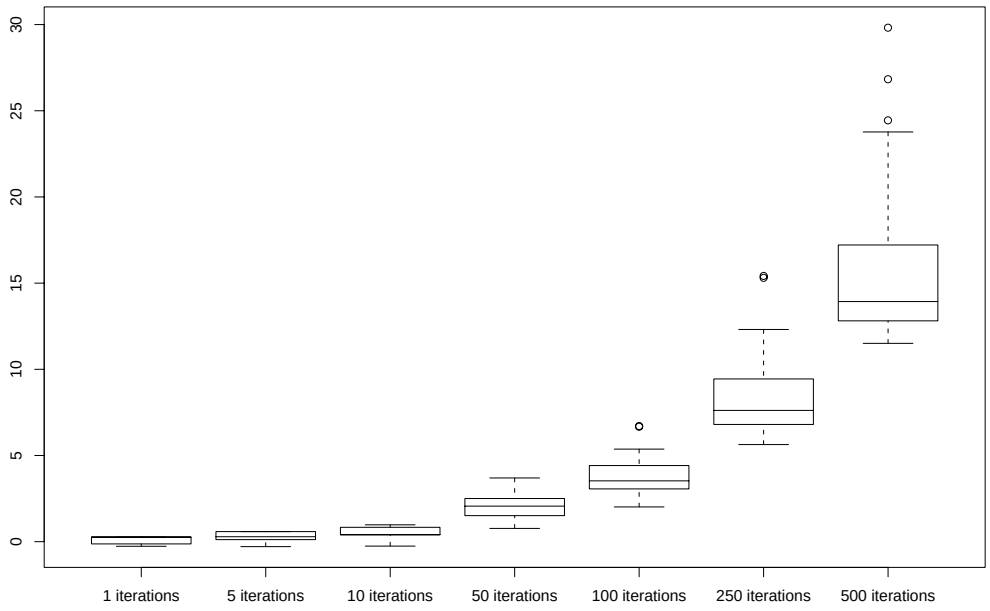


Figure 8.8: Distribution of the margin (Adaboost fit to the breast tumour dataset)

Bibliography

Agresti, A. (1990) *Categorical Data Analysis*. New York: John Wiley & Sons.

Alizadeh, A., Eisen, M., Davis, R., Ma, C., Lossos, I., Rosenwald, A., Boldrick, J., Sabet, H., Tran, T., Yu, X., Powell, J., Yang, L., Marti, G., Moore, T., Hudson, J., Lu, L., Lewis, D., Tibshirani, R., Sherlock, G., Chan, W., Greiner, T., Weisenburger, D., Armitage, J., Warnke, R., Levy, R., Wilson, W., Grever, M., Byrd, J., Botstein, D., Brown, P. and Staudt, L. (2000) Different types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, **403**, 503–511.

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D. and Levine, A. J. (1999) Broad patterns of gene expression revealed by cluster analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Natl. Acad. Sci. USA*, **96**, 6745–6750.

Anderson, E. (1935) The irises of the Gaspé peninsula. *Bulletin of the American Iris Society*, **59**, 2–5.

Barash, Y. and Friedman, N. (2001) Context specific bayesian clustering for gene expression data. In *Fifth Annual International Conference on Computational Biology* (Ed. T. Lengauer), Montreal, Canada. RECOMB 2001.

Beckmann, C. and Smith, S. (2002) Probabilistic independent component analysis for FMRI. Technical report, FMRIB, University of Oxford, UK.

Ben-Hur, A., Elisseeff, A. and Guyon, I. (2002) A stability based method for discovering structure in clustered data. *Pac Symp Biocomput. 2002*.

Bottou, L. and Bengio, Y. (1995) Convergence properties of the K-means algorithms. In *Advances in Neural Information Processing Systems* (Eds G. Tesauro, D. Touretzky and T. Leen), volume 7, pp. 585–592. The MIT Press.

Breiman, L. (1996a) Heuristics of instability and stabilization in model selection. *Annals of Statistics*, **24(6)**, 2350–2383.

Breiman, L. (1996b) Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (1999) Random forests. Technical report, University of California, Berkeley.

Breiman, L., Friedman, J., Olshen, R. and C., S. (1984a) *Classification and Regression trees*. Belmont (CA): Wadsworth Int.

Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984b) *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks/Cole.

Bruton, D. and Owen, A. (1988) The Norwegian Upper Ordovician illaenid trilobites. *Norsk Geologisk Tidsskrift*, **68**.

Bühlmann, P. and Yu, B. (2003) Boosting with the l_2-loss: Regression and classification. *Journal of the American Statistical Association*, **98**(462), 324–339.

Campbell, N. and Mahon, R. (1974) A multivariate study of variation in two species of rock crab of genus *leptograptus*. *Australian Journal of Zoology*, **22**.

Chipman, H., Hastie, T. and Tibshirani, R. (2003) Clustering microarray data. In *Statistical analysis of gene expression microarray data* (Ed. T. Speed), chapter 4. New York: Chapman and Hall.

Cho, R., Campbell, M., Winzeler, E., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T., Gabrielian, A., Landsman, D., Lockhart, D. and Davis, R. (1998) A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular Cell*, **2(1)**, 65–73.

Dempster, A., N.M.Laird and D.B.Rubin (1977) Maximum likelihood from incomplete data via the EM algorithm. *Journal Royal Stat. Soc., Series B*, **39**(1), 1–38.

Devijver, P. and Kittler, J. (1982) *Pattern Recognition: A Statistical Approach*. Prentice Hall.

Devlin, S., Gnanadesikan, R. and Kettenring, J. (1981) Robust estimation of dispersion matrices and principal components. *Journal of the American Statistical Association*, **76**, 354–362.

Diaconis, P. and Freedman, D. (1984) Asymptotics of graphical projection pursuit. *Annals of Statistics*, **12**, 793–815.

Duan, K., Keerthi, S. and Poo, A. N. (2003) Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, **51**, 41–59.

Duffy, D. and Quiroz, A. (1991) A permutation-based algorithm for block clustering. *Journal of Classification*, **8**, 65–91.

Efron, B. (1975) The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association*, **70**(352), 892–898.

Efron, B. (1983) Estimating the error rate of a prediction rule. Improvements on cross-validation. *JASA*, **78**, 316–331.

Eisen, M., Spellman, P., Brown, P. and Botstein, D. (1998) Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, **95**, 14863–14868.

Eslava-Gómez, G. (1989) *Projection Pursuit and Other Graphical Methods for Multivariate Data*. D. Phil. thesis, University of Oxford.

Fauquet, C., Desbois, D., Fargette, D. and Vidal, G. (1988) Classification of furoviruses based on the amino acid composition of their coat proteins. In *Viruses with Fungal Vectors* (Eds J. I. Cooper and M. J. C. Asher), pp. 19–38, Edinburgh. Association of Applied Biologists.

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, **7**, 179–188.

Frawley, W. J., Piatetsky-Shapiro, G. and Matheus, C. J. (1992) Knowledge discovery in databases - an overview. *Ai Magazine*, **13**, 57–70.

Freund, Y. and Schapire, R. E. (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pp. 23–37.

Friedman, J. (1999) Greedy function approximation: a gradient boosting machine. Technical report, Stanford University.

Friedman, J. and Tukey, J. (1974) A projection pursuit algorithms for exploratory data analysis. *IEEE Transactions on Computers*, **23**.

Friedman, J., Hastie, T. and Tibshirani, R. (1998) Additive logistic regression: a statistical view of boostin. Technical report, Stanford University.

Friedman, J. H. (1989) Regularized discriminant analysis. *Journal of the American Statistical Association*, **84**(165).

Getz, G., Levine, E. and Domany, E. (2000) Coupled two-way clustering analysis of gene microarray data. *Proc. Natl. Acad. Sci.*, **97(22)**, 12079–12084.

Ghosh, D. and Chinnaiyan, A. (2002) Mixture modelling of gene expression data from microarray experiments. *Bioinformatics*, **18**.

Gower, J. and Hand, D. (1996) *Biplots*. London: Chapman & Hall.

Hamadeh, H., Bushel, P., Jayadev, S., DiSorbo, O., Sieber, S., Bennett, L., Li, L., Tennant, R., Stoll, R., Barrett, J., Blanchard, K., Paules, R. and Afshari, C. (2002) Gene Expression Analysis Reveals Chemical-Specific Profiles. *Toxicological Sciences*, **67**, 219–231.

Hand, D. (1997) *Construction and assessment of classification rules*. Wiley: Chichester.

Hartigan, J. (1972) Direct clustering of a data matrix. *Journal of the American Statistical Association*, **6**, 123–129.

Hastie, T., Tibshirani, R. and Buja, A. (1994) Flexible discriminant analysis by optimal scoring. *Journal of the American Statistical Association*, **89**(428), 1255–1270.

Hastie, T., Buja, A. and Tibshirani, R. (1995) Penalized discriminant analysis. *Annals of Statistics*, **23**, 73–102.

Hastie, T., Tibshirani, R. and Friedman, J. (2001) *The elements of statistical learning: data mining, inference and prediction*. New York: Springer.

Holst, P. A., Kromhout, D. and Brand, R. (1988) For Debate: Pet birds as an Independent Risk Factor of Lung Cancer. *British Medical Journal*, **297**.

Hori, G., Inoue, M., Nishimura, S. and Nakahara, H. (2001) Blind gene classiffication based on ICA of microarray data. In *Proc. 3rd Int, Workshop on Independent Component Analysis and Blind Separation*, pp. 332–336. ICA2001.

Hyvärinen, A. (1999) Survey on independent component analysis. *Neural computing surveys*, **2**, 94–128.

Kari, L., Loboda, A., Nebozhyn, M., Rook, A. H., Vonderheid, E. C., Nichols, C., Virok, D., Chang, C., Horng, W., Johnston, J., Wysocka, M., Showe, M. K. and C., S. L. (2003) Classification and prediction of survival in patients with the leukemic phase of cutaneous t cell lymphoma. *Journal of Experimental Medicine*, **197**(11), 1477–88.

Kaufman, L. and Rousseeuw, P. (1990) *Finding groups in data: An Intorduction to cluster analysis*. New York: Wiley.

Kerr, M. and Churchill, G. (2001) Bootstrapping cluster analysis: assessing the reliability of conclusions from microarray experiments. *Proc. Natl. Acad. Sci.*, **98(16)**, 8961–8965.

Kohonen, T. (1982) Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**, 59–69.

Kohonen, T. (1988) Learning vector quantization. *Neural Networks*, **1**(Supplement 1), 303.

Kustra, R. and Strother, S. (2001) Penalized discriminant analysis of $[^5O]$-water PET brain images with prediction error selection of smoothness and regularization hyperparameters. *IEEE Transactions on Medical Imaging*, **20**(5), 376–387.

Lazzeroni, L. and Owen, A. (1992) Plaid models for gene expression data. *Statistica Sinica*, **12**, 61–86.

Lee, S. and Batzoglou, S. (2003) Application of independent component analysis to microarrays. *Genome Biology*, **4:R76**, 658—666.

Lee, S.-M. (2003) *Estimating the number of independent components via the SONIC statistic*. Master's thesis, University of Oxford, United Kingdom.

Liebermeister, W. (2002) Linear modes of gene expression determined by independent component analysis. *Bioinformatics*, **18**, 51–60.

Macnaughton-Smith, P., Williams, W., Dale, M. and Mockett, L. (1964) Dissimilarity analysis: a new technique of hierarchical sub-division. *Nature*, **202**.

Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979) *Multivariate analysis*. London: Academic Press.

Marriott, F. (1974) *The interpretation of multiple observations*. London: Academic Press.

Massart, D. L., Kaufman, L. and Plastria, F. (1983) Non-hierarchical clustering with masloc. *Pattern Recognition*, **16**, 507–516.

McLachlan, G. and Peel, D. (1998) *Robust cluster analysis via mixtures of multivariate t distributions*, pp. 658–666. Berlin: Springer-Verlag.

McLachlan, G. and Peel, D. (2000) *Finite Mixture Models*. New York: Wiley.

McLachlan, G., Bean, R. and Peel, D. (2002) A mixture model-based approach to the clustering of microarray expression data. *Bioinformatics*, **18**.

McQueen, J. (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematics, Statistics and Probability*, volume 1.

McShane, L., Radmacher, M., Freidlin, B., Yu, R., Li, M. and Simon, R. (2002) Methods for assessing reproducibility of clustering patterns observed in analyses of microarray data. *Bioinformatics*, **18**.

Michie, D., Spiegelhalter, D. J. and Taylor, C. C. (1994) *Machine Learning, Neural and Statistical Classification*. Chichester: Ellis Horwood.

Quenouille, M. (1949) Approximate tests of correlation in time series. *JRSSB*, **11**, 68–84.

Quinlan, J. R. (1993) *C4.5: Programs for Machine Learning*. Morgan Kauffman.

Ramsey, F. L. and Schafer, D. W. (2002) *The statistical sleuth: A course in methods of data analysis*. Pacific Grove, CA: Duxbury, second edition.

Ripley, B. D. (1996) *Pattern recognition and neural networks*. Cambridge: Cambridge University Press.

Rosenblatt, F. (1962) *Principles of Neurodynamics*. Washington D.C.: Spartan Books.

Rumelhart, D. E. and McClelland, J. L. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. Cambridge: MIT Press.

Schölkopf, B. and Smola, A. (2002) *Learning with kernels: Support vector machines, regularization, optimization and beyond*. The MIT Press, Cambridge (MA).

Shipp, M., Ross, K., Tamayo, P., Weng, A., Kutok, J., Aguiar, R., Gaasenbeek, M., Angelo, M., Reich, M., Pinkus, G., Ray, T., Koval, M., Last, K., Norton, A., Lister, T., Mesirov, J., Neuberg, D., Lander, E., Aster, J. and Golub, T. (2002) Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine*, **8**, 68–74.

Singh, D., Febbo, P., Ross, K., Jackson, D., Manola, J., Ladd, C., Tamayo, P., Renshaw, A., D'Amico, A., Richie, J., Lander, E., Loda, M., Kantoff, P., Golub, T. and Sellers, R. (2002) Gene expression correlates of clinical prostate cancer behaviour. *Cancer Cell*, **1(2)**.

Sørlie, T., Perou, C., Tibshirani, R., Aas, T., Geisler, S., Johnsen, H., Hastie, T., Eisen, M., van de Rijn, M., Jeffrey, S., Thorsen, T., Quist, H., Matese, J., Brown, P., Botstein, D., Eystein, L. and Borresen-Dale, A. (2001) Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proc. Natl. Acad. Sci.*, **98(19)**.

Stone, M. (1974) Cross-validatory choice and assessment of statistical predictions (with discussion). *JRSSB*, **36**, 111–147.

Tibshirani, R., Walther, G. and Hastie, T. (2001) Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society B*, **63(2)**, 411–423.

Vapnik, V. (1995) *The Nature of Statistical Learning Theory*. New York: Springer.

Vapnik, V. and Chervonenkis, A. J. (1964) On the one class of the algorithms of pattern recognition. *Automation and Remote Control*, **25**(6).

Vapnik, V. and Chervonenkis, A. J. (1968) On the uniform convergence of relative frequencies of events to their probabilities. *Sov. Math. Dokl.*, pp. 781–787.

Vapnik, V. and Chervonenkis, A. J. (1971) On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Apl.*, **16**, 264–280.

Vapnik, V. and Lerner, A. J. (1963) Generalized portrait method for pattern recognition. *Automation and Remote Control*, **24**(6).

van 't Veer, L., Dai, H., van de Vijver, M., He, Y., Hart, A., Mao, M., Peterse, H., van der Kooy, K., Marton, M., Witteveen, A., Schreiber, G., Kerkhoven, R., Roberts, C., Linsley, P., Bernards, R. and Friend, S. (2002) Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, **415**.

Venables, W. and Ripley, B. (2002) *Modern Applied Statistics with S*. New York: Springer-Verlag, fouth edition.

Widrow, B. and Hoff, M. E. (1960) Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, volume 4, pp. 96–104. New York: IRE.

Witten, I. H. and Frank, E. (1999) *Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations*. San Francisco: Morgan Kaufmann.

Yeung, K. and Ruzzo, W. (2001) Prinicipal component analysis for clustering gene expression data. *Bioinformatics*, **17**.

Yeung, K., Fraley, C., Murua, A., Raftery, A. and Ruzzo, W. (2001a) Model-based clustering and data transformations for gene expression data. *Bioinformatics*, **17**.

Yeung, K., Haynor, D. and Ruzzo, W. (2001b) Validating clustering for gene expression data. *Bioinformatics*, **17**.