

Pattern Recognition for the Natural Sciences Studyguide

Ron Wehrens
Institute of Molecules and Materials
Analytical Chemistry
Radboud University Nijmegen

edited by: Geert Postma, Lionel Blanchet

Version:

January 3, 2012

Contents

1	Introduction	5
1.1	Targets	5
1.2	Requirements	5
1.3	Lecturers	5
1.4	Format of the course	6
1.5	Examination	7
1.5.1	Written report	7
1.5.2	The exam	8
1.6	Computer exercises	9
2	Visualization techniques	11
2.0	Visualization of data and understanding models	13
2.1	Principal Component Analysis (PCA)	15
2.2	Independent Component Analysis (ICA)	21
2.3	Multi-dimensional Scaling (MDS)	23
2.4	Self-organising maps (SOMs)	26
2.5	Multi-variate Curve Resolution (MCR)	31
2.6	ANOVA-simultaneous component analysis (ASCA)	33
3	Unsupervised methods: clustering	35
3.1	Hierarchical clustering	36
3.2	Crisp Partitioning methods	39
3.3	Probabilistic, or fuzzy, clustering	41
4	Supervised methods: classification	45
4.1	Discriminant analysis	46
4.2	Nonparametric classification	51
4.4	Support Vector Machines	55
A	R packages used in PRiNS	59

Chapter 1

Introduction

The 6 EC MSc-level course *Pattern Recognition in the Natural Sciences*, in short PRiNS, aims to provide a comprehensive overview of the pattern recognition techniques as they are often applied in the natural sciences. The course web site,

<http://www.webchem.science.ru.nl/PRiNS>,

provides up-to-date information for the students, including information on installing R and the PRiNS package, as well as this document.

1.1 Targets

At the end of the course, the student should know the principles behind the most important pattern recognition methods, when and how to apply them, and will have practical experience in applying these methods to several data sets. The students should be able to reason which of the methods treated in the course are most useful in a given situation.

1.2 Requirements

To be able to do this course, it is necessary to have passed the Statistics course in the second year of Chemistry, Molecular Life Sciences and Natural Sciences, or an equivalent course outside the Chemistry curriculum. This means that a basic knowledge of subjects like confidence intervals, hypothesis tests, basic linear regression and experimental design should be present.

1.3 Lecturers

Lecturers are prof.dr. L. Buydens, dr. Jeroen Jansen, dr. G. Postma, and dr. L. Blanchet from the Analytical Chemistry Department of the Radboud University. The contact person for the course is G. Postma (g.j.postma@science.ru.nl, HG01.712, tel. 52345).

1.4 Format of the course

The course will run over a period of about 10 weeks. In principle each week there will be a lecture. During the week following the lecture the student can perform computer practicals on the lecture subject him-/herself using the exercises as described in this study guide. Possibilities will be provided for asking (conceptual) questions. The students are expected to do the computer exercises of a certain topic before the next meeting. The course will be concluded with a written examination and can include an assignment as well.

The subjects of the course are summarized in Table 1.1. One of the books that is used as background material is “*The Elements of Statistical Learning*” by Hastie, Tibshirani and Friedman. A pdf version of this book is available for free at the website of one of the authors (<http://www-stat.stanford.edu/~tibs/ElemStatLearn/>). The corresponding key paragraphs in the book are gathered in the column labelled *TESL*; in many other places in the book you will find related material. Especially the introductory chapters and introductory paragraphs of individual chapters provide interesting reading. Details on other background material will be provided during the course.

Table 1.1: Pattern Recognition in the Natural Sciences, edition 2010–2011: topics.

Topic	Subject	Lecturer	Hours	<i>TESL</i>
Visualization	Visualization+Models, PCA	Jansen	2	14.5
	ASCA, MCR	Jansen, Blanchet	2	–
Clustering	Clustering techniques	Postma	2	6.8, 7.7, 8.5, 14.3
Classification	Discriminant Analysis	Postma	2	4.3, 12.4 – 12.7 (13.3)
	SVMs	Buydens	2	12.1 – 12.3
	Bayes Classifiers	Lucas / Velikova / ..	2	–

Background material on ASCA is available in the paper “ANOVA-simultaneous component analysis (ASCA): a new tool for analyzing designed metabolomics data”, Age K. Smilde et al., *Bioinformatics*, 21 (13), 2005, 3043-3048 (http://sfx.ubn.kun.nl/sfx_local?&url_ver=Z39.88-2004&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&rft_val_fmt=info:ofi/fmt:kev:mtx:journal&rft.atitle=ANOVA-simultaneous%20component%20analysis%20%28ASCA%29%3A%20a%20new%20tool%20for%20analyzing%20designed%20metabolomics%20data&rft.aufirst=AK&rft.aulast=Smilde&rft.date=2005&rft.epage=3048&rft.genre=article&rft.issn=1367-4803&rft.issue=13&rft.jtitle=BIOINFORMATICS&rft.pages=3043-3048&rft.spage=3043&rft.stitle=BIOINFORMATICS&rft.volume=21&rfr_id=info:sid/www.isinet.com:WoK:WOS&rft.au=Jansen%2C%20JJ&rft.au=Hoefsloot%2C%20HCJ&rft.au=Lamers%2C%20RJAN&rft.au=van%20der%20Greef%2C%20J&rft_id=info:doi/10%2E1093%2Fbioinformatics%2Fbti476)

Background material on MCR is available in the paper “Elucidating the composition profiles of alcoholic fermentations by use of ALS methodology”, M. Blanco, A.C. Peinado, J. Mas, *Analytica Chimica Acta*, 544, (1-2), 2005, 199-205 (http://sfx.ubn.kun.nl/sfx_local?

```
sid=Elsevier:Scopus&iissn=00032670&isbn=&volume=544&issue=1-2+SPEC.+ISS.&spage=199&epage=205&pages=199-205&artnum=&date=2005&title=Analytica+Chimica+Acta&atitle=Elucidating+the+composition+profiles+of+alcoholic+fermentations+by+use+of+ALS+methodology&aufirst=M.&aunit=M.&aunit1=M&aualast=Blanco&_service_type=getFullTxt)
```

1.5 Examination

The final mark for this course consists of two parts:

1. A written report about a pattern recognition problem of the student's choice in the natural sciences;
2. An exam.

Both count for 50% of the final mark.

1.5.1 Written report

In the written report, the student should show that (s)he is able to find and define a problem in the natural sciences that deals with pattern recognition. The report is split into two tasks:

1. A small report (max 2 A4, excluding figures) on the fundamentals of the problem. For this, the student should visit different departments within the IMM and/or NCMLS to look for problems. Problems found in literature or in every-day life are also possible, as long as they have a connection to the natural sciences. The problem should be well defined in the report and the student should describe which data should be measured in order to find an answer to the defined problem. This deadline for handing in this report, is **Friday, December 16, 2011**.
2. A report of max 5 A4 (again excluding figures) in which the student discusses the techniques described in the course which will help in finding the answer to the problem from the first report. Explain how the techniques should be applied and how the outcome should be interpreted. For techniques that are not applicable to the problem, discuss why they are not applicable. Also pick some technique(s) as preferred to solving the problem and explain why these techniques are chosen as preferred. **The real data analysis is not required!** Deadline for handing in this report: **Wednesday, February 1, 2012**.

The first report makes up 1/3 of the mark for the report, the second report 2/3. Both reports should be sent in .pdf or .doc(x) format to g.j.postma@science.ru.nl.

1.5.2 The exam

The exam will consist of both theoretical and practical questions on the methods discussed during the course. For the practical part, the student should be able to perform little programming in R. Basically, the student must be able to perform the methods from the computer practicals on new data and make the required plots.

The date on which the exam will be held, will be set in mutual agreement with all participating students.

1.6 Computer exercises

The computer exercises will be done in R, a statistical programming language that has become the *de facto* standard in many areas of applied statistics, such as bioinformatics and econometrics. It is freely available from the web site <http://www.r-project.org>. The base program is already very powerful and offers a plethora of statistical tools and visualisation possibilities; on top of that several hundred “packages” are available implementing specific procedures. For this course, a special PRiNS package has been developed, that is available from the course website. R, and the PRiNS package, have been installed at the computers in the computer rooms at the Huygens building; students working at home should download and install both R and PRiNS¹. After installation, the PRiNS package can be loaded as follows²:

```
> library(PRiNS)
> library(help = PRiNS)
```

where the second line gives information on all functions and data sets contained in the package. Data sets can then be loaded with the `data` command, e.g.

```
> data(wines)
> ?wines
```

The second line is the general way in R to get information on a function or a data set. To get all help in a browser window rather than in the console, type

```
> help.start()
```

The computer exercises in this studyguide show what can be done; ideally, the students should rework them and try different “what-if” scenarios. More extensive versions of most exercises, with more elaborate pieces of code, are available from the PRiNS package. To see an extended version of exercise 2.3.1, for example, type:

```
> doex(2, 3, 1)
```

The R-code for this exercise is available as function `exer.2.3.1`; it can be viewed as follows:

```
> exer.2.3.1
function ()
{
  cat("*****\n")
  cat("This demo shows different kinds of distance measures.\n")
  cat("The winesLight data are used leading to 60x60 distance\n")
  cat("matrices. First we plot histograms of distances.\n")
  cat("*****\n\n")
  data(winesLight)
```

¹see the course web site for more information and installation scripts.

²Windows users may find it easier to use the pull-down menus.

To save space, only the first couple of lines of the function are shown. Any other R function can be inspected this way. Note that functions of individual packages are sometimes hidden in “namespaces” to avoid name conflicts; in that case you can see the function by typing `pkgname:::function`. You can find out which package contains a specific function by using double question marks. As an example, suppose that you want to see how the function `predict.rpart` is implemented:

```
> ??predict.rpart
```

```
Help files with alias or concept or title matching 'predict.rpart'
using regular expression matching:
```

```
rpart::predict.rpart    Predictions from a Fitted Rpart Object
```

```
Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

```
> rpart:::predict.rpart
```

```
function (object, newdata = list(), type = c("vector", "prob",
      "class", "matrix"), na.action = na.pass, ...)
{
  if (!inherits(object, "rpart"))
    stop("Not legitimate tree")
  ...
}
```

For more information on namespaces and the double and triple colon operators, type `? ":: "`. Inspecting the demonstration functions such as `exer.2.3.1` is useful for trying out other settings, and should help you in applying these functions for your own data. The computer exercises and demos serve to help you answer the questions in the reader. Both forms of the exercises are important!

Several additional exercises are available at the end of each section; these are not numbered, and no demo versions are available. Rather, the student should use the commands from the earlier exercises to reach a conclusion.

Chapter 2

Visualization techniques

Goals

After doing the exercises below you should be able to

- calculate, plot and interpret Principal Components, based on the R function *PCA* from the *PRiNS* package.
- work with ANOVA-simultaneous Component Analysis, based on the R functions from the *PRiNS* package.
- calculate, plot and interpret results from Independent Component Analysis, obtained with functions from the *fastICA* package.
- calculate, plot and interpret multi-dimensional scaling models using the R functions *isoMDS* and *sammon*.
- train, interpret and use self-organising maps from the *kohonen* package.
- analyse the strengths and weaknesses of these visualization methods, and *a priori* indicate which would be most useful in a given situation.
- calculate and visualize (several types of) distances between the rows of a data matrix, using R's *dist* function

The course starts with a general introduction on how to properly visualize data and information in a graph and on how to understand models (models are the results of the application of all techniques that are dealt with during this *PRiNS* course).

If the number of variables is relatively small, pairs plots or coplots can be used to visualize data, but very quickly these lose their attractiveness when the dimensionality of the data increases over, say, ten. In those cases, we are forced to reduce the number of variables or to concentrate on derived properties such as distances or similarities. This leads to methods like Principal Component Analysis (PCA), ANOVA-simultaneous component

analysis (ASCA), Independent Component Analysis (ICA) and Multi-Dimensional Scaling (MDS); a discrete version that focusses on topology rather than distance is known as Self-Organising Maps (SOM). ICA, MDS and SOMs are not part of the current PRiNS course, but the exercises are still available for the interested student. The exercises will highlight strengths and weaknesses of the methods, and show in which they will lead to the same or different conclusions.

2.0. PRiNS Course Study guide: VISUALIZATION OF DATA

2.0 Exercises / Questions for Visualisation course

An important first step in pattern recognition is looking at the data itself. This will tell you about trends in the data, deviating samples and variables and possible information in the data. You will visualise data from a table in Time magazine, that showed the amounts of alcoholic beverages consumed in various countries, together with some health-related figures. For now we will use Excel for visualisation because this facilitates nice and easy construction of plots and is frequently used in laboratories and industry.

	Liquor (gallon/person)	Wine (gallon/person)	Beer (gallon/person)	Life Expectancy (Year)	Heart Disease (cases/100,000 persons)
France	2.5	63.5	40.1	78	61.1
Italy	0.9	58	25.1	78	94.1
Switzerland	1.7	46	65	78	106.4
Australia	1.2	15.7	102.1	78	173
Great Britain	1.5	12.2	100	77	199.7
U.S.A.	2	8.9	87.8	76	176
Russia	3.8	2.7	17.1	69	373.6
Czech	1	1.7	140	73	283.7
Japan	2.1	1	55	79	34.7
Mexico	0.8	0.2	50.4	73	36.4

Exercise. Visualise this data in Microsoft Excel, showing:

- Alcohol preferences around the world
- Relations between life expectancy and heart disease rate
- Relation between alcohol intake and life expectancy and Heart Disease

Use for example Pie charts (think of Minard!), Bar Charts, Bubble plots but try out all visualisations that were mentioned in the course.

If you see outliers or 'special' points, hypothesise why these countries do not follow the trends!

Also make some additional (standard) visualisations from Excel and mention what you do and do not like about them on an artistic and an information-level.

The end result (and not your Excel skill) counts, so copy, paste and Powerpoint all you like!

2.1 Principal Component Analysis (PCA)

PCA is often used as an exploratory tool in data analysis. In the exercises, the suitability of PCA for this purpose will be explored. Although R contains two functions for carrying out PCA (*prcomp* and *princomp*), the function *PCA* from the PRiNS package will be used. This function has a somewhat easier interface, especially for making plots.

Exercise 2.1.1 Load the “spheres” data and plot the three variables:

```
> data(spheres); pairs(spheres)
```

What is the single most ‘interesting’ direction in the spheres data? Do you expect the first PC to lie in that direction? Why (not)?

Perform PCA on the spheres data and make score and loading plots of PCs 1 and 2:

```
> PC.spheres <- PCA(spheres)
> scoreplot(PC.spheres)
> loadingplot(PC.spheres)
```

Can you distinguish classes in this plot? Which PC(s) is (are) responsible for the class separation? Which variables are contributing most to which PC? Confirm this by looking at the loadings:

```
> loadings(PC.spheres)
```

To get the complete picture, look at the *pairs* plot of the scores:

```
> pairs(scores(PC.spheres), main="spheres, PCs")
```

Did you miss anything by looking only at the first two PCs?

Note: the above commands lead to black-and-white plots; for information on how to obtain colour variants, type

```
> exer.2.1.1
```

in your R console, and study the differences with the above set of commands.

Exercise 2.1.2 Now, do Exercise 2.1.1 again for the “discs” data, also included in the *spheres* data set. The “discs” set can be thought of as two discs, approximately lying along the body diagonal of a cube on two opposite sides. Do your answers for this set differ from the ones you gave for the spheres data? In which cases do you expect PCA to be a good way of finding interesting directions in data? And when do you think it is not? Why?

Exercise 2.1.3 In the two previous exercises, you have performed PCA on uncentered data. In general, it is recommendable to *mean-center* the data. Do you have any idea why?

Look again at the *pairs* plot for the spheres data. What will be the difference between the direction of the PCs obtained from the uncentered and the mean-centered data? Why?

Verify your answer by performing PCA on the mean-centered data:

```
> spheres.mc <- scale(spheres, scale = FALSE)
> PC.spheres.mc <- PCA(spheres.mc)
> loadings(PC.spheres.mc)
```

Compare the loadings with the loadings of the uncentered data.

Now load ‘spheres2’, a slightly adapted version of the spheres data, and visualize it in a *pairs* plot. What will be the difference between the direction of the PCs obtained from the uncentered and the mean-centered data in this case? Why?

Here also, perform PCA on the uncentered and mean-centered data and compare the loadings. Is there a difference between the loadings now? Why (not)? In which case do you expect the first PC to lie along the ‘most interesting direction’? Verify your answer by comparing the *pairs* plots of the scores for the two cases:

```
> pairs(scores(PC.spheres2))
> pairs(scores(PC.spheres2.mc))
```

Exercise 2.1.4 Load the *winesLight* data, visualize the variables in a *pairs* plot and calculate the correlations between them:

```
> data(winesLight); pairs(winesLight)
> cor(winesLight)
```

Which two variables show the highest (positive) correlation, and which ones show the largest anti-correlation? Which variables are least correlated? Do the plots agree with the correlation matrix? In what way?

Exercise 2.1.5 The PCs that result from a PCA are orthogonal to each other. If the data are mean-centered, orthogonality implies uncorrelatedness (another reason why mean-centering is a good idea).

Verify that the scores of the uncentered *winesLight* data are orthogonal but correlated:


```
> PC.winesLight <- PCA(winesLight)
> crossprod(scores(PC.winesLight))
> cor(scores(PC.winesLight))
```

Now consider the mean-centered *winesLight* data:

```
> winesLight.mc <- scale(winesLight, scale = FALSE)
```

Check that for these data the scores are both orthogonal *and* uncorrelated.

Exercise 2.1.6 The squares of the singular values that result when SVD is applied to a data matrix are equal to the sums of squares of the associated PC scores. Calculate both of these quantities for the *winesLight* data and verify that they are identical:

```
> winesLight.svd <- svd(winesLight)
> winesLight.svd$d^2
> colSums(scores(PC.winesLight))^2
```

For *mean-centered* data, dividing the sums of squares of the scores by the number of objects minus one (the $n - 1$ *degrees of freedom*) produces the scores' variances. Verify that this is indeed true for mean-centered data:

```
> colSums(scores(PC.winesLight.mc)^2) / (nrow(winesLight.mc) - 1)
> diag(var(scores(PC.winesLight.mc)))
```

Check that it does not hold for uncentered data.

Compare the total variances of the original data, of the PC scores for the uncentered data, and of the PC scores for the mean-centered data:

```
> sum(diag(var(winesLight)))
> sum(diag(var(winesLight.mc)))
> sum(diag(var(scores(PC.winesLight))))
> sum(diag(var(scores(PC.winesLight.mc))))
```

Can you make a statement about the distribution of the total variance over the different PCs for uncentered and mean-centered data?

In which case would you expect the first PC to “contain most information”?

Exercise 2.1.7 Assuming that variance equals information, the first few PCs of a mean-centered data set should contain a large part of all information that is present in the original data. A reconstruction, using only those PCs, should therefore already closely resemble the original data.

Use a scree plot to determine the appropriate number of PCs to retain for the *winesLight* data and, using only those PCs, reconstruct the data:

```
> screeplot(PC.winesLight.mc)
> winesLight.mc.rec <- reconstruct(PC.winesLight.mc, npc = 2)
```

Visualize the *error matrix* and have a look at the *root mean squared error* (RMSE):

```
> image(winesLight.mc - winesLight.mc.rec$reconstruction)
> winesLight.mc.rec$error
```

Do you think the data have been reconstructed well? Try a couple of different numbers of PCs for reconstruction. What is the optimal number to use?

Exercise 2.1.8 Make a score plot for the first two PCs of the mean-centered *winesLight* data:

```
> scoreplot(PC.winesLight.mc)
```

We are now going to project other data in the PC space. Let us start with the *winesLightA* matrix, already available from the *winesLight* data. We should scale in the same way, so we first subtract the mean of the *winesLight* data. Then we project the objects into the *winesLight* score space and add them to the score plot:

```
> winesLightA.mc <- sweep(winesLightA, 2, colMeans(winesLight))
> winesLightA.mc.scr <- project(PC.winesLight.mc, npc = 2, winesLightA.mc)
> points(winesLightA.mc.scr[,1:2], col = "red")
```

Why would you use the mean of the *winesLight* data to center the *winesLightA* data rather than the mean of the *winesLightA* data itself? Do you think the *winesLight* data is representative for the *winesLightA* data? Why (not)?

Now, mean-center the combined *winesLight* and *winesLightA* data, perform PCA and again make a score plot:

```
> morewine <- rbind(winesLight, winesLightA)
> PC.morewine.mc <- PCA(scale(morewine, scale=FALSE))
> scoreplot(PC.morewine.mc, col = "red")
> points(PC.morewine.mc$scores[1:60,1:2], col = "black")
```

Does the plot differ much from the previous one? Why (not)?

Exercise 2.1.9 Repeat the previous exercise for the *winesLightB* data. An earlier name for this data set was “BaaadWines”. Do the results live up to the suggestive name of this data set? What do you think is the difference (are the differences) between the *winesLight* and *winesLightB* data?

Exercise 2.1.10 As we have seen, the loadings contain information on which variables are important in what PCs. In some special cases, the loading plots themselves reveal important features, especially when there is some sort of ordering in the variables.

As an example, consider the `yeast.alpha` data set, containing microarray measurements of gene activity (roughly spoken); it is a time series of 18 points on 613 genes. First perform a PCA on the mean-centered data, and then have a look at the loading plot.

```
> data(yeast.alpha)
> ya.PC <- PCA(scale(yeast.alpha, scale = FALSE))
> loadingplot(ya.PC, show.names = TRUE)
```

Look carefully at the variable names in the loading plot. Do you notice something? Connect subsequent points in time with a red line:

```
> lines(loadings(ya.PC)[,1:2], col = "red")
```

Can you estimate the length of the cell cycle in yeast?

Exercise 2.1.11 A second example where loadings are informative in themselves is in analysing spectral data. The loading of the first PC often resembles a spectrum! Larger loadings (peaks) indicate more important variables for that particular PC.

Let us look at a NIR data set, recorded at five different temperatures. Nineteen different mixtures of ethanol, water, and iso-propanol have been measured (95 samples in total); 200 wavelength numbers have been recorded (850 – 1049 nm).

```
> data(nir.spectra)
> nir.PC <- PCA(scale(nir.spectra, scale = FALSE))
> plot(850:1049, loadings(nir.PC)[,1], type = "l",
+       ylab = "Loadings", xlab = "Wavelengths (nm)")
```

The loadings of PCs 2, 3 and further do not look like a (mean-centered) NIR spectrum; they should be orthogonal to the first PC! However, they do show smooth and continuous curves.

Exercise 2.1.12 As a final example, consider the influence of scaling. This can be dramatic! Suppose we have data where the individual variables have been measured at different scales. An example is the (complete) wines data set, of which the `wines-Light` set is a subset. Let us compare score plots and loading plots for mean-centered and autoscaled data: in autoscaling, every variable is standardized to zero mean and unit variance, so that units of measurement are no longer relevant. First, load the data, and perform PCA:

```
> data(wines)
> PC.wines <- PCA(scale(wines, scale = FALSE))
> PC.wines.as <- PCA(scale(wines))
```

Then, plot score and loading plots:

```
> par(mfrow = c(1, 2))
> scoreplot(PC.wines)
> loadingplot(PC.wines, show.names = TRUE)
```

Which variables are most important on PCs 1 and 2?

Do the same for the autoscaled data. Now, which variables are most important? Where are the variables that dominated the mean-centered PCs?

Extra exercise The *winesLight* data (60 objects, 6 variables) are a subset of the wines data (177 objects, 13 variables). The six variables in the *winesLight* data are variables numbers 2, 3, 6, 7, 9 and 12 in the original wines data set, and have been selected for the similar scale of their variances.

Suppose we would want to use the *winesLight* data as a *representative* subset of the wines data (of course, only for the selected variables). We can investigate whether this is possible by making a PCA model on the *winesLight* data, projecting the other wine samples into the space of – say – the first two PCs, and to check the score plot for strange phenomena.

Using the commands from the previous exercises, perform this check. Are the *winesLight* data a representative subset? How about assessing each of the three separate cultivars separately? Would it be useful to reverse the procedure, i.e. to make a PCA model on the complete set, and to project the *winesLight* data into that space?

2.2 Independent Component Analysis (ICA)

ICA is another exploratory variable reduction technique. As in PCA, the technique defines new, informative directions in the original high-dimensional data. ICA maximizes the statistical independence between the extracted components. In the fastICA algorithm that will be used in this course, the independence is achieved by maximizing an approximation of a quantity called *negentropy*, which one can also see as a degree of “information content” or “interestingness”. The *fastICA* algorithm applied here is a particularly fast approximation that has become very popular in the last decade.

Exercise 2.2.1 The fastICA algorithm first mean-centers, then performs PCA and finally applies ICA. Both the PCs and the ICs can be obtained from the result of the algorithm. Perform ICA on the spheres and discs data and visualize the PCs and ICs for both sets in *pairs* plots:

```
> data(spheres)
> IC.spheres <- fastICA(spheres, 3)
> IC.discs <- fastICA(discs, 3)
> par(mfrow = c(2, 2))
> pairs(IC.spheres$X %*% IC.spheres$K, main="spheres, PCs")
> pairs(IC.spheres$S, main="spheres, ICs")
> pairs(IC.discs$X %*% IC.discs$K, main="discs, PCs")
> pairs(IC.discs$S, main="discs, ICs")
```

What is your opinion about the directions of the ICs compared to those of the PCs? Are the directions more, less, or equally ‘interesting’?

Exercise 2.2.2 Calculate the variances of the original discs data and of the discs ICs:

```
> diag(var(discs))
> diag(var(IC.discs$S))
```

What is strange about the results?

The fact that all variances are (nearly) equal to one is the result of an ambiguity in ICA: the magnitude (“scale”) of the individual ICs is arbitrary and therefore generally chosen to be such that the variance of the ICs equals one. This also means there is no hierarchy in the ICs: which IC is the “first”, which the “second”, etc. is indeterminate. If you run the fastICA algorithm several times (for the discs data for example), you will encounter differently ordered ICs.

Exercise 2.2.3 Perform ICA on the *winesLight* data, project the *winesLightA* data (scaled in the correct way of course) in the IC-space of the *winesLight* data and make a *pairs* plot, showing the *winesLightA* objects in red:

```
> data(winesLight)
> IC.winesLight <- fastICA(winesLight, 6)
> S.winesLightA <- project(npc = 2, newdata = winesLightA.mc,
+                          ldngs = IC.winesLight$K)
> colors <- c(rep(1,nrow(winesLight)), rep(2,nrow(winesLightA)))
> pairs(rbind(IC.winesLight$S, S.winesLightA), col= colors)
```

Do the same for the *winesLightB* data; also investigate ICA on the combined *winesLight* and *winesLightB* data matrix. Is there a large difference between the projections into the ICA space and the ICA on the combined data? Is the ICA behaviour similar to PCA in this respect, or not?

Extra exercise Perform ICA on the yeast data, and compare the results with the PCA results.

Extra exercise Perform ICA on the NIR data, and compare the results with PCA.

Extra exercise Perform ICA on the wine data, with and without autoscaling; again, compare the results with PCA.

2.3 Multi-dimensional Scaling (MDS)

Multi-Dimensional Scaling, sometimes also called nonlinear mapping (NLM), visualizes a distance matrix, rather than a data matrix. The aim is to find a two-dimensional set of points whose distance matrix – in some sense – approximates the distance matrix from the original high-dimensional data as well as possible. In many cases, this is extremely valuable: think, for example of a typical biological data set where the number of variables (e.g., gene expressions in transcriptomics, m/z values in proteomics, or ppm values in metabolomics) often is in the order of 10^5 , but the number of objects (typically, patients and controls) rarely exceeds 100. Moreover, in some cases specialized distance functions are needed to express the “true” similarities between objects.

The price to be paid is a certain loss of information: the original data are no longer visible. Therefore, it is not directly possible to investigate which variables correlate with a possible group separation. And, of course, many data sets exist where the number of objects is huge: in those cases MDS is of no help. Only calculating the distance matrix would take too much computer resources, let alone calculating the optimal 2D configuration.

Several different distance measures have been implemented in R. Although the Euclidean distance is by far the most used one (and is also the default in R’s *dist* function), there are several situations in which other distances are more appropriate. An important one is the Mahalanobis distance (function *mahalanobis*). The exercises scratch the surface of the topic; the take-home message is that the concept of “distance” has many facets.

Exercise 2.3.1 Calculate the Euclidean distance between the objects (rows) in the *winesLight* data matrix, and make a histogram of the distances:

```
> data(winesLight)
> winesLight.euclid <- dist(winesLight, method = "euclidean")
> hist(winesLight.euclid, main = "Euclidean")
```

Do this for the other distances implemented in the *dist* function, and compare the histograms (remember the *par(mfrow = c(3,2))* trick from Example 2.1.12?).

Note that the numbers on the x-axis change, as well as the shape of the distribution.

Another useful visualisation is given by the *image* function:

```
> image(as.matrix(winesLight.euclid), main = "Euclidean")
```

Compare the images of the different distance matrices. Can you see whether the *winesLight* samples are ordered by class?

Exercise 2.3.2 For the Euclidean, maximum, Manhattan and Minkowski ($p = .5$) metrics, calculate the distances between the rows in the *winesLight* data, and compare the results by plotting, e.g.;

```
> plot(winesLight.euclid, winesLight.maximum, xlab = "Euclidean",
+      ylab = "maximum")
```

Which ones are similar? Which ones are different?

Exercise 2.3.3 Calculate the (Euclidean) distances between the *scores* of the *winesLight* data, using only 2 PCs. How similar is this distance matrix to the “real” distance matrix? Visualize the difference between the two distance matrices in an image. You may want to use the function *as.matrix* on the result of a call to *dist...*

Now, perform Sammon mapping of the *winesLight* data:

```
> wl.sammon <- sammon(dist(winesLight))

Initial stress      : 0.02990
stress after 10 iters: 0.01511, magic = 0.500
stress after 20 iters: 0.01488, magic = 0.500
stress after 30 iters: 0.01487, magic = 0.500
```

Compare the scores of the PCA model with the points of the Sammon mapping. To see the latter, type:

```
> plot(wl.sammon$points, main = "winesLight: Sammon mapping",
+      col = winesLight.classes, pch = winesLight.classes)
```

Are the results different?

Exercise 2.3.4 Visualize the different distance matrices, from Exercise 2.3.2, for the *winesLight* data with Sammon mapping. Is your earlier conclusion confirmed?

Exercise 2.3.5 Non-metric MDS is implemented in R in the function *isoMDS*. Apply this technique to the *winesLight* data, and visualise the result.

Calculate distance matrices from the PCA scores, Sammon mapping and non-metric MDS solutions. Visualize the differences of these distance matrices with the original distance matrix. Do you see any patterns?

Exercise 2.3.6 Perform the MDS variants on the combined *winesLight* and *winesLightB* data. Note that in MDS it is not possible to “map” new data in the low-dimensional space; you *have* to recalculate the mapping with the combined data set. What can you say about “outlying” observations and the effect on the mapping? Which observations have the largest influence on the overall shape of the point cloud?

Extra exercise The *yeast* data contain 613 genes (objects, rows) and 18 variables. Can you perform the MDS variants on the yeast data matrix? Which ones are fast, and which ones take more time?

What happens if you do MDS on the *transposed* yeast data matrix? What can you learn from these plots? Is this in agreement with earlier conclusions?

2.4 Self-organising maps (SOMs)

The power of SOMs will become immediately obvious upon performing a few mappings. You will see that similar objects will land in the same region in the map, leading to an immediate and interpretable view of a large data set. You will be able to experience the consequences of different choices for map size and topology, different settings for training parameters, etcetera.

Exercise 2.4.1 To get an overview of the kinds of plots we can expect when using SOMs, we use the wine data (the complete set). To make sure all variables are equally important, we first perform autoscaling. The R package implementing the SOMs is called `kohonen` (automatically loaded with the `PRiNS` package), in honour of the “inventor” of SOMs, Teuvo Kohonen. Training the self-organising map is done by the function `som`, which requires a data matrix and a unit grid as arguments. Several other arguments are optional (see the help page).

```
> data(wines)
> wn <- scale(wines)
> wine.map <- som(wn, somgrid(4, 4, "hexagonal"))
```

Now the map has been trained, and we can start some visualisation. The mapping itself is shown by entering “mapping” as the plotting type:

```
> plot(wine.map, type = "mapping",
+ col = wine.classes, pch = wine.classes)
```

The vector `wine.classes` contains the class labels, and in these plots is used for defining plotting colours and symbols.

The number of objects per unit is shown with a value of “counts” for the type argument, whereas the codebook vectors are shown when type equals “codes”. The changes in the codebook vectors, usefull to assess whether the map has converged, are shown by using “changes” as the type argument. Finally, the average dissimilarity of an object to the corresponding codebook vector is shown by the “quality” argument. The more similar the objects are to the codebook vectors (the smaller the distance), the better the mapping. Obviously, this measure is not very useful to compare maps of different size: larger maps will almost always show a better agreement between codebook vectors and data.

```
> plot(wine.map, type = "counts")
> plot(wine.map, type = "codes")
> plot(wine.map, type = "changes")
> plot(wine.map, type = "quality")
```

It should now be clear that the maps offer quite a good view on the similarities in the data, even for very high numbers of objects.

Exercise 2.4.2 One difficulty in interpreting SOMs is that they start from a random initialisation. Typically, random objects are taken as codebook vectors in the first iteration. The effect is clear when you repeatedly train SOMs; every time a different random seed is used. If you want, you can explicitly set a random seed:

```
> set.seed(7)
```

Repeating the training with the same random seed will start from the same initial configuration, and therefore will lead to the same mapping.

Study the effect of repeated training, with or without setting an explicit random seed. You should use the visualisation functions of the previous exercise. Which of these is (are) most useful in this context?

Exercise 2.4.3 The most obvious setting is the size of the map. Maps that are too large contain many empty regions, whereas too small maps may not be able to fit in the data. What is the correct size depends on the application; in most cases, one should not have too many objects per unit (unless they really *are* similar), and not have too many empty units. The penalty for the latter is an unnecessary long computation time and a more difficult interpretation.

Again, train a map with a specified size and topology, e.g. a 6-by-6 hexagonal one, and plot the mapping of the samples:

```
> wine.map <- som(wn, somgrid(6, 6, "hexagonal"))
> plot(wine.map, type = "mapping",
+      col = wine.classes, pch = wine.classes)
```

Train SOMs of different sizes on the wine data, and observe the different mappings. Instead of the hexagonal unit organisation one may also try out rectangular arrangements.

Exercise 2.4.4 In so-called toroidal maps, the edges are joined so that the resulting surface resembles a closed tube (see the figure below).



In such a map, all units are exactly equal; in non-toroidal maps the units on the outer edges are by definition different from units more to the center. This is not necessarily a bad thing; outlying objects or groups of objects tend to like a hiding place near the edge of the map.

```
> set.seed(34)
> wine.map <- som(wn, somgrid(12, 6, "hexagonal"), toroidal = TRUE)
> plot(wine.map, "mapping", col = wine.classes, pch = wine.classes)
```

Note that the three groups – in some mappings, at least – appear to be split in the mapping plot; however, in many cases this is an artefact due to the necessarily non-toroidal graphical representation.

Exercise 2.4.5 For large data sets, training a SOM can take a lot of time. One may be tempted to squeeze back the number of training iterations. In some cases this may work, but in others the mapping may become suboptimal. For a small set like the wine data there is no reason not to be generous and take a reasonably large number of iterations. The effects of too little training are easily seen; just look at the quality plots.

```
> set.seed(7)
> wine.map <- som(wn, somgrid(6, 6, "hexagonal"), rlen = 50)
> plot(wine.map, type = "quality")
```

Try training a map with only very few iterations, and plot the quality map.

Compare different numbers of iterations; note the change of scale in the colour key!

Exercise 2.4.6 One can also use SOMs for prediction. By mapping the objects into a SOM, one also gets an idea of where different classes are located; you have already seen several examples of that. To estimate the class of a new wine sample, for example, we may want to check where it is mapped in the SOM, and then assume it has the same class as the (majority of the) other wine samples in the neighbourhood.

```
> wine.map <- som(wn, somgrid(6, 6, "hexagonal"))
> prediction <- predict(wine.map,
                        trainY = classvec2classmat(wine.classes))
```

We did not use the class of the wine samples in training the map, so this information must be supplied explicitly to the `predict` function. The function `classvec2classmat` converts a class vector to a matrix where the columns indicate class membership. Next, we can visualise which classes occur in which units:

```
> unit.classification <-
+   as.numeric(classmat2classvec(prediction$unit.prediction))
> plot(wine.map, "property", main = "Unit classification",
+       property = unit.classification, palette = rainbow,
+       contin = FALSE)
```

Note that with this implementation also empty units will be predicted; in that case, the majority vote for the neighbouring units is used. Other software packages may make other choices.

Exercise 2.4.7 The differences between SOM mapping on the one hand, and the other mappings we have studied on the other, can be nicely illustrated by imagining a multivariate normal distribution, i.e., a point cloud where the point density decreases with the distance from the origin:

```
> data(pts)
```

Try to (mentally) visualize the score plots for PCA and ICA, the MDS result, and the SOM result. Using data sets *pts2* (a two-dimensional data set, so the raw data can be plotted as well) and *pts5* (a five-dimensional multivariate normal distribution), examine the different mappings. How close were you?

Extra exercise Visualize the mapping of the *yeast* data in a SOM. This is more like it: the number of objects is reasonably large, yet training should go fast. What size of SOM is appropriate? Try a few different sizes.

Visualize the codebook vectors (try the *codeRendering = "lines"* argument to the plotting function). Can you see the cell cycle in the codebook vectors?

Compare the mapping of the genes in the SOM to the PCA mapping. What can you say about the relation between individual genes – or groups of genes – and the timepoints in both the SOM mapping and the PCA mapping?

Extra exercise Visualize the *nir.spectra* data in a SOM. Pay attention to scaling! Again, pick the appropriate map size.

Use temperature (*as.integer(temp)*) to colour the points (and perhaps to use different plotting symbols). Are the samples ordered in the map according to temperature? Why (not)?

What about ordering after water content? You can use

```
as.integer(cut(water, seq(0, 1, length = 5)))
```

for colours or plotting symbols.

2.5 Multivariate Curve Resolution (MCR)

2.5 Multivariate Curve Resolution (MCR)

Lecturer: Dr. Lionel Blanchet

The Multivariate Curve Resolution (MCR) methods are widely used in the analysis of mixtures in chemistry and biology. The main interest of this method is to perform a matrix decomposition allowing obtaining the individual characteristics (spectra, chromatograms ...) of the chemical compounds of mixtures. Although MCR was mainly developed and used in Matlab, the main functions have been transposed in *R* into the *ALS* package.

Exercise 2.5.1 Load the “NIRdata” and plot the data set

```
> data(NIRdata)
> matplot(wv, t(NIRdata), type="l", lty=1)
```

The data set is here composed of 2000 variables. The 30 near infrared spectra represent 30 mixtures of chemical compounds. The vector “wv” contains the corresponding wavelengths. Do you observe any evolution in the spectral shape between the different samples? Are you able using this graphical representation to imagine the number and the shapes of the spectra of the pure chemical compounds forming the mixture?

Exercise 2.5.2 Perform exploratory analysis

In the previous chapter, you have performed Principal Component Analysis (PCA) on multiple examples. Use the same approach to visualize the “NIRdata”.

CHAPTER 2. VISUALIZATION TECHNIQUES

```
> NIRdata.mc <- scale(NIRdata, scale=FALSE)
> PC.NIRdata.mc <- PCA(NIRdata.mc)
> scoreplot(PC.NIRdata.mc, main = "Scores")
> loadingplot(PC.NIRdata.mc, main = "Loadings")
```

Look at the score plot obtained. Do you visualize a logical evolution between the samples? From this observation are able to estimate the number of chemical compounds present in the mixture?

The score plot provides you information about the different samples. The information corresponding to variables is also of interest in this problem. Visualize the loading plot as you learn in chapter 2. Are you able to determine the chemical rank of this data set (i.e. number of compounds present)?

The resulting plot is difficult to read. Two reasons can explain it. First, the number of variables is very high (2000 variables), so the loading plot is too crowded. Second, a lot of variables are colinear: the different wavelengths belonging to the same peak appear grouped on the loading plot. However, the information about variables may be visualized in a more usual way for a chemist: as a spectrum.

```
> matplot(wv, loadings(PC.NIRdata.mc)[,1:2], type="l", lty=1)
```

Can you identify the most important peaks for the PCA model? Are you able to interpret these loadings as spectra? In such a “toy-example”, the interpretation is still doable. However, this task remains quite difficult and non intuitive for most of the chemists or biologists.

One of the objectives of MCR is to obtain more readable results. To do so the main constraint of PCA is removed: the orthogonality of the different components. Indeed two spectra of two pure chemical compounds can share multiple features and are therefore non orthogonal. The objective is to obtain an interpretable solution directly after

2.5 Multivariate Curve Resolution (MCR)

statistical analysis. The components extracted must therefore stay within a meaningful “chemical subspace”.

Exercise 2.5.3 Construct an initial guess of the pure spectra

The first approach in MCR is trying to estimate the pure spectra directly from the data. This is the approach proposed in SIMPLISMA (SIMPLe to use Interactive Self-Modelling Algorithm). The idea is to find the most different spectra within the dataset and assume that they are the most pure ones.

```
> simplisma(t(NIRdata), 2, 0.1)
```

Apply SIMPLISMA to the “NIRdata” for two components. Observe the resulting estimation of the pure spectra. Do they look like NIR spectra?

```
> simplisma(t(NIRdata), 3, 0.1)
```

Redo the SIMPLISMA to the “NIRdata” but for three components. Observe the resulting estimation of the pure spectra. Do they still they look like NIR spectra? Do the two first components resemble the ones obtained before?

Exercise 2.5.4 Construct an initial guess of the concentration profiles

In the previous exercise pure spectra were estimated. One could be more interested in the evolution of the compound concentrations in the mixture. One possible approach is to apply SIMPLISMA on the transposed NIRdata. (You can do it as an extra exercise.)

CHAPTER 2. VISUALIZATION TECHNIQUES

A second approach aims to detect and characterize evolution in the data set. This is done using Evolving Factor Analysis (EFA). As previously the calculation can be done using different chemical rank. Apply EFA for 2 and 3 components.

```
> efa(NIRdata, nf=2, plot=1)
```

```
> efa(NIRdata, nf=3, plot=1)
```

Can you interpret the results obtained? Does it make sense? How comparable are the two results obtained?

Exercise 2.5.5 Unconstrained Multivariate Curve Resolution

The two precedent methods only use explicitly one of the two directions available in the NIRdata: the concentrations or the spectra. The algorithm MCR ALS (Multivariate Curve Resolution by Alternating Least Squares) is using both directions alternatively until it converges to a stable solution.

Use the initial estimates for EFA as initial guess for MCR-ALS. Calculate the best decomposition for 2 and 3 components without constraints.

```
> bla2 <- efa(NIRdata, nf=2)
```

```
> test0 <- als(CList=list(bla2$E), PsiList=list(NIRdata), S=matrix(1, nrow=2000,  
ncol=2), nonnegC=FALSE, nonnegS=FALSE, normS=2, x=wav)
```

```
> matplot(test0$CList[[1]], type="l")
```

```
> x11()
```

```
> matplot(test0$S, type="l")
```

```
> bla3 <- efa(NIRdata, nf=3)
```

```
> test1 <- als(CList=list(bla3$E), PsiList=list(NIRdata), S=matrix(1, nrow=2000  
, ncol=3), nonnegC=FALSE, nonnegS=FALSE, normS=2, x=wav)
```

2.5 Multivariate Curve Resolution (MCR)

```
> x11()
> matplot(test1$CList[[1]], type="l")
> x11()
> matplot(test1$S, type="l")
```

Compare the results first graphically and then using the statistical criterions R^2 and lack of fit (lof). Which solution appears to be the best? What would you conclude about the chemical rank of this data set?

Exercise 2.5.6 Effect of a wrong initial guess and constrained MCR-ALS

We will repeat the previous example using different estimates. The calculations produce different results. Let's try using a bad starting point. Load the initial guess IG_wrong, and use it in MCR-ALS.

```
> test0 <- als(CList=list(IG_wrong), PsiList=list(NIRdata), S=matrix(1,nrow=2000,
ncol=3), nonnegC=FALSE, nonnegS=FALSE, normS=2, x=vv)
> x11()
> matplot(test0$CList[[1]], type="l")
> x11()
> matplot(test0$S, type="l")
```

What can you conclude about the results?

Now have a look at the initial guess you used.

```
> matplot(IG_wrong, type="l")
```

CHAPTER 2. VISUALIZATION TECHNIQUES

As you can see one component is entirely negative, which means that this component is really far from a real chemical contribution. Try now to use again IG_wrong in MCR-ALS but this time using the non negativity constraint.

```
> test1 <- als(CList=list(IG_wrong), PsiList=list(NIRdata), S=matrix(1,nrow=2000,
ncol=3), nonnegC=TRUE, nonnegS=FALSE, normS=2, x=wav)
> x11()
> matplot(test1$CList[[1]], type="l")
> x11()
> matplot(test1$S, type="l")
```

Is the constrained result better than the previous one? How do you explain it?

Extra exercise (can be executed as exercise 2.5.7). The intensity ambiguities

The drawback of the loss of orthogonality (compare to PCA) is that multiple solutions are possible for one data set, i.e. multiple sets of components equally good in terms of statistics but leading to different interpretations.

The simplest problem is the called “intensity ambiguities”. The interpretation of the results can be troubled by this effect but in most of the case it is easy to detect.

Let’s multiply the concentration output of the exercise 2.5.5 for three components by a vector [1 5 1].

```
> bla3 <- efa(NIRdata, nf=3)
> test1 <- als(CList=list(bla3$E), PsiList=list(NIRdata), S=matrix(1,nrow=2000,
ncol=3), nonnegC=FALSE, nonnegS=FALSE, normS=2, x=wav)
> C <- test1$CList[[1]]
> C[,2] <- C[,2] * 5
```

2.5 Multivariate Curve Resolution (MCR)

```
> S <- test1$S  
> S[,2] <- S[,2] / 5
```

The spectra must be divided by the same vector. The matrix of residuals remains the same i.e. the new model is as good as the previous one (same R^2 and lof).

Now look at the concentration profile. The results look quite different with an higher concentration of one of the compounds. However if you look at the spectra it is easy to detect a scale problem.

```
> x11()  
> matplot(C, type="l")  
> x11()  
> matplot(S, type="l")
```

Extra exercise (can be executed as exercise 2.5.8). The rotation ambiguities

The second problem is more complex. Instead of multiplying (or dividing) the different components by a scalar, the model is now multiply (or divided) by a rotation matrix. Use the matrix “m_rotation” to transform the MCR model form the exercise 3.3.1 (with 3 components)

```
> bla3 <- efa(NIRdata, nf=3)  
> test1 <- als(CList=list(bla3$E), PsiList=list(NIRdata), S=matrix(1,nrow=2000,  
ncol=3), nonnegC=FALSE, nonnegS=FALSE, normS=2, x=vv)  
> rot.mat <- matrix(c(0.36,0.48,-0.8,-0.8,0.6,0,0.48,0.64,0.6), ncol=3, nrow=3,  
byrow=TRUE)  
> C <- test1$CList[[1]]  
> C[,2] <- C[,2] %*% rot.mat
```

CHAPTER 2. VISUALIZATION TECHNIQUES

```
> S <- test1$S  
> S[,2] <- S[,2] %*% ginv(rot.mat)  
> x11()  
> matplot(C, type="l")  
> x11()  
> matplot(S, type="l")
```

Again you can check the concentration and the spectra. In this case you observed a strong effect. Do you think that the use of constraint could help to limit or prohibit this effect completely?

2.6 ANOVA-simultaneous component analysis (ASCA)

ANOVA-Simultaneous Component Analysis (ASCA) is a method that is specifically suited to determine the chemical influences different factors may have. Therefore, it combines ideas from Analysis of Variance (ANOVA) with those of Principal Component Analysis. In these exercises you will use ASCA to describe the chemical defence that plants have against their natural enemies.

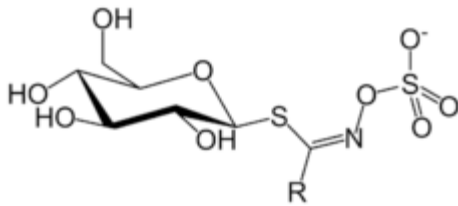


Figure 1 Glucosinolate

Plants in the Cabbage family (*Brassicaceae*) make very specific defence compounds when under attack by herbivores. These compounds are called glucosinolates: about 120 different species exist, varying only in the group R (Figure 1). To simulate a herbivore attack and thus elicit the plant to make glucosinolates, the plant hormone Jasmonic Acid (JA) was administered to either the roots (Root-induced) or to the leaves (Shoot-induced) of cabbage plants (*B. oleracea*). Then the glucosinolate levels were measured 1, 3, 7 and 14 days after treatment. This measurement was destructive such that on every time-point different plants were analysed. The plants contain 11 different glucosinolates. We want to know now:

- **How the treatment affects the glucosinolate composition.**
- **Whether these changes also vary in time.**

Table 1 Glucosinolates	
Variable #	Glucosinolate
1	PRO
2	RAPH
3	ALY
4	GNL
5	GNA
6	4OH
7	GBN
8	GBC
9	4MeOH
10	NAS
11	NEO

Exercises

1. Raw data

- a. Describe in words what kinds of chemical variation you expect there to be present in this experiment? Which factors and interactions may be present in the data (go back to ANOVA)? How would you split up the variation in the data caused by the experiment?
 - b. Inspect the raw data using 'exer.2.6.1'. Which glucosinolates change in time? Which differ between treatments?
-

2. ASCA model

- a. Make an ASCA model of the mean-centered data, use 'exer.2.6.2' for this. Discuss how many PCs each submodel needs, taking into account the three criteria described in the lectures.
 - b. Use 'exer.2.6.3' to make an ASCA model of the plant data. Discuss the different submodels and which information they describe.
 - c. Compare and discuss the ASCA model with the PCA models you made in the previous session. Which advantages do each of the models have (parsimony, interpretability)? In which case does it not make sense anymore to make an ASCA model?
-

3. ANOVA and PCA

- a. Discuss how ASCA can overfit the data. Try to describe an approach by which you could test for this overfit.
 - b. How could you combine ANOVA and PCA in another way? How would this change your view on the experiment?
-

Chapter 3

Unsupervised methods: clustering

Goals

After doing the exercises below you should be able to

- calculate, plot and interpret hierarchical clustering models, based on the R functions *hclust*, *plot.hclust* and *cutree*.
- calculate kmeans clustering models using the functions *kmeans* and *pam*.
- compare the results of different clusterings with each other and with the “true” partition using the *table* function.
- build probabilistic clustering models using the *mclust* package, and interpret and plot the results.
- assess the strengths and weaknesses of the different clustering strategies, and indicate which is most appropriate in a given situation.

We distinguish between two forms of clustering in this course. On the one hand, we have the hierarchical methods which build a tree – a dendrogram – of progressively smaller and better-defined clusters. Cutting the tree at a certain height leaves a discrete set of clusters, where every object is assigned to a group without any doubt. On the other hand, there are the partitioning methods, which divide a data set in a predefined number of clusters. They are iterative in nature; in contrast to hierarchical methods, objects initially in different clusters can come together during the iterations. Partitioning methods come in two flavours, “crisp” (discrete) and “fuzzy” (continuous). The latter calculate (approximations of) probabilities; if one needs a discrete clustering result, an object is typically assigned to the cluster having the highest probability.

After these exercises one should be aware of the underlying forces driving the algorithms, and one should be able to apply them, and visualise and interpret the results.

3.1 Hierarchical clustering

Many forms of agglomerative hierarchical clustering exist¹, and although they share the same philosophy (repeatedly merge objects or clusters until only one big cluster is left), the results may vary wildly. The objective of these exercises is to show how to perform, visualise and interpret hierarchical clustering, and from there obtain an understanding of the characteristics of the individual methods. Keep in mind that these methods start with a distance matrix; if that does not capture relevant information on grouping (e.g., because of inappropriate scaling), any hierarchical clustering method is doomed to fail...

Exercise 3.1.1 For the first five objects in the *winesLight* data set, calculate the (Euclidean) distance matrix. Which two objects are closest?

Define the combination of these two as the first merger, with the center at the average of the coordinates. Calculate the respective distances of the other three objects to the new cluster, and create the resulting four-by-four distance matrix.² Which distance now is the smallest?

Continue creating merges until only two clusters are left, and note the distances (the heights) at which the merges take place.

Now, let R do the job:

```
> X <- winesLight[1:5,]
> X.dist <- dist(X)
> X.ave <- hclust(X.dist, method = "average")
> plot(X.ave)
```

The final statement creates the dendrogram.

Is the order of the merges the same as with your hand-crafted model? And the *y*-values at which the merges take place?

Exercise 3.1.2 For the first five objects in the *winesLight* data, construct hierarchical models for single linkage, average linkage and complete linkage (using the *method* argument of the *hclust* function). Plot the dendrograms. What differences do you see between the different methods?

Do the same for the first ten objects from the *winesLight* data. While you are at it, take the first twenty. What conclusions can you draw?

¹There is such a thing as divisive hierarchical clustering, where one continuously splits clusters, but it is much harder to implement and is hardly ever used; we will not mention it again in this course.

²You can do this by hand if you are not proficient in R; if you want to program your own script, look at the code in the computer demo.

Exercise 3.1.3 Repeat the previous exercise, but now apply autoscaling to the *winesLight* data (*before* calculating the distance matrix). Are the results very different? Why (not)?

Exercise 3.1.4 A dendrogram in itself is not a clustering; only after we “cut” the dendrogram at a certain height do we obtain the cluster labels. R allows us to do that in two ways. Either we specify the number of clusters we desire:

```
> c11 <- cutree(X.ave, k = 3) # three groups
> plot(X.ave)
> rect.hclust(X.ave, k = 3) # show the three groups in the dendrogram
```

or we specify the height at which to cut:

```
> c12 <- cutree(X.ave, h = 0.2) # cut at a specific height
> plot(X.ave); rect.hclust(X.ave, h = 0.2, col = "red")
```

The vector that is returned by *cutree* contains the class labels per object.

Compare the clustering of the *winesLight* data using complete linkage, average linkage and single linkage, when three clusters are chosen. The comparison is most conveniently done with the *table* function:

```
> table(c11, c12)
```

Which of these three methods lead to similar results, if any?

Exercise 3.1.5 Suppose we scramble the order of the variables, so we put column 7 first, and column 13 next, and so on. Do you think the distances will change? Do you think a clustering based on these distances will change?

And what about scrambling the order of the objects? This can be done both in the distance matrix (remember to scramble rows and columns in the same way!) or in the original data matrix, before calculating the distances.

Ideally, scrambling variables *or* objects should not lead to different results. However, life is not ideal... Sometimes two merges are equally good. In most implementations (also in R), the first possible merge is taken. This may lead to a difference if the order of the objects changes.

You can check this using the following piece of R code, which will continue to scramble the order of the rows in the data matrix until a “different” clustering is found:

```

> X.ave <- hclust(dist(winesLight[1:10,]), method = "average")
> ntries <- 50 # at most 50 attempts to find something
> for (i in 1:ntries) {
+   new.order <- sample(nrow(X))
+   X2.dist <- as.matrix(X.dist)[new.order, new.order] # scramble
+   X2.ave <- hclust(as.dist(X2.dist), method = "average")
+   if (!all(X2.ave$height == X.ave$height)) {
+     plot(X2.ave, main = "Alternative clustering")
+     break # stop if you found a different clustering
+   }
+ }

```

The ordering of the variables never influences the result, but a different ordering of the objects may lead to a completely different dendrogram! Few people realise this.

Exercise 3.1.6 Perform hierarchical clustering on the *wines* data, rather than the *winesLight* data. Plot the dendrograms for different forms of hierarchical clustering. We know that the true number of classes equals three; plot the three-cluster solution in the dendrograms by issuing a command similar to

```

> rect.hclust(myclustering, k=3)

```

after plotting the dendrogram. Is the result convincing? Can you obtain a better clustering by taking more clusters? Confirm this using the *cutree* and *table* functions. How should you interpret the results when the number of clusters is larger than the actual number of classes?

Exercise 3.1.7 To take this one step further, load the *yeast.alpha* data:

```

> load(yeast.alpha)

```

and perform hierarchical clustering. Plot the dendrograms. Is there any way in which you can guess the number of classes?

You will notice that it is not at all easy to interpret the dendrograms. Nevertheless, hierarchical clustering has become very popular, even in areas such as microarray data analysis where large data sets are the rule rather than the exception.

3.2 Crisp Partitioning methods

Crisp partitional clusterings lead to a clustering where every object is unambiguously assigned to one cluster. Most algorithms require the number of clusters as an input parameter; if one wants to compare clusterings with different numbers of clusters, one is forced to repeat the procedure several times.

Exercise 3.2.1 The simplest partitional method is k-means clustering. We will apply this to the *winesLight* data first. Note that the calculation of a distance matrix is not necessary; a huge advantage for data sets with many objects. In partitional methods, it is necessary to specify the number of clusters in advance. Let us start with the correct number for the wine data: three.

```
> wine.kmeans <- kmeans(winesLight, 3)
> wine.kmeans
```

The last command will cause R to show information about the number of objects in each cluster, the cluster centers (means) and the “tightness” of the clusters (see the next exercise).

Now repeat the procedure a couple of times:

```
> wine.kmeans1 <- kmeans(winesLight, 3)
> wine.kmeans2 <- kmeans(winesLight, 3)
```

... and so on. Compare the results:

```
> table(wine.kmeans1$cluster, wine.kmeans2$cluster)
```

Are all class labels equal? Why (not)? Which of the clusterings is the best? Why?

Exercise 3.2.2 The criterion that is optimised in k-means is the sum of squared distances of objects to their respective cluster centers, a measure of the “tightness” of the clusters. This means that it is possible to compare different results, and to keep only the best one. In R, the procedure is automated: one can give an extra argument to the *kmeans* function:

```
> wine.kmeansS25 <- kmeans(winesLight, 3, nstart=25)
```

Only the best solution is returned. Compare the quality of each model by inspecting the resulting sum-of-squared distances:

```
> wine.kmeansS25$withinss
```

Which of your models is best?

Plot the wine data (using a PCA, e.g.), and colour them according to the k-means cluster labels. Is the best model really the best?

Exercise 3.2.3 Load the *yeast.alpha* data. Make k-means clustering models for 3, 4, ..., 12 clusters. Use at least 10 restarts every time. Make a plot of the sums-of-squares of the best solution for every number of clusters. What can you conclude? Is there a natural cut-off that points to a specific number of clusters?

Exercise 3.2.4 Instead of k-means, we can also use a robust version called k-medoids. In that case, the class centers are not calculated by taking the means of the coordinates, but given by “typical” objects. The distance to these centers – or medoids – is not measured in Euclidean distance, but in the sum of the absolute marginal distances. This is implemented in function *pam*. Read the manual pages on *pam* to see how to apply the function.

For the *winesLight* data set, compare the k-medoids clustering solution to the k-means clustering (both using three clusters). Do you see any specific differences?

If yes, which of the two is better, i.e. which shows the best agreement with the true class labels?

Exercise 3.2.5 The function *pam* optimizes a specific measure of tightness, returned in the form of *x\$objective*. It should be as large as possible.

For the *winesLight* data, investigate how many clusters are appropriate according to this criterion, similar to what you did in Exercise 3.2.3.

Extra exercise Cluster the complete *wines* data, using k-means and k-medoids with three clusters. Visualize the results using PCA plots, where every cluster is plotted in a different colour. How good is the agreement with the true labels in each case?

If the results look surprising, you may have forgotten to think about appropriate scaling...

Use different numbers of clusters. Again, assess the agreement with the true labels.

Extra exercise Do the same for the *yeast.alpha* data.

3.3 Probabilistic, or fuzzy, clustering

Rather than a black-and-white answer placing every object without any doubt in one of the possible categories, one often would like to have a measure of confidence for individual cluster assignments, as well as for the clustering as a whole. In case there is doubt, the previous clustering methods would not easily show you; especially the hierarchical clusterings do not give you any idea which classes are more likely and which are less likely for a given object.

Non-crisp clustering methods lead to a result where, for each object, the relative “probabilities” for the separate clusters are calculated. Depending on the method, these can be interpreted as real probabilities or as some other confidence measure (hence the quotes). Several methods exist; all have their specific strengths and weaknesses. Here, we will focus on a clustering strategy that fits mixtures of Gaussians, known as model-based clustering (MBC). For every cluster a center and a covariance matrix is to be estimated, as well as a “mixing probability”, a weight.

In many cases, especially with high-dimensional data, the number of parameters is too large to estimate with any confidence, and constrained models have to be considered. One can only allow clusters, for example, with equal covariance matrices, or with equal shapes or sizes. This significantly reduces the number of parameters to estimate. Using criteria such as the Bayesian Information Criterion (BIC), one can compare different clusterings.

Model-based clustering is implemented in an R package called `mclust`, automatically loaded with the `PRiNS` package.

Exercise 3.3.1 We will introduce model-based clustering using the `winesLight` data. Function `mclustBIC` takes a data matrix and returns a BIC table, which can be plotted and inspected.

```
> wl.mbc <- mclustBIC(winesLight)
> plot(wl.mbc)
```

This plot shows the BIC values for several models; in the three-letter codes, signifying volume, shape and orientation, respectively, a symbol “E” stands for “Equal”, the “V” is “Variable”, and an “I” stands for unity. Thus, in an “EEE” model all clusters have the same volume, shape, and orientation, whereas in a “VEV” model only the shape is the same for all clusters, and volume and orientation may vary.

Study the BIC plot in order to answer questions like: what number of clusters is optimal? Which models perform better, which ones are worse?

How would you interpret the three different BIC values at the one-cluster solution?

Differences in BIC values over 10 units are considered highly significant. Would you say a three-component model is competitive with the very best model? If not, can you think of a reason?

Exercise 3.3.2 You can convert a BIC table into a clustering model using the `summary` function for this type of object. This construction allows you to specify what exactly you are interested in without having to redo the (expensive) calculation of BIC values. Suppose we are only interested in relatively unconstrained models with 3 – 6 clusters:

```
> wl.model <- summary(wl.mbc, G = 3:6, data = winesLight,
+                    modelNames = c("VVV", "VEV", "EEE"))
> wl.model
```

The last line will give you information on the optimal set of three models (as defined by their BIC value), and the number of objects in each cluster, according to the single optimal model. For this particular set of models, a fully unconstrained VVV model with three clusters appears to be best.

Construct and inspect the best 3-cluster model from the BIC table.

Construct and inspect the best “EEE” model from the BIC table.

Exercise 3.3.3 Once you have a clustering model, you can extract information by inspecting its components. To see what is available, type

```
> names(wl.model)
```

The most important element is `z`, the matrix containing the (conditional) probabilities of the objects to belong to the respective clusters. One can plot this, with the objects along the x -axis and the probabilities of clusters 1–3 indicated with the height of the corresponding symbol:

```
> matplot(wl.model$z)
```

From this figure, it is clear that for some objects there is a slight uncertainty. Uncertainty, for that matter, can be defined as the difference between 1 and the largest z value for an object:

```
> plot(wl.model$uncertainty, xlab = "objects", ylab = "uncertainty")
```

For different clustering models, investigate the overall uncertainty, i.e. the sum of the uncertainties of individual objects.

Does overall uncertainty increase or decrease when the number of clusters grows? Is there a minimum uncertainty at a specific number of clusters? Why (not)?

Do you think you can use the total uncertainty to assess the “correct” number of clusters?

Exercise 3.3.4 A discrete, crisp, clustering can be obtained from the z matrix by assigning every object to the most likely class. For this, function `map` is available:

```
wl.cl <- map(wl.model$z)
```

The result is a vector of integer numbers, which can – again – be used to colour objects in, e.g., a PCA scoreplot. For convenience, the classification resulting from applying `map` is also returned in the clustering object: `wl.model$classification`.

Investigate the clustering of the `winesLight` data by making scoreplots of the optimal 2-, 3- and 4-cluster models.

Also check the differences between several three-cluster models. Can you see the difference between a three-cluster model with a high BIC value and one with a low BIC?

Exercise 3.3.5 Instead of first calculating the BIC values and extracting models from the BIC table, one can do this in one go using the `Mclust` function. The result can be plotted in a variety of ways: the default shows all BIC values, and, for the optimal model, the classification for all variables (in a pairs plot), and the classification and uncertainty of the objects in two chosen dimensions (indicated with the `dimens` argument).

```
> wl.mcl <- Mclust(winesLight, G = 1:6,
+                 modelNames = c("VVV", "VEV", "EEE"))
> plot(wl.mcl, data = winesLight, dimens = c(1, 2))
```

The uncertainty plots show the objects with high uncertainty in large, black symbols; accordingly, objects that are classified with high confidence are the small, lightgray dots. There are three levels – see the man page.

Plot several clustering models, and study the differences.

Investigate which objects are most uncertain. Are these the same in different models?

Plot the clustering model obtained with the argument `G = 3`; why is the BIC plot empty?

Exercise 3.3.6 One disadvantage of model-based clustering is the number of parameters that have to be estimated, especially for high-dimensional data sets. This has two effects: the clustering takes a long time, and in some cases gets instable. In the latter case, `Mclust` will not return the model coefficients. In other words: not all desired models may be feasible.

Apply `Mclust` for the (appropriately scaled) `wines` data; look at the BIC plot. Why is the line for the “VVV” model discontinued after more than five clusters?

Of course, there is always the trick of dimension reduction: we can cluster the PCA scores, rather than the original data matrix. Do this for the *wines* data: pick an “optimal” number of principal components, and cluster the scores. Compare the result to the clustering of the original data.

Extra exercise If you want to be surprised, and just would like to look at the data from several different angles, use *randProj*³:

```
> with (wl.model,
+       randProj(winesLight, seeds = 1:8, z = z,
+               parameters = parameters))
> with (wl.model,
+       randProj(winesLight, seeds = 1:8, z = z,
+               parameters = parameters,
+               what = 'uncertainty'))
```

This will show you eight random projections. In some directions, the clusters appear nicely separated; in others, they are completely overlapping. The second plot shows eight random projections of the uncertainties of the clustering for individual objects.

Again, study the uncertainties in different models. Are your previous conclusions still valid?

Extra exercise Apply MBC to the *yeast.alpha* data. Note that there are no problems with the “VVV” model (I checked 1 – 9 clusters). Why is that?

Compare the results with the clustering obtained from the scores rather than the original data.

Warning: clustering the *yeast.alpha* data on my computer (a 3.20 GHz Pentium) took almost a whole minute...

Extra exercise Perform hierarchical clustering, k-means and k-medoids on the scores of the yeast data. Do you see any differences with clusterings based on the “raw” data?

Which clustering method do you prefer for these data? Why?

³The construction using *with* causes R to look for the relevant elements (in this case *z* and *parameters*) within object *wl.model*. This saves a lot of typing.

Chapter 4

Supervised methods: classification

Goals

After doing the exercises below you should be able to

- use and interpret discriminant analyses models such as LDA, QDA, as well as more elaborate versions.
- use and interpret non-parametric approaches, such as k-nearest neighbours, and tree-based classification methods.
- use and interpret neural networks, especially backpropagation networks and radial-basis function networks.
- use and interpret support vector machines.
- estimate prediction accuracies, compare the different characteristics of the classification methods, and be able to choose appropriate methods in a given situation.

Just as in clustering, classification aims to assign individual objects to separate groups. The difference is that in the case of classification, the characteristics of the individual groups are defined beforehand, in the majority of cases indirectly, through a training set. The modelling then consists of finding the optimal discrimination rule between the classes present in the training data. Note that one of the main problems in clustering, finding the “optimal” number of clusters, is no longer present in classification.

Several techniques can be used. Some estimate a kind of density for every class, and assign an object to the class that has the highest density at that position in high-dimensional space. These methods are labelled “Discriminant Analysis”, and we will see linear, quadratic and model-based forms. Others are non-parametric: prime examples are tree-based methods, artificial neural networks, and support vector machines. All methods, again, have their strengths and weaknesses. We focus on when and how to use several of the methods, and how to estimate prediction success. Neural networks and tree-based classifiers are not subject of the current PRiNS course, but are still available for the interested student.

4.1 Discriminant analysis

In Discriminant Analysis, we describe the individual class densities by fitting distributions, usually normal distributions, to the training data. New objects are assigned to the class with the highest density. In Linear Discriminant Analysis (LDA), all classes are fit with the same normal distribution. Interestingly, for two classes the resulting classifier is exactly equal to Fisher's LDA, which does not assume normality but instead aims at separating classes as much as possible (not treated in this course).

When all classes are allowed to be different in shape, orientation and size, we speak of Quadratic Discriminant Analysis (QDA): every class is summarised with a single normal distribution. An even more complex model can be obtained by describing every class with a mixture of normals. This is called Model-Based Discriminant Analysis (MBDA).

Exercise 4.1.1 We have already seen the wine data several times, and you may have wondered what all the fuss was about – we already know everything about the stuff! But now, suppose that we get new samples where the labels have been lost, perhaps because of an unfortunate leakage of a broken bottle, or a particularly heavy rain shower. We know the new bottles belong to one of the three cultivars, but which? The difference with clustering is that in this case, the classes have been determined *beforehand*, by means of a training set.

To visualise things more easily, let us concentrate on two variables: flavonoids and proline concentrations (numbers 7 and 13 in the wine data). Plotting the 177 wine samples for these two variables already shows a good separation between the three classes. The question is: where exactly are the boundaries?

Linear Discriminant Analysis (LDA) gives one solution:

```
> X <- wines[, c(7, 13)]
> wine.lda <- lda(X, grouping = wine.classes)
> wine.predictions <- predict(wine.lda)
> table(wine.predictions$class, wine.classes)
```

This classification method predicts 160 out of 177 objects correctly – better than any form of clustering we have seen in this course. The reason of course is that we have not used the class label information, until now.

Investigate the *wine.lda* and *wine.predictions* objects: use the *names* function to see the individual elements.

The *posterior* element in the *wine.predictions* object gives the posterior probabilities of belonging to a certain group, and can be interpreted just like the *z*-matrix in model-based clustering.

Which objects are classified with the lowest certainty? Visualise that in a plot. Are you surprised by the positions of these objects?

Exercise 4.1.2 Repeat the previous exercise for the 13-dimensional wine data, and plot the objects in the space of the first two LDs.

Which objects are classified with the highest uncertainty? Which are classified with the highest certainty?

What is the performance of LDA on the 13-dimensional data? Is it better or worse than LDA on variables 7 and 13, as in the previous exercise?

Can you think of a reason?

Exercise 4.1.3 Just like PCA, LDA performs a rotation of your data to yield new axes. This time the criterion is not the fraction of variance explained, but the maximal discrimination between the separate classes. You can visualise the new “point of view” by plotting the LDA object:

```
> plot(wine.lda, col = wine.classes)
```

Repeat the LDA analysis of the wine data using mean- and autoscaling data. Compare the LDA plots for both the raw data and the scaled data. What do you notice? Can you explain this?

What happens when you consider other forms of scaling, e.g. log-scaling? Can you explain differences (similarities)?

Exercise 4.1.4 You may have wondered why the technique is called *Linear* Discriminant Analysis. Without stressing the mathematics, there is an easy way to show this. For the two-dimensional LDA model from the first exercise, generate random uniformly distributed data in the appropriate range, and predict the LDA class:

```
> X <- wines[, c("flavonoids", "proline")]
> x <- runif(n = 2000, min = min(X[,1]), max = max(X[,1]))
> y <- runif(n = 2000, min = min(X[,2]), max = max(X[,2]))
> newX <- data.frame(flavonoids = x, proline = y)
> new.predictions <- predict(wine.lda, newdata = newX)$class
```

Plot the new predictions¹ and colour them according to the predicted class. Notice the shape of the class boundaries. See?

¹You can convert class labels to integers which can be used for colouring in the following way: `as.integer(factor(new.predictions))`.

Exercise 4.1.5 Quadratic discriminant analysis (QDA) is based on the same principles as LDA. The difference is that the covariance matrices of the individual classes are now allowed to vary, in contrast to LDA where all classes are supposed to have the same covariance matrix.

Function `qda` does exactly what you would expect; use it to find predictions for the wines data in two dimensions as well as in 13 dimensions. How many objects are predicted incorrectly? Is that better or worse than LDA?

Which objects are predicted with the highest uncertainty? Any differences to the LDA predictions here?

Visualise the boundaries of the two-dimensional QDA model, just like you did for LDA in the previous exercise. Are you surprised?

Exercise 4.1.6 When the number of objects is relatively small, and the number of variables relatively large, QDA – and even LDA – will choke: the necessary matrix inversions are no longer possible. One approach is to do data reduction *before* the discriminant analysis step. That is, you perform LDA (or QDA) on the PC scores.

Do this for the wine data. First find a number of PCs that does not throw away too much information, and next apply LDA and QDA. Compare the results with your previous results. Do you see any differences?

What about scaling, now? Do you think it matters? Why (not)?

Exercise 4.1.7 An important question is how well future data can be predicted. We can simulate this by setting apart a fraction of the data, say, one-third, to be used as a test set. We use the remainder – the training set – to build the model, and evaluate the predictive abilities on the test set.

Do this for the wine data: randomly (this is important!) select one-third of the wine samples, and predict their class, based on the LDA model from the training set.

Repeat this exercise several times: is the expected error the same in every case?

Compare the prediction errors (and the variation in prediction errors) for QDA and LDA. Which one is better? What does “better” mean, anyway?

Exercise 4.1.8 A formalisation of the procedure in the previous exercise is called crossvalidation. One repeatedly generates test sets of a certain size until every object has been present in the test set exactly once. The prediction errors for objects in the test sets can be used to calculate the overall crossvalidation error, which is a reasonable estimate of prediction error.

Create leave-ten-percent-out crossvalidation estimates for the `yeast.alpha` data², using LDA. Repeat this several times, for different leave-out schemes.

Leaving *one* object out at a time is taking crossvalidation to the limit. R functions `lda` and `qda` have this built-in; just set the argument `CV = TRUE`.

Compare your ten-percent crossvalidation errors to the leave-one-out (LOO) crossvalidation error. Which of the two is more stable, do you think?

LOO errors are the same, when repeatedly executed, whereas ten-percent CV estimates can differ. Can you explain this?

Exercise 4.1.9 One way to get more complicated decision boundaries than in LDA or QDA, is to describe each class separately with a mixture of gaussians. Using the formalism already encountered in the context of model-based clustering, one can pick the number of clusters and the set of constraints that optimises a measure like BIC. We will call this model-based discriminant analysis (MBDA). This is implemented in the R function `mclustDA`:

```
> X <- scale(wines)
> training <- sample(nrow(X), 100)
> testing <- (1:nrow(X))[-training]
> wines.MDA <- mclustDA(train = list(data = X[training,],
+                                   labels = wine.classes[training]),
+                       test = list(data = X[testing,],
+                                   labels = wine.classes[testing]),
+                       G = 2)
> wines.MDA
```

The last line shows a summary of the result, including error rates for training and test sets.

Experiment with the flexibility provided by the `G` argument. Does this parameter influence the quality of the predictions?

Which model corresponds to LDA? Which to QDA?

Extra exercise Investigate whether QDA is insensitive to mean-centering and/or autoscaling.

Extra exercise Investigate whether MBDA is insensitive to mean-centering and/or autoscaling.

²Just to make life a *bit* more interesting...

Extra exercise Make a classification model for the microarray data available from

```
> data(golub)
```

Read the manual page for more information.

Compare the prediction errors for the classification methods LDA, QDA and MBDA; do you need to do a dimension reduction before building the classification model?

4.2 Nonparametric classification: k-NN, tree-based methods, and ensemble methods

One may also consider approaches that do not rely on distributional assumptions. The most extreme case is to rely solely on the examples in the training set: a new object is assigned to the class of the most similar object in the training data. This is called Nearest Neighbour classification. When one not only looks at the single most similar object, but a limited set of k most similar objects, the procedure is called k -nearest-neighbours (k -NN). There is no explicit model; in a way, the training set *is* the model.

Other popular approaches are based on binary decision trees. A sequence of splits is set up in such a way that every split maximises the homogeneity of the branches below. Such a procedure has important advantages. It is possible to use both qualitative and quantitative predictors; it nicely combines making a classification model with variable selection, and the final result can easily be interpreted by humans. Moreover, there are elegant ways to circumvent missing values (not treated in this course).

Unfortunately, the predictive power of single tree-based models is usually quite low. Combining several of these shows in many cases an enormous improvement of prediction quality, keeping many of the advantages of the single-tree models. The interpretability of such a “random forest” decreases, the price one has to pay for the better predictions. Note that combining several weak classifiers into one strong classifier is a general procedure, not only applicable to tree-based methods (although it does work rather well there).

Exercise 4.2.1 Apply k -NN to a two-dimensional subset of the *wine* data (e.g., variables 7 and 13 again), and visualise the results for different values of k :

```
> training <- sample(nrow(wines), 100)
> testing <- (1:nrow(wines))[-training]
> test.predictions <- knn(train = wines[training, c(7, 13)],
+                          test = wines[testing, c(7, 13)],
+                          cl = wine.classes[training], k = 3)
> plot(wines[, c(7, 13)], cex = .3, col = wine.classes)
> points(wines[testing, c(7, 13)], col = as.integer(test.predictions))
```

Visualize the class boundaries. See for an example on how to do this Exercise 4.1.4.

Some of the plots will look very different from the earlier LDA and QDA plots. Find out which variables are the cause of this, and explain the phenomenon.

Exercise 4.2.2 To get a feel for the method, apply k -NN to the autoscaled *wine* data. Use either crossvalidation or a division in training and test set to assess the predictive abilities of the method. For leave-one-out crossvalidation, the R function `knn.cv` is available.

Compare the predictions for several values of k : which is optimal in this case?

Exercise 4.2.3 For the *wine* data, select two variables, and use the R function `rpart` to create a tree-based model:

```
> wine.frame <-
+   data.frame(class = factor(wine.classes), wines[, c(7, 13)])
> wine.trees <- rpart(class ~ ., data = wine.frame)
> plot(wine.trees, margin = .1); text(wine.trees)
```

The `rpart` call is an example of the R way to define a formula: it says that the element `class` from `wine.frame` should be modelled with by all other elements (the dot) in the data frame.

Use your model to predict points on a regular grid (you guessed it: `predict.rpart`), and visualize the result. What shape do the class boundaries have?

Do you think mean-centering or autoscaling will influence the predictions?

Exercise 4.2.4 Build `rpart` models for the full, 13-dimensional *wine* data.

Play around with the function parameters, such as `split` (which chooses the criterion that is optimised in individual splits) or `minsplits` (the minimal number of objects in a node, for that node to be considered as a potential split).

```
> wine.tr10 <- rpart(class ~ ., data = wine.frame,
+                   control = list(minsplit = 10))
> wine.trInfo <- rpart(class ~ ., data = wine.frame,
+                      parms = list(split = "information"))
```

What are the effects of the different settings?

Exercise 4.2.5 Use crossvalidation to determine the “optimal” settings – say, for *split* and *minsplit* for the *wine* data. This is conveniently implemented as follows:

```
> wine.frame <- data.frame(class = wine.classes, wines)
> wine.rpart <- rpart(class ~ ., data = wine.frame)
> predictions <- xpred.rpart(wine.rpart)
> sum(diag(table(wine.classes, predictions[, ncol(predictions)])))
```

The object *predictions* contains, for all splits, the estimated class labels. The final split therefore corresponds to the actual prediction by the complete tree.

Exercise 4.2.6 It is not too difficult to grow a tree where every leaf contains only objects of one class: in the limit every leaf would only contain one object, period! You can obtain such a tree as follows:

```
> cntrl <- list(cp = 0, minbucket = 1, minsplit = 2)
> wine.tree <- rpart(class ~ ., data = wine.frame, control = cntrl)
```

Find out what the effect of the extra parameters *cp*, *minbucket* and *minsplit* is.

Two mechanisms are commonly employed to prevent this situation. First, one can prevent certain nodes from splitting, e.g. when there are too few objects in that node. This leads to a tree which is not as big as it could be. And secondly, given a big tree, one can “prune” away leaves that do not contribute significantly to the overall classification:

```
> pruned.tree <- prune(wine.tree, cp = .2)
```

Visualize the full and pruned trees. Assess their prediction capabilities by using separate training and test sets, or do a crossvalidation. What do you expect? Can you see it in the results as well?

Exercise 4.2.7 Trees have many advantages: they are simple, easily understood, flexible, generally applicable, etcetera. They have one quite big disadvantage: they suck (*excusez le mot*). Now, there are different ways in which to suck (*e.l.m.*). The particular way in which trees fail is that they are so variable: given a slightly different training set, they may construct a quite different model. Hence, their predictions are sometimes way off.

The idea now is that many trees put together (a *forest*) will – on average – give much less variable answers. Indeed, this turns out to be true. The technique “Random Forests” exploits this property, where every single tree is fit on a so-called *bootstrap data set*. Such a set is obtained by drawing – with replacement – objects from

the training data; the left-over objects (often termed out-of-bag (oob) objects) are predicted. Thus, the method gives a kind of cross-validated predictions for free.

One particular feature of this method is that at every split, only a part of all possible variables is considered. This brings in extra variability, which improves the predictive properties of the method.

The call is simple:

```
> wine.frame <- data.frame(class = factor(wine.classes), wines)
> wine.rf <- randomForest(class ~ ., data = wine.frame)
> wine.rf$confusion
```

Apply the *randomForest* method (using default parameter settings) to the *wine* data several times; check the performance, and visualize which objects are misclassified. Is it better than *rpart* classification?

A very attractive property of the *randomForest* procedure is that it can give information on what variables are important:

```
> wine.rf <- randomForest(class ~ ., data = wine.frame,
+                          ntree = 1000, importance = TRUE)
> varImpPlot(wine.rf, type = 2)
```

Does the result correspond with the variables that are usually selected in *rpart* models?

Extra exercise Study the variability in *rpart* trees in crossvalidation. That is, divide the wine data set in (say) ten sections, and leave out each section once, while making a tree-based model on the other nine sections.

Display the ten resulting models. Are they very different? Do you think ten individual *randomForest* models would be more different? Why would this be useful?

Extra exercise Apply both *rpart* and *randomForest* models to the *golub* data.

What variables are used in both cases?

Which of the two techniques is better in prediction of the test set?

4.4 Support Vector Machines

Support Vector Machines (SVMs) have emerged as a powerful modelling technique for classification as well as regression purposes. However, due to the explicit flexibility of the SVM, some vital model parameters need to be selected (e.g., the so-called kernel function and its parameter(s), and a penalty weight) and optimized to achieve a good generalization performance.

Exercise 4.4.1 In this exercise you will investigate the influence of scaling of the input data. Because the kernel matrix for SVMs is based on a covariance or distance matrix the choice of the scaling of the input data might change the classification model considerably. Consider two cases for the *wine data*: mean-centered data and autoscaled data.

Create a training and test set for the wine data, considering only variables 7 and 13; note that the class information has to be presented as a *factor* object. The default for the *svm* function in R is to apply autoscaling. This can be prevented by the *scale* argument:

```
> y <- as.factor(wine.classes)
> ytrain <- y[training]
> ytest <- y[testing]
> model <- svm(xtrain, ytrain, scale = FALSE)
> reprod <- predict(model, xtrain)
> pred <- predict(model, xtest)
```

Check the confusion tables for the reproduction of the training set, and the prediction of the test set. Plot the classification result for the wine data. Are there differences between the classification result of the training and test set?

Do you think there will be differences between mean-centered data and raw data? Why (not)?

Repeat this exercise for the autoscaled data. Note that for scaling the test set, the means (and standard deviations) of the *training* set should be used!

The bad results of mean-centered data are caused by the large distances, that by application of the radial-basis kernel function lead to uninformative values. Apart from scaling, one can counter this by tweaking the parameters of the kernel function. Try this, for instance:

```
> model <- svm(xtrain, ytrain, gamma = 0.0001, scale = FALSE)
> reprod <- predict(model, xtrain)
> pred <- predict(model, xtest)
```

Can you explain what you see?

Exercise 4.4.2 Repeat the previous exercise, now by taking other combinations of the autoscaled variables of the *wine* data into account. Which combinations of variables lead to good classification?

Inspect for several pairs of variables the shape of the class decision boundaries.

Exercise 4.4.3 Again build an SVM classification model for variables 7 and 13 of the *wine* data. Visualize the performance for the training and test set in the variable space. Visualize for the training set the support vectors, and indicate which objects are wrongly classified in both training and test sets.

Repeat this exercise for other combinations of two variables. Is there a relation between the quality of the prediction and the number of support vectors?

Exercise 4.4.4 Use the autoscaled variables 7 and 13 of the *wine data*, and construct an SVM classifier with a linear kernel function:

```
model.lin <- svm(xtrain, ytrain, scale = FALSE, kernel = "linear")
```

Calculate the confusion tables and the overall model accuracies. Show the classification result in the variable space, as well as the class boundaries.

Repeat this exercise for polynomial, radial basis function and sigmoidal kernel functions. Feel free to vary the kernel function parameter(s).

Here is an example to select a polynomial kernel function and its parameters *coef0* (the offset value), *gamma* (a multiplication constant) and *degree* (the degree of the polynomial function):

```
Model.poly <- svm(xtrain, ytrain, kernel = "polynomial",
+               coef0 = 1, gamma = 0.5, degree = 2)
```

Which kernel function and parameter setting yields the best reproduction and prediction accuracy for the *wine* data set?

Exercise 4.4.5 An SVM classifier tries to fit a line, including two parallel margins, to separate two classes in feature space. But what should be done if the two classes are overlapping? In that case, no separating line exists. To overcome this, one can assign a penalty – a *cost* – to the “bad” objects causing the class overlap.

Generate the following model with the *cost* parameter set to 5:

```
model.C <- svm(xtrain, ytrain, cost=5)
```

Calculate the confusion tables, and model accuracies for training and test sets, and visualize the classification. Visualize the model performance and the class boundaries in the variable space.

Repeat this exercise a few times for other penalty values. Which *cost* values yield good and (very) bad classification models?

Exercise 4.4.6 Instead of finding the optimal kernel parameter *gamma* (for the radial basis kernel function) and the SVM *cost* parameter by trial and error, one can also perform a grid search. To do this, define a number of possible *gamma* and *cost* values:

```
GamVal <- seq(0.1, 2, by=0.2)
CostVal <- seq(1, 100, by=20)
```

One should use three data sets, one for building the SVM model for each parameter combination, and the second to test the performance of the model for selecting the optimal settings. The third set is the independent validation set, to estimate the prediction error. Note that we now use all 13 variables of the *wine* data set!

Function *OptimSVM* then can be used to assess the quality of all models:

```
OptMod <- OptimSVM(xtrain, ytrain, xtest, ytest, GamVal, CostVal)
```

How many models are generated to find the optimal one? Which kernel type is used?

The optimal SVM model can be validated by rebuilding the model for the training and test set together (!), using the optimized parameter settings, and predicting the validation set. Show the results for the validation set numerically as well as graphically.

The result, *OptMod*, also contains two matrices, giving information on the number of support vectors, and the accuracy for the internal test set, respectively, as a function of the parameters *gamma* and *cost*. Display the content of these matrices as line plots and images. What do you conclude regarding the predefined parameter ranges?

How are the number of support vectors and the prediction accuracy related to the parameters *gamma* and *cost*? Which combinations of parameters yield the best results? Which are worst?

Finally, visualize the reproduction (training and internal test set) and the prediction (external validation set) abilities of the model. To do this, apply a PCA analysis and consider – for the visualization – only the first and the second PC.

Extra exercise Repeat the previous exercise for the *golub* data.

Appendix A

R packages used in PRiNS

Table A.1: R functions and data sets within the PRiNS package.

Name	Type
<i>PCA</i>	function
<i>scores</i>	function
<i>loadings</i>	function
<i>variances</i>	function
<i>screeplot</i>	function
<i>reconstruct</i>	function
<i>scoreplot</i>	function
<i>loadingplot</i>	function
<i>biplot.PCA</i>	function
<i>OptimSVM</i>	function
<i>golub</i>	data set
<i>nir.spectra</i>	data set
<i>pnts</i>	two data sets
<i>spheres</i>	three data sets
<i>winesLight</i>	three data sets
<i>yeast.alpha</i>	data set

Table A.2: External R packages, available from CRAN, used in the PRiNS package.

Package	Name	Type
fastICA	<i>fastICA</i>	function
MASS	<i>sammon</i>	function
	<i>isoMDS</i>	function
	<i>lda</i>	function
	<i>predict.lda</i>	function
	<i>plot.lda</i>	function
	<i>qda</i>	function
	<i>predict.qda</i>	function
kohonen	<i>som</i>	function
	<i>predict.kohonen</i>	function
	<i>plot.kohonen</i>	function
	<i>classmat2classvec</i>	function
	<i>classvec2classmat</i>	function
	<i>unit.distances</i>	function
	<i>wines</i>	data set
cluster	<i>pam</i>	function
mclust	<i>mclustBIC</i>	function
	<i>plot.mclustBIC</i>	function
	<i>summary.mclustBIC</i>	function
	<i>Mclust</i>	function
	<i>plot.Mclust</i>	function
	<i>coordProj</i>	function
	<i>mclustDAttrain</i>	function
	<i>mclustDAtest</i>	function
class	<i>knn</i>	function
	<i>knn.cv</i>	function
rpart	<i>rpart</i>	function
	<i>predict.rpart</i>	function
	<i>xpred.rpart</i>	function
	<i>prune</i>	function
randomForest	<i>randomForest</i>	function
	<i>varImpPlot</i>	function
e1071	<i>svm</i>	function
	<i>predict.svm</i>	function
lattice	<i>levelplot</i>	function