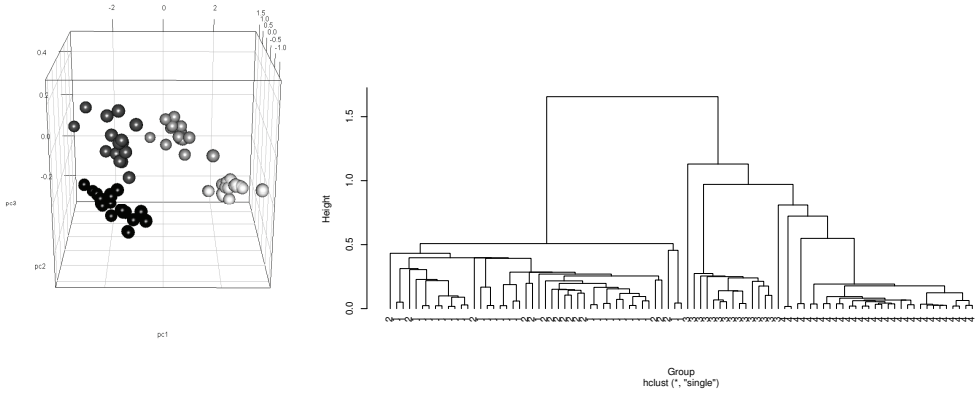
Zürcher Hochschule
für Angewandte Wissenschaften

Life Sciences und
Facility Management

MSc Life Sciences
D3:Modelling and Exploration of Multivariate Data
Part 1: Exploratory Data Analysis



October 2019

Institute of Applied Simulation, ZHAW

Thomas Ott

Contents

Introduction

The first part of the D3 is dedicated to the field of exploratory data analysis. The main goal of this field is to identify patterns in data. We aim to uncover the underlying structure and extract important information in the data. Exploratory data analysis is sometimes introduced as a pre-step to classical statistical hypothesis testing. However, the field has evolved into something that goes beyond classical statistics by offering methods whose starting points and goals are different from the parametric models used in statistics. In fact, the field of exploratory data analysis (as we understand it) is inspired by the machine learning and pattern recognition philosophy and by data visualisation techniques with the goal to tell a story about and with the data. While classical statistics often makes tight assumptions about the problems and the data distributions, the goal of exploratory data analytics – as we understand it – is simply to learn from data. To this end, also methods that make little assumptions of the data, the structure of the data sources or even heuristics[1] are employed. We may see this approach in line with a general understanding of machine learning ML (more precisely, we may mainly think of unsupervised ML) that has received much attention lately. Driving factors of the development are the availability of 'big data', i.e. large amounts of data, and the increase in computing power. Again, the general question is: what can be learnt from a heap of data?

Plain data usually do not have much value. Only when we can interpret them, data reveal their power. Hence, exploratory data analysis wants to 'make sense out of data'. In many cases this means that we have to make data "manageable" in order to be able to extract important information. For this

[1] A heuristic is a method for learning from data or for problem solving for which no rigid mathematical proof for its optimality might be available. Often, such a method is inspired by previous experience or by examples in nature. In machine learning, many methods can be considered to have developed from heuristics such as neural networks.

purpose, visualisation of data structures is a central tool of communication. This not only holds for the communication between humans, but also for letting data themselves speak to us. Learning about suitable visualisation techniques thus will be a central aspect of our lectures.

You will notice that the terminology used in this part can slightly differ from the nomenclature in traditional statistics. We will try to make connections where helpful or necessary.

About these Lecture Notes:
These lecture notes are supposed to provide supportive material for the lectures. You should not expect completeness and these notes can still be improved in many ways. Please report any mistakes you find. Suggestions and ideas for next year's edition are also most welcome.

All the programs (listings) and soltutions to the exercises in these notes are available as separate R scripts.

Chapter 1

Some Preparatory Material

In this chapter, we briefly discuss some concepts and methods that are assumed to be prerequisites for the following parts. You are supposed to prepare the material before the first lessons.

1.1 Covariance and Correlation

Consider two variables x and y. The variance $var(x) = \sigma_x^2$ of a variable x is a measure of its statistical spread. It is estimated on the basis of the corresponding sample values s_x^2, i.e.

$$var(x) \approx s_x^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{1.1}$$

where $\{x_1, ...x_N\}$ is the sample data set. Note: For samples, usually the factor $\frac{1}{N-1}$ is used.[1] For large data sets, the difference is, however, marginal. The standard deviation σ_x is the square root of $var(x)$. $var(y)$ and σ_y for y are defined analogously.
The covariance $covar(x, y)$, in turn, is a measure of the interdependence of x and y. The sample-based estimation is calculated as follows:

$$covar(x, y) = covar(y, x) \approx \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x}) \cdot (y_i - \bar{y}) \tag{1.2}$$

[1] In many books, estimations based on samples are distinguished from the population values by using different letters: s and s^2 are used for the sample standard deviation and variance, σ and $var(x) = \sigma^2$ are used for the population values.

From the covariance we can derive the Pearson correlation index r. In fact, it is just the covariance, normalised with the corresponding standard deviations σ_x and σ_y.

$$r = \frac{covar(x,y)}{\sigma_x \cdot \sigma_y} \tag{1.3}$$

This operation assures that r is dimensionless and $-1 \leq r \leq 1$. The correlation measures the strength of the linear relation between two variables.

Variances and covariance(s) are usually put together in a matrix, the so called **covariance matrix** Σ. For two variables, Σ is a 2×2 matrix:

$$\Sigma = \begin{pmatrix} var(x) & covar(x,y) \\ covar(x,y) & var(y) \end{pmatrix} \tag{1.4}$$

Attention: Do not confuse the covariance Σ with the sigma sign for sums! The covariance matrix determines the shape of a two (or higher)-dimensional normal distribution, as it contains all information about the spread and the correlation of the variables. In R we can define a 2×2 covariance matrix as follows:

```
covm=array(c(varx,covarxy,covarxy,vary),dim=c(2,2))
```

Similarly, we can define the correlation matrix. In many cases, we want to calculate the covariance matrix or the correlation matrix from given data. For this, we use the commands

```
cov(data)
cor(data)
```

It is often convenient to visualise the correlation matrix. In Program 1a, this is done for the Iris data set.[2] The result is depicted in Fig.1.1.

```
#Program 1a
library(corrplot)  #install first if necessary
cor(iris[,1:4])
corrplot(cor(iris[,1:4]),method="color")
corrplot(cor(iris[,1:4]),method="number")
plot(iris[,1:4]) #scatterplot for all pairs of variables
```

Listing 1.1: Correlation plots

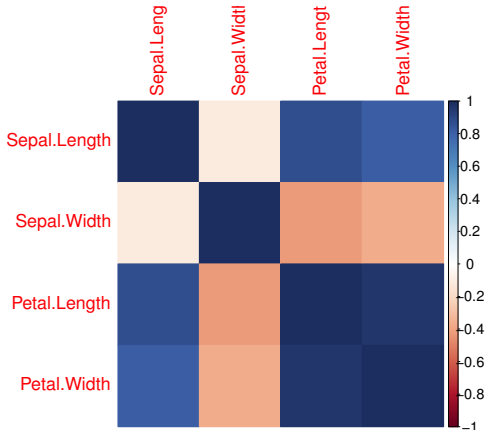


Figure 1.1: Correlation plot for the Iris data set

What can we learn from this?
Variables with a strong negative or positive correlation contain similar information about the data distribution. If the correlation is close to 1 or -1, we can thus often reduce the number of variables by using only one of the variables.

[2]The Iris data set is available in R by default and you should have encountered it in module D1.

Exercises:

Exercise 1.1.1 *a) Which of the Iris data variables are most strongly correlated? Take a look at the corresponding scatterplot (see Program 1a). How does it look like?*
b) The command corrplot allows for different methods. Try them out!

Exercise 1.1.2 *You may also use the library ggplot2 for correlation plots. The commands are*

```
library(ggplot2)
library(reshape2)
ggplot(data = melt(cor(dataset)), aes(x=Var1, y=Var2, fill=value)) +
geom_tile()
```

Adapt the commands and apply them to the Iris data set.
What is the role of the function melt again?

1.2 Generating Data Distribution Samples

In some situations it is useful to generate data on your own. It allows you to create toy data sets whose characteristics are known and can be controlled. Based on this, you can for instance more easily test and understand an analysis method that is new to you. In practice, data are often assumed to be normally distributed (the Central Limit Theorem gives reasons why many variables are normally distributed). The corresponding distributions are called *normal distributions* or *Gaussian distributions*.

♣ Example: Consider a collection of 150 leaves from one tree for which you examine the two features/variables *length* and *weight*. Both features are assumed to be normally distributed. How can we generate a corresponding sample data set? Is it enough to know the mean and the variance of each variable? ♣

This is an example of bivariate normally distributed data. The data consist of two normally distributed variables (features).

R−application: The following programme randomly generates a sample of 150 leaves (details will become clear in the next subsection). The size of the leaves is stored in x and the weight is stored in y. The units of the variables are arbitrary in this example.

```
#Program 1b
library(MASS)
mu=c(1,4)
varx=1; vary=1;covarxy=0.8
covar=array(c(varx,covarxy,covarxy,vary),dim=c(2,2))
   #covariance matrix
leaves=mvrnorm(150,mu,covar)  # two-dimensional Gaussian
x=leaves[,1]
y=leaves[,2]
hist(x)
hist(y)
plot(x,y)
```

Listing 1.2: 2D normal distribution

We can use the generated values to examine some characteristics of 2D normal

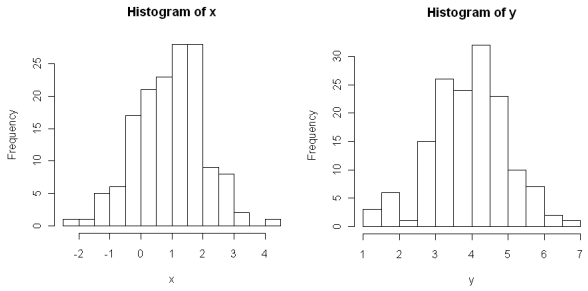


Figure 1.2: x and y are normally distributed.

distributions:

1. **Normality of each variable:** You can check the assumption that each single variable is normally distributed. Your histograms should look similar to the ones in Fig. 1.2 (since the numbers are generated by a random process, the results will differ slightly from trial to trial).

2. **Mean and standard deviation of each variable:** The mean of the size variable x is approximately $\bar{x} \approx 1$ and the mean of the weight variable is approximately $\bar{y} \approx 4$. The corresponding standard deviations are approximately $s_x \approx s_y \approx 1$.

3. In general, the 2D (or 3D, 4D,...) data distribution of a normal distribution is not just a simple product of one-dimensional normal distributions. This is only the case if the variables are independent, i.e. uncorrelated. For the example this would mean that the weight of a leaf y is not correlated to its length x, which is of course wrong (in general, we expect: the longer a leaf, the greater its weight). We can actually determine the correlation (Pearson index) using the R command *cor(x,y)*, which should yield a value $r \approx 0.8$.

Some useful R-Commands for Data Generation

```
library(MASS);
rnorm(n=5,mean=0,sd=1) #1D normal distribution
runif(n=5,min=0,max=1) #1D uniform distribution
mvrnorm(150,mu,covar)
#multivariate normal distribution
```

Exercises:

Exercise 1.2.1 *a) Change Program 1b such that it generates a 2D normal distribution that is centered around the point* $(5, 5)$.
b) What happens if you increase or decrease the covariance?

Exercise 1.2.2 *Using the following program you can generate a scatter plot that corresponds to a simple multiplication of 1D normal distributions.*

```
-----------------------------------------------------------------
mux=1; sigmax=1
x=rnorm(150,mux,sigmax)
muy=4; sigmay=1
y=rnorm(150,muy,sigmay)
plot(x,y)
-----------------------------------------------------------------
```

a) Compare the scatter plot with the one obtained from Program 1b.
b) What is the correlation of x *and* y*? Compare it for both programs using the R-command* cor(x,y).

Exercise 1.2.3 *Run Program 1b several times with different covariance matrices and observe the change in the shape of the data distribution, using a scatter plot (R command: plot(x,y)).*
a) For the first runs, use the following covariance matrices:

$$\Sigma_1 = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \quad \Sigma_2 = \begin{pmatrix} 1 & -0.5 \\ -0.5 & 1 \end{pmatrix} \quad \Sigma_3 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

What would the leaves from a tree look like in these three cases? (assume that x: length, y: width) What is the correlation of the two variables in these cases?

b) How should you choose the covariance values if you want to have a perfect circular distribution? What would the leaves then look like?

c) How do the values $var(x), var(y)$ and $covar(x, y)$ influence the shape of the data distribution? Describe it in your own words.

1.3 Feature Scaling

If we are dealing with independent variables (features) of data, the scales of the features can be very different. This makes comparisons between data instances (different observations of the data sample) difficult or even almost impossible as the distances between data points are governed by the features with the broadest range of values.
As an example, let us consider a sample of male human being for which we measure the *height* in and the *weight*, using the SI-units *meter* and *kilogram*. As the height is typically around 1.80 m and the weight is perhaps around 80 kg, the scales are clearly different. If we do not correct for the different scales, the differences between observations will be largely dominated by the weight. Make it plausible with the following lines.

```
-------------------------------------------------------------
mux=1.8; sigmax=0.1   #height
x=rnorm(150,mux,sigmax)
muy=80; sigmay=10
y=rnorm(150,muy,sigmay)
plot(x,y,xlim=c(0, 120), ylim=c(0, 120))
#compare to plot(x,y)

-------------------------------------------------------------
```

The plot-function (plot(x,y)) is very nice because it automatically rescales the axes for the visual eye if you do not fix the limits. But this can also be dangerous as it hides potential problems that may occur when you further analyse the data.

It is often advantageous to explictly standardise/normalise the features in a preprocessing step, where a value x (or y) is mapped onto a normalised or rescaled value x' (or y'). The following scaling methods are very common:

- **Min-Max Rescaling:**

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}. \tag{1.5}$$

  The range of x' is $[0, 1]$.

- **Mean Normalisation:**

$$x' = \frac{x - \bar{x}}{\max(x) - \min(x)}. \tag{1.6}$$

- **Standardisation:**

$$x' = \frac{x - \bar{x}}{\sigma_x}, \tag{1.7}$$

  where σ_x is the standard deviation. This operation is performed by the R-command *scale*.

Exercises:

Exercise 1.3.1 *Apply standardisation (R-command* scale*) to the data generated in this section (see lecture notes) and display the data in a scatterplot with suitable limits for the axes. Use different values for the variances of x and y.*

Chapter 2

About Multi- and High-Dimensional Data

If you're lucky, and a building succeeds, the real product has many more dimensions than you can ever imagine. You have the sun, the light, the rain, the birds, the feel.

Peter Zumthor, architect

2.1 What are High-Dimensional Data?

High-dimensional data are typically data that are represented in a data space with many dimensions. That is, each observed data item is described by many variables and can be represented by a vector with many dimensions. By pure statisticians the term 'high-dimensional data' is usually – if at all – only used for data for which the number of dimensions is larger than the sample size. In exploratory data analysis, we will however use the term generally for all kinds of data with at least four dimensions. Moreover, for this kind of data, we will use the terms *high-dimensional* and *multi-dimensional* as equivalents.
Real data, in this sense, are often very high-dimensional. The problem with this type of data is that a direct visual analysis, e.g. by means of scatter

plots, is not possible in a straightforward way. If the dimensionality increases drastically, which is for instance often the case for most modern applications of machine learning, then also the so-called *curse of dimensionality* comes into plays. Generally, this describes the phenomenon that with increasing dimensionality, the data becomes more and more sparsely distributed and the distances between data points become more equal. Hence, it becomes increasingly harder to detect meaningful structure. In our lectures, we will become familiar with some special techniques that – nonetheless – allow us to make statements about high-dimensional data.

Why are real data often high-dimensional?
The reason is because real data often belong to one (or both) of the following types.

- **Multivariate measurements:** We are often interested in many features when measuring an object. E.g., when examining a patient, the doctor measures temperature, blood pressure and many other parameters. Each feature or variable corresponds to one dimension. If the variables are measured repeatedly for different objects, we obtain several **data instances** (observations). They can be interpreted as data points in a **feature space**. In the case of two (or three) dimensions, we can plot the data distribution in the feature space by means of a scatter plot. More than three dimensions, of course, cannot directly be visualised in this way.

- **Time series:** Many measurements are repeated in time for the same object. If the repetition can be performed automatically, it may give rise to a very long time series. We may interpret each measurement as a data instance of one dimension. However, it is often advantageous to interpret each time point as one dimension, as this facilitates the comparison of time series. Such time series vectors or feature spaces respectively are typically very high-dimensional.

2.1.1 Examples of high-dimensional data sets

R includes a variety of data sets in the *datasets* and *MASS* packages, some of them are well established in the scientific literature about data analysis as test and benchmark data sets. After including the packages, you can see an overview of the data sets with the following command:

Data set	short description	numeric dimensions	instances
Iris data set	measurements of flowers	4	150
Fever curves	time series	10	340
Colon cancer data set	gene expression levels	2000	62
Isis chemical data set	chemical descriptors	166	153
Mammal teeth data set	dental formulas	8	23

Table 2.1: Overview of the data sets used

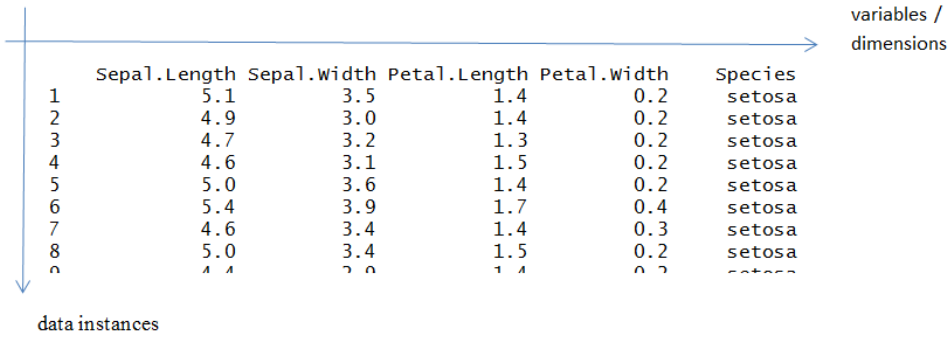


Figure 2.1: Structure of the data sets

```
------------------------------------------------------------
> data()
------------------------------------------------------------
```

In the following section, we will introduce a couple of additional data sets that we will use for further analysis and exercises. An overview is given in Table 2.1.

All the data sets are used in the same structure with the variables/features corresponding to columns and the data instances (observations) stored in rows (Fig. 2.1).

Detailed Description

1. The **iris data set** gives the measurements in centimetres of the variables sepal length and width and petal length and width, respectively,

for 50 flowers from each of 3 species of Iris. The species are Iris setosa, versicolor, and virginica.
The data set has four numeric dimensions. You may plot a 3D-projection of the first 3 dimension by executing the following commands:

```
-------------------------------------------------------------
library(rgl) #possibly you have to install the package first
x=iris[,1]; y=iris[,2]; z=iris[,3];
plot3d(x,y,z)
#alternative: plot3d(iris)
-------------------------------------------------------------
```

The result is depicted in Fig. 2.2.

2. We will use the **fever curves** data set as an example for time series. The data set was artificially created and serves didactical purposes. It contains the temporal courses of the temperature of 340 patients with different diseases. For each patient, 10 measurements at fixed points in time were recorded. The times are called $T1, T2, ..., T10$. Hence, technically we have to deal with 10-dimensional vectors. Time series are usually depicted in line charts, where the $x-$axis shows the time and the $y-$axis the measurements. The fever time series are depicted in Fig. 2.3. For multiple time series there is a specific command in R:

```
-------------------------------------------------------------
library (graphics) #install first if necessary
ts.plot(t(fever))
-------------------------------------------------------------
```

For the *ts.plot* command, the single time series vectors have to correspond to columns. As this switches the structure of features/dimensions and data instances as described above, the data matrix hast to be transposed (command t()).
How many different diseases can you distinguish by viewing the time series?

3. The **colon cancer data set** (Details in: *Alon et al., Proc. Natl. Acad. Sci. USA. 96. p. 6745-6750, 1999*) contains gene expression levels of 40 tumor and 22 normal colon tissues for 2000 genes. One of the

tasks is to distinguish colon cancer from normal tissues by identifying important genes and special expression patterns. Formally, the data set is 2000-dimensional and contains 62 samples. The expression levels are contained in the file *cancer.txt*. The identity of the 62 tissues is given in the file *canceridentity.txt*. The numbers correspond to patients, a positive sign to a normal tissue, and a negative sign to a tumor tissue. The expression profiles are depicted in Fig. 2.4 in a level plot. Can you discriminate between cancerous and healthy tissues?

4. The **isis chemical data set** contains so-called chemical fingerprints of 153 chemical substances. Chemical fingerprints (or chemical keys) are vectors whose values are based on a coding scheme for chemical substances, indicating the presence or absence of a few hundred predefined functional structures in a substance. For instance, one structure could be the carboxyl group $-CO_2H$, defining one component of the fingerprint. If the substance A contains such a group, the component is set to 1, otherwise it is 0. The isis chemical data set includes 166 predefined structures, hence it is 166-dimensional. Details can be found in: *Ott et al, J. Chem. Inf. Comput. Sci., 44 (4), pp 1358-1364, 2004.*

5. The **mammal teeth data set** contains the dental formulas of 23 mammals. These formulas record the number of teeth of each kind on one side of the upper and lower jaws (deutsch: Kiefer). For example, the formula of old-world monkey is 2-1-2-3 / 2-1-2-3. These values before the slash record the upper jaw: 2 incisors (Schneidezahn), 1 canine (Eckzahn), 2 premolars (vorderer Backenzahn), 3 molars (Backenzahn); after the slash they record the lower jaw. These 8 numbers are joined in one vector $(2, 1, 2, 3, 2, 1, 2, 3)$ which defines the first column of the matrix below (be aware that coloumns and rows are interchanged with respect to data instances and dimensions in this figure).

	incisor1	incisor2	canine	premolar1	premolar2	molar1	molar2	molar3
Old-world monkeys	2	1	2	3	2	1	2	3
New-world monkeys	2	1	3	3	2	1	3	3
Marmosets	2	1	3	2	2	1	3	2
Insectivorous bats	2	1	3	3	3	1	3	3
Frugivorous bats	2	1	2	3	2	1	3	3
Hedgehogs	3	1	3	3	2	1	2	3
Moles	3	1	4	3	3	1	4	3
Cats	3	1	3	1	3	1	2	1
Dogs	3	1	4	2	3	1	4	3
Racoons	3	1	4	2	3	1	4	2
Hyaenas	3	1	4	1	3	1	3	1
Weasels	3	1	4	1	3	1	4	2
Eared seals	3	1	4	1	2	1	4	1
Ordinary seals	3	1	4	1	3	1	4	1
Walruses	1	1	3	0	0	1	3	0
Hippopotamuses	2	1	4	3	2	1	4	3
Pigs	3	1	4	3	3	1	4	3
Camels	1	1	3	3	3	1	2	3
Chevrotains	0	1	3	3	3	1	3	3
Hollow-horned ruminants	0	0	3	3	3	1	3	3
Tapirs	3	1	4	3	3	1	3	3
Horses	3	1	3	3	3	1	3	3
Rabbits	2	0	3	3	1	0	2	3

Table: Dental formulas of 23 mammals.

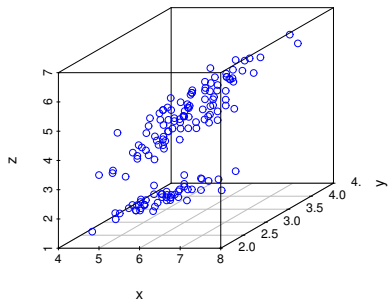


Figure 2.2: A 3D-plot of the iris data set: x: sepal length, y sepal width and z: petal length.

R−application: You can load all the data set into your workspace using Program a. Make sure that your working directory is set correctly.

```
#Program a:
#make sure that you are in the right directory

#loading iris data set
library(MASS)
data(iris)

#loading fever data set
fever=read.csv(file="fever.csv",header=TRUE)

#loading colon cancer data set
cancer=t(read.table("cancer.txt",header=FALSE))

#loading isis chemical data set
isisall=read.csv(file="isisall.csv",header=TRUE)
isis=isisall[,2:167] #only numerics
# use the data set 'isis' for the analysis

#loading mammal teeth data set
dental=read.table("dental.csv",header=TRUE,sep=",")
```

Listing 2.1: Loading all the data sets

The fever curve data set is an example of a set of multiple time series. Time series data are often depicted in a line plot as shown in Fig. 2.3. A similar kind of plot can also be used for other data sets with multivariate data (no time series) for which it is known as **parallel coordinates plot**. The corresponding command in R automatically rescales the axis to the (visual) interval $[0, 1]$. An example for the Iris data set is given in Fig. 2.5.

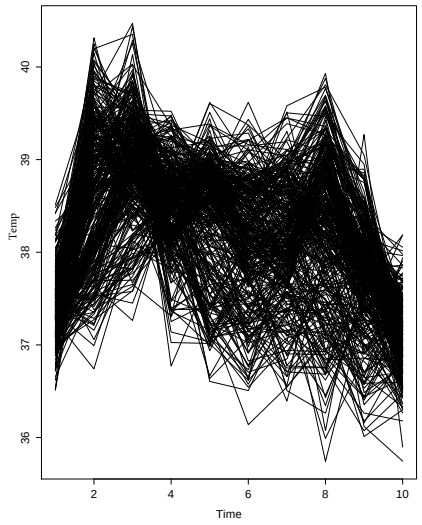


Figure 2.3: Fever curves. How many different diseases are there?

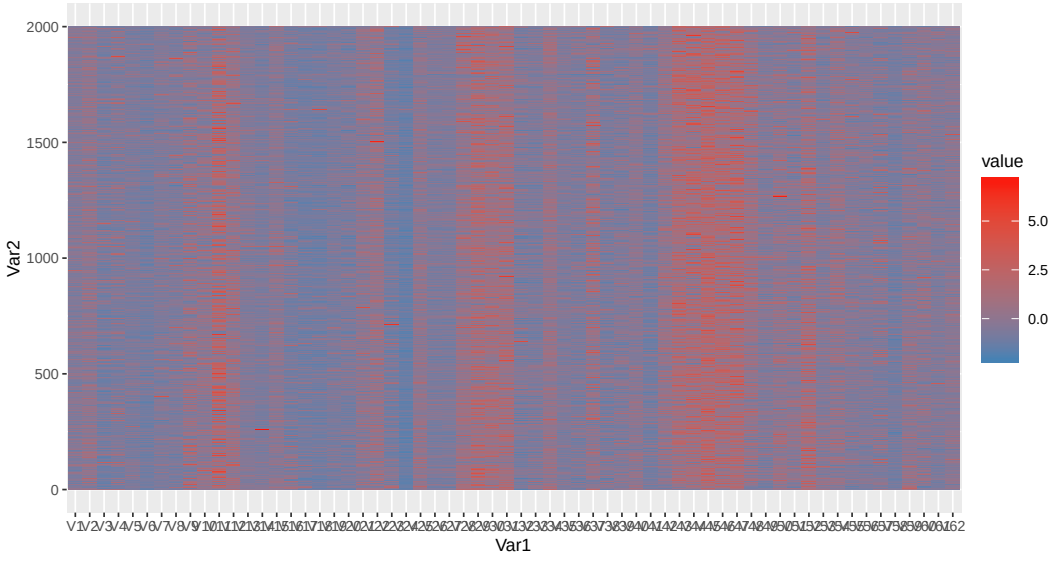


Figure 2.4: Gene expression profiles after rescaling shown in a level plot

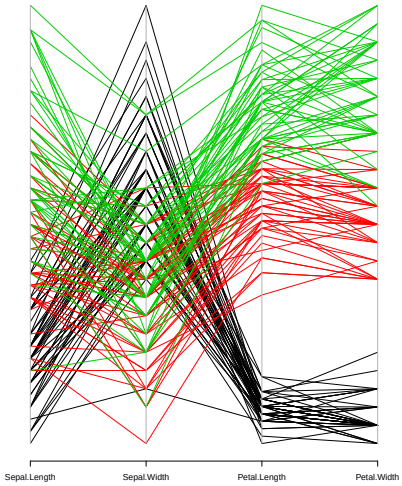


Figure 2.5: Parallel coordinates plot for the Iris data set.

2.1.2 Summary of some useful visualisation techniques

Together with the data sets we have already got to know some useful techniques in the last section that allow for a visual inspection of the structure of potentially high-dimensional data.

- **Rotatable 3D scatter plots:** The command *plot3d(data)* from the package/library *rgl* allows for 3-dimensional rotatable scatterplots.[1]

- **Parrallel coordinates plots**: In a parallel coordinates plot, the axes of all the variables are placed in parallel to each other. As each axis can have a different scale, the data is visually rescaled. This is often advantageous for the visual inspection and further analysis, but one has to be aware that the scale information is lost. An example is given in Fig. 2.5.
  You may use the command *parcoord(data)* from the package *MASS*, as in

[1]Mac-users have reported that they needed to install an extra program to make it run. Use Google to find a quick fix.

```
library(MASS); parcoord(iris[1:4],col=iris$Species)
```

Alternative options exist such as

```
library(ggplot2);library(GGally)
ggparcoord(iris,columns=1:4,groupColumn = 5)
```

- **Time series plots:** The command *ts.plot* from the package *graphics* allows for producing classical time series plots such as the one in Fig. 2.3. Strictly, the plotting form is reserved for time series. However, for the purpose of a visual inspection, you might also apply it to other data in some cases with appropriate data scaling (but do not publish it in this form). In this case, you have to first transpose the data, as in

  ```
  library(graphics); ts.plot(t(iris[1:4]))
  ```

  The difference to the parallel coordinates plots is the fact that the axes are not rescaled.

- **Level plots:** Level plots as in Fig. 2.4 can also be very useful to visually recognize the main structure of high-dimensional data. In this plot, the original data matrix as in Fig. 2.1 (or its transpose) is actually displayed and the single data values are colour-coded according to some colouring scheme. A possibility to do this is offered by *ggplot* as demonstrated by the following lines.

  ```
  library (ggplot2)
  miris=melt(as.matrix(iris[,1:4]))  #only numeric dimensions
  ggplot(data=miris,aes(x=Var1, y=Var2)) +geom_tile(aes(fill=value))
  +scale_fill_gradient(low="steelblue",high="red")
  ```

  The command *melt* is necessary to put the data in a tidy form (see module D1) which is the standard format for *ggplot*.

Exercises - First (Visual) Inspections:

The purpose of the following exercises is that you inspect the data sets, mainly visually, and you try to find out something about the structure of the data. The exercises will lead you through the tasks.
You are asked to summarise the insights gained from these exercises on an extra sheet that will be distributed in class.

Exercise 2.1.1 *a) Load all the data sets mentioned from the Moodle course into your R-workspace using Program a (basic data sets: iris, fever, isis, dental, cancer).*
b) Study the characteristics of these data sets by displaying the data in R (number of points, dimension, ...). Do your observations support the descriptions of the data sets given above?
You may use the following commands:

```
dim(dataset)
summary(dataset)
str(dataset)
```

Which command is useful for the different data sets?

c) For each data set: display the data in a 2-dimensional ***scatter plot*** *by choosing any two dimensions that seem reasonable to you. You can use the normal* plot *command for this. Can you identify any structure (e.g. groups of similar data items)? If yes: what structure? If no: why not? What could be the problem in the case of the particular data set? (all the data sets are in fact strongly structured)*
d) No, use the command plot3d. *For which of the data sets is this useful? Example:*

```
library(rgl); plot3d(iris)
```

Exercise 2.1.2 *Illustrate all the data sets in a time series plot (line plot) or in a parallel coordinates plot. Can you identify any structure?*
Examples:

```
library(MASS); parcoord(iris[1:4])
library(graphics); ts.plot(t(iris[1:4]))
#ts.plot is actually for time series plots
#for ts.plot you have to transpose the data => t(data)
#such that a column contains a time series
#(or a data item respectively)
```

For which data set(s) is which kind of plot useful?
What's the difference between ts.plot *and* parcoord*?*
For the iris data set you may also colour the different classes as follows.

```
parcoord(iris[1:4],col=iris$Species)
```

Exercise 2.1.3 *Finally, we want to explore the data using a kind of level plot as shown in Fig. 2.4.*
We are using ggplot *that was introduced in module D1. First, we need to prepare the data using the command* melt*. The* melt *function stacks a set of columns into a single column of data which is used for a level plot. The following lines show the commands using the example of the iris data set.*

```
miris=melt(as.matrix(iris[,1:4]))  #only consider numeric dimensions
ggplot(data=miris,aes(x=Var1, y=Var2)) +geom_tile(aes(fill=value))
+scale_fill_gradient(low="steelblue",high="red")
```

a) What is the effect of the melt *function? Run it and answer the question.*
b) Now consider the level plot for the iris data set. How can the plot be interpreted? Is there any structure you can recognise?
c) Adapt the code for the other data sets (fever, isis, cancer, dental) and interpret the plots. For which data set(s) is this kind of plot useful?
d) Sometimes, e.g. for the cancer data set, it is advantageous to rescale the data using the command scale(cancer)*. Find out about the effect of this command. Plot it again in a level plot. Why is rescaling advantageous?*

Chapter 3

Principle Component Analysis (PCA)

PCA is a data analysis technique that is often used as a tool for data exploration. Usually the goal of a PCA is to reduce the dimension of high-dimensional data in order to get rid of noisy[1] data dimensions and/or to illustrate the relevant information content in 2D or 3D. To illustrate the concept, we first focus on 2D. We start with a little detour and discuss how data can be reinterpreted by rotating the axes.

3.1 Principle components in two dimensions

Rotation of axes

We start with an **example**:
Consider again a collection of leaves from a tree for which we examine the two features *length* and *width* and let us focus on the measurements of one leaf. This data can be depicted as a point in a scatter plot whose axes correspond to the two measured features (see Fig. 3.1). We could have measured many other features of the leaf of course, for instance its *thickness* or its *weight*. Some of these other features might be completely independent of the two features *length* and *width*, some of them depend on the features *length* and *width* in a functional way. That is, if we know the measurements for *length*

[1]Noisy data are randomly distributed data that do not contain useful information to describe the measurement.

and *width*, we can calculate the values for the new features. For instance the *thickness* of a leaf might be dependent on *length* and *width*, since long and broad leaves may tend to be thicker as well.

For the sake of the example, let us assume that there are two features, *thickness* and *shape abnormality*, that are supposed to be completely functionally dependent on *length* and *width*. The feature *shape abnormality* measures the deviation from the normal leaf shape. It is increased for short and broad leaves – the shorter and broader a leaf, the more abnormal it looks.
Let us assume that *thickness* and *abnormality* functionally depend on *length* and *width* as follows:

$$\begin{aligned} \text{thickness} &= \frac{1}{\sqrt{2}}\text{length} + \frac{1}{\sqrt{2}}\text{width} \\ \text{abnormality} &= \frac{-1}{\sqrt{2}}\text{length} + \frac{1}{\sqrt{2}}\text{width} \end{aligned} \tag{3.1}$$

This functional dependence allows us to characterise a leaf by means of the new dimensions *thickness* and *abnormality* instead of the dimensions *length* and *width*. For instance a leaf with length= 1 and width= 1 is characterised by a thickness= $\sqrt{2}$ and a abnormality= 0 (see Fig. 3.1, right).
The transformation from *length* and *width* to *thickness* and *abnormality* has an illustrative geometric interpretation. It is obtained by rotating the axes

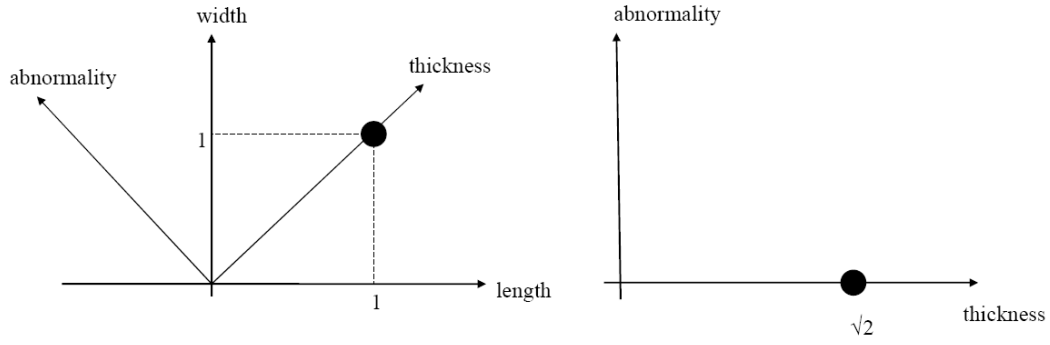


Figure 3.1: Left: The variables/dimensions *thickness* and *abnormality* functionally depend on *length* and *width*. Right: A leaf with length= 1 and width= 1 is characterised by a thickness= $\sqrt{2}$ and an abnormality= 0.

45° counter clockwise and describing the points of the scatter plot relative to these new axes.

In principle, the original axes could be rotated by any angle, see Fig. 3.2 (left side). Of course, the coordinates of a point in a new system (x', y') always differ from the coordinates (x, y) of the point in the original system, but both pairs of coordinates denote the same point. What is the interpretation of the new axes? Can we always make an interpretation such as „they denote *thickness* and *abnormality*"? It turns out that an interpretation is not always so easy, unless the new dimensions are defined in advance as in our example.[2] So why should we rotate the axes (i.e. change the variables) at all? And – if we insist on rotating – by what angle should we rotate? This will be discussed in the next section.

Exercise 3.1.1 *Draw a coordinate system with two perpendicular axes corresponding to the variables 'size'(in cm) and 'weight' (in kg) of a man. Let us define a new variable 'pear-shape' that shall depend on the other variables in a simple (functional) way: 'pear-shape'='weight'-'size'. What direction would the corresponding axis point at? What would be the pear-shape value of a man of 1.80 m and 65 kg? What can be said about people with pear-shape value 0? Would you prefer a positive or a negative pear-shape value (of yourself)?*

---

[2]The construction of the example is quite a bit artificial to make it more comprehensible.

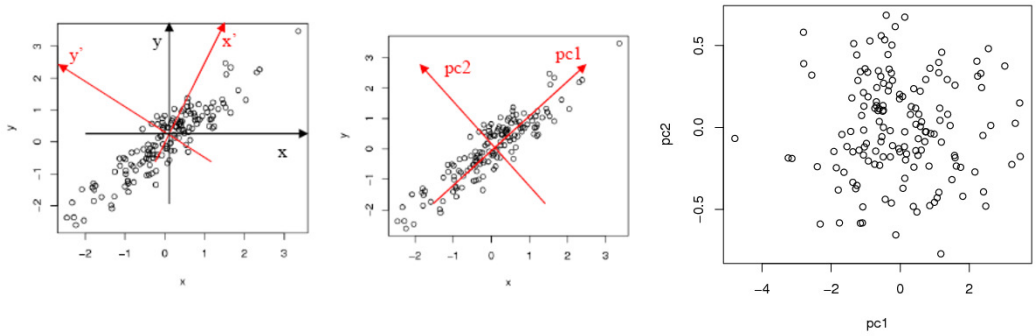


Figure 3.2: Left: The axes can be rotated, defining a new coordinate system. Middle: The principle components. Right: Relative to the coordinate system of the principal components, the data is decorrelated.

Finding principal components to decorrelate the data

There is one rotation of the coordinate system which is especially useful. It is the one shown in Fig. 3.2 in the middle. On the right side in Fig. 3.2, the data points are drawn relative to these new axes. You may notice that the distribution does no longer exhibit any correlation. This is one of the characteristics of the principal component system – the axes (or data) are decorrelated. Decorrelated data can be beneficial in many situations and decorrelation is often used as a pre-processing step for other methods. Some methods in data analytics assume that the input data is decorrelated and other methods show better performance when applied to decorrelated data. For instance, in multivariate regression, (multi-)collinearity can hinder the correct quantification of the contribution of single variables. So, one potential remedy is to apply PCA first.

In our artifical example, the variable *abnormality* is not correlated to the variable *thickness*. In other words, the fact that a leaf is abnormal does not give you any hint of whether the leaf is thick or not. Hence the data are decorrelated with respect to these variables.

How can we determine the principal components?

- Conceptually, we see that in order to decorrelate the axes, the first axis has to be chosen in such a way that it contains the greatest variance of the data. Or in other words, if we project onto this axis, then we obtain the greatest possible spread of the data. The second axis lies orthogonal to the first axis.

- **R**−application: R provides a routine to calculate the principal components. You should first generate a data distribution with Program c choosing $\mu_x = \mu_y = 0, var(x) = var(y) = 1, covar(x, y) = 0.8$. The principal components can be calculated with the following R programme:

```
#Program d
#run Program c first to create a 2d data distribution
# the data will be written into the variable 'leaves'

leaves_pca=prcomp(leaves)  #calculates rotation
   (matrix)
print(leaves_pca)
pc=predict(leaves_pca)   #calculates new coordinates
pc1=pc[,1] #write coordinates relative to first p.c.
   into pc1
pc2=pc[,2] #write coordinates relative to second
   p.c. into pc2
plot(pc1,pc2)
```

Listing 3.1: Principal components

As a result we get the scatter plot shown in Fig. 3.2 (right side) and a response from R similar to the following lines[3]:

```
Standard deviations:
[1] 1.3223918 0.4134313

Rotation:
         PC1        PC2
[1,] 0.6978152 -0.7162778
[2,] 0.7162778  0.6978152
```

[3]There will never be exactly the same results since the points are randomly generated

The block *Standard deviations* contains the information about the standard deviations of the data projected onto the principal components. The block *Rotation* contains the information about how the coordinate system has to be rotated in order to get the principal component system. The columns contain the coordinates of the vectors that indicate the direction of the principal components. It allows us to decompose the variables x and y into their principal components. For the example, we have

$$x = 0.6978152 \cdot pc1 - 0.7162778 \cdot pc2 \quad (3.2)$$
$$y = 0.7162778 \cdot pc1 + 0.6978152 \cdot pc2 \quad (3.3)$$

where the coordinates of the data points relative to the principal components are stored in the variables $pc1$ and $pc2$.
Conversely, one can calculate the new coordinates $(pc1, pc2)$ after the rotation for given (x, y) (for the vector geometry experts: the coordinates are actually obtained from the scalar product of the PC1 and PC2 vectors with (x, y)):

$$pc1 = 0.6978152 \cdot x + 0.7162778 \cdot y \quad (3.4)$$
$$pc2 = -0.7162778 \cdot x + 0.6978152 \cdot y \quad (3.5)$$

Attention: The equations only hold if $\bar{x} = \bar{y} = 0$. This is why a shift of the coordiate system has to be performed first if necessary (the command *prcomp* automaticall does it).

- The rotation matrix[4]

```
Rotation:
          PC1         PC2
[1,] 0.6978152 -0.7162778
[2,] 0.7162778  0.6978152
```

can also be interpreted geometrically by noticing that a general rotation matrix has the following structure:

$$\begin{bmatrix} \cos(\Theta) & -\sin(\Theta) \\ \sin(\Theta) & \cos(\Theta) \end{bmatrix}$$

[4]We added a, b to denote general entries.

From this we can derive that the axes are rotated counterclockwisely by the angle $\Theta = \arccos(a) = \arcsin(-b)$. In the actual case we have $\Theta = \arccos(0.6978152) = \arcsin(0.7162778) = 0.7984$. Note that the unit is rad from which we can easily derive the angle in degree: $0.7984/(2\pi) \cdot 360° = 45.7°$

- Be aware that sometimes an axis can be inverted in its direction, depending on the details of the data. For determining the directions of the principal components, this fact is irrevant. However, in the rotation matrix, the signs of PC1 or PC2 can switch.

- It is relatively easy to identify the rough directions of the principal components in a 2D scatterplot. Using R, the directions relative to the original data (shifted to $\mu_x = \mu_y = 0$) ca be drawn with the commands

```
plot(x-mean(x),y-mean(y))
abline(0,leaves_pca$rotation[2,1]/leaves_pca$rotation[1,1])
abline(0,leaves_pca$rotation[2,2]/leaves_pca$rotation[1,2])
```

Exercises

Exercise 3.1.2 *Program d: Show that the correlation vanishes if the data is considered in the principal component system (use R-function* cor(.,.)*).*

Exercise 3.1.3 *Consider the three points* $P1 = (1, 0), P2 = (0, 1), P3 = (-1, -1)$.
a) Draw them in a 2D scatterplot (using R)
b) Calculate the principal components with the command prcomp *and draw their directions in the plot using the commands that have been introduced in the last remark above.*
c) Now consider the rotation matrix with the entries

```
Rotation:
     PC1        PC2
[1,] a          b
[2,]-b          a
```

Determine a and b (use the command print *as in Program 5d to view the rotation matrix). Then manually calculate the new coordinates of the points* $P1, P2, P3$ *relative to the principal components. According to equations (3.4) and (3.5) the general rule is*

$$pc1 = a \cdot x + -b \cdot y \tag{3.6}$$
$$pc2 = b \cdot x + a \cdot y \tag{3.7}$$

d) Now compare your results with the results that you get with the command predict. *Do you get the same?*
e) Determine the rotation angle.

3.2 Dimensionality reduction using PCA

PCA is often used to reduce the dimension of high-dimensional data in order to illustrate the relevant information content in 2D or 3D. To achieve this goal, we determine the first two or three principal components, following the principal sketched in the last section for 2D. Given a $d-$dimensional centered[5] data distribution,

1. we first determine the axis with the greatest variance. This is the first principal component. The original coordinate system (with d coordinate axes) is rotated such that one new axis is the first principal component.

2. We then consider the projection onto the remaining $d-1$ dimensions and determine again the axis with the greatest variance (second principal component). The coordinates of the data points relative to the first and second principal component gives a 2D picture of the data.

3. For a 3D (4,5....D) picture, step 2) should be repeated accordingly.

In accordance with the last section, the principal components can be calculated using Program d. Using the following listing Program e, we can study the concept of a PCA for a dimensionality reduction from 3D to 2D. The question is how we should project the data in a best possible way, i.e. in such a way that we can see as much structure as possible (see also Fig. 3.3).

```
#Program e
library(MASS)
library(rgl)

#Define 3D covariance matrix:
varx=1.5;vary=1;varz=0.05
covar=array(c(varx,0,0,0,vary,0,0,0,varz),dim=c(3,3))
data1=mvrnorm(300,c(0,0,0),covar)
plot3d(data1,xlim=c(-3,3),ylim=c(-3,3),
zlim=c(-3,3),xlab="x",ylab="y",zlab="z")

#determine the directions of pc
data_pca=prcomp(data1)
```

[5] all the means are zero.

```
pc1d=data_pca$rotation[,1]
pc2d=data_pca$rotation[,2]
pc3d=data_pca$rotation[,3]

#draw the lines
segments3d(c(0,3*pc1d[1]),c(0,3*pc1d[2]),c(0,3*pc1d[3]))
segments3d(c(0,3*pc2d[1]),c(0,3*pc2d[2]),c(0,3*pc2d[3]))
segments3d(c(0,3*pc3d[1]),c(0,3*pc3d[2]),c(0,3*pc3d[3]))
```

Listing 3.2: Principal components 2

Conclusion: Since the first principal components correspond to new axes that maximise the variance of the data, these axes typically contain the most information about the data. Hence, we should project onto these axes if we want to reduce the number of dimensions.

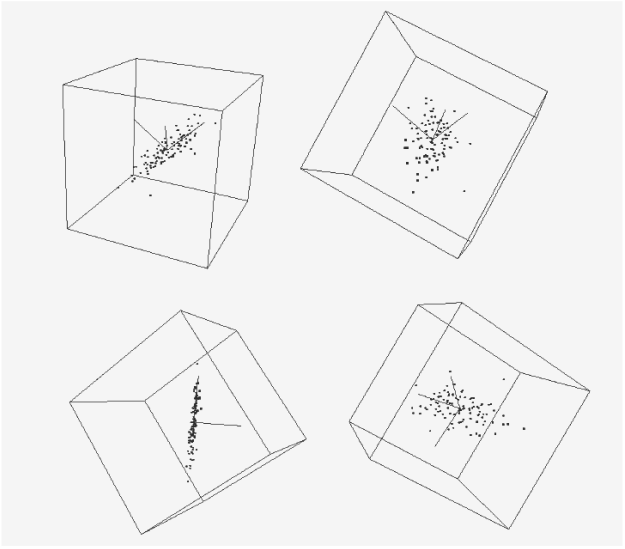


Figure 3.3: 3D PCA: different projections

$R-$application: Time series discrimination with PCA

The time course of fever can be different for different diseases. Let us consider again the time series in the fever data set. How many different diseases can be distinguished in the data set? Our goal is to find a simple method that allows to distinguish the diseases by just observing the fever curves.
Our strategy is to perform a PCA and plot the data in 2D (Program f). The underlying hope is that the first two principal components contain the most relevant aspects of the data so the group structures are revealed.

```
#Program f
#fever curves: run Program a first

#plotting curves as time series
library(graphics) #install first if necessary
feverdata=data.frame(t(fever))
ts.plot(feverdata) #without colors

#PCA and plotting
fever_pca=prcomp(fever)
pc1=predict(fever_pca)[,1]
pc2=predict(fever_pca)[,2]
plot(pc1,pc2)
```

Listing 3.3: PCA for the fever data set

Note: To perform a PCA, each column must correspond to one dimension.

A PCA can help identify the classes of different fever curves hidden in the confusing set of time series (Fig. 3.4, left). The projection onto the first two principal components clearly reveals the classes and makes it easy in most cases to assign a point (representing a time series) to one of the classes (Fig. 3.4, right).
The classes or clusters (including a cluster prototype) can also be detected automatically. We will discuss such techniques in the section about cluster analysis.

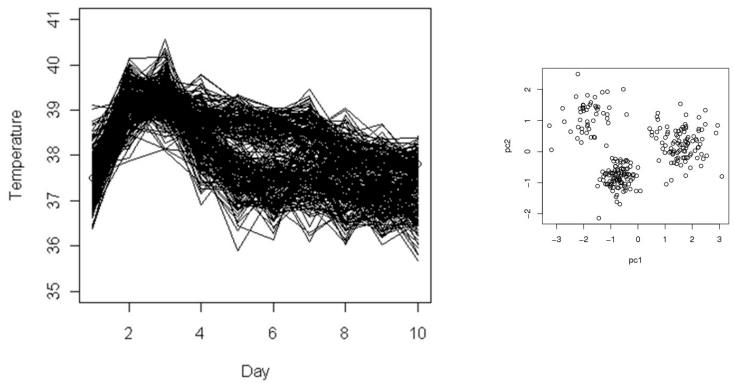


Figure 3.4: Examination of a data set that is similar (but not equal) to the fever data set we examine in the course. Left: the original time series data set is hard to analyse visually. Right: a PCA clearly reveals three classes in this case.

Important dimensions

PCA not only allows to reduce the dimension in an optimal way, but it also tells us something about the differing importance of the dimensions. With regard to the example of fever curves in the last section, this means that for some points in time (=dimensions) the difference between the curves of the three diseases may be more pronounced than for others. Hence these dimensions contain the important information about the differences. We will concentrate on the first principal component PC1. If the data is projected onto the direction of PC1, we preserve as much information as possible in a 1D projection because it is the direction of the greatest spread of the data. Hence PC1 indicates the 'direction of importance'. PC1 can be described with respect to the original dimensions, consider the first column in the rotation matrix. The PCA for the fever curves yields:

```
PC1
T1  -0.06034735
T2  -0.40920750
T3  -0.25671416
T4   0.08439626
T5   0.35497791
```

```
T6   0.36347060
T7   0.35252178
T8   0.52479506
T9   0.30993754
T10  0.04493993
```

For this vector, a ranking for the importance of the different dimensions (with respect to PC1) can be derived. The higher the absolute value of a coordinate the more important is the corresponding dimension because PC1 tends to point into this direction. Here we conclude that the 8th and 2nd dimension $T8$ and $T2$ carry the most information about PC1.
Variables that load (as we say) a lot on PC1, however, tend to be correlated. It thus can make sense to also consider PC2 or even more principal components. A tool for doing so are so called biplots.

Biplots

A biplot is a display that allows us to represent both the observations (or data instances = single fever curves in our example) and the variables in the same plot. A biplot uses points to represent the observations and it uses vectors to represent the variables. We may also focus only on the distribution of the variables by using white as the colour for the observations.
Both the direction and the length of the vectors can be interpreted. Vectors that point in the same direction correspond to variables that have a similar meaning or influence in the context of the data. The length of the vector is a measure of the correlation of the original variable and the corresponding direction in the system of the principal components. That is, for example, a long vector that points in the direction of a principal component corresponds to a variable that is a basic component of this principal component. Very short vectors, in turn, correspond to variables that are perpendicular to the principal components used. Hence they carry little information.
Observations, i.e. points, that lie close to the tip of a vector, are essentially determined by the corresponding variable.
In our example (Program f), we can run the command

```
-------------------------------------------------------------
biplot(fever_pca, expand=0.9,col=c(0,1))
-------------------------------------------------------------
```

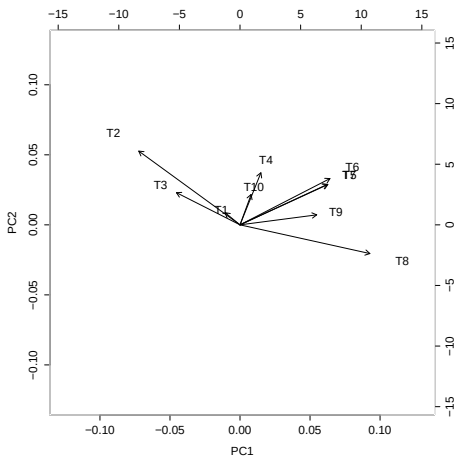


Figure 3.5: Representation of the old variables in the system of the first two principal components

with the following output
We see that the first principal component is essentially determined by variable 8 and 2, while the second principle component is less clear. It is essentially a mixture of the variables 2 and 4.

Remark: The components of the observations (with respect to the principal components) are called **scores** and the the components of the variables are called **loadings**.

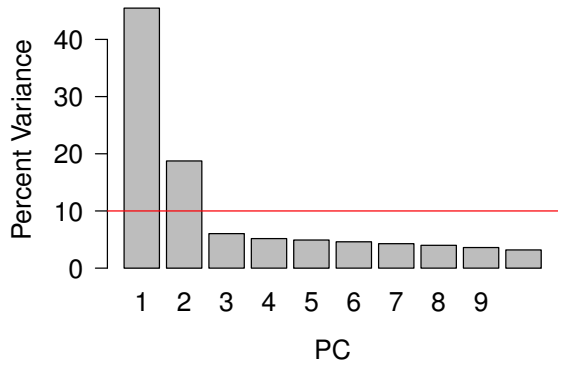


Figure 3.6: Percentage of variance explained by pc

Importance of principal components

We may also be interested in the question how important the principal components are. This question is of special interest if we don't fix the number of pc's to 2 or 3 (as done for visualisation purposes). For this you may take a look at the variance that is carried or explained by a principal component. The standard deviations are given from the prcomp-command. Using the following programme, you can calculate and plot the percentage of variance (= square of standard deviation) explained by each pc. In the plot we add a line that indicates the amount of variance each variable contributes if all contributed the same amount. The line can serve as a selection criterion for the important principal components. The result is shown in Fig. 3.6.

```
#Program g
# run Program f first
sd=fever_pca$sdev  #read out the standard deviation
var = sd^2 #determine the variance
var.percent = var/sum(var) * 100 #percentage of
   variance

#plot:
barplot(var.percent, xlab="PC", ylab="Percent
   Variance",
```

```
    names.arg=1:length(var.percent),
    las=1, ylim=c(0,max(var.percent)), col="gray")
    abline(h=1/ncol(fever)*100, col="red")
```

Listing 3.4: Percentage of variance explained by pc

Note: The number of principal components cannot exceed the number of samples in the dataset. To see this, consider two points in a 3D space. These two points lie on a straight 1D line. This defines the only meaningful direction, i.e. the first principal component. The second pc is any direction perpendicular to it. A third pc does not exist. The following code will examplify it:

```
x12=rbind(c(-1,-1,-1),c(1,1,1)) #two points in 3D
pc_x12=prcomp(x12)
pc_x12 #only two pcs obeserved
```

Limitations of PCA

Although PCA has become a very popular method that has been successfully applied in many fields, there are also some limitations one has to be aware of. PCA struggles or is unable to catch the relevant data structures if the data points form a nonlinear, i.e. a curved, manifold, see Fig.3.7. Also, if there are clusters in the data set and the distances between points within a cluster are similar or even larger than the distances between the cluster centres or if the data is very noisy, PCA may struggle to map the correct cluster structures.

To overcome these limitations, a couple of other approaches (such as kernel PCA, Independent Component Analysis ICA, Isomap, t-SNE,..) have been developed. These methods, however, are beyond the scope of this introductory course.

Another limitation concerns the structure of the input data. PCA relies on the availability of data vectors. For some applications, distance or (dis)similarity measurements rather than data vectors might be available. This aspect is addressed in the next section.

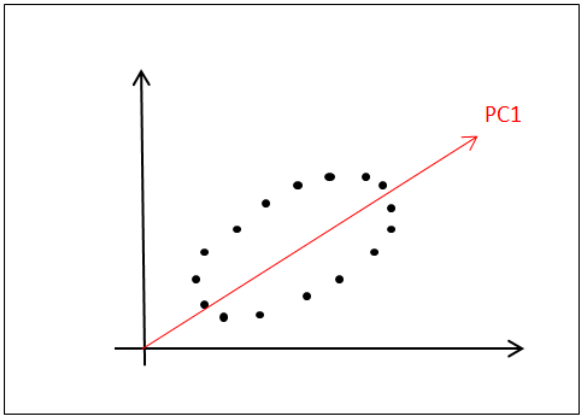


Figure 3.7: Even though the data lie on a (intrinsically) one-dimensional manifold, the correct information about neighbouring points is lost after projecting the data onto the first principal component.

3.2.1 Exercises

Exercise 3.2.1 *Take a look at the code (Program f) and repeat the PCA for the fever data set.*

Exercise 3.2.2 ***Iris data set**:*
Load the Iris data set and perform a PCA. Illustrate the data in 2D and 3D. Make a biplot and interpret it.

Exercise 3.2.3 ***Colon cancer data set**:*
a) Load the colon cancer data set and perform a PCA. Illustrate the data in 2D and 3D. Colour the points according to the criteria 'normal tissue' and 'tumor tissue'. The information is contained in the data set canceridentity.txt*. Use the following commands:*

```
#load label information:
cancerid=as.matrix(read.delim("canceridentity.txt",header=FALSE))
cancerid=apply(cancerid,1,sign)+3;
#cancerid indicates whether tissues are cancerous or not and defines colour
plot(pc1,pc2,col=cancerid)
```

Does a PCA facilitate the analysis of the data set?
b) What would be an appropriate number of pcs according to the criterion of variance explained? Adapt Program g to answer the question.
c) Given that PC1 clearly explains the most variance in the data: which are the four most important genes in the colon cancer data set? Which genes are not important? Follow the instruction in section 3.2.

Exercise 3.2.4 ***Dental data set****:*
Load the mammal teeth data set and perform a PCA (Attention: use the numeric dimensions only!). Illustrate the data in 2D. You can label the points using the command

```
text(pc1, pc2, dental[,1], pos=1,cex=0.7, col="red")
```

According to the PCA plot of dental formulas, which species are most similar to horses? Can you identify other groups?

Exercise 3.2.5 ***Chemical isis data set****:*
Load the chemical data set isis*.*
a) Perform a PCA and display the data in 2D. How many classes can you recognise?
b)In fact, the data set consists of 7 classes. You can load the class labels as follows:

```
isisall=read.csv(file="isisall.csv",header=TRUE)
isislist=isisall$class #only labels
```

Colour the plots in the PCA plot according to their actual classes. Which classes are easy to distinguish? Which classes are hard to distinguish? Why? Hint: To colour the points use

```
col=as.numeric(isislist)
```

Bonus material***: calculating principal components

Performing a PCA involves a singular value decomposition or an eigenvector decomposition of the covariance matrix.:[6] People with some experience in linear algebra may remember that a symmetric matrix such as the covariance matrix Σ, is diagonalised by a matrix of its orthogonal eigenvectors. The eigenvectors $\mathbf{e_i}$ and the corresponding eigenvalues λ_i are the solutions of the equation

$$\Sigma \mathbf{e_i} = \lambda_i \mathbf{e_i} \tag{3.8}$$

The eigenvalues reflect the variances of the data in the direction of the associated eigenvectors. They can be found by finding the solutions of the characteristic equation

$$det(\underbrace{\Sigma - \lambda I}_{M}) = 0 \tag{3.9}$$

where I is the identity matrix and $det(M)$ denotes the determinant of the matrix M. If the data vector has n components, the characteristic equation becomes a polynomial equation of order n. This is easy to solve only if n is small. For $n > 3$ the equation can only be solved using numeric methods. Afterwards, the linear system of equations (3.8) must be solved for each eigenvalue. By ordering the eigenvectors in the order of descending eigenvalues (largest first), one can create an ordered orthogonal basis with the first eigenvector having the direction of largest variance of the data. In this way, we can find the principal components.

As an **example**, we perform (by hand) a PCA for the 2D covariance matrix

$$\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \tag{3.10}$$

The characteristic equation is

$$det\left[\begin{pmatrix} 1-\lambda & 0.5 \\ 0.5 & 1-\lambda \end{pmatrix}\right] = (1-\lambda)^2 - 0.25 = 0 \tag{3.11}$$

This is a quadratic equation with the solutions $\lambda_1 = 1.5$ and $\lambda_2 = 0.5$. In order to find the first principal component, one has to solve the system

$$\begin{aligned} 1x + 0.5y &= 1.5x \\ 0.5x + 1y &= 1.5y \end{aligned} \tag{3.12}$$

[6] This section is intended for those with a particular interest in mathematics. It is not vital you understand it.

with the solution $\mathbf{e_i} = (x, y) = \frac{1}{\sqrt{2}}(1, 1)$ (the factor $\frac{1}{\sqrt{2}}$ is used for normalisation). Conclusion: The first principal component points into the direction of the bisecting line of the first quadrant.

3.3 Multidimensional Scaling

The basis of PCA is data given as data vectors. In such a data space, we can calculate the distance (or dissimilarity) between two data vectors $\mathbf{x} = (x_1, ..., x_d)$ and $\mathbf{y} = (y_1, ..., y_d)$ by means of the Euclidean distance[7]

$$|\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{l=1}^{d} (x_l - y_l)^2} \tag{3.13}$$

For some applications, the original data vectors might not be available and we are provided with a matrix of distance or dissimilarity values instead. In other cases, e.g. in chemometics or image analysis, the Euclidean distance might not be an appropriate measure to compare the data vectors. In these cases, we can use (classical) multidimensional scaling that uses a distance matrix rather than data vectors as input. The goal of classical multidimensional scaling MDS is to find low-dimensional (often 2-dimensional) reconstruction vectors r_i that minimise the following cost function

$$\boldsymbol{\Phi(\{r\}) = \sum_{(i,j)} (d_{ij} - |r_i - r_j|)^2} \tag{3.14}$$

where $d_{ij} = d_{ji}$ is the distance between the original data points and $|r_i - r_j|$ denotes the Euclidian distance between the ith and the jth point in the reconstruction space. That is, the goal is to find points in a reconstructed scatterplot such that the distances between the points equals the given distances/dissimilarities as much as possible.
As a simple intuitive example consider the distance matrix of some Swiss cities and reconstruct the positions (Program g).

```
   #Program h
library(vegan)

#we first construct the distance matrix:
cities=c("Zurich","Geneva","Basel","Bern","Lugano",
"Lausanne","St.Gallen","St. Moritz","Schaffhausen")
N = length(cities), dstMat = matrix(numeric(N^2),
   nrow=N)
```

[7]The formula reflects just Pythagoras' Theorem in 2D

```
##             GE   BS  BE   LUG   LA    SG  SM   SH
cityDst <- c(225,  87, 96,  156,  174, 64,  140, 37, ##
   ZH
187, 130, 218,  51,  282, 286, 251,  ## GE
69,  202,  137, 136, 208, 80,  ## BS
156,  78,  155, 190, 123, ## BE
188, 162, 87,  190,  ## LUG
232, 246, 201,  ## LA
110, 63, ## SG
162) ## SM
dstMat[upper.tri(dstMat)] = rev(cityDst)
dstMat = t(dstMat[ , N:1])[ , N:1]
dstMat[lower.tri(dstMat)] = t(dstMat)[lower.tri(dstMat)]
dimnames(dstMat) = list(city=cities, city=cities)

#now, we perform MDS
mds = cmdscale(dstMat, k=2)

#plotting:
xLims = range(mds[ , 1]) + c(0, 250)
plot(mds, xlim=xLims, ylab="North-South",
   xlab="West-East", pch=16,
main="City locations according to MDS")
text(mds[ , 1]+5, mds[ , 2], adj=0, labels=cities)
```

Listing 3.5: Multdimensional Scaling

The results are depicted in Fig.3.8. Note that for Euclidean distances, the results of MDS are equivalent to PCA (up to rotations/inversion of the data).

Exercise 3.3.1 *Consider the isis chemical dataset and perform a MDS using the different distance measures 'Euclidean', 'Jaccard', 'Manhattan'. How does it compare to a normal PCA?*
Tip: Use the function vegdist *from the library* vegan *to calculate the distances and consider for plotting exercise 9.2.5.*

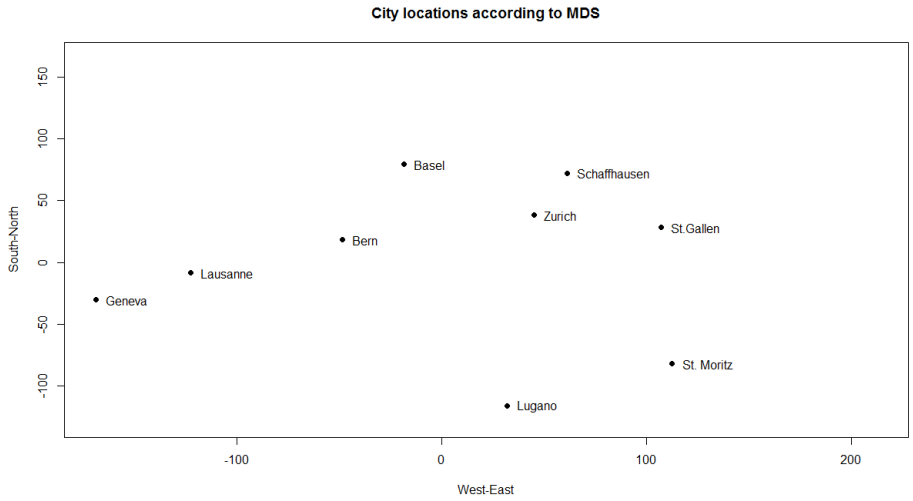


Figure 3.8: Multidimensional scaling: Reconstruction of the positions of some Swiss cities.

Chapter 4

Clustering

4.1 What is clustering?

Birds of a feather flock together.

Proverb

Clustering deals with finding a structure in a collection of data. In other words, the goal of clustering is to find groups of data items that are similar in some way. Consider the fever curve problem in Fig. 3.4. Here, the problem is to group all those time series or points together that indicate the same disease. There is a variety of different clustering methods, but we will not provide a systematic overview. We will concentrate on two widely-used methods that are also implemented in R: K-means clustering and hierarchical clustering.
In many cases, the problem of clustering is how to detect clouds of points in a data set. These clouds or clusters typically represent natural classes, such as in Fig. 3.4. In 2D the detection of clusters can be done visually. However, as real data is often high-dimensional, we need methods that are able to detect point clouds in high-dimensional spaces for which we cannot generate a visualisation (a PCA projection onto 2D is not always optimal). But even in 2D, it is often advantageous to have an automated allocation of points to clusters.

Examples of clustering applications:

1. **Marketing:** Help marketers discover distinct groups in their customer bases to develop marketing programs.

2. **Land use:** Identification of areas of similar land use in an an earth observation database.

3. **Earthquake studies:** Observed earthquake epicenters should be clustered along continent faults.

4. **Combinatorial chemistry:** Clustering is used to identify groups in chemical databases to develop new drugs.

5. **Microarray analysis:** Genes can be clustered into functional groups.

6. **Neuroscience:** Neurons can be clustered according to their firing behaviour for further analysis.

7. **Genetics:** Hierarchical clustering approaches are used to determine phytogenic trees (showing the evolutionary relationships among species).

4.1.1 Taxonomy of clustering methods/algorithms

There is a plethora of different clustering algorithms and each algorithm comes along with certain advantages and disadvantages. Generally, there is no best algorithm as the best choice depends on the data and the problem given.

Jain et al provided a taxonomy of clustering algorithms in a hierarchical structure (*A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. ACM Comput. Surv., 31(3):264-323, 1999.*) as depicted in Fig.4.1.

Generally, we may classify the different algorithms/methods according to the following criteria:

- Most of the clustering algorithms are divided into hierarchical and partitioning clustering algorithms.

- Agglomerative vs. divisive: The agglomerative approach is a bottom-up construction and the divisive is a top-down approach.

- Monothetic vs. polythetic: It refers to the different use of features in the process, either sequential or simultaneous.

- Hard vs. fuzzy: This aspect refers to the membership of the data. A hard clustering assigns one data point to only one cluster while a fuzzy clustering assigns one to multiple clusters.

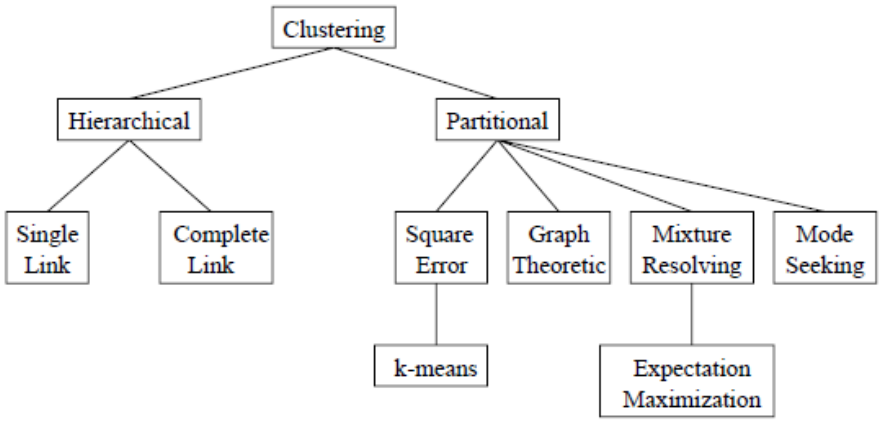


Figure 4.1: Taxonomy of clustering algorithms

- Deterministic vs. stochastic: It is related to the optimal techniques the clustering algorithm is using. Most of partitioning clustering algorithms use either a deterministic objective function or a random search technique for optimization.
- Incremental vs. non-incremental: An algorithm where the size of data can be increased is considered as incremental otherwise non-incremental.

In the following, we focus on two very popular classes of algorithms, K-means and linkage methods.

4.2 K-means

K-means clustering is well suited if the number of clusters is known and the clusters tend to be spherical and are clearly separated. The goal of K-means clustering is to find cluster centres $\mathbf{c_j} = (c_{j1}, ..., c_{jd})$ (where j denotes the jth centre and d is the dimension) in such a way that each point $\mathbf{x_i}$ is assigned to the closest centre and the sum of the (squares of the) distances from the points to the allocated centres is as small as possible. Hence K-means is an optimisation problem; the goal is to determine the coordinates of the centres such that the following function is minimised:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \epsilon_{ij} |\mathbf{c_j} - \mathbf{x_i}|^2 \tag{4.1}$$

where k denotes the number of clusters (number of cluster centres), n denotes the number of data points and $|\mathbf{c_j} - \mathbf{x_i}|$ denotes the distance between a point $\mathbf{x_i}$ and a cluster centre $\mathbf{c_j}$. $\epsilon_{ij} = 1$ if $\mathbf{c_j}$ is the closest centre to $\mathbf{x_i}$, otherwise $\epsilon_{ij} = 0$.
The problem is usually solved using the following algorithm (K-means algorithm):

1. Set the k centre vectors to random positions (= coordinates).
2. Determine for each data vector $\mathbf{x_i}$ the closest centre $\mathbf{c_j}$ and assign the label j to $\mathbf{x_i}$. All points with the same label form a cluster.
3. Calculate the mean (= centre) of each cluster and identify it with $\mathbf{c_j}$.
4. Go back to 2 until the assignments no longer change.

The algorithm can be shown to reduce the costs J, but sometimes it can get trapped in a local minimum and yields poorer results (see Fig. 4.2). This is why the algorithm should be run several times and the best result is picked in the end.

The R function $kmeans(data, k)$ starts the algorithm. *data* is the data array that contains the coordinates of all data points, k is the number of clusters.

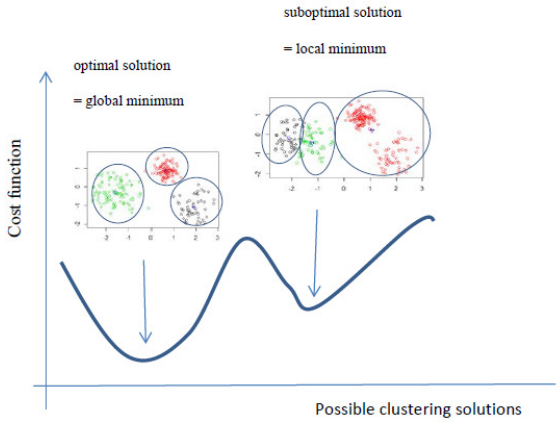


Figure 4.2: The k-means algorithm can get trapped in local minima.

R−application: Program i performs K-means clustering for the fever curve data set based on the first two principal components and compares it to the actually expected results.

```
#Program i

#using the results from Program f
#run 5f first if necessary
pcmatrix=cbind(pc1,pc2) #data in one matrix
plot(pcmatrix) #you can see 4 clusters

# clustering
k=4 #number of clusters
cl=kmeans(pcmatrix,k)
plot(pcmatrix,col=cl$cluster)
points(cl$centers,pch=25,col="blue")
cl

# confusion matrix
feverlabels=read.csv(file="feverlabels.csv")
actual_class=as.matrix(feverlabels) #actual class labels
predicted_cluster = cl$cluster   #predicted cluster
   labels
```

```
confusion_matrix = table(actual_class,
   predicted_cluster)
confusion_matrix
```

Listing 4.1: K-means clustering

Result:

```
K-means clustering with 4 clusters of sizes 50, 94, 98, 98

Cluster means:
      pc1        pc2
1  2.0936927 -0.5355651
2  0.5448976 -0.4494856
3 -1.8800702 -0.2610865
4  0.2892027  0.9654733

Clustering vector:
[1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4
[74] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[147] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[220] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[293] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1]  9.964211 16.369614 48.635708 16.978476
(between_SS / total_SS =  88.9 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
 "betweenss"    "size"         "iter"         "ifault"
```

`confusion_matrix`

actualclass/predictedclass	1	2	3	4
1	0	2	0	98
2	0	2	98	0
3	50	0	0	0
4	0	90	0	0

- The variable *cluster means* provides the coordinates of the cluster centres.
- The variable *clustering vector* indicates the cluster to which each point is allocated. E.g. points 1,2,... are allocated to cluster 4, point 147 is allocated to cluster 3, etc.
- Since the cluster means are initialised randomly, the final results can differ, i.e. the clusters found and the sequence of cluster means may be different for each trial. To find an optimal solution you should run the algorithm several times (see section about quality measures).
- The confusion matrix compares the results of the clustering to a benchmark classification in tabular form. This layout makes it easy to see how many of the items have been misclassified. Ideally, there is just one nonzero entry per row and column. Note: as the numbering of the clusters is arbitrary, the number of the actual class and the predicted cluster do not have to coincide. In the example, 98 data points of class 1 were assigned to cluster 4 and 2 data points were (mis-)classified to cluster 2, etc.

Remark on PCA and clustering:
In the example above, we first applied PCA (Program f) to reduce the number of dimensions to two, so that the cluster structure of the data could be recognised visually. Then we applied K-means clustering (Program i). This procedure is often, but not always, advantageous. Sometimes, it is better to keep more dimensions (the 'optimal' number can be determined using Program g) and sometimes, especially if there are highly nonlinear structures, it might be even better not to use PCA at all (see Fig.3.7).

4.2.1 Quality measures

If the actual results are known we may use a confusion matrix to assess the quality of the clustering achieved. However, usually the correct classification is unknown and we have to employ cluster validation measures that give a hint on the quality, i.e. 'clusterness' of the found classes.

- In the case of clear, i.e. compact and spatially separated clusters, the distances within the clusters are much smaller than the distances between the clusters. For k-means clustering this is measured by the ratio of $between_{SS}$ (sum of squares of distances between means of two different clusters) and $total_{SS}$ (total sum of squares of distances between all points and the overall mean).[1] In general we have

$$between_{SS} = total_{SS} - \sum_i within_{SS}(Cluster\ i)$$

  where $within_{SS}(Cluster\ i)$ is the sum of squares of distances between points and the mean of cluster i. A value of the ratio $between_{SS}/total_{SS}$ close to 100% means that the clusters tend to be compact. However, the measure is mainly suitable if you keep k fixed as it naturally grows for increasing k. For fixed k, you may run kmeans several time and keep the best result as follows

```
clratio=0 #initiate ratio
for (i in 1:15)   #run kmeans 15 times
{
cl=kmeans(pcmatrix,k)
if (cl$betweenss/cl$totss > clratio){ #check whether result is better
clratio=cl$betweenss/cl$totss;
clk=cl; #write current best result into clk
} #end if
}# end i-loop
plot(pc,col=clk$cluster)
```

[1] In fact, this is akin to the partionings of the sum of squares that you probably know from ANOVA and regression.

- The **Connectivity** measures to what extent observations are placed in the same cluster as their nearest neighbours in the data space. The connectivity has a value between 0 and 1, and should be minimized.
- The **Dunn Index** is the ratio of the smallest distance between observations not in the same cluster to the largest intra-cluster distance. The Dunn Index has a value between 0 and 1, and should be maximized.

Connectivity and **Dunn Index** can be calculated using the package *clValid*.

```
-----------------------------------------------------------------
 #Example: calculating Dunn index, install package clValid first
 #run Program f and i first
 library(clValid)
 d=dist(pcmatrix) #calculating all the distances
 cluster=cl$cluster
 dunn(d,cluster)
-----------------------------------------------------------------
```

Determine the number of clusters

If the number of clusters is not known or unclear in advance, which is often the case, then a visual inspection of the data (e.g. by means of PCA) might give some insight. Alternatively, many different measures have been proposed in the literature in order to automatically evaluate the number of clusters. If the data contains well separated spherical clusters, the Dunn index might be a good choice as illustrated in the following example:

```
#Program j: calculating Dunn index for different k

#extract first two classes in fever data set:
fever2=fever[1:200,]
fever2_pca=prcomp(fever2)
pcc= predict(fever2_pca) # calculates new coordinates
pc1=pcc[,1] # write coordinates relative to first p.c.
pc2=pcc[,2] # write coordinates relative to second p.c.
plot (pc1,pc2)

library(clValid)
pcmatrix=cbind(pc1,pc2) #data in one matrix
```

```
d=dist(pcmatrix) #calculating all the distances
dindex=0;
for (k in 2:30){
 cl=kmeans(pcmatrix,k)
 dindex=c(dindex,dunn(d,cl$cluster))}
plot(dindex)
lines(dindex)
```

Listing 4.2: Dunn index

However, for many applications the Dunn index is not very reliable (see exercises below). To gain a good view, it is usually a good strategy to use different quality measures as well as different clustering algorithms to come to a conclusion abouth the cluster structure of a data set.

4.2.2 Exercises

Exercise 4.2.1 *Perform a K-means clustering analysis directly on the original 10-dimensional fever curve data set (do not apply PCA first). How many instances are misclassified? Run the algorithm several times and compare the results in terms of* $between_{SS}/total_{SS}$.

Exercise 4.2.2 *Using the results of the first exercise, use Program j to determine the optimal number of clusters that should be chosen. What is your observation?*

Exercise 4.2.3 *Load the isis chemical data set (Program a). Determine the number of different classes contained in the data set (see Exercise 3.2.5). Perform an appropriate K-means clustering analysis on the isis chemical data set. Compare the resulting assignments with the correct assignments (contained in the list). This could be done using a confusion matrix.*

Exercise 4.2.4 *Use the k-means algorithm to determine the clusters in the Iris flower data set. Use different numbers of clusters* $k = 2, 3, 4, 5, 6, ...$ *and compare the results visually and by means of the ration* $between_{SS}/total_{SS}$ *and the Dunn index.*

4.2.3 Choosing a distance measure

K-means exploits the distance between points and corresponding cluster centres. If the components of the data vectors are all in the same physical units, then the normal Euclidean distance is usually a good choice for this distance. In many situations, the data vectors combine different types of data with different scales. In the case of different scales it is often advantageous to rescale the data (e.g. all values lie between 0 and 1). In other situations, it is advantageous to use another type of distance measure. Unfortunately, the R-function *kmeans*() only supports the Euclidean distance directly. Choosing the best distance is not always straightforward and often requires some trials and experience. We will not dwell on this issue, but you should consider this if you observe poor performances with the Euclidean distance.
Here is a list of some common measures to determine the distance between two $d-$dimensional vectors $\mathbf{x} = (x_1, ..., x_d)$ and $\mathbf{y} = (y_1, ..., y_d)$:

Euclidean distance	$\lvert\mathbf{x}-\mathbf{y}\rvert = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$	
Manhattan distance	$\lvert\mathbf{x}-\mathbf{y}\rvert = \sum_{i=1}^{d}\lvert x_i - y_i\rvert$	
Tanimoto distance	$\lvert\mathbf{x}-\mathbf{y}\rvert = \frac{\text{number of } j \text{ with } x_j \neq y_j}{\text{number of } j \text{ with } x_j=1 \text{ or } y_j=1}$	for binary vectors
Mahalanobis distance	$\lvert\mathbf{x}-\mathbf{y}\rvert = \sqrt{(\mathbf{x}-\mathbf{y})^T S^{-1}(\mathbf{x}-\mathbf{y})}$	S is the covariance matrix of the data.

List of some common distance measures

Exercise 4.2.5 *Determine the Euclidean distance, the Manhattan distance and the Tanimoto distance between the vectors* $(1, 0, 0, 1)$ *and* $(0, 1, 1, 0)$*. See if you can find a R function which can be used to calculate some of these distances.*

Exercise 4.2.6 *What distance measure could be suitable for comparing chemical fingerprints (as contained in the isis data set)?*

4.3 Hierarchical clustering

In hierarchical clustering the data is not partitioned into clusters in a single step. Instead, a series of partitions takes place, which may range from a single cluster containing all objects to n clusters each containing a single object. So hierarchical clustering provides a cluster hierarchy. There are a couple of related hierarchical algorithms implemented in R. They are all agglomerative methods in the sense that they start with singletons and then build up a cluster hierarchy by merging clusters to bigger clusters. In every step, the two clusters that are closest to each other, i.e. that have the smallest inter-cluster distance, are merged.
The simplest method is called the **single linkage** method, where the distance between two clusters is defined as the shortest link between the clusters (see Fig. 4.3). The other methods (complete linkage, average linkage and Ward's method) rely on different definitions of the distance between clusters A and B.[2] The cluster distances for the different methods are defined as follows:

Single linkage: shortest distance between two members	$d(A,B) = \min_{\mathbf{x}\in A,\mathbf{y}\in B} \lvert\mathbf{x}-\mathbf{y}\rvert$
Complete linkage: longest distance between two members	$d(A,B) = \max_{\mathbf{x}\in A,\mathbf{y}\in B} \lvert\mathbf{x}-\mathbf{y}\rvert$
Ward's method: increase of sum of squares	$d(A,B) = \frac{n_A n_B}{n_A+n_B}\lvert c_A - c_B\rvert^2$ $c_{A,B}$: centres of clusters, $n_{A,B}$: size of clusters similar versions exist as well
Average linkage: average distance between two members	$d(A,B) = \frac{1}{n}\sum_{\mathbf{x}\in A,\mathbf{y}\in B} \lvert\mathbf{x}-\mathbf{y}\rvert$ n: number of distances

[2]Do not confuse the distance between clusters with the distance between points, described in the last section. The distance between clusters can be based on any distance between points.

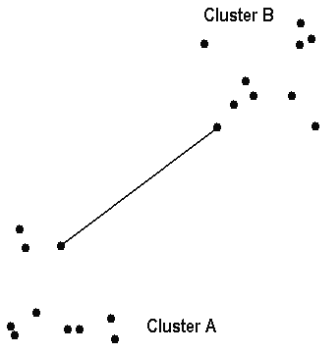


Figure 4.3: Single linkage method: The distance between two clusters is defined as the shortest link between the clusters.

List of some common distance measures for hierarchical clustering

In practice, Ward's method is often considered the most reliable hierarchical clustering method (yielding the most reliable results).
The results of hierarchical clustering are displayed in a **dendrogram**, see Fig.4.4.

R-code: The following programme performs hierarchical clustering for a generated toy data set.

```
#Program k

  library(MASS)
 #generating a toy data set with two Gaussian clusters
   mu1=c(0,0); #center cluster 1
   mu2=c(4,4); #center cluster 2
   sd=1; #standard deviation of cluster distributions
   cl1x=rnorm(20,mu1[1],sd);cl1y=rnorm(20,mu1[2],sd);
   cl1=cbind(cl1x,cl1y)
   #cluster 1: cl1x are x-coordinates, cl2y are
      y-coordinates
   #randomly drawn from a normal distribution with 20
      points
   cl2x=rnorm(20,mu2[1],sd);cl2y=rnorm(20,mu2[2],sd);
```

```
   cl2=cbind(cl2x,cl2y)
   cl=rbind(cl1,cl2)

   plot(cl)

 # hierarchical clustering with the function hclust
  d=dist(cl) #creates a object with the interpoint
     distances
  hc=hclust(d, method="single") #the result of hclust
     is written into hc
  plot(hc, hang=-1)

 democut=cutree(hc,k=5)  #try and find out
 democut
 democut2=cutree(hc,h=0.65)
 democut2

# new plot:
  plot(hc, hang=-1)
  # draw dendogram with red borders around the 5
     clusters
  rect.hclust(hc, k=5, border="red")
```

Listing 4.3: Hierarchical clustering

The $y-$axis of the dendrogram is the variable *height* that can be interpreted as the distance (or similarity) between clusters according to the method chosen. The cluster analysis can be "sliced" horizontally to produce unique clusters either by specifying a height (distance) or the number of clusters desired. For example, to get 5 clusters, use

```
democut=cutree(hc,k=5)   #hc is the result from hclust
```

To cut the tree at a specific height, specify the explicit "h" argument second with the specified similarity (or "height").

```
democut2=cutree(hc,h=0.65)
```

Since the height (y-axis) represents how close together observations were when they were merged into clusters, clusters whose branches are very close together (in terms of the heights at which they were merged) are not very

well separated. If there is a big difference along the y-axis between the last merged cluster and the currently merged one, then that indicates that the clusters are well separated.

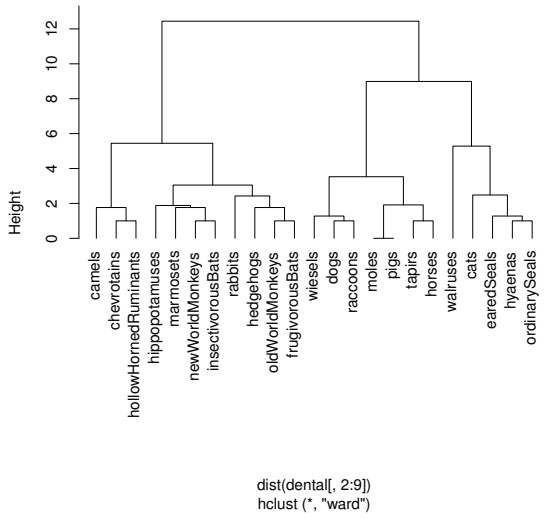


Figure 4.4: The dendrogram you obtain from a hierarchical clustering of the mammal teeth data set.

Exercises

Exercise 4.3.1 *a) Interpret the results from Program k.*
b) Use other linkage methods (e.g. method="ward.D") and compare the results.

Exercise 4.3.2 *Load the mammal teeth data set. The matrix is difficult to make sense of, but this changes when it is cluster-analysed. Perform a hierarchical cluster analysis. How do you interpret the results? Use the following command for labelling:*

```
plclust(hc,labels=dental[,1],hang=-1) #hc is the result from hclust
```

Exercise 4.3.3 *Load the isis chemical data set and perform a hierarchical clustering. Determine the cluster assignments for 7 clusters and compare the results with the correct assignments. A confusion matrix could be used for this.*

4.4 Fuzzy c-means clustering

For many data sets the cluster structures are not very clear. In these situations it can make sense to look for a fuzzy cluster assignment rather than for a hard cluster assignment. According to the fuzzy clustering idea, data points can belong to different clusters with a varying degree of cluster membership. For instance, points on the edge of a cluster may belong to the cluster to a lesser degree than points in the center of cluster. Fuzzy $c-$means clustering is an algorithm that is similar to the $k-$means which allows for calculating the degree of cluster memberships for each point. That is, the algorithm attempts to partition a finite collection of points into a collection of c fuzzy clusters such that the weighted sum of the distances from the points to the allocated centres is as small as possible. The weights in the sum reflect the degrees of cluster membership of the points.
The following program illustrates $c-$ means clustering for the example of the Iris flower data set.

```
#Program 1
# install.packages("e1071") #install if necessary
library(e1071)
numberofclusters=3;
cl_result = cmeans(iris[,-5], numberofclusters,
   iter.max=100, m=2, method="cmeans")
# iter.max is the max. number of iterations of the
   algorithm
#m=2 defines the degree of fuzziness (m must be larger
   than 1)

plot(iris[,1], iris[,2], col=cl_result$cluster)
points(cl_result$centers[,c(1,2)], col=1:3, pch=8,
   cex=2)
cl_result$membership[,]

library(rgl)
plot3d(iris[,1], iris[,2],iris[,3],
   col=cl_result$cluster)
```

Listing 4.4: Fuzzy $c-$means

- The degree of cluster membership can be interpreted as the probability

that a certain point belongs to a cluster. Hence the values for one point have to add up to 1. For example for the first point we have:

```
> cl_result$membership[1,]
          1           2           3
0.001072050 0.996623577 0.002304373
```

Here, the sum of the values has to be 1:

```
> sum(cl_result$membership[1,])
[1] 1
```

Exercises

Exercise 4.4.1 *a) Interpret the results from Program l. What happens when you make m very large or almost 1?*
b) Run the program for the first 2 components of the dataset Iris only (iris[,1:2]). How is the flower number 135 classified? What do you think about the classification?

Exercise 4.4.2 *Load the isis chemical data set. Determine the number of different classes contained in the data set from the list (isislist). Perform an appropriate fuzzy c-means clustering analysis on the isis chemical data set. Compare the resulting assignments with the correct assignments (contained in the list) and with the results you get from K-means.*
