Teorikompendium ”Statistical Modelling and Computing” för kursen Multivariat Datanalys och Försöksplanering VT 2015

Kompendiet är skrivet av Dr. Nick Fieller, University of Sheffield, UK och har bla annat används för undervisning vid Tammerfors Universitet i Finland

Statistical Modelling and Computing

0. Introduction

0.1 Books and Websites

Venables, W. N. & Ripley, B. D. (2002) ***Modern Applied Statistics with S****,* (4th Edition), Springer.

This is the main course book. The software (including versions in R) and datasets used in this book are available from various websites such as

http://www.stats.ox.ac.uk/pub/MASS4

This course will use many of the data sets and functions from the MASS library.

Verzani, J. (2005) ***Using R for Introductory Statistics***, Chapman & Hall. This book provides many good examples of the more elementary techniques.

Nolan, D. & Speed, T. P. (2000), ***Stat Labs: Mathematical Statistics Through Applications.*** Springer. Support material is available at:

http://www.stat.Berkeley.edu/users/statlabs

This book is recommended for additional reading.

Ripley, B.D. (1996) ***Pattern Recognition and Neural Networks***. Cambridge University Press.

This book provides much fuller details of neural nets from the practical statistical point of view.

Much of the course will be focused around the computing system R which provides various statistical facilities including high quality graphics. It is an open source system and is available free. It is 'not unlike' the expensive commercial package S-PLUS, the prime difference is that R is command-line driven without the standard menus and dialog boxes in S-PLUS. Otherwise, most code written for the two systems is interchangeable.

The sites from which R and associated software (extensions and libraries) and manuals can be found are listed at

http://www.ci.tuwien.ac.at/R/mirrors.html

The nearest ones are at

http://cran.dk.r-project.org (in Denmark)

and

http://cran.uk.r-project.org (in Bristol, UK)

Free versions of full manuals for R (mostly in PDF format) can be found at any of these mirror sites. There is also a wealth of contributed documentation. Particularly useful are:

Using R for Data Analysis and Graphics by John Maindonald (PDF [702kB], 106 pages). Many of the topics in this course are covered in these notes. This is also available as a hardback book.

R for Beginners by Emmanuel Paradis, (PDF [152kB], 31 pages). This provides a useful introduction to R. The notes are translated from the original version in French (but not always very accurately).

R reference card by Jonathan Baron, (PDF [58kB], LaTeX [5kB], 1 page)

These can be consulted online during R sessions or downloaded and printed to take away.

0.2 Objectives

The overall objective of this course is to provide an introduction to some of the techniques of modern statistical methodology. An integral part of modern statistical analysis is directed towards understanding data, discovering structure in it and making inferences about the wider world. Applied Statistics is not a subset of mathematics, though mathematics is a useful tool in developing statistical methods and techniques, just as it is a useful tool in the various forms of engineering. In some ways, this course regards applied statistics as '*data engineering*' — this includes actually doing practical things with data. Inevitably, some attention has to be given to the computational side and there will be some pointers to the mathematical aspects.

A great revolution in statistical practice occurred with the development of the language S and later the development of S-PLUS.

(R is essentially the same language as S-PLUS but is free)

This integrated computing system has allowed the statistical community to extend traditional methods and to try out new techniques to provide new ways of investigating practical statistical problems. Often these are based not on mathematical development but on more intuitive ideas. This course aims to give a flavour of this new approach to statistical thinking and an introduction to implementing them in practice.

0.3 Outline of Course

1. Overview of S-PLUS and R:– how does it work and what can it do.
2. Exploratory Data Analysis:– standard summary descriptions and plots, robust summaries, improved alternatives to histograms.
3. Classical Univariate Statistics:– revision and implementation of one and two sample tests, analysis of variance, bootstrap and permutation methods.
4. Linear Statistical Models:– classic linear regression and diagnostics. Robust methods, smooth regression and additive models.
5. Multivariate Methods:– multivariate EDA, principal components and biplots, discrimination and classification, cluster analysis.
6. Tree-based Methods:– Classification and Regression Trees, trees for decision making.
7. Neural Networks:– use for classification and regression problems.

1. Overview of S-PLUS and R

1.0 Introduction

S-PLUS (and its public domain equivalent R) is an integrated suite of software facilities for data analysis and graphical display. It offers:–

- an extensive and coherent set tools for statistics and data analysis
- a language for expressing statistical models and tools for using linear and non-linear statistical models
- graphical facilities for interactive data analysis and display
- an object-orientated programming language that can easily be extended
- an expanding set of publicly available libraries of routines for special analyses

S-PLUS is available as a commercial package from Insightful (formally known as MathSoft) and is an implementation of the language S developed at Bell Laboratories by Becker, Chamberlain and Wilks. R is a very similar implementation but is available free from many different websites. The prime differences between R and S-PLUS (apart from the cost!) are:

- R is an ***Open Source* system** — it is possible to examine the source code and determine precisely what variation on a statistical method has been implemented. This is less important for e.g. t-tests (although even for these there are *equal variance* or *unequal variance* versions of t-tests) but much more important for the more heuristic methods of *robust analysis* and semi-parametric methods, i.e. those modern methods based more on practical consideration than on mathematical theory.
- S-PLUS has *menus and dialogs* as well as a command-line interface, but R has only the command-line.
- S-PLUS has ways to edit graphs and more facilities for multi-panel plots.
- R is better at annotating with mathematical notation.
- R is small with many extensions, S-PLUS is monolithic.
- R runs on less powerful machines.

1.1 Some Features of R

1.1.1 R is a function language

All commands in R are regarded as *functions*, they operate on *arguments*, e.g. `plot(x, y)` plots the vector x against the vector y — that is it produces a scatter plot of x *vs.* y. Even Help is regarded as a function:— to obtain help on the function `plot` use `help(plot)`. To obtain general help use `help()`, i.e.use the function `help` with a null argument. To end a session in R use `quit()`, or `q()`, i.e. the function `quit` or `q` with a null argument. In fact the function `quit` can take optional arguments, type `help(quit)` to find out what the possibilities are.

1.1.2 R is an *object orientated* language

All entities (or 'things') in R are ***objects***. This includes vectors, matrices, data arrays, graphs, functions, and **the results of an analysis.** For example, the set of results from performing a two-sample t-test is regarded as a complete single object. The object can be displayed by typing its name or it can be summarized by the function `summary()`.

1.1.3 R is a *case-sensitive* language

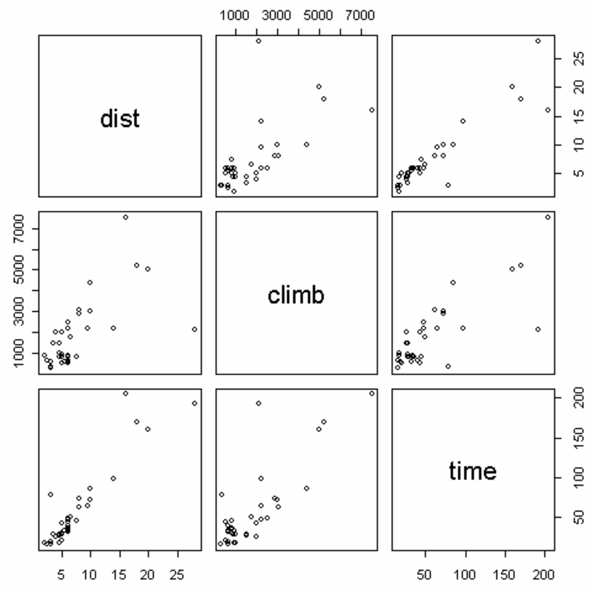
Note that R treats small letters and big letters as different, for example a two sample t-test is performed using the function `t.test()` but R does not recognize `T.test()`, nor `T.TEST()`, nor `t.Test()`, nor

1.1.4 Brief Example

```
R : Copyright 2004, The R Foundation for Statistical
Computing
Version 2.0.1 Patched (2004-11-19), ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain
conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in
publications.

Type 'demo()' for some demos, 'help()' for on-line help,
or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.


> library(MASS)

> data(hills)

> summary(hills)

      dist             climb            time

 Min.   : 2.000   Min.   : 300   Min.   : 15.95

 1st Qu.: 4.500   1st Qu.: 725   1st Qu.: 28.00

 Median : 6.000   Median :1000   Median : 39.75

 Mean   : 7.529   Mean   :1815   Mean   : 57.88

 3rd Qu.: 8.000   3rd Qu.:2200   3rd Qu.: 68.63

 Max.   :28.000   Max.   :7500   Max.   :204.62
```

```
> hills
                 dist climb    time
Greenmantle       2.5   650  16.083
Carnethy          6.0  2500  48.350
Craig Dunain      6.0   900  33.650
Ben Rha           7.5   800  45.600
Ben Lomond        8.0  3070  62.267
Goatfell          8.0  2866  73.217
Bens of Jura     16.0  7500 204.617
Cairnpapple       6.0   800  36.367
Scolty            5.0   800  29.750
Traprain          6.0   650  39.750
Lairig Ghru      28.0  2100 192.667
Dollar            5.0  2000  43.050
Lomonds           9.5  2200  65.000
Cairn Table       6.0   500  44.133
Eildon Two        4.5  1500  26.933
Cairngorm        10.0  3000  72.250
Seven Hills      14.0  2200  98.417
Knock Hill        3.0   350  78.650
Black Hill        4.5  1000  17.417
Creag Beag        5.5   600  32.567
Kildcon Hill      3.0   300  15.950
Meall Ant-Suidhe  3.5  1500  27.900
Half Ben Nevis    6.0  2200  47.633
Cow Hill          2.0   900  17.933
N Berwick Law     3.0   600  18.683
Creag Dubh        4.0  2000  26.217
Burnswark         6.0   800  34.433
Largo Law         5.0   950  28.567
Criffel           6.5  1750  50.500
Acmony            5.0   500  20.950
Ben Nevis        10.0  4400  85.583
Knockfarrel       6.0   600  32.383
Two Breweries    18.0  5200 170.250
Cockleroi         4.5   850  28.100
Moffat Chase     20.0  5000 159.833
>
```

```
> pairs(hills)
```



```
> cor(hills)
           dist     climb      time
dist  1.0000000 0.6523461 0.9195892
climb 0.6523461 1.0000000 0.8052392
time  0.9195892 0.8052392 1.0000000
```

```
> data(shoes)

> shoes

$A

 [1] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3


$B

 [1] 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6


> attach(shoes)

> t.test(A,B)


         Welch Two Sample t-test


data:  A and B

t = -0.3689, df = 17.987, p-value = 0.7165

alternative hypothesis: true difference in means is
not equal to 0

95 percent confidence interval:

 -2.745046  1.925046

sample estimates:

mean of x mean of y

    10.63     11.04
```

```
> T.test(A,B)

Error: couldn't find function "T.test"

> t.test(a,b)

Error in t.test(a, b) : Object "b" not found

> summary(t.test(A,B))

            Length Class  Mode

statistic   1      -none- numeric

parameter   1      -none- numeric

p.value     1      -none- numeric

conf.int    2      -none- numeric

estimate    2      -none- numeric

null.value  1      -none- numeric

alternative 1      -none- character

method      1      -none- character

data.name   1      -none- character

>

>  mean(A)

[1] 10.63

> mean(B)

[1] 11.04
```

1.1.5 Comments on example

- 1:– The first command opened the library of routines and data sets `MASS`. There are many libraries of routines available in R and many can be downloaded from the various R websites listed in §0.1. To find out what libraries are available in your system type `library()` and you will obtain a list of them. To find out what routines are available in [for example] `MASS` type `library(help=MASS)`.

- 2:– The second command `data(hills)` made the data set `hills` available to the session. The base system of R and many of the available libraries come with example data sets for testing routines and for illustrations and `hills` is one of those that come in the library `MASS`. To find out what data sets are currently available to the session type `data()`. It is of course possible to read in data from files, not only ordinary ASCII text files but also files produced by most other packages such as Excel, SAS, SPSS, Minitab, STATA, . In addition data can be typed in direct from the keyboard.

- 3:– `summary(hills)` produced a basic summary of the *object* `hills`. Typing `summary(name-of-object)` will produce some sort of summary whatever type of object it is, though what is produced depends on the type of the object (i.e. whether it is a data set or the results of an analysis or whatever.

- 4:– `hills` produced a complete list of the object hills. Typing `name-of-object` will print it out, whatever sort of object it is. Note that this data set consists of three variables: `dist`, `climb`, `time`, and that the rows are labelled with names. These are the record times in minutes taken for ***hill races*** in Scotland. The distance `(dist)` is in kilometres and `climb` gives the total cumulative height in metres climbed in the race.

- 5:– Note the commands `pairs(hills)` and `cor(hills)` are functions operating on the ***object*** `hills`.

- 6:– Finally, a further data set, `shoes`, is opened. Given are measures of the wear of shoes of materials `A` and `B` for one foot each of ten boys. Illustrated are the results of a Two Sample t-test of `A` *vs* `B` and reminders that R is ***case-sensitive***.

- 7:– In fact, it would be better to do a ***paired t-test*** on these data, since each boy is wearing material `A` on one foot and `B` on the other and since there is likely to be great differences between the different boys but not between the different feet of individual boys. This can be done by the same function `t.test()` on the differences, i.e. `t.test(A-B)`. In fact `t.test()` is an example of a ***generic function*** (as is `summary()`) whose result depends on the type of argument given to it

```
> t.test(A-B)

         One Sample t-test

data:  A - B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of x
    -0.41
```

1.2 Summary so far

- The aim of this course is to give a flavour of recent developments in applied statistics that have been made possible by the development of a computer language **S** (implemented as the commercial package **S-plus** and as the free language **R**).

- It may seem at first as if the course is more about the computer package **R** than about statistics, but have patience — it really is about statistics.

- **R** is an *object-orientated* language providing facilities for manipulating *objects* such as vectors, matrices, data sets, results of analyses as well as inbuilt statistical procedures and integrated (and interactive) graphical facilities.

- **R** consists of a *base* system supplemented by various *libraries* of routines. Additionally, various standard data sets are included that can be used to illustrate the techniques. The extensive Help System can be used to find out what libraries are available, what each of them contains, what data sets are included and what the data refer to.

- §1.1.4 gives a record of a short **R** session with comments and explanations given in §1.1.5. These contain some key tools for getting started when using the system.

2. Exploratory Data Analysis

2.1 Data Summaries

2.1.1 Introduction

Standard summaries `mean()`, `median()` and `var()` are available for summarizing data. The first two take individual variables as arguments, and the argument for `var()` can be either a single variable or a data matrix. If the latter then a complete variance-covariance matrix is returned. `summary()` will return the minimum, 1st quartile, median, 3rd quartile and maximum, together with the mean. The first five of these are the (0,0.25,0.5,0.75,1) quantiles and can be produced by `quantile()`. This can also be used to produce any arbitrary quantiles by including a vector of the required probabilities:

```
> attach(hills)
> summary(dist)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.500   6.000   7.529   8.000  28.000
> quantile(dist)
  0%  25%  50%  75% 100%
 2.0  4.5  6.0  8.0 28.0
> quantile(dist,c(0.25,0.33,0.4,0.8))
25% 33% 40% 80%
4.5 5.0 5.3 9.6
```

Note the use of `c(0.25,0.33,0.4,0.8)` to ***concatenate (=join together)*** the numbers into a vector.

[The quantiles are obtained by linear interpolation in the ordered sample] However, these summaries (especially `mean()` and `var()`)are ***sensitive to outliers***, i.e. they are not ***robust***.

2.1.2 Robust Summaries

```
> data(chem)

> chem

[1] 2.90 3.10 3.40 3.40 3.70 3.70 2.80 2.50 2.40 2.40
                                                2.70 2.20

[13] 5.28 3.37 3.03 3.03 28.95 3.77 3.40 2.20 3.50
                                            3.60 3.70 3.70
```

The data above are values of 24 determinations of copper in ppm in wholemeal flour. *The* `[1]` *and* `[13]` *indicate that these lines begin with the 1st and 13th element of the object* `chem`.

Note the very large value `28.95`. It is an ***outlier***.

```
> mean(chem)
[1] 4.280
```

The value of the mean is highly influenced by this outlier (it is larger than all but two of the observations).

The sample mean $\bar{x} \to \pm\infty$ if any data value $x_i \to \pm\infty$, whereas the median is hardly affected if any single value of tends to $\pm\infty$.

In fact, the median will not be affected until **50%** of the data are grossly contaminated.

The median is ***resistant to gross errors***, but the mean is not.

The median has a ***breakdown point of 50%***, the mean has a breakdown point of 0%.

A more robust estimate of location is a trimmed mean, i.e. the mean when a percentage of the largest and smallest observations are *trimmed* away from the sample. Specifically, an α-trimmed mean is the mean of the sample after removal of the upper and lower $100\times\alpha$% portions of the sample, i.e. of the middle $1-2\alpha$ part of the distribution.

Example (chemical data above):

```
> mean(chem, trim=0.01)
[1] 4.280417
> mean(chem, trim=0.04)
[1] 4.280417
> mean(chem, trim=0.05)
[1] 3.253636
> mean(chem, trim=0.1)
[1] 3.205
```

Questions:

1. What breakdown point does an α-trimmed mean have?
2. Which observations have been trimmed and why in the four calculations above?
3. What will an 0.5-trimmed mean give?

Other robust estimators of location are ***M-estimators***, see e.g. Venables & Ripley, (1999), but these are not implemented as standard in **R** [yet] but are available in S-plus.

Robust estimators of scale:

Consider the following estimators of scale

1. s, where $s^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2$
2. $\tilde{\sigma} = \sqrt{\frac{\pi}{2}}\frac{1}{n}\sum|x_i - \bar{x}|$
3. IQR=$0.741\times(x_{[3n/4]} - x_{[n/4]})$

   (Inter-Quartile Range)
4. MAD=$1.4826\times\text{median}\{|x_i - \text{median}(x_j)|\}$

   (Median Absolute Deviation)

All of these are [approximately] unbiased estimators of σ (or their squares of σ^2) if the $x_i \sim N(\mu,\sigma^2)$.

Questions:

1. How resistant are these to outliers?, i.e. what are their breakdown points?
2. How can these be calculated in R using functions `mad()`, `sum()`, `mean()`, `median()`, `abs()`?
3. Do we need to use the function `mad()`?

Relative Efficiency: This measures what price is paid in using a robust estimator instead of an alternative one. The relative efficiency of two estimators $\tilde{\theta}$ and $\hat{\theta}$ is RE($\tilde{\theta};\hat{\theta}$)=(variance of $\hat{\theta}$)/(variance of $\tilde{\theta}$) where the variances are calculated for the particular distribution that the sample comes from (***assuming we know what this is***). This will probably depend on the sample size n and we can consider the Asymptotic Relative efficiency as $n\to\infty$

e.g. for Normal data, (1) ARE($\tilde{\sigma}^2;s^2$)=88%, (2) ARE(MAD;s)=37% and ARE(median;mean)=64%.

We can interpret these as saying that for Normal data we need roughly only 37% of the sample size to estimate σ with s to achieve the same precision of estimation as we would have with MAD. This does not look attractive — it is a high price to pay for protection against outliers.

However, these calculations are based on the sample **really** coming from a Normal distribution.

If the data come from a student t-distribution on 5 d.f., t_5, the ARE(median; mean)=96% (not 64%)

If the data come from a Normal distribution with ε% contamination from a Normal with the same mean but 3 times the standard deviation, i.e. from $(1-\varepsilon)N(\mu,\sigma^2)+\varepsilon N(\mu,9\sigma^2)$ then the table of ARE($\tilde{\sigma}^2;s^2$) values is

ε(%)	**ARE($\tilde{\sigma}^2;s^2$)**
0	87.6%
0.1	94.8%
0.2	101.2%
1	144%
5	204%

Thus we can see that $\tilde{\sigma}^2$ is ***robust to model deviation***, i.e. if the data do not come from the Normal model that we have assumed but instead from a slightly different model then this estimator provides good protection.

As well as robust data summaries (and implicitly estimators) we can consider methods of more general statistical analysis that are ***robust*** or ***resistant*** to ***model deviation*** and ***data contamination***.

2.2 Graphical Summaries

2.2.1 Stem-and-leaf plots

Examples:

(1) Scottish hill race data

```
> data(hills)
> dist
 [1] 2.5 6.0 6.0 7.5 8.0 8.0 16.0 6.0 5.0 6.0 28.0
                                  5.0 9.5 6.0 4.5
[16] 10.0 14.0 3.0 4.5 5.5 3.0 3.5 6.0 2.0 3.0 4.0
                                  6.0 5.0 6.5 5.0
[31] 10.0 6.0 18.0 4.5 20.0
```

```
> stem(dist)

The decimal point is 1 digit(s) to the right of the |

  0 | 2333344
  0 | 5555556666666667888
  1 | 0004
  1 | 68
  2 | 0
  2 | 8
```

(2) Durations and intervals between eruptions of Old Faithful.

```
> data(geyser)
> summary(geyser)
    waiting          duration
 Min.   : 43.00   Min.   :0.8333
 1st Qu.: 59.00   1st Qu.:2.0000
 Median : 76.00   Median :4.0000
 Mean   : 72.31   Mean   :3.4608
 3rd Qu.: 83.00   3rd Qu.:4.3833
 Max.   :108.00   Max.   :5.4500
```

```
> stem(duration)

 The decimal point is 1 digit(s) to the left of the |

   8 | 3
  10 |
  12 |
  14 |
  16 | 223370023357778
  18 | 00022223333335557778880022333333555557778
  20 | 00000000000000000000000223578023578
  22 | 0278
  24 | 7807
  26 | 05
  28 | 373
  30 | 00
  32 | 583
  34 | 523
  36 | 00235
  38 | 0277802377
  40 | 00000000000000000000000000000000000000000000000000000023780233355777
  42 | 00222222355555777880233355778888
  44 | 00222255555557777800000233888
  46 | 00002225577778800033357778
  48 | 0033782277788
  50 | 30
  52 | 7
  54 | 5
```

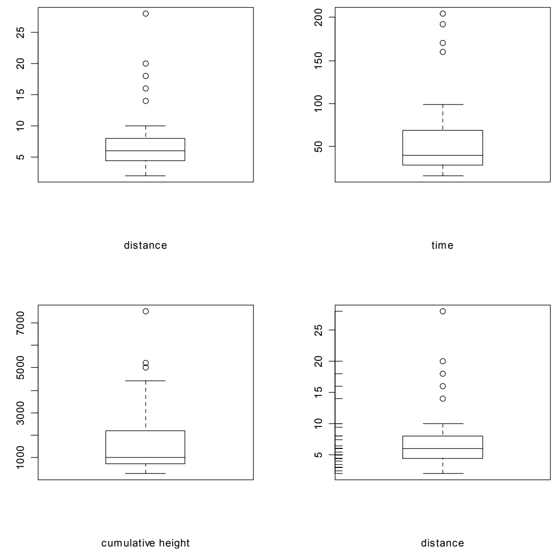
```
> stem(waiting)

The decimal point is 1 digit(s) to the right of the |

   4 | 3
   4 | 577888889999999
   5 | 00000000000011111222223333333444444444
   5 | 5556677777777788888999
   6 | 0000001112222234
   6 | 5555555668889999
   7 | 01111122222223333334444444
   7 | 5555555556666666677777777778888888888888888899999999999
   8 | 00000000000001111111111112222222233333344444444444
   8 | 5555555666667777777777777788888888999999
   9 | 0011222333333334
   9 | 668
  10 |
  10 | 8
```

Comments: Quick, easy, no data are lost — actual values are retained

2.2.2 Boxplots

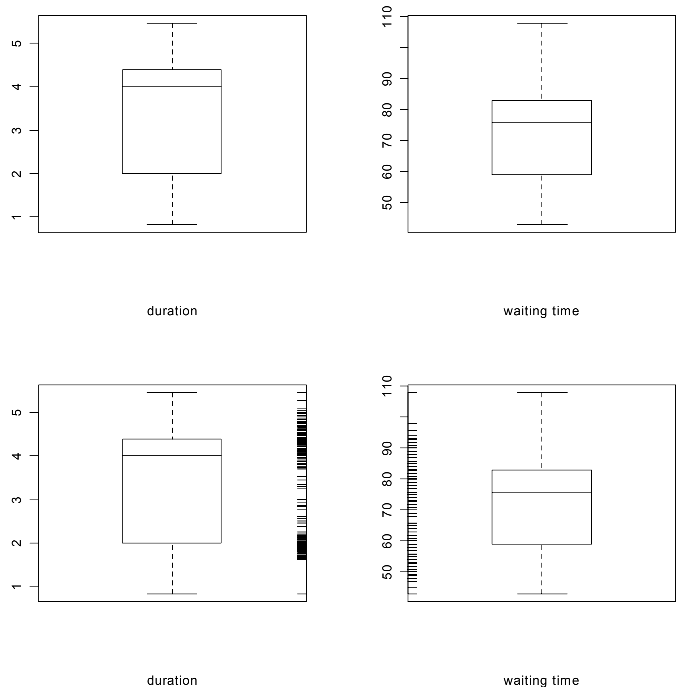
Examples:

```
> data(hills)
> par(mfrow=c(2,2))
> boxplot(dist,sub="distance")
> boxplot(time,sub="time")
> boxplot(climb,sub="cumulative height")
> boxplot(dist,sub="distance")
> rug(dist,side=2)
>
```



Note use of `par(mfrow=c(2,2))` and `rug()`

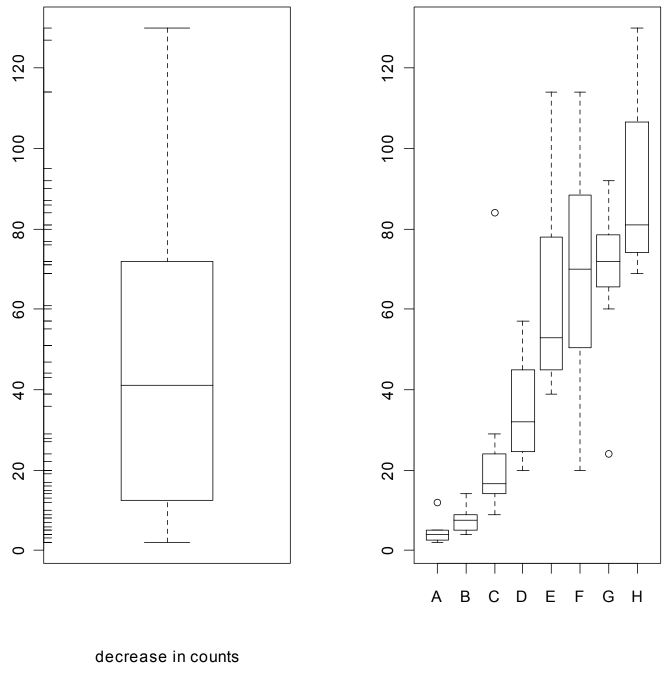
```
> data(geyser)
> boxplot(duration,sub="duration")
> boxplot(waiting,sub="waiting time")
> boxplot(duration,sub="duration")
> rug(duration,side=4)
> boxplot(waiting,sub="waiting time")
> rug(waiting, side=2)
```



Comments: quick summaries for data but may miss gross features, e.g. bimodality, though addition of a *rug-plot* can help. However, most useful for plotting several related data sets for comparison, see example below.

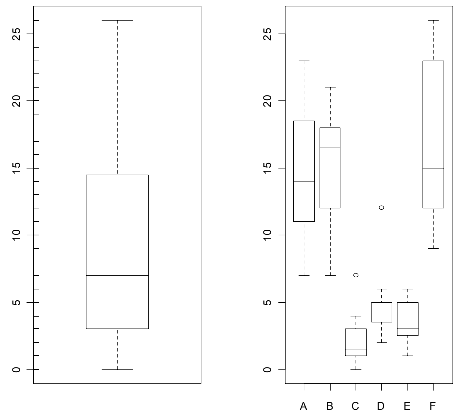
Example: OrchardSpray data give decrease in counts on bees in response to 8 levels of sulphur treatment. The experiment was performed as an 8×8 Latin Square with row and column positions. Here we ignore the Latin Square structure and treat the data as one-way classification example.

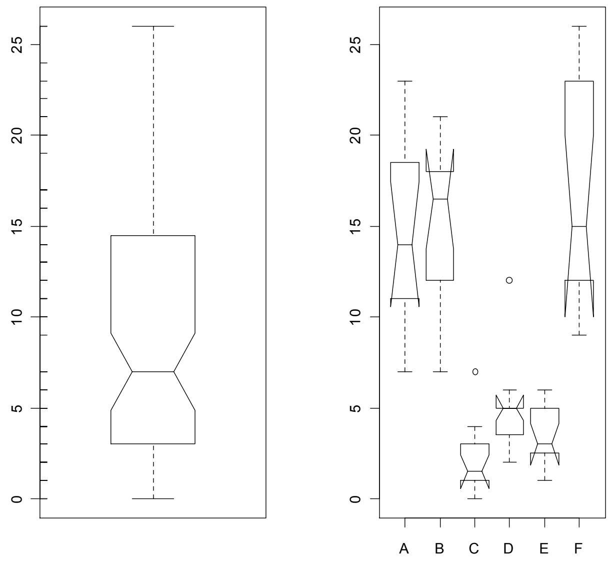
```
> data(OrchardSprays)

> attach(OrchardSprays)

> summary(OrchardSprays)
    decrease           rowpos          colpos        treatment
 Min.   :  2.00   Min.   :1.00   Min.   :1.00   H      : 8
 1st Qu.: 12.75   1st Qu.:2.75   1st Qu.:2.75   G      : 8
 Median : 41.00   Median :4.50   Median :4.50   F      : 8
 Mean   : 45.42   Mean   :4.50   Mean   :4.50   E      : 8
 3rd Qu.: 72.00   3rd Qu.:6.25   3rd Qu.:6.25   D      : 8
 Max.   :130.00   Max.   :8.00   Max.   :8.00   C      : 8

                                                  (Other):16

> par(mfrow=c(1,2))

> boxplot(decrease, sub="decrease in counts")

> rug(decrease,side=2)

> boxplot(decrease~treatment)
```



Example: InsectSprays, similar data to above with six treatments. Note use of the ***logical parameter*** `notch` in the second two plots. This indicates a 'sort of confidence interval' for the median, in the sense that if two notches do not overlap then the medians of those samples are 'significantly different' at the 5% level.

```
> data(InsectSprays)
> attach(InsectSprays)
> boxplot(count)
> rug(count, side=2)
> boxplot(count~spray)
> boxplot(count,notch=TRUE)
> rug(count,side=2)
> boxplot(count~spray,notch=TRUE)
```





Note that the notches may be bigger than the boxes e.g. for spray F, this is likely to happen with small amounts of data.

Question: Why, in this example, is the rug plot not very informative?

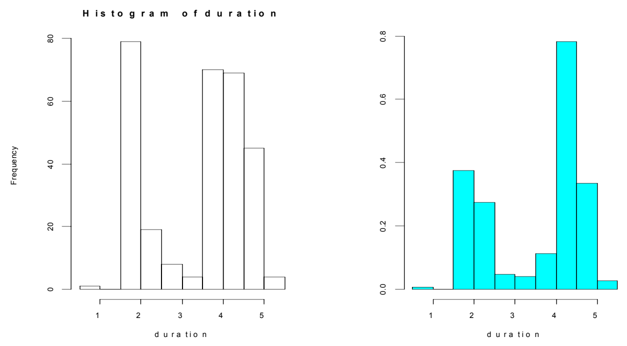
2.2.3 Histograms and density estimation.

2.2.3.1 Histograms

Histograms provide a very simple *density estimate* of the data.

Two functions are useful for drawing histograms, `hist()`(shownon the left below) and `truehist()` (on the right). The first comes from the base library of **R** and by default plots frequencies vertically, the second comes from the MASS library of Venables & Ripley and plots *relative frequencies* vertically,so the total area under the histogram in the second one is 1. Both take many optional arguments controlling the bin width, the number of bins, the class boundaries and it is possible to use unequal bin widths. Type `help(truehist)` to find out more.

```
> data(geyser)
> hist(duration)
> truehist(duration)
```



Question: why are these different (e.g. in range 1.5 to 2.5)?

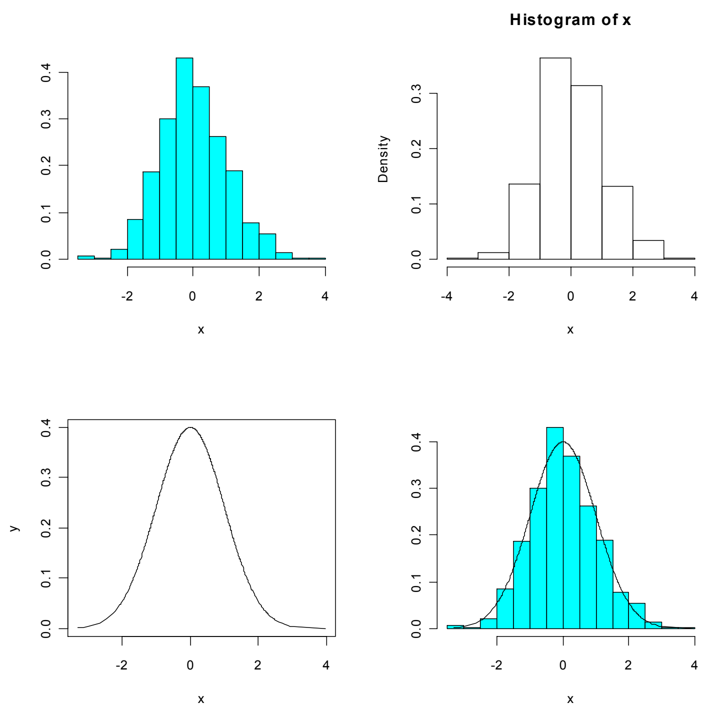
If we think of the data as coming from some density f(.) [i.e. that the data are observations of a random variable with probability density function f(.)] then for any value of x the histogram gives an estimate of f(x),

Specifically, if the class intervals are $c_0,c_1,\ \ ,c_k$ and x is in interval (c_i,c_{i+1}) then the histogram estimate of f(x) is $\tilde{f}(x)=\dfrac{\#(x;c_i \le x < c_{i+1})}{n(c_{i+1}-c_i)}$

If the number of points is large then this will provide quite a good estimate of the true density, but it will depend on the number of bins and the starting values. It is possible to make choices of these based on measures of optimality for sampling from specific distributions, resulting in rules such as *Sturges' formula:* $h=range(x)/(\log_2(n)+1)$ to give the bin width *h* for a sample of *n* observations. Another is *Scott's formula* which gives $h=3.5s(n^{-1/3})$ where *s* is the sample standard deviation or [better] a robust estimate of standard deviation.

The **R** code below produces a histogram of a random sample taken from N(0,1), with superimposed the 'true' density.

```
> x<- rnorm(1000)
> x<- sort(x)
> y<- exp(-x*x/2)/sqrt(2*pi)
> truehist(x)
> hist(x,probability=TRUE)
> plot(x,y,type='l')
> truehist(x)
> lines(x,y,type='l')
```



[Asides: Note the use of the **assign operator** <− which assigns names to objects. Note also the use of `lines()` to add lines to an existing plot (the most recent one), just as `rug()` does.]

2.2.3.2 Kernel Density Estimates

Definition: If we have data $x_1, x_2, \quad x_n$ which are observations of a density f(.) and if K(.) is any probability density function then the ***Kernel Density Estimate*** of f(x), with kernel K(.) and bandwidth b is given by

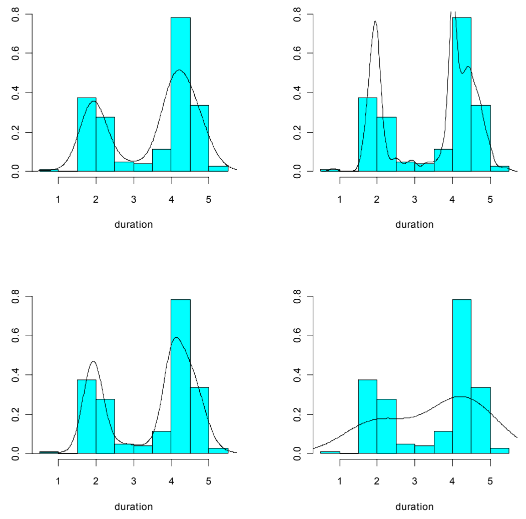
$$\hat{f}(x) = \frac{1}{nb}\sum_{j=1}^{n} K\left(\frac{x - x_j}{b}\right)$$

[It is easy to check that this is a genuine probability density provided that K(.) is, i.e. $\hat{f}(x) \geq 0$ for all x and $\int f(t)dt = 1$]

The ***smoothing parameter*** or bandwidth b is open to choice and is similar to the bin width in histograms. If b is small then the kernel estimate is very rough, if it is large then the estimate is smooth. Similar arguments to choosing the bin width for histograms can be used to show that the best bandwidth is proportional to $n^{-1/5}$ with the constant of proportionality dependent both on the kernel used and on the underlying distribution (which you are trying to estimate of course).

A common choice of kernel function is the standard Normal or Gaussian, i.e. $f(x) = (2\pi)^{-½}\exp(-½x^2)$ but other choices are available (e.g. rectangular, triangular and Epanechnikov) and there are various theoretical results available for choosing them.

```
> library(MASS)
> data(geyser)
> attach(geyser)
> par(mfrow=c(2,2))
> truehist(duration)
> lines(density(duration))
> truehist(duration)
> lines(density(duration, adjust=0.3))
> truehist(duration)
> lines(density(duration,adjust=0.7))
> truehist(duration)
> lines(density(duration, adjust=2.5))
```



Kernel density estimates of Old Faithful data with default, 0.3×default, 0.7×default and 2.5×default bandwidths.

Comments: Kernel density estimates are an easy and attractive alternative or additional tool to histograms. Although you have to choose the bandwidth, as you do in histograms, they do not depend upon choices of starting values of class intervals nor upon whether you regard the classes as open or closed on the left/right.

A more important reason for considering them is that they can be used in more sophisticated methods, e.g. in problems of testing for mixtures of distributions the minimum value of the bandwidth (b_{crit} say) for which the data is unimodal can be used as a test statistic for bimodality.

2.2.3.3 Two Dimensional Kernel Density Estimates

Extensions to two dimensions (and more) are straightforward. In **R** they can be calculated using functions `kde2d()`, and displayed using `contour()` and `persp()`.

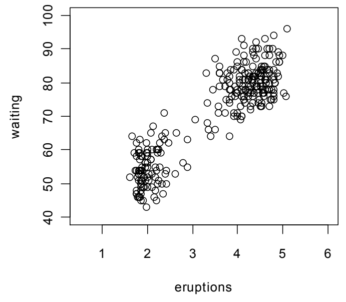
The two dimensional kernel density estimate is defined by

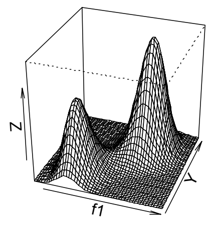
$$\hat{f}(x,y)=\frac{\sum_i \phi((x-x_i)/h_x)\phi((y-y_i)/h_y)}{nh_xh_y}$$ where $\phi(.)$ is a probability density function (e.g. the standard normal) and h_x, h_y are the ***two*** bandwidths.

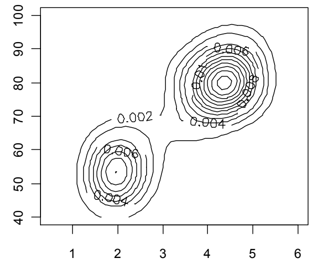
Example: (the data set `faithful` is in the base library and is the same as `geyser` but with a different variable name)

```
> data(faithful)
> attach(faithful)
> summary(faithful)

   eruptions         waiting
 Min.   :1.600   Min.   :43.0
 1st Qu.:2.163   1st Qu.:58.0
 Median :4.000   Median :76.0
 Mean   :3.488   Mean   :70.9
 3rd Qu.:4.454   3rd Qu.:82.0
 Max.   :5.100   Max.   :96.0
```

```
> plot(eruptions,waiting,xlim=c(0.5,6),ylim=c(40,100))
> f1 <- kde2d(eruptions, waiting, n=50, lims=c(0.5,6,40,100))
>  persp(f1, phi=30, theta=20, d=5)
> contour(f1)
```







It is possible to choose the angle of view in the perspective drawing, the levels of contours plotted, put labels on the axes etc, etc, .

2.2.4 Choice of bandwidth:

The theoretical optimal choice of bandwidth depends on what the true density is that we are estimating. However, we do not know what this is (which is why we are estimating it). However, we do have an estimate of the density (!!), provided of course we know what the optimal bandwidth is. Can we use this somehow?

Yes, by using ***cross-validation.*** The idea is to leave one observation out and then estimate the density using the other n–1 observations and compare the estimate with the observation left out in some way. Then we do this again, leaving out the next observation, and then the next. We then choose the bandwidth *b* to make the match as good as possible.

Specifically, in this case we choose b to minimize

$\mathrm{UCV}(b)=\int\hat{f}(x;b)^2dx-\frac{2}{n}\sum_{i=1}^{n}\hat{f}(x_i;b)_{-i}$ where $\hat{f}(x_i;b)_{-i}$ is the kernel density estimate based on the n–1 observations leaving out x_i .

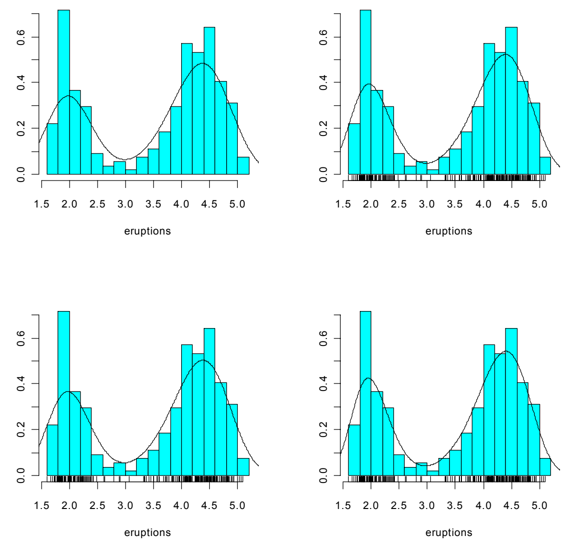
The idea of cross-validation is used in many different contexts in statistical analysis.

2.2.5 Another use of kernel density estimates:

Suppose we want to estimate the median of a set of data (e.g. of the geyser eruptions). Obviously the sample median is a sensible estimate but how do obtain it's standard error? It can be shown that in large samples, the median of a sample from f(.) with true median *m* is asymptotically Normally distributed N(*m,* 1/{4n[f(*m*)]^2}). So, the standard error depends upon the value of the density at the median. This can be estimated from the kernel density estimate.

Example:

```
> library(MASS)
> data(faithful)
> attach(faithful)
> summary(faithful)
> median(eruptions)
[1] 4
> truehist(eruptions, nbins=15)
> lines(density(eruptions))
> truehist(eruptions, nbins=15)
> lines(density(eruptions,adjust=0.8))
> rug(eruptions)
> truehist(eruptions, nbins=15)
> lines(density(eruptions,adjust=0.9))
> rug(eruptions)
> truehist(eruptions, nbins=15)
> lines(density(eruptions,adjust=0.7))
> rug(eruptions)
> density(eruptions,n=1,from=3.99, to=4.01)$y
[1] 0.3808035
> density(eruptions,n=1,from=3.99, to=4.01,adjust=0.9)$y
[1] 0.3858891
> density(eruptions,n=1,from=3.99, to=4.01,adjust=0.8)$y
[1] 0.3901167
> density(eruptions,n=1,from=3.99, to=4.01,adjust=0.7)$y
[1] 0.3937933
```



Note that we first found that the sample median was 4. Then investigation of the kernel density estimates suggested that the default choice of bandwidth was a little too large, so try a few other values slightly smaller. Then note use of `density` with `n=1`, to ensure only one value calculated, over a range around the sample median and also note the use of `density(…….)$y` to extract the y-coordinate.

Conclusion, $\hat{f}(m) \simeq 0.39$ so standard error of the estimate 4.0 of the median is $(4\times 272\times 0.39^2)^{-½}$ =0.078

2.3 Summary

The key ideas introduced here have been problems of

- ways of summarizing and displaying data, perhaps informally
- **robustness** & **resistance** to **model deviation** and **data contamination**
- **kernel density estimates** and their use for a variety of problems
- idea of **cross-validation**

These ideas are especially useful because they allow us to examine assumptions made in statistical analyses and they provide a starting point for developing methods which are not so sensitive to failures in assumptions.

3. Classical Univariate Statistics

3.1. Standard tests

Standard one– and two–sample Normal theory and non-parametric classical univariate tests are readily available in **R** and S-plus.

Many of these are *generic* functions and what is returned depends on the context, i.e. whether it is a one-sample or two-sample test depends on whether you give the function `t.test()` the names of one or two samples.

Example: (Data shoes in MASS library but note how to enter the data direct into vectors `A` and `B`)

```
> data(shoes)
> shoes
$A
 [1] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
$B
 [1] 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6
```

Or enter the data directly:

```
> A<- c(13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3)
> B<- c(14.0, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6)
```

```
> t.test(A,B)

         Welch Two Sample t-test

data:  A and B
t = -0.3689, df = 17.987, p-value = 0.7165
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -2.745046  1.925046
sample estimates:
mean of x mean of y
    10.63     11.04
```

```
> t.test(A-B)

         One Sample t-test

data:  A - B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of x
    -0.41

> t.test(A,B,paired=TRUE)

         Paired t-test

data:  A and B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of the differences
                  -0.41
```

The full list of tests available is

```
binom.test     chisq.test     cor.test         fisher.test

friedman.test  kruskal.test   mantelhaen.test  mcnemar.test

prop.test      t.test         var.test         wilcox.test

chisq.gof      ks.gof
```

And details can be found in `help()`.

Note that the results of each of these functions is ***an object*** and individual elements of these objects can be accessed separately by using a $ sign with `name-of-ofbject$name-of-element`:

```
> t.test(A-B)$p.value
[1] 0.00853878
```

```
> t.test(A-B)$conf.int
[1] -0.6869539 -0.1330461
attr(,"conf.level")
[1] 0.95
```

Again, a list of elements of each of these tests is given in the help system.

Some of these tests depend upon assumptions on the underlying distribution of the data and others do not. For example the t-test presumes data are normally distributed but the non-parametric Wilcoxon does not. Both of them can test whether the measures of location are the same for two samples or whether the measure has a specific value (e.g. 0) for one sample, but the t-test works in terms of the ***mean*** as the measure of location and the Wilcoxon uses the ***median*** as the measure.

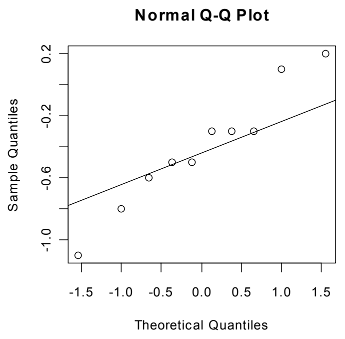
Not only is the median more resistant to outliers but the probability argument used to obtain the p-value is based on combinatorial arguments rather than one assumptions about probability distributions of the data.

Example (shoe data again, paired test):

```
> t.test(A-B)

         One Sample t-test

data:  A - B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of x
    -0.41

> wilcox.test(A-B)

         Wilcoxon  signed  rank  test  with  continuity
correction

data:  A - B
V = 3, p-value = 0.01431
alternative hypothesis: true mu is not equal to 0

Warning message:
Cannot  compute  exact  p-value  with  ties  in:
wilcox.test(A - B)
```

Note that the Wilcoxon returns a larger p-value than the t-test, this is largely because the t-test is assuming more about the data and so you 'get more out of the analysis' (more in ⇒ more out). This is fine provided the assumptions made for the t-test are sensible. It is of course possible to check some of the assumptions (e.g. using a normal probability plot for checking normality) but with small samples it is difficult to detect non-normality.

```
> qqnorm(A-B)
> qqline(A-B)
```



This plot suggests that there are at least doubts about normality for these data.

One solution is to use only 'non-parametric' methods, but even these make some assumptions.

An alternative is to use permutation methods or simulation techniques, some of which come under a general heading of Monte Carlo Methods. **The Bootstrap** is a particular form of simulation technique.

3.2 The Bootstrap

3.2.1. Introduction

Suppose we have estimated the median by the sample median from a set of data and want to know how variable this estimate is. If we knew what the true density was then we could *simulate* more samples from the same density, taking samples of the same size as the one we have, calculate the median of each and then see how variable our answers were in each of these separate simulated samples.

However, we don't know what the true distribution is. So, either we have to estimate it (e.g. fit a normal distribution) or we have to find some other estimate. It might be possible to use a kernel density estimate but then simulating from this might be complicated. The is a much simpler estimate of the distribution and that is ***the sample itself.***

Specifically, if we have a sample $x_1,\ \ ,x_n$ from a distribution F(.) and we calculate the sample distribution function, $F_n(.)$ based on our sample, we can then use $F_n(.)$ directly to generate more samples from our 'best estimate' of the unknown F(.). In fact this is just the same as taking a random sample of size n, ***with replacement***, from our actual data set $x_1,\ \ ,x_n$. This may look very strange but it is a very powerful technique and with the use of the **R** function `sample(….., replace=TRUE)` it can be done very easily.

3.2.2 Simple Simulation

First, we give an illustration of the basic idea of estimation by simulation.

Suppose we take a sample of size 20 from a Normal distribution N(5,2.7^2), i.e. mean 5 and standard deviation, calculate the sample mean and then want to calculate a 95% confidence interval for the true mean. The standard way of doing this is to use classical distributional theory and say the 95% confidence interval is given by

$$\bar{x} \pm t_{19}(0.975)s/\sqrt{n}$$

where s is the sample standard deviation and $t_{19}(0.975)$ is the two-sided 95% point of a t-distribution on 19 d.f. (which is 2.093, but can be calculated directly in **R** as

```
> qt(0.975,19);
[1] 2.093024
```

(here the function is `q`uantile of the `t`-distribution for the `0.975` point for `19` degrees of freedom.)

```
> x<- rnorm(20,mean=5,sd=2.7)
> mean(x)
[1] 4.75921
> var(x)
[1] 7.10922
> confupper<-mean(x)+qt(0.975,19)*sqrt(var(x)/length(x))
> conflower<-mean(x)-qt(0.975,19)*sqrt(var(x)/length(x))
> confinterval<- c(conflower,confupper)
> confinterval
[1] 3.511338 6.007082
```

This gives the 95% confidence interval based on our sample as (3.511,6.007)

However if we could do a practical experiment to see how variable our estimate of the means is. If we 'simulate' our particular sample by taking lots of similar samples from N(5,2.7^2) and calculate the mean of each of them, then we can see experimentally what the range of values they have. We could then estimate a confidence interval by taking a range which includes 95% of our simulated values. We would not have used the sample standard deviation nor the t-distribution, nor any of the mathematical theory involving the t-distribution.

```
> simulate <-numeric(100)
> for (i in 1:100) simulate[i]<-mean(rnorm(20,mean=5,sd=2.7))
> z<-sort(simulate)
> z
  [1] 3.683628 3.876367 3.901581 3.977876 4.059417 4.084968 4.103193 4.127214
  [9] 4.145423 4.151252 4.151807 4.158069 4.197285 4.220176 4.250503 4.393266
 [17] 4.467750 4.500474 4.521293 4.566604 4.574698 4.613527 4.631142 4.652405
 [25] 4.684332 4.696737 4.699455 4.707846 4.720326 4.732183 4.737479 4.754224
 [33] 4.761708 4.828722 4.832087 4.848576 4.873959 4.897021 4.903855 4.919861
 [41] 4.926004 4.936150 4.955231 4.962549 4.982843 4.985068 5.014348 5.025469
 [49] 5.060322 5.068560 5.070019 5.078719 5.081987 5.086046 5.097905 5.134257
 [57] 5.148494 5.156743 5.162649 5.177213 5.184232 5.190236 5.216297 5.245337
 [65] 5.249008 5.276213 5.296429 5.308889 5.310640 5.316523 5.352162 5.357711
 [73] 5.372045 5.376082 5.423826 5.440574 5.448857 5.477199 5.484830 5.511197
 [81] 5.517907 5.585537 5.598260 5.657312 5.659240 5.662326 5.692230 5.709956
 [89] 5.714891 5.859206 5.926859 5.989662 6.009138 6.072194 6.139593 6.217246
 [97] 6.337749 6.383891 6.385220 6.449417
> z[3]
[1] 3.901581
> z[98]
[1] 6.383891
>
```

Then we could say that an approximate 95% confidence interval is given by (3.90, 6.39), — more precisely this is a 96% interval since 96% of our values lie inside it and 4% outside.

Computational notes:

1: note the declaration of the vector `simulate[.]` of length 100 using `numeric(100)`.

2: note the construction of a simple loop with `for (i in 1:100)`, This can be notoriously slow in packages such as **R** and S-plus and advanced programmers would try to replace loops etc by matrix calculations (but I don't intend doing this here).

3: note that we do not need to store all the values in each simulated sample, we just need the mean of them.

4: note the use of `sort(.)`

Difficulty: In this example we 'knew' that our sample came from N(5,2.7^2) and we used this to simulate further samples. Of course we could never really know this and so the best we could do is to simulate from our best guess at the distribution, i.e. N($\bar{x}$,s^2), i.e. N(4.76, 2.67^2) since 4.76 and 2.66 were the mean and standard deviation of our original sample:

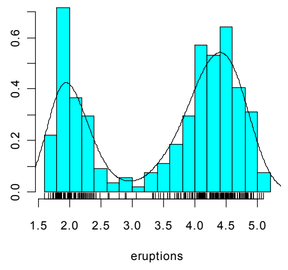
```
> simulate <- numeric(100)
> for(i in 1:100)simulate[i]<-mean(rnorm(20,mean=4.76,sd=2.67))
> z<-sort(simulate)
> z
  [1] 3.396327 3.502508 3.841749 3.870058 3.923347 3.969280 4.050996 4.071518
  [9] 4.110478 4.115113 4.163969 4.182597 4.185243 4.195840 4.277520 4.286212
 [17] 4.289088 4.295636 4.365233 4.374939 4.386417 4.404138 4.417440 4.425339
 [25] 4.460611 4.471175 4.471656 4.501675 4.510588 4.517224 4.528234 4.535420
 [33] 4.551477 4.552511 4.557144 4.557668 4.569633 4.573537 4.578601 4.582115
 [41] 4.583091 4.583715 4.598089 4.599501 4.609323 4.613675 4.666297 4.671486
 [49] 4.671580 4.679536 4.681736 4.721742 4.723026 4.734635 4.738312 4.738548
 [57] 4.796185 4.800435 4.800466 4.874767 4.887359 4.891548 4.893593 4.899882
 [65] 4.908909 4.925140 4.935247 4.964652 4.970336 4.970521 4.992720 5.080825
 [73] 5.081477 5.091885 5.106060 5.110453 5.129278 5.154696 5.159185 5.184879
 [81] 5.197338 5.206724 5.229970 5.256769 5.271497 5.304417 5.348681 5.356202
 [89] 5.364906 5.376544 5.411127 5.441553 5.545112 5.605137 5.680706 5.789276
 [97] 5.855591 5.902711 5.946993 6.147668
> z[3]
[1] 3.841749
> z[98]
[1] 5.902711
```

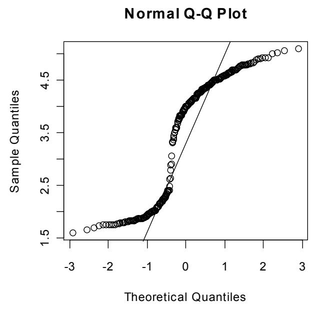
This gives an estimated confidence interval of (3.84, 5.90) — not very different from our previous estimates, in fact slightly closer to the 'true answer' of (3.511,6.007) but this is just an accident.

Difficulty: Although we estimated the mean and variance from our sample, we still ***assumed*** that our data came from a Normal distribution. Of course, we can test this and in the simple case we had above it might seem reasonable, but in other cases it we might know that a Normal distribution was not sensible and we might have no idea of a sensible distribution to use in simulation.

Consider again the problem of estimating the median of the eruption durations of Old Faithful. We have already seen that the distribution is bimodal and so cannot possibly be Normal of any sort but here is how we would check:

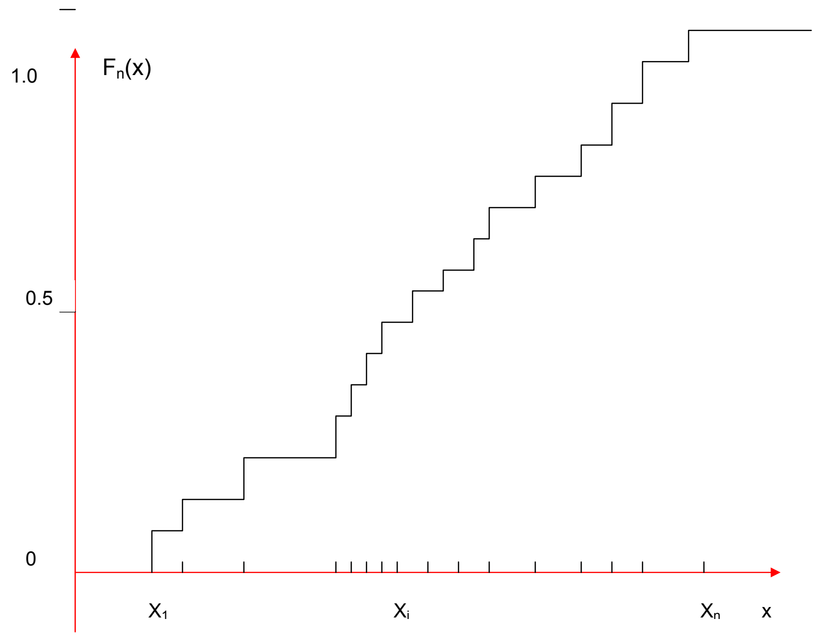
```
> data(faithful)
> attach(faithful)
> par(mfrow=c(2,2))
> library(MASS)
> truehist(eruptions,nbins=15)
> rug(eruptions)
> lines(density(eruptions,adjust=0.7))
> qqnorm(eruptions)
> qqline(eruptions)
```





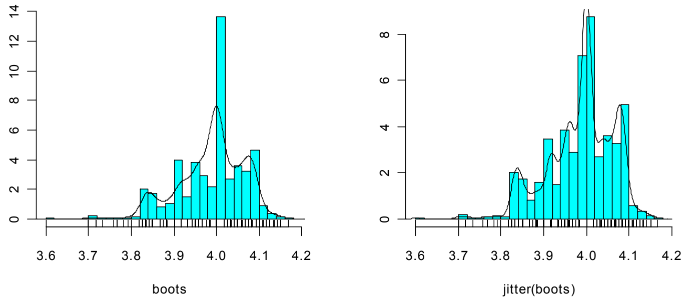
3.2.3 Bootstrap Simulation

We cannot possible pretend that the distribution of the eruption durations is normal so if we want to simulate samples that are like the actual sample we need another distribution. Now the simplest estimate we have of the 'true' distribution of eruption durations is given by the sample itself, i.e. by the sample distribution function $F_n(x)$



If we sample from $F_n(x)$ this is equivalent to taking a sample ***with replacement*** from our original observations.

```
> data(faithful)

> attach(faithful)
> set.seed(137)
> help(numeric)
> boots<- numeric(1000)
> for (i in 1:1000) boots[i]<-
              median(sample(eruptions,replace=TRUE))

> mean(boots-median(eruptions))
[1] -0.013551
> sqrt(var(boots))
[1] 0.07662417
> truehist(boots)
> lines(density(boots))
> rug(boots)
> truehist(jitter(boots))
> lines(density(boots,adjust=0.7))
> rug(jitter(boots))
```



This shows that the bias of the *bootstrap estimate* is –0.0.136 (i.e. quite small) and the estimated standard error is 0.077, quite close to the kernel density estimate of 0.078, which was based in part on a Normal distribution.

Computational notes

1: `Set.seed(137)` chooses the **‘seed’** of the random number generator as 137. This means that I can get precisely the same bootstrap sample again if I set the seed to be 137. If I were to set the seed to another number then I would get a different sample and so a different estimate at the end. For example:

```
> set.seed(731)
> for (i in 1:1000) boots[i]<-
              median(sample(eruptions,replace=T))
> mean(boots-median(eruptions))
[1] -0.0180855
> sqrt(var(boots))
[1] 0.08026924
```

— slightly different but not enough to be of practical importance.

2: Note the use of `jitter` in drawing the histogram, there were clearly problems of observations being exactly on the class boundary (not surprising since we know the median is 4) and the use of the jitter makes the histogram a better *density estimate* — if we just wanted to display the actual data then the use of jitter would not be statistically justified (and we should use stem anyway).

3.2.4 Other Types of Bootstrap

The bootstrap sampling used above took the sample distribution function $F_n(x)$ as an estimate of the 'true' distribution function F(x). This is a very 'rough' estimate and it makes intuitive sense to use a smoother one, i.e. to use a ***Smooth Bootstrap.*** We can do this by adding a small amount to each sampled value, rather like using jitter(.).

The procedure used in the second set of simulations illustrating the simple simulation technique (i.e. when we presumed the underlying distribution was Normal but estimated the mean and variance from our sample) is sometimes known as a ***Parametric Bootstrap***.

The general ideas of bootstrapping are very powerful and very widely used for statistical analyses that do not depend very much on assumptions that are difficult to verify. They are one of the techniques that were initially called ***computer intensive*** but now these are becoming so routine and ordinary that this term is becoming old-fashioned.

3.3 Randomization and Permutation Tests

When W.S. Gosset (who was also known as 'Student') first derived the t-distribution he did not actually work it out as a result of assuming that the original data were Normally distributed but from a different argument.

Consider the problem of comparing the means of two samples A and B. Each of the observations is labelled either A or B. If the null hypothesis that there is no difference between the two is samples is true then these labels are entirely random. This means that the true distribution of a test statistic (such as the t-statistic) could be assessed by considering random re-labelling of each observation.

We could do this by experiment (or a type of simulation) by doing the following:

Step 1: calculate our two-sample test statistic, t_{obs} say.

Step 2: randomly label each observation as either A or B (keeping the sample sizes the same) and then calculate the same test statistic for comparing all the A observations with the B observations, getting a value t_1 say.

Steps 3, 4, , 1001: repeat step 2 for a total of a 1000 times getting a thousand values t_1, t_2, , t_{1000} .

Final step: compare our observed value t_{obs} with the simulated values. If there is no difference between the original samples A and B then the labels are arbitrary and so our t_{obs} will not look unusual amongst the simulated t_1, t_2, , t_{1000}. However, if our value t_{obs} is amongst the most extreme 5% then we would have evidence (at the 5% level) that there really was a difference between the samples. Specifically, if we order the values so that $t_{(1)}<t_{(2)}< \quad <t_{(1000)}$ then we would reject the hypothesis that the two samples were the same if either $t_{obs}<t_{(25)}$ or if $t_{obs}>t_{(975)}$.

What Student showed was that if you consider the theoretical distribution of the randomly re-labelled t-values then this was very well approximated by the 'student t-distribution'. [The mathematics involved much the same approximations and limits as are involved in proving the Central Limit Theorem]. It was only later that it was shown that you could get the same result by assuming that the observations were Normally distributed.

Anyway, it is now easy to perform these tests empirically and so avoid either the assumption of Normality or the inaccuracy of the approximations (whichever approach you use). Many packages (not just **R** and S-plus) now offer this facility for many tests, e.g. in SPSS you will find it under Options and Monte Carlo.

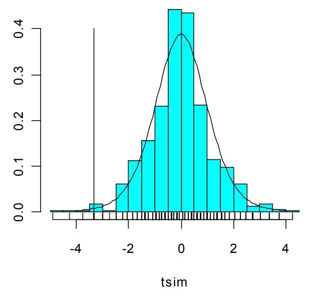
Sometimes, the sample is so small that you can consider all possible relabellings, in which case you just need to calculate the statistics for each labelling once and then the resulting test is know as a ***permutation test.*** Otherwise, it is known as a ***randomization test***.

Example: Paired t-test

Consider the shoes example and consider it is a paired t-test. There are 10 pairs and the paired t-test is just the same as a one-sample test on the differences that the mean is zero. If we randomly relabel each pair as either A-B or B-A then the numerical values of the differences in values stays the same, it is just the sign that is changed (with probability 0.5). In fact there are only 2^{10}=1024 different possibilities so it is practical to consider a permutation test but here we will do it by simulation.

To do this in **R** we will first define a *function* to calculate the t-statistic which is $t_{sim} = \frac{\bar{x}}{\sqrt{var(x)/n}}$. Then to change the signs of the differences randomly we will use the R function `sign()` which is either +1 or –1 according to whether the argument is positive or negative, together with a random Uniform(0,1) number which is generated by the function `runif()`, subtracting 0.5 from it. Note that `sign(runif(10)-0.5)` will produce a vector of length 10 consisting of +1 or –1 with probability 0.5.

```
> data(shoes)
> attach(shoes)
> ttest<- function(x) mean(x)/sqrt(var(x)/length(x))
> d<- A-B
> d
 [1] -0.8 -0.6 -0.3  0.1 -1.1  0.2 -0.3 -0.5 -0.5 -0.3
> ttest(d)
[1] -3.348877
> t.test(A,B,paired=TRUE)

         Paired t-test

data:  A and B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true difference in means is
not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of the differences
                  -0.41

> tsim<- numeric(1000)
> ttest(d)
[1] -3.348877
> for(i in 1:1000)tsim[i]<-ttest(d*sign(runif(10)-0.5))
> truehist(tsim)
> rug(tsim)
> z<- seq(-4,4,0.1)
> lines(z,dt(z,9))
> tobs<-ttest(d)
> markx<- c(tobs,tobs)
> marky<- c(0,0.4)
> lines(markx,marky)
> tsorted<-sort(tsim)
> tsorted[1:10]
 [1] -4.920934 -4.258442 -3.753745 < tobs <-3.348877 -
3.348877 -3.348877 -3.348877
 [8] -3.348877 -3.011905 -3.011905
```



The picture above gives a histogram of the simulated values (and note from the rug plot how few distinct values there are — partly this is because there are only 7 (not 10) distinct values of abs(d).

The vertical line towards the left of the plot marks our observed value of –3.35 and we can see that only **three** of our 1000 simulated values are less than this. We can thus reject the hypothesis that the means are equal at level $2\times3/1000=.006$, (note that the t-approximations gives a level of 0.0085, quite close as can be seen from the density of t_9 superimposed on the histogram).

3.4 Summary

This section has given a brief summary of classical univariate tests. The key ideas introduced are that

- many of these tests rely on assumptions which may be difficult to verify or may in fact be wrong (e.g. for bimodal data)
- tests involving sample means and variances are more at risk than those depending on medians and permutation arguments
- we can **simulate** similar samples to obtain estimates of quantities such as standard errors or p-values
- **bootstrapping** provides a way of making no assumptions about the distribution of the data at all (except independence!)
- **randomization and permutation** tests are easy to do and provide good protection. If they are available in your favourite statistics package (e.g. SPSS) then **USE THEM**, especially for small sample problems such as 2×2 tables and chi-squared tests.

4. Linear and Generalized Linear Models

4.1 Introduction

Linear models relate a response (or dependent variable) to a set of predictors (or independent variables) by a linear expression of unknown parameters which are estimated from the data, i.e. they are termed *linear* because we estimate a *linear function* of the unknown parameters. The actual response may depend in a non-linear way on the predictors, e.g. there may be a polynomial relationship but this can still be expressed as a *linear function of the parameters*.

If the response is a continuous quantitative variable then we may model the response as a Normal random variable with mean depending upon the predictors. The next section provides a brief review and illustration of regression models, including simple regression diagnostics. Ideas of *robust regression* and *bootstrapping* are covered.

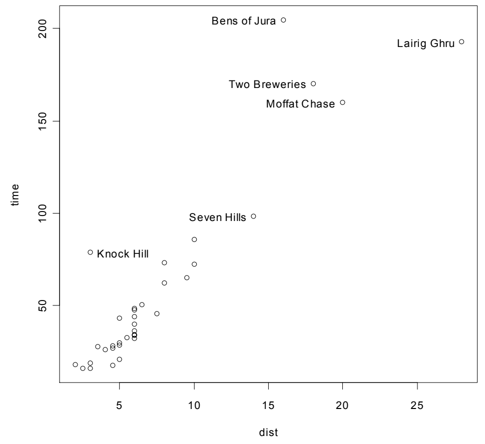
Generalized linear models cover situations where the response is some other measure, e.g. success/failure, and we may then model some other function of the response leading to techniques such as *log-linear* and *logistic regression* which are considered briefly in later sections.

All the methods will be illustrated on specific examples.

4.2 Simple Linear Regression

4.2.1 Example: Scottish Hill Races

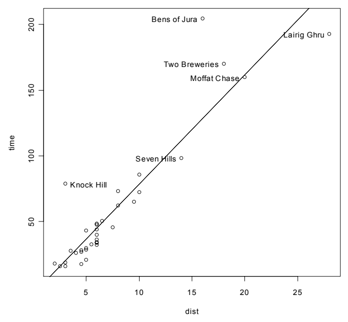
```
> library(MASS)
> data(hills)
> attach(hills)
> plot(dist,time)
> identify(dist,time,row.names(hills))
 [1]  7 11 17 18 33 35
```



Note the use of `identify()` which allowed several of the points to be named interactively with the row names just by pointing at them with the mouse and clicking the left button. Clicking with the right button and choosing stop returns control to the Console Window. It is clear that there are some outliers in the data, notably Knock Hill and Bens of Jura which may cause some trouble.

Next, we fit a simple linear model to relate time to distance, storing the analysis in object `hillslm1`, and add the fitted line to the plot:

```
> hillslm1<- lm(time~dist)
> summary(hillslm1)
Call:
lm(formula = time ~ dist)
Residuals:
    Min      1Q  Median      3Q     Max
-35.745  -9.037  -4.201   2.849  76.170
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -4.8407     5.7562  -0.841    0.406
dist          8.3305     0.6196  13.446    6e-15 ***
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1

Residual standard error: 19.96 on 33 degrees of freedom
Multiple R-Squared: 0.8456,     Adjusted R-squared: 0.841
F-statistic: 180.8 on 1 and 33 degrees of freedom,        p-
value: 6.106e-015

> lines(abline(hillslm1))
>
```



Although this shows a highly significant result for the slope we need to investigate the adequacy of the model further.

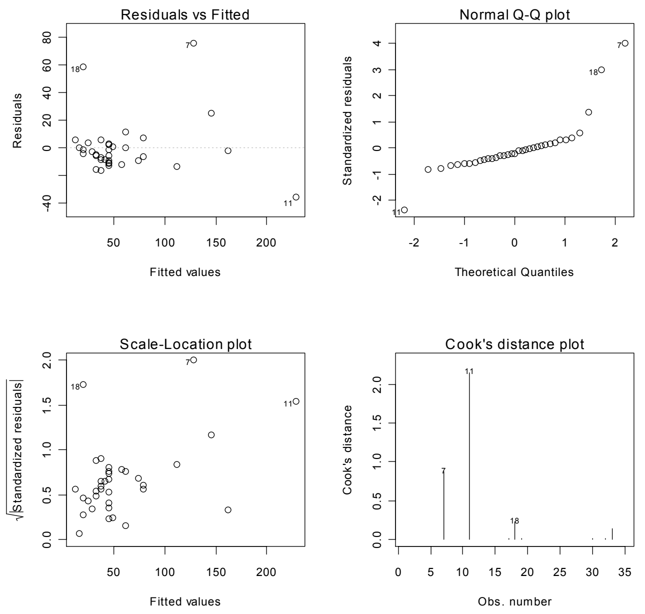
4.2.2 Regression Diagnostics:

The model that we have used is that if the time and distance of the ith race are y_i and x_i respectively then our model is that $y_i=\alpha+\beta x_i+\varepsilon_i$ where the ε_i are independent observations from $N(0,\sigma^2)$. If we look at the ***residuals*** $\hat{\varepsilon}_i$ given by $\hat{\varepsilon}_i = y_i - \hat{y}_i = y_i - \hat{\alpha} - \hat{\beta}x_i$ then these residuals should look as if they are independent observations from $N(0,\sigma^2)$, and further they should be independent of the fitted values $\hat{y}_i$.

A further question of interest is whether any of the observations are ***influential***, that is, do any of the observations greatly affect the estimates of α and β. The way to check this is to leave each observation out of the data in turn and estimate the parameters from the reduced data set. ***Cooks Distance*** is a measure of how much the estimate changes as each observation is dropped.

All of these ***diagnostics*** can be performed graphically using the function `plot.lm()` which take as its argument the results of the `lm()` analysis (which was stored as an object `hillslm1`).

```
> par(mfrow=c(2,2))
> plot.lm(hillslm1)
> hills
> row.names(hills)
 [1] "Greenmantle"   "Carnethy"         "Craig Dunain"    "Ben Rha"
 [5] "Ben Lomond"    "Goatfell"         "Bens of Jura"    "Cairnpapple"
 [9] "Scolty"        "Traprain"         "Lairig Ghru"     "Dollar"
[13] "Lomonds"       "Cairn Table"      "Eildon Two"      "Cairngorm"
[17] "Seven Hills"   "Knock Hill"       "Black Hill"      "Creag Beag"
[21] "Kildcon Hill"  "Meall Ant-Suidhe" "Half Ben Nevis"  "Cow Hill"
[25] "N Berwick Law" "Creag Dubh"       "Burnswark"       "Largo Law"
[29] "Criffel"       "Acmony"           "Ben Nevis"       "Knockfarrel"
[33] "Two Breweries" "Cockleroi"        "Moffat Chase"
```



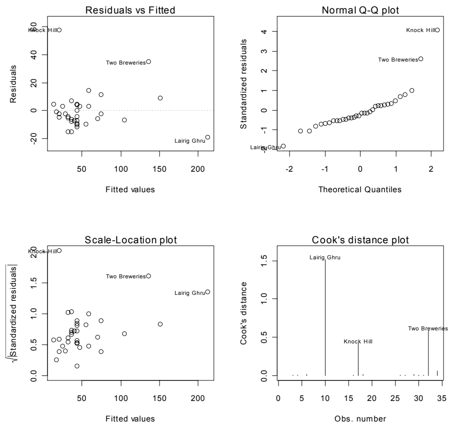
The function automatically identifies (with row numbers) the outlying and most influential points.

`7: Bens of Jura, 11: Lairig Ghru, 18: Knock Hill`

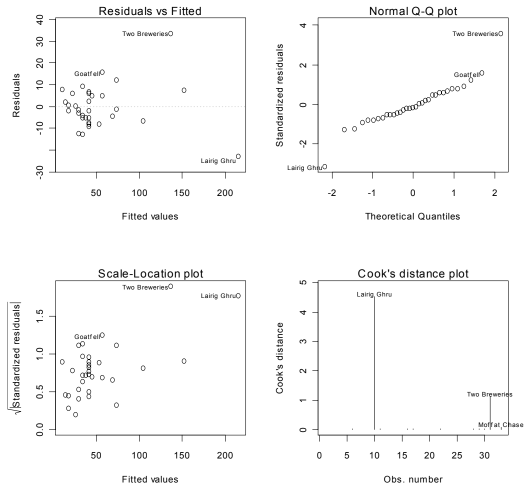
Of these, 7, Bens of Jura, is the most serious — it is both an outlier and is influential so the estimates depend strongly on it. 18, Knock Hill, is also an outlier but not nearly so influential so the results will not change so much if that observation is removed. 11, Lairig Ghru is very influential but does not appear to be 'out of line' with the others, it is just the longest race.

We can investigate the effect of dropping individual observations from the data set by `lm(time~dist,data=hills[-7,])` (to drop the 7th observation) and `lm(time~dist,data=hills[-c(7,18),])` for both:

```
> hillslm2<- lm(time~dist,data=hills[-7,])
> summary(hillslm2)
Call:
lm(formula = time ~ dist, data = hills[-7, ])
Residuals:
    Min       1Q  Median       3Q      Max
-19.221  -7.412  -3.159   3.692  57.790
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.0630     4.2073   -0.49    0.627
dist          7.6411     0.4665   16.38   <2e-16 ***
Residual standard error: 14.48 on 32 degrees of freedom
Multiple R-Squared: 0.8934,     Adjusted R-squared: 0.8901
F-statistic:  268.3 on 1 and 32 degrees of freedom,        p-
value:     0
> plot.lm(hillslm2)
```



```
> hillslm3<- lm(time~dist,data=hills[-c(7,18),])
> summary(hillslm3)
Call:
lm(formula = time ~ dist, data = hills[-c(7, 18), ])
Residuals:
    Min      1Q  Median      3Q     Max
-23.023  -5.285  -1.686   5.981  33.668
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -5.8125     3.0217  -1.924   0.0636 .
dist          7.9108     0.3306  23.926   <2e-16 ***

Residual standard error: 10.16 on 31 degrees of freedom
Multiple R-Squared: 0.9486,     Adjusted R-squared: 0.947
F-statistic:  572.5  on  1  and  31  degrees  of  freedom,        p-
value:     0
> plot.lm(hillslm3)
```

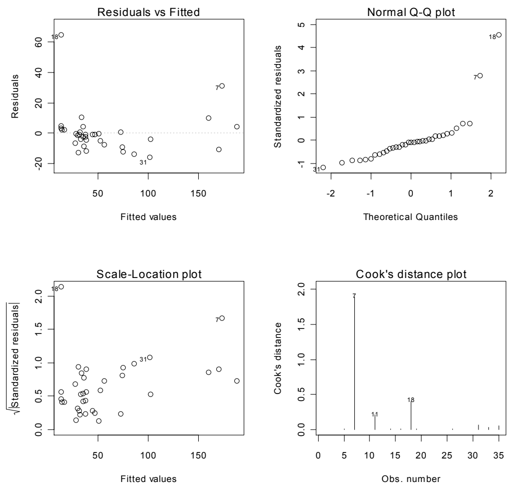


4.2.3 Several Variables

We can fit regression models involving several variables just by extending the formula in the `lm()` function in a natural way and we still have the same available diagnostics. To include the variable climb we have:

```
> hillslm4<-lm(time~dist+climb)
> summary(hillslm4)
Call:
lm(formula = time ~ dist + climb)

Residuals:
    Min      1Q  Median      3Q     Max
-16.215  -7.129  -1.186   2.371  65.121
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -8.992039   4.302734  -2.090   0.0447 *
dist         6.217956   0.601148  10.343 9.86e-12 ***
climb        0.011048   0.002051   5.387 6.45e-06 ***
Residual standard error: 14.68 on 32 degrees of freedom
Multiple R-Squared: 0.9191,     Adjusted R-squared: 0.914
F-statistic:  181.7  on  2  and  32  degrees  of  freedom,        p-
value:     0
```

```
> plot.lm(hillslm4)
```



Note that the 11th observation is no longer the most influential one but we still have problems with outliers. These could be dropped in just the same way as previously.

4.3 Robust Regression

Just as we have robust and resistant summary measures, or more technically robust estimates of location and scale, it is possible to define ***robust regression*** methods which are not so influenced by outliers. The only techniques provided in **R** is the function `rlm()`, (but in S-plus there are several others).

Consider again the Scottish Hill Races Data where it was found that observations 7 and 18 were outliers. The ordinary least squares fits to the full data set and after dropping the two outliers is given below:

```
> lm(time~dist+climb)

Call:
lm(formula = time ~ dist + climb)

Coefficients:
(Intercept)         dist        climb
   -8.99204      6.21796      0.01105

> lm(time~dist+climb,data=hills[-c(7,18),])

Call:
lm(formula = time ~ dist + climb, data = hills[-c(7,
18), ])

Coefficients:
(Intercept)         dist        climb
 -10.361646     6.692114     0.008047
```

Note how much the estimates change after dropping the two outliers.

Now consider the results of the robust regression using `rlm()`

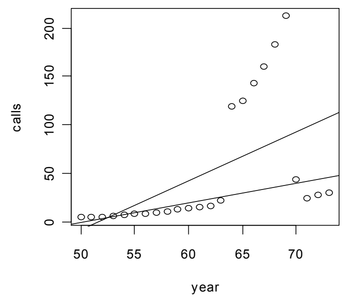
```
> rlm(time~dist+climb)
Call:
rlm.formula(formula = time ~ dist + climb)
Converged in 10 iterations

Coefficients:
 (Intercept)         dist        climb
-9.606716592  6.550722947  0.008295854

Degrees of freedom: 35 total; 32 residual
Scale estimate: 5.21
```

The estimates are nearly the same as using ordinary least square on the reduced data set.

Another example is a set of data giving the numbers of phone calls (in millions) in Belgium for the years 1950-73. In fact, there had been a mis-recording and the data for six years was recorded as the total length and not the number.

```
> plot(year,calls)
> lines(abline(lm(calls~year)))
> lines(abline(rlm(calls~year,maxit=50)))
```

gives a plot of the data with the fitted least squares line and the line fitted by a robust technique.



The diagnostic function `plot.lm()` can also be used with the results of a robust fit using `rlm()` and you are encouraged to try it, as well as investigating the full summary output of both `lm()` and `rlm()` for this data set.

4.4 Bootstrapping linear models

Care needs to be taken when bootstrapping in linear models. The obvious idea of taking samples with replacement of the actual data points $\{(x_i,y_i);I=1, \quad ,n\}$ is not usually appropriate if we regard the values of the independent variable as fixed. In the `whiteside` data set the x-values were temperatures and it might be argued that you could select pairs of points then.

The more usual technique is first to fit a model and then resample, with replacement, the residuals and create a bootstrap data set by adding on the resampled residuals to the estimated fits.

Specifically, if our model is $y_i=\alpha+\beta x_i+\varepsilon_i$ then we estimate α and β to obtain $\hat{\varepsilon}_i = y_i - \hat{\alpha} - \hat{\beta}x_i$, then we create a new bootstrap data set $\{(x_i, y_i^*);i = 1,\dots,n\}$ where $y_i^* = \hat{\alpha} + \beta x_i + \varepsilon_i^*$ and the (ε_i^*) are resampled with replacement from the $(\hat{\varepsilon}_i)$.

More details of this and other bootstrap techniques are given in Davison & Hinkley (1997) *Boostrap Methods and their Applications, C.U.P.* The library boot contains routines described in that book. It may also be noted that the library Bootstrap contains routines from the book *An Introduction to the Bootstrap* by Efron & Tibshirani (1993), Chapman and Hall.

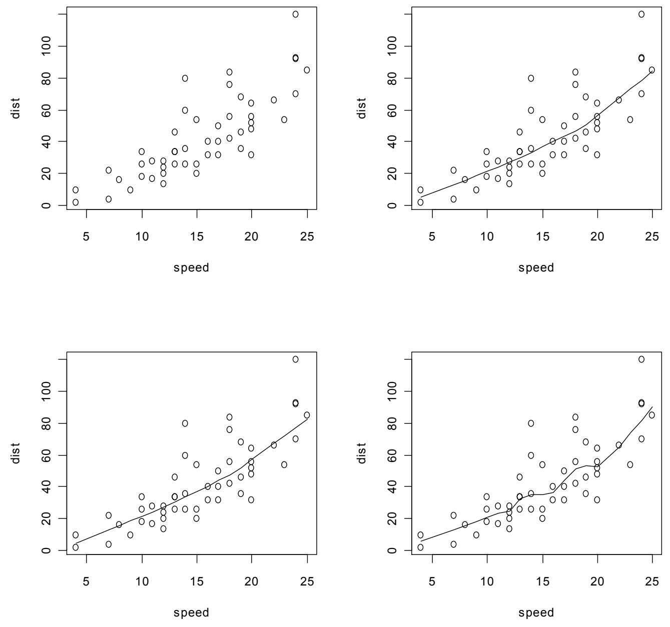
4.5 Scatterplot smoothing and smooth regression

A useful function for investigating structure in a scatterplot is the `lowess()` function. In outline, this fits a curve to a small window of the data, sliding the window across the data. The curve is calculated as a locally weighted regression, giving most weight to points close to the centre of the window with weights declining rapidly away from the centre to zero at the edge. It is possible to control the width of the window by a parameter which specifies the proportion of points included. The default value is 2/3 and larger values will give a smoother line, smaller ones a less smooth line.

Example: The data set `cars` gives the stopping distances for various speeds of 50 cars.

```
> plot(cars)
> plot(cars)
> lines(lowess(cars))
> plot(cars)
> lines(lowess(cars,f=0.8))
> plot(cars)
> lines(lowess(cars,f=0.3))
```

The plots are given on the next page. Note that for this data set of two variables we do not need to specify the names of the variables. If the data set had several variables the we would have to `attach` the data set to be able to plot specific variables, although if it is just the first two we need we could give just the name of the data set as here.

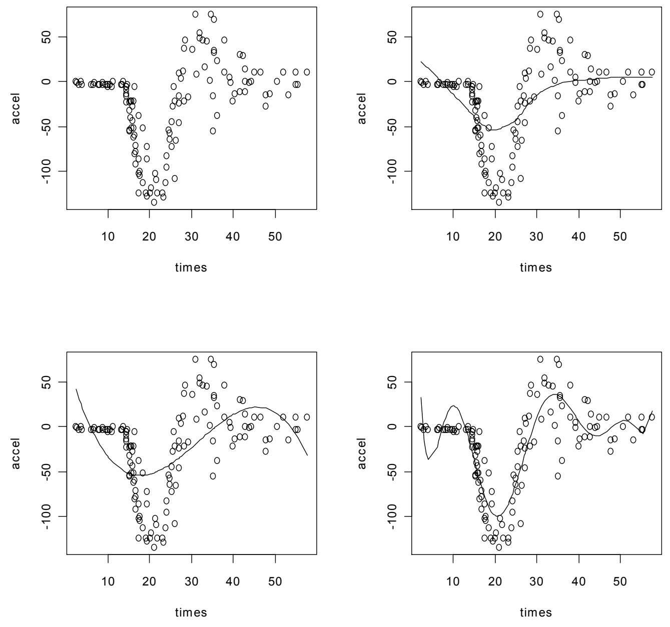


Although the data are very ‘noisy’ we can see definite evidence that the stopping distance increases more sharply with higher speeds, i.e. that the relationship is not completely linear. It provides an informal guide to how we should model the data.

The `lowess` smoother is an example of a smooth regression function.

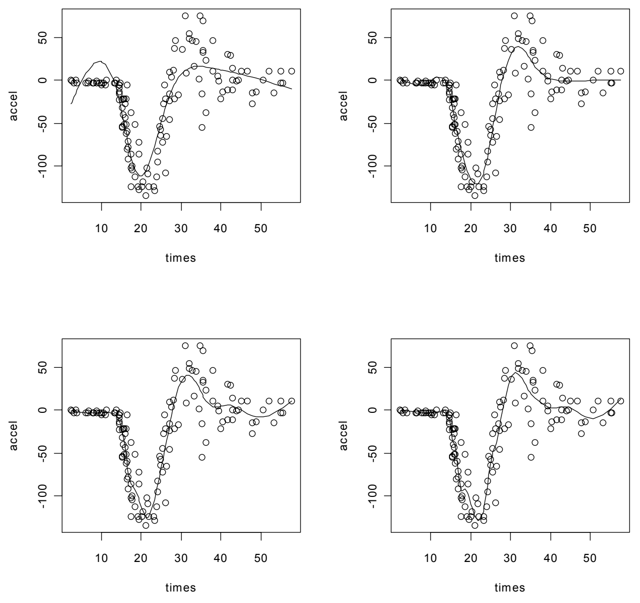
Other smooth regressions can be done with *natural splines* using function `ns()` in the `splines` library and kernel smoothing local regression. Generally these work much more satisfactorily than polynomial regression. We illustrate some of these on a data set `mcycle` which gives the accelerations at times in milliseconds after impact.

```
> data(mcycle)
> attach(mcycle)
> summary(mcycle)
     times            accel
 Min.   : 2.40   Min.   :-134.00
 1st Qu.:15.60   1st Qu.: -54.90
 Median :23.40   Median : -13.30
 Mean   :25.18   Mean   : -25.55
 3rd Qu.:34.80   3rd Qu.:   0.00
 Max.   :57.60   Max.   :  75.00
> plot(mcycle)
> plot(mcycle)
> lines(lowess(mcycle))
> plot(mcycle)
> lines(times,fitted(lm(accel~poly(times,3))))
> plot(mcycle)
> lines(times,fitted(lm(accel~poly(times,8))))
```



Note how unsatisfactory even a very high order polynomial regression is.

```
> library(splines)
> plot(mcycle)
> lines(times,fitted(lm(accel~ns(times,df=5))))
> plot(mcycle)
> lines(times,fitted(lm(accel~ns(times,df=10))))
> plot(mcycle)
> lines(times,fitted(lm(accel~ns(times,df=15))))
> plot(mcycle)
> lines(times,fitted(lm(accel~ns(times,df=20))))
```



The degree of freedom parameter controls the smoothness and it can be seen that a sensible choice is some around 10 for this data set.

More general regression models are ***generalised additive models*** which have the form

$$Y = \alpha + \sum_{j=1}^{p} f_j(X_j) + \varepsilon$$

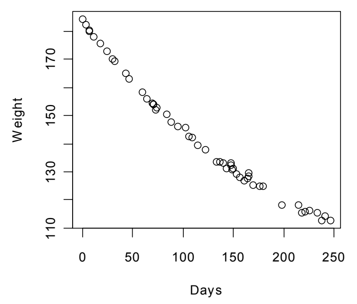
for some smooth functions $f_j(.)$

4.6 Example of non-linear regression

This example fits a model to data giving the weight loss in kg of a subject over a period of 250 days. The data are in dataset `wtloss` in the MASS library. There are strong theoretical grounds supporting a model of the form $y = \alpha + \beta.2^{-t/\theta} + \varepsilon$ which is linear in the parameters α and β but non-linear in the parameter θ.

To fit this model we need to have starting values for the iterative estimation of the parameters and this can be difficult to determine in general cases.

```
> data(wtloss)
> attach(wtloss)
> library(nls)
> wtlossfm<- nls(Weight~a+b*2^
        (-Days/theta),start=c(a=90,b=95,theta=120))
> wtlossfm
Nonlinear regression model
  model:  Weight ~ a + b * 2^(-Days/theta)
   data:  list
        a          b      theta
 81.37375 102.68417 141.91052
 residual sum-of-squares:  39.2447
> plot(Days,Weight)
```



A summary of the fitted model will produce standard errors of the estimates and inference can be carried out [almost] as with any other linear model. It is left as an exercise to plot the fitted model on the data.

4.7 Summary

- In this section we have presented the standard methods of fitting regression models on one or more variables, including cases where the independent variables are continuous covariates and/or factors with discrete levels.

- Diagnostic methods, including searches for **outliers** and **influential** observations, were mentioned and illustrated. Ideas of **robustness** and **bootstrapping** were extended to regression situations.

- **Smooth regression** was introduced with the idea of the **lowess** function (also available in many other packages) and illustrations using **natural splines** were presented.

- An example of a **non-linear** regression model was given.

The overall theme of this chapter is that there are many widely different regression models that can be handled in very similar ways without necessarily knowing the full details of all the mathematical theory underlying them.

5. Multivariate Methods

5.1 Introduction

Earlier chapters have considered a variety of statistical methods for *univariate data*, i.e. the response we are interested in is a one-dimensional variable although we have considered data sets which have several variables, e.g. `hills` consists of three variables `dist`, `time` and `climb` but we regarded `time` as the response and considered its dependence on the explanatory variables `dist` and `climb`.

Multivariate analysis is concerned with datasets that have more than one response variable for each observational unit, we may also have several explanatory variables or maybe none at all — the key feature is that we want to treat all the response variables simultaneously and equally, none is 'preferred' over the others. In this brief review of multivariate methods we will consider primarily problems where all the ***n*** observations are on ***p*** response variables.

References:

Gnanadesikan, R. (1997) *Methods for Statistical Data Analysis of Multivariate Observations.* (2nd) Edition). Wiley.

Mardia, K., Kent, J. & Bibby, J. (1981) *Multivariate Analysis.* Wiley.

Krzanowski, W. (1990) *Multivariate Analysis.* (Oxford)

Ripley, B.D. (1996) ***Pattern Recognition and Neural Networks***. Cambridge University Press

Everitt, B.S. & Dunn, G. (1991) *Applied Multivariate Data Analysis.* Arnold

Manly, B. J. (1986) *Multivariate statistical methods: a primer*, Chapman & Hall.

Examples of multivariate data sets:

(i) body temperature, renal function, blood pressure, weight of 73 hypertensive subjects (*p=4, n=73*).

(ii) petal & sepal length & width of 150 flowers (*p=4, n=150*).

(iii) amounts of 9 trace elements in clay of Ancient Greek pottery fragments (*p=9*).

(iv) amounts of each of 18 amino acids in fingernails of 92 arthritic subjects (*p=18, n=92*).

(v) presence or absence of each of 286 decorative motifs on 148 bronze age vessels found in North Yorkshire (*p=286, n=148).*

(vi) grey shade levels of each of 1024 pixels in each of 15 digitized images (*p=1024, n=15*)

(vii) Analysis of responses to questionnaires in a survey (*p= number of questions, n=number of respondents*)

(viii) Digitization of a spectrum (*p=20000, n=20 is typical)*

(ix) Activation levels of all genes on a genome (p=60000, n=50 is typical)

Notes

- Measurements can be *discrete* e.g. (v) & (vi), or *continuous*, e.g. (i)-(iv) or a *mixture* of both, e.g.(vii).
- There may be more observations than variables, e.g. (i)-(iv), or they may be more variables than observations, e.g. (v) & (vi) and especially (viii) and (ix).
- Typically the variables are ***correlated*** but individual sets of observations are ***independent***.

Subject Matter/ Some Multivariate Problems

(i) Obvious generalizations of univariate problems: t-tests, analysis of variance, regression, multivariate general linear model. e.g. model data Y' by $Y'=X\Theta + \varepsilon$,

where Y' is the $n\times p$ data matrix, X is an $n\times k$ matrix of known observations of k-dimensional regressor variables, Θ is $k\times p$ matrix of unknown parameters, ε is $n\times p$ with n values of p-dimensional error variables.

(ii) Reduction of dimensionality for

(a) exploratory analysis

(b) simplification (MVA is easier if p=1 or p=2)

(c) achieve greater statistical stability

(e.g. remove variables which are highly correlated)

Methods of principal component analysis, factor analysis, non-metric scaling....

(iii) Discrimination

Find rules for discriminating between groups, e.g. 2 known variants of a disease, data X', Y' on each. What linear combination of the p variables best discriminates between them. Useful to have diagnostic rules and this may also throw light onto the conditions (e.g. in amino acids and arthritis example there are two type of arthritis :— psoriatic and rheumatoid, determining which combinations of amino acids best distinguishes between them gives information on the biochemical differences between the two conditions). Sometimes referred to as a ***supervised*** problem.

(iv) Cluster Analysis/Classification

Do data arise from a homogeneous source or do they come from a variety of sources, e.g. does a medical condition have sub-variants. This is sometimes referred to as an ***unsupervised*** problem.

(v) Canonical Correlation Analysis

Of use when looking at relationships between sets of variables, e.g. in particular in questionnaire analysis between response to 2 groups of questions, perhaps first group of questions might investigate respondents expectations and the second group their evaluation.

This chapter will look at (ii), (iii) and (iv).

Quotation:

“Much classical and formal theoretical work in Multivariate Analysis rests on assumptions of underlying multivariate normality — resulting in techniques of very limited value”.

(Gnanadesikan, page 2).

5.2 Data Display

5.2.1 Preliminaries

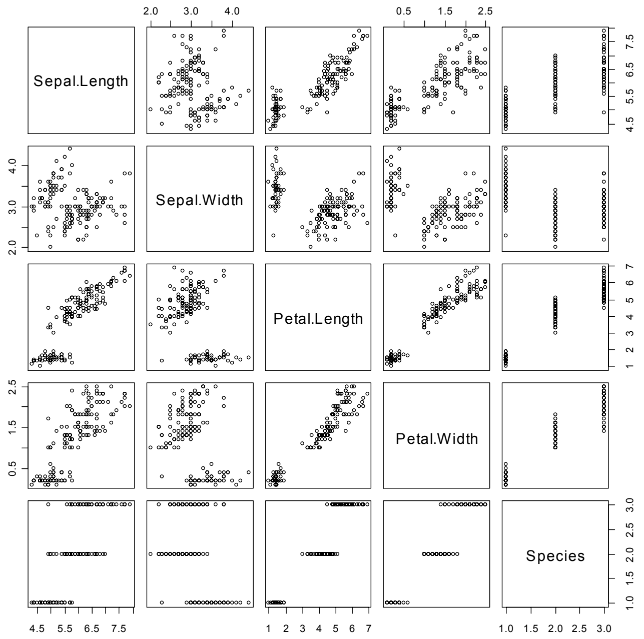
The key tool in displaying multivariate data is scatterplots of pairwise components arranges as a *matrix plot*. The function `pairs()` makes this easy but the examples below illustrate that as soon as the number of dimensions becomes large (i.e. more than about 5) it is difficult to make sense of the display. If the number of observations is small then some other technique can handle quite large numbers of variables. Star plots are one possibility and are illustrated, others are Andrews' Plots and Chernoff Faces which are not available in **R** but you may find them in other packages (e.g. S-plus, Minitab).

A classic set of data that is used for illustration of many multivariate problems is Anderson's Iris Data which give the sepal and petal lengths and widths of each of 50 flowers from each of threes varieties (Setosa, Versicola and Virginica). This is held in the base library of **R** in two formats, data sets `iris` and `iris3`, the first is as a complete dataframe and the second has a slightly different structure. Although we 'know' *a priori* (because the botanist has told us and it is recorded) that there are three different varieties we will ignore this fact in some of the analyses below.

For many multivariate methods in **R** the data need to be presented as a *matrix* rather than a *dataframe*. This is a slightly obscure technicality but is the reason for the use of `as.matrix(.)`, `rbind(.)` and `cbind(.)` etc at various places in what follows.

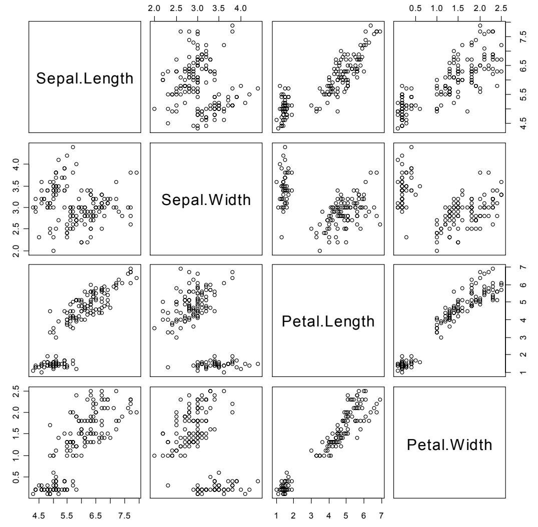
5.2.2 Matrix Plots and Star Plots

```
> data(iris)
> pairs(iris)
```



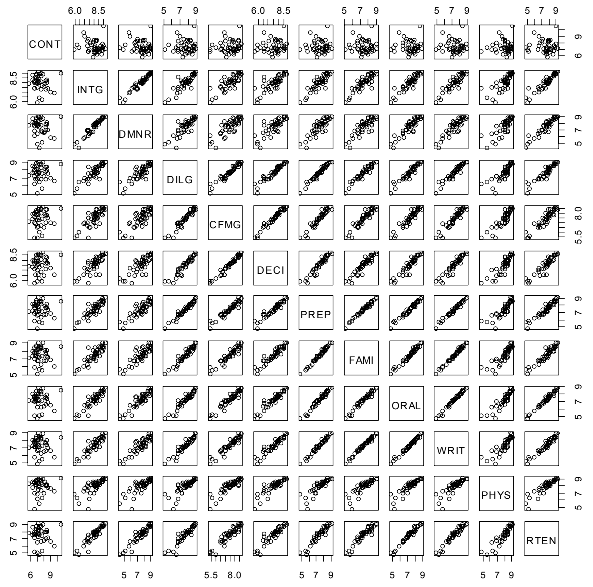
The 5th 'variable' held in the dataframe iris is the name of the species and perhaps it is not useful to include that

```
> attach(iris)
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50

> ir<- cbind(Sepal.Length, Sepal.Width, Petal.Length,
Petal.Width)
> pairs(ir)
```



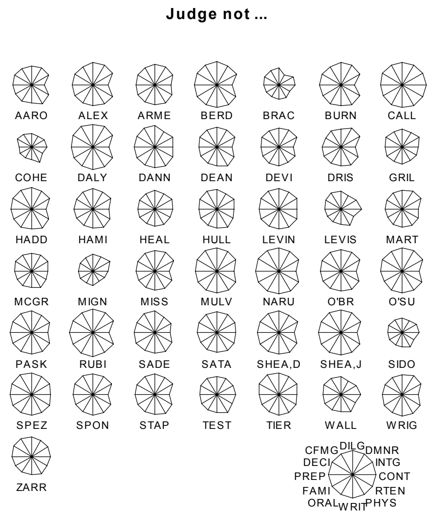
It is clear that we can see some structure in the data and might even guess that there are three distinct groups of flowers. Now consider an example with 12 variables (and 43 observations). Data set `USJudgeRatings` gives 12 measures of ability and performance on 43 US judges.

```
> data(USJudgeRatings)
> pairs(USJudgeRatings)
```



The matrix plot above is difficult to comprehend. An alternative is a *star plot* where each observation is represented by a star or polygon where the length of the vector to each vertex corresponds to the value of a particular variable.

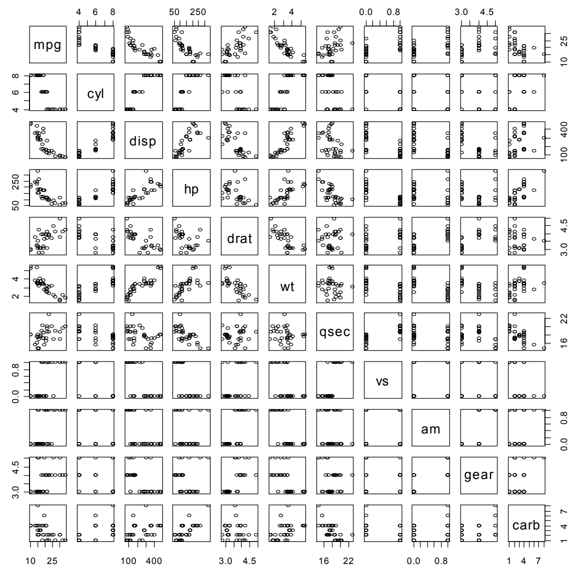
```
>  stars(USJudgeRatings, labels =
abbreviate(case.names(USJudgeRatings)),
+           key.loc = c(13, 1.5), main = "Judge not ...", len
= 0.8)
```



We can begin to see something about similarities and differences between individual observations (i.e. judges) but not any relationships between the variables.

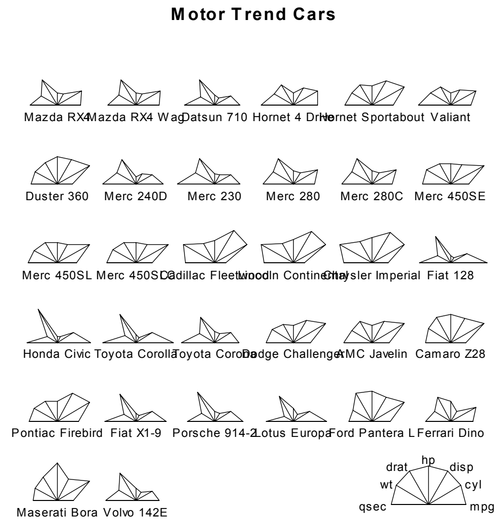
Another example: data set `mtcars` gives measurements of 11 properties of 32 cars (fuel consumption, engine size etc).

```
> data(mtcars)
> pairs(mtcars)
```



Again, not informative.

```
> stars(mtcars[, 1:7], key.loc = c(12, 2),
+             main = "Motor Trend Cars", full = FALSE)
```



This is perhaps more informative on individual models of cars but again not easy to see more general structure in the data.

Conclusions: The simple scatter plots arranged in a matrix work well when there are not too many variables. If there are more than about 5 variables it is difficult to focus on particular displays and so understand the structure. If there are not too many observations then we can construct a symbol (e.g. a star or polygon) for each separate observation which indicates the values of the variables for that observation. The other techniques mentioned (Andrews' Plots and Chernoff Faces, as well as many other ingenious devices) all have similar objectives and drawbacks as star plots. Obviously star plots will not be useful if either there are a large number of observations (more than will fit on one page) or if there are a large number of variables (e.g. > 20).

However, many multivariate statistical analyses involve very large numbers of variables, e.g. 50+ is routine, 1000+ is becoming increasingly common in areas such as genomics.

What is required is a display of 'all of the data' using just a few scatterplots (i.e. using just a few variables). That is we need to select 'the most interesting variables'. This may mean concentrating on 'the most interesting' actual measurements or it may mean combining the variables together in some way to create a few new ones which are the 'most interesting'. That is, we need some technique of ***dimensionality reduction.***

The most useful routine method of dimensionality reduction is ***principal component analysis.***

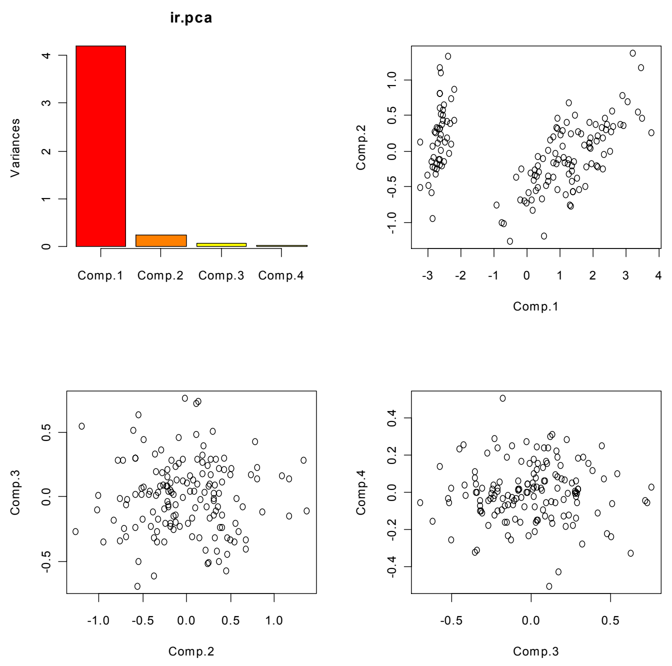
5.3 Principal Component Analysis

Principal Component Analysis (or PCA) looks for components in the data which contain the most information. Often, transforming data to principal components will reduce the dimensionality of the data and we need only look at a very few components. Details are not given here (they can be found in many of the textbooks listed at the start of the chapter). It is best, to begin with, to use the technique and see what happens.

Note that some of the key routines in **R** are contained in a special library for multivariate analysis called `mva`.

Example: Iris data.

```
> library(mva)
> ir.pca<-princomp(ir)
> ir.pca
Call:
princomp(x = ir)

Standard deviations:
   Comp.1    Comp.2    Comp.3    Comp.4
2.0494032 0.4909714 0.2787259 0.1538707

 4  variables and  150 observations.
> summary(ir.pca)
Importance of components:
                          Comp.1         Comp.2         Comp.3
Comp.4
Standard  deviation          2.0494032  0.49097143  0.27872586
0.153870700
Proportion  of  Variance   0.9246187  0.05306648  0.01710261
0.005212184
Cumulative  Proportion     0.9246187  0.97768521  0.99478782
1.000000000
> plot(ir.pca)
> par(mfrow=c(2,2))
> plot(ir.pca)
> loadings(ir.pca)
                  Comp.1      Comp.2      Comp.3     Comp.4
Sepal.Length  0.36138659  0.65658877 -0.58202985 -0.3154872
Sepal.Width  -0.08452251  0.73016143  0.59791083  0.3197231
Petal.Length  0.85667061 -0.17337266  0.07623608  0.4798390
Petal.Width   0.35828920 -0.07548102  0.54583143 -0.7536574
> ir.pc<- predict(ir.pca)
> plot(ir.pc[,1:2])
> plot(ir.pc[,2:3])
> plot(ir.pc[,3:4])
>
```



Comments

Generally, if data X’ are measurements of p variables all of the same ‘type’ (e.g. all concentrations of amino acids or all linear dimensions in the same units, but ***not*** e.g. age/income/weights) then the coefficients of principal components can be interpreted as ‘loadings’ of the original variables and hence the principal components can be interpreted as contrasts in the original variables.

Further Example of Interpretation of Loadings

This data for this example are given in Wright (1954), The interpretation of multivariate systems. In *Statistics and Mathematics in Biology* (Eds. O. Kempthorne, T.A. Bancroft J. W. Gowen and J.L. Lush), 11–33. State university Press, Iowa, USA.

Six bone measurements x_1, ,x_6 were made on each of 275 white leghorn fowl. These were: x_1 skull length; x_2 skull breadth; x_3 humerus; x_4 ulna; x_5 femur; x_6 tibia (the first two were skull measurements, the third and fourth wing measurements and the last two leg measurements). The table below gives the coefficients of the six principal components calculated from the covariance matrix.

Original	**Principal Components**					
variable	**a_1**	**a_2**	**a_3**	**a_4**	**a_5**	**a_6**
x_1 skull l.	0.35	0.53	0.76	–0.04	0.02	0.00
x_2 skull b.	0.33	0.70	–0.64	0.00	0.00	0.03
x_3 humerus	0.44	–0.19	–0.05	0.53	0.18	0.67
x_4 ulna	0.44	–0.25	0.02	0.48	–0.15	–0.71
x_5 femur	0.44	–0.28	–0.06	–0.50	0.65	–0.13
x_6 tibia	0.44	–0.22	–0.05	–0.48	–0.69	0.17

To interpret these coefficients we 'round' them heavily to either just one digit and ignore values 'close' to zero, giving

Original	**Principal Components**						
variable	**a_1**	**a_2**	**a_3**	**a_4**	**a_5**	**a_6**	
x_1 skull l.	0.4	0.6	0.7	0	0	0	**skull**
x_2 skull b.	0.4	0.6	–0.7	0	0	0	
x_3 humerus	0.4	–0.2	0	0.5	0	0.7	**wing**
x_4 ulna	0.4	–0.2	0	0.5	0	–0.7	
x_5 femur	0.4	–0.2	0	–0.5	0.6	0	**leg**
x_6 tibia	0.4	–0.2	0	–0.5	–0.6	0	

Original	**Principal Components**						
variable	**a_1**	**a_2**	**a_3**	**a_4**	**a_5**	**a_6**	
x_1 skull l.	+	+	+				**skull**
x_2 skull b.	+	+	–				
x_3 humerus	+	–		+		+	**wing**
x_4 ulna	+	–		+		–	
x_5 femur	+	–		–	+		**leg**
x_6 tibia	+	–		–	–		

We can then see that the first component $\mathbf{a_1}$ is proportional to the sum of all the measurements. Large fowl will have all x_i large and so their scores on the first principal component y_1 (=x'a_1) will be large, similarly small birds will have low scores of y_1. If we produce a scatter plot using the first p.c. as the horizontal axis then the large birds will appear on the right hand side and small ones on the left. Thus the first p.c. measures ***overall size***.

The second component is of the form (skull)–(wing & leg) and so high positive scores of y_2 (=x'a_2) will come from birds with large heads and small wings and legs. If we plot y_2 against y_1 then the birds with *relatively* small heads for the size of their wings and legs will appear at the bottom of the plot and those with relatively big heads at the top. The second p.c. measures ***overall body shape***.

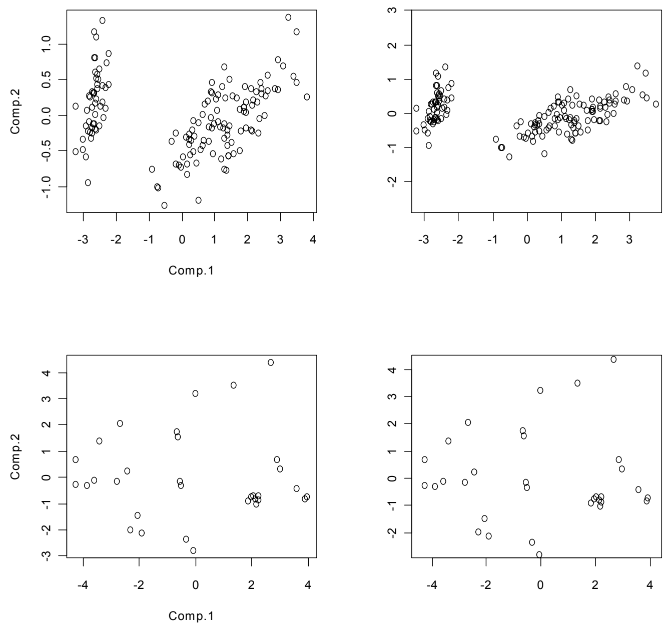
The third component is a measure of ***skull shape*** (i.e. skull width *vs* skull width), the fourth is wing size *vs* leg size and so is also a measure of ***body shape*** (but not involving the head). The fifth and sixth are contrasts between upper and lower parts of the wing and leg respectively and so y_5 measures ***wing shape*** and y_6 measures ***leg shape.***

Notes:

- The full mathematical/algebraic theory of principal component analysis strictly applies **ONLY** to continuous data on comparable scales of measurements using the covariance matrix. Using the correlation matrix brings the measurements onto a common scale but a little care is needed in interpretation, especially in interpretation of the loadings.

- Since the key objective of pca is to extract information, i.e. *partition the internal variation* it is sensible to plot the data using equal scaling on the axes. This can be done using the MASS function `eqscplot()`:

```
> plot(ir.pc[,1:2])
> eqscplot(ir.pc[,1:2])
> plot(mtcars.pc[,1:2])
> eqscplot(mtcars.pc[,1:2])
```



Note that unlike `plot()` the axes are not automatically labelled by `eqscplot()` and you need to do this by including

```
,xlab="first p.c.",ylab="second p.c."
```

in the call to it.

5.4 Discriminant Analysis

So far the data analytic techniques considered have regarded the data as arising from a homogeneous source — i.e. as all one data set. A display on principal components might reveal unforeseen features:– outliers, subgroup structure as well as perhaps singularities (i.e. dimensionality reduction).

Suppose now we know that the data consist of observations classified into k groups (c.f. 1-way classification in univariate data analysis) so that data from different groups might be different in some ways. We can take advantage of this knowledge

- to produce more effective displays
- to achieve greater dimensionality reduction
- to allow informal examination of the nature of the differences between the groups.

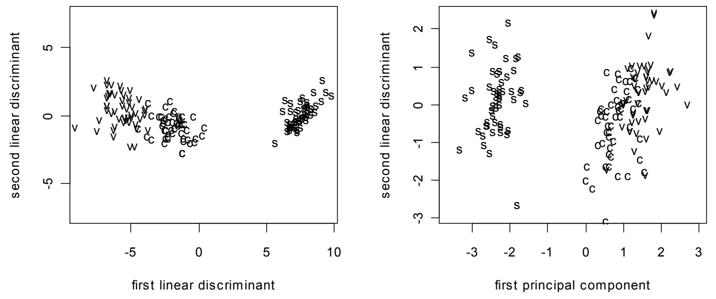
Linear Discriminant Analysis finds the best linear combinations of variables for separating the groups, if there are k different groups then it is possible to find k–1 separate linear discriminant functions which partition the *between groups variance* into decreasing order, similar to the way that principal component analysis partitions the *within group variance* into decreasing order. The data can be displayed on these *discriminant coordinates.*

Example: Iris Data

```
> ir.species<-
factor(c(rep("s",50),rep("c",50),rep("v",50)))
> ir.lda<-lda(ir,ir.species)
> ir.lda
Call:
lda.matrix(ir, grouping = ir.species)
Prior probabilities of groups:
      c       s       v
0.33333 0.33333 0.33333
Group means:
  Sepal L. Sepal W. Petal L. Petal W.
c    5.936    2.770    4.260    1.326
s    5.006    3.428    1.462    0.246
v    6.588    2.974    5.552    2.026

Coefficients of linear discriminants:
              LD1       LD2
Sepal L.  0.82938  0.024102
Sepal W.  1.53447  2.164521
Petal L. -2.20121 -0.931921
Petal W. -2.81046  2.839188

Proportion of trace:
   LD1    LD2
0.9912 0.0088
```

Now plot the data on discriminant coordinates and compare with a principal component plot:

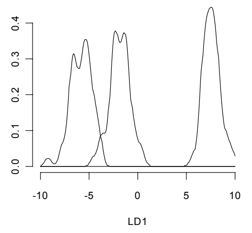
```
> ir.ld<-predict(ir.lda,dimen=2)$x
> eqscplot(ir.ld,type="n",
+ xlab="first linear discriminant",
+ ylab="second linear discriminant")
> text(ir.ld,labels=as.character(ir.species))
> eqscplot(ir.pc,type="n",
+ xlab="first principal component",
+ ylab="second principal component")
> text(ir.pc,labels=as.character(ir.species))
```



There is perhaps a very slightly better separation between the groups labelled `v` and `c` in the left hand plot than in the right hand one.

Another way of examining the overlap between the groups is to look at kernel density estimates based on the values on the first linear discriminant coordinate separately for the three groups, conveniently done by:

```
> plot(ir.lda,type="density",dimen=1)
```



A key objective of discriminant analysis is to classify further observations. This can be done using the predict function `predict.lda(lda-object,newdata)`. In the Cushings data we can perform the lda on the first 21 observations and then use the results to classify the final 7 observations of unknown categories. Note that we have to turn the final 7 observations into a data matrix `cushu` in the same way as we did with the *training data.*

```
> cush<-log(as.matrix(Cushings[1:21,-3]))
> cushu<-log(as.matrix(Cushings[22:27,-3]))
> tp<-factor(Cushings$Type[1:21])

> cush.lda<-lda(cush,tp)

> upredict<-predict.lda(cush.lda,cushu)
> upredict$class
[1] b c b a b b
```

These are the classifications for the seven new cases.

We can plot the data on the discriminant coordinates with

```
> plot(cush.lda)
```

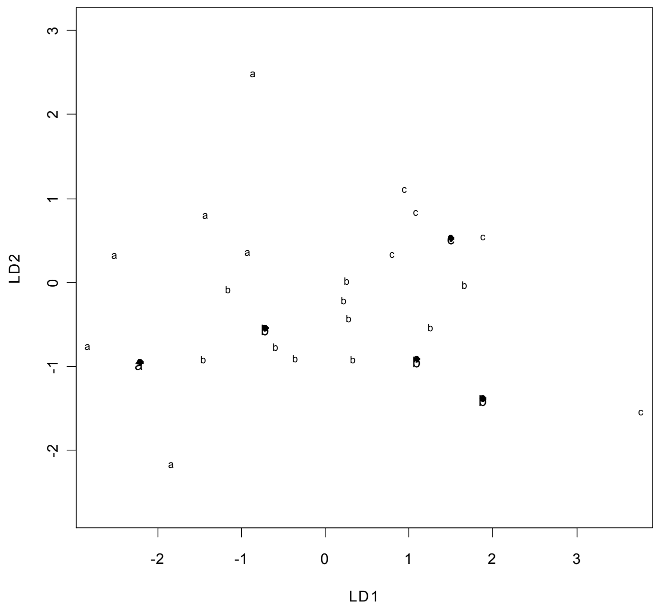
and then add in the unknown points with

```
> points(jitter(predict(cush.lda,cushu)$x),pch=19,)
```

and finally put labels giving the predicted classifications on the unknown points with

```
> text(predict(cush.lda,cushu)$x,pch=19,
+ labels=as.character(predict(cush.lda,cushu)$class))
```

(where the `+` is the continuation prompt from **R**) to give the plot below. The use of `jitter()` moves the points slightly so that the labels are not quite in the same place as the plotting symbol.



Quadratic Discriminant Analysis generalizes lda to allow quadratic functions of the variables. Easily handled in **R** `qda()`.

```
> cush.qda<-qda(cush,tp)
> predict.qda(cush.qda,cushu)$class
[1] b c b a a b
```

It can be seen that the 5th unknown observation is classified differently by `qda()`. How can we see whether `lda()` or `qda()` is better? One way is to see how each performs on classifying the training data (i.e. the cases with known categories.

```
> predict.lda(cush.lda,cush)$class
 [1] a a a b b a b a a b b c b b b b c c b c c
```

and compare with the 'true' categories:

```
> tp
 [1] a a a a a a b b b b b b b b b b c c c c c
```

We see that 6 observations are misclassified, the 5th,6th,9th,10th,13th and 19th. To get a table of predicted and actual values:

```
> table(tp,predict.lda(cush.lda,cush)$class)
```

```
tp  a b c
  a 4 2 0
  b 2 7 1
  c 0 1 4
```

Doing the same with `qda()` gives:

```
> table(tp,predict.qda(cush.qda,cush)$class)
tp  a b c
  a 6 0 0
  b 0 9 1
  c 0 1 4
```

so 19 out 21 were correctly classified, when only 15 using `lda()`.

If we want to see whether correctly classifying 15 out of 21 is better than chance we can permute the labels by sampling `tp` without replacement:

```
> randcush.lda<-lda(cush,sample(tp))
> table(tp,predict.lda(randcush.lda,cush)$class)

tp  a b c
  a 3 2 1
  b 1 9 0
  c 0 5 0
```

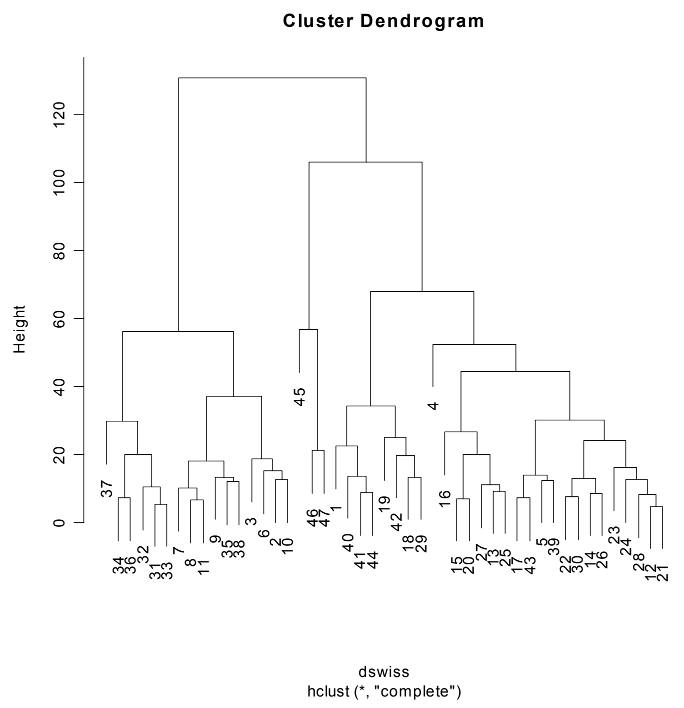
i.e. 12 were correctly classified even with completely random labels. Repeating this a few more times quickly shows that 15 is much higher than would be obtained by chance. It would be easy to write a function to do this 1000 times say by extracting the diagonal elements which are the 1st,5th and 9th elements of the object `table(.,.)`, i.e. `table(.,.)[1]`, `table(.,.)[5]` and `table(.,.)[9]`.

```
> randcush.lda<-lda(cush,sample(tp))
> table(tp,predict.lda(randcush.lda,cush)$class)
tp  a  b c
  a 1  5 0
  b 0 10 0
  c 0  5 0
> randcush.lda<-lda(cush,sample(tp))
> table(tp,predict.lda(randcush.lda,cush)$class)
 tp a b c
  a 1 5 0
  b 2 8 0
  c 1 4 0
> randcush.lda<-lda(cush,sample(tp))
> table(tp,predict.lda(randcush.lda,cush)$class)
tp  a  b c
  a 1  5 0
  b 0 10 0
  c 1  4 0
```

5.6 Cluster Analysis

Cluster Analysis is a collection of techniques for *unsupervised* examination of multivariate data with an objective of discovering 'natural' groups in the data, often used together with scaling methods. The results are displayed in a ***dendogram*** rather like a family tree indicating which family objects belong to. The first example is on data `swiss` which gives demographic measurements of 47 provinces in Switzerland. The first step is to calculate a distance matrix, using `dist()` and then to perform hierarchical cluster analysis using `hclust()`, The result can then be plotted as a dendogram using the generic function `plot()`. This example has used the default clustering method of complete linkage, others you might try are average linkage, single linkage or Wards method

```
> data(swiss)
> summary(swiss)
   Fertility       Agriculture      Examination      Education
 Min.   :35.00   Min.   : 1.20   Min.   : 3.00   Min.   : 1.00
 1st Qu.:64.70   1st Qu.:35.90   1st Qu.:12.00   1st Qu.: 6.00
 Median :70.40   Median :54.10   Median :16.00   Median : 8.00
 Mean   :70.14   Mean   :50.66   Mean   :16.49   Mean   :10.98
 3rd Qu.:78.45   3rd Qu.:67.65   3rd Qu.:22.00   3rd Qu.:12.00
 Max.   :92.50   Max.   :89.70   Max.   :37.00   Max.   :53.00
    Catholic       Infant.Mortality
 Min.   :  2.150   Min.   :10.80
 1st Qu.:  5.195   1st Qu.:18.15
 Median : 15.140   Median :20.00
 Mean   : 41.144   Mean   :19.94
 3rd Qu.: 93.125   3rd Qu.:21.70
 Max.   :100.000   Max.   :26.60
> dswiss<-dist(swiss)
> h<- hclust(dswiss)
> plot(h)
```



This suggests three main groups, we can identify these with

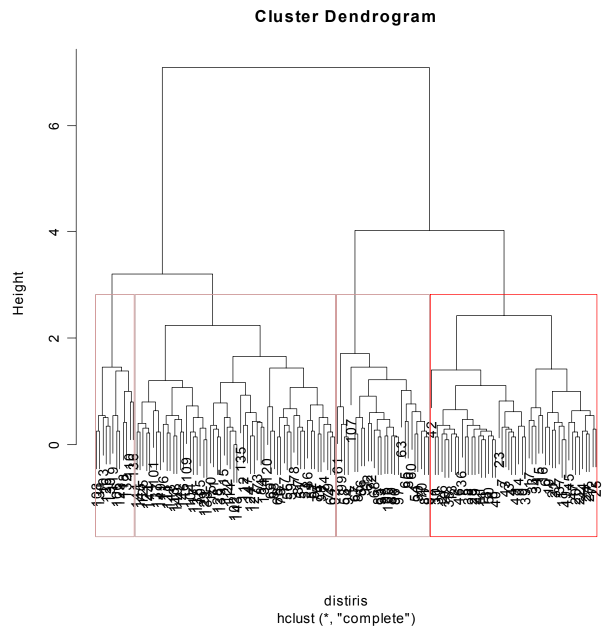
```
> cutree(h,3)
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26
 1  2  2  1  1  2  2  2  2  2  2  1  1  1  1  1  1  1  1  1  1
1  1  1  1  1
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 1  1  1  1  2  2  2  2  2  2  2  2  1  1  1  1  1  1  3  3  3
```

which gives the group membership for each of the provinces.

Next, we look at the iris data (yet again) and use the interactive function `identify.hclust()` which allows you to point with the mouse and click on vertical bars to extract the elements in the family below. Click with the right button and choose stop to leave it.

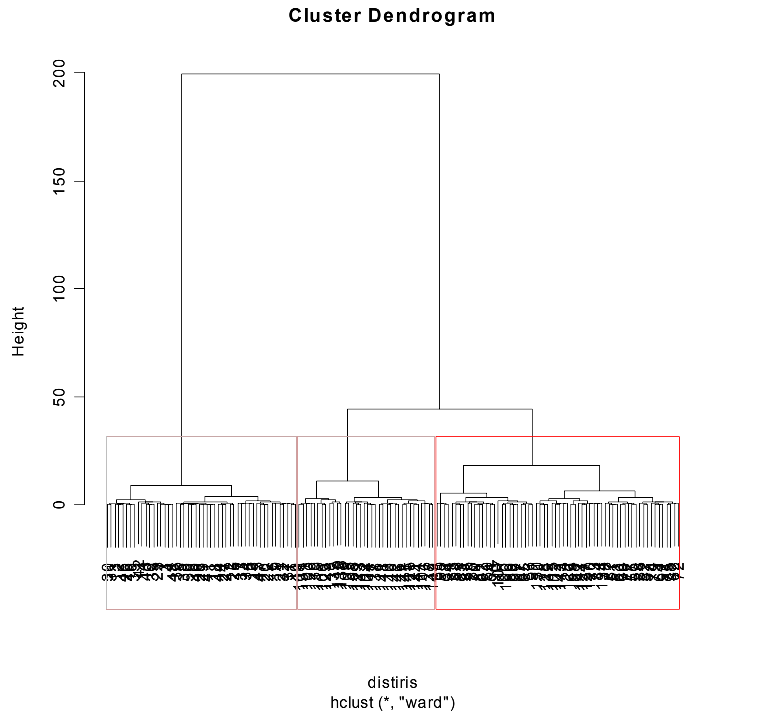
```
> distiris<-dist(ir)
> hiris<- hclust(distiris)
> plot(hiris)
> help(identify.hclust)
>  identify.hclust(hiris, function(k) print(table(iris[k,5])))

    setosa versicolor  virginica
         0          0         12

    setosa versicolor  virginica
         0         23         37

    setosa versicolor  virginica
         0         27          1

    setosa versicolor  virginica
        50          0          0
>
```

The dendogram on the next pages shows four groups, and `identify.clust` was used to click on the four ancestor lines. Note that one of the groups is obviously the overlap group between versicolour and virginica.



Using a different method (Ward's) gives:

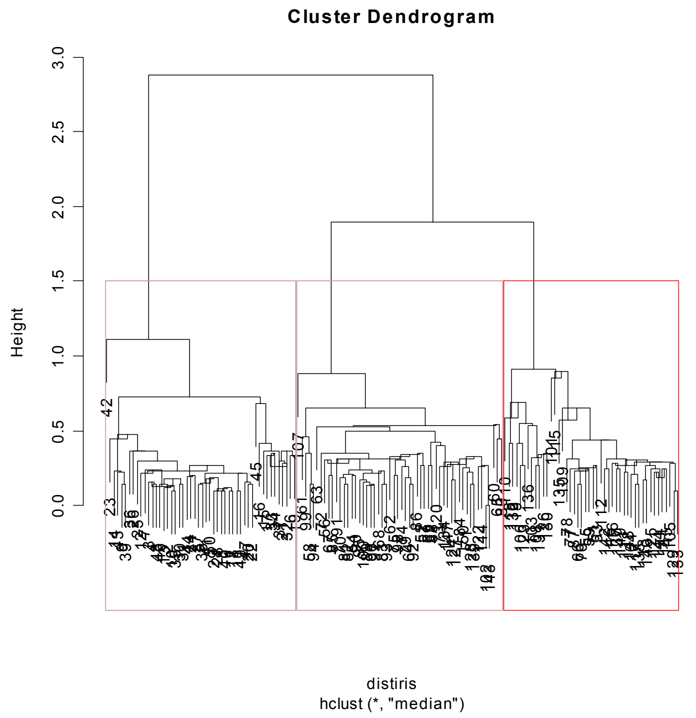
```
> hirisw<- hclust(distiris,method="ward")
> plot(hirisw)
>  identify.hclust(hirisw,function(k) print(table(iris[k,5])))
    setosa versicolor  virginica
        50          0          0
    setosa versicolor  virginica
         0          0         36
    setosa versicolor  virginica
         0         50         14
```



And finally, using the 'median' method gives

```
> hirimed<- hclust(distiris,method="median")
> plot(hirimed)
>  identify.hclust(hirimed,function(k)print(table(iris[k,5])))
    setosa versicolor  virginica
        50          0          0
    setosa versicolor  virginica
```

```
         0         41         13
    setosa versicolor  virginica
         0          9         37
```



Further Directions: The library `cluster` contains a variety of routines and data sets. The `mclust` library offers model-based clustering (i.e. a little more statistical).

Key reference: **Cluster Analysis, 4th Edition, (2001)**, by Brian Everitt, Sabine Landau and Morven Leese.

6 Tree-Based Methods

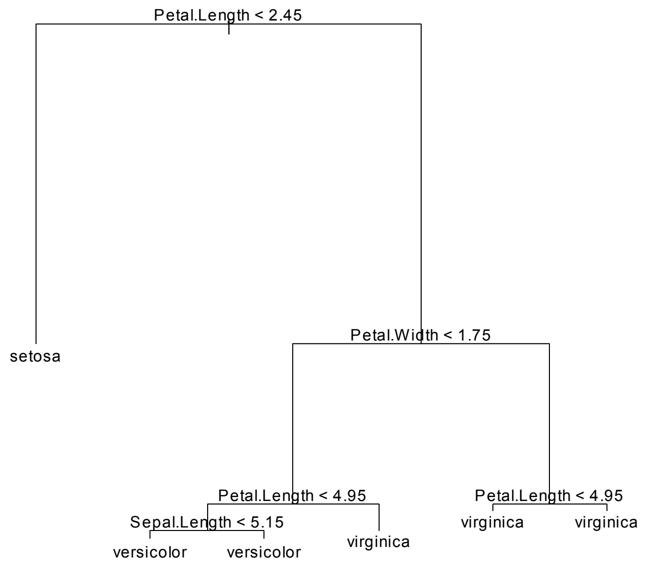
Classification and regression trees are similar *supervised* techniques which are used to analyse problems in a step-by-step approach. We start (even yet again) with the iris data where the objective is to find a set of rules, based on the four measurements we have, of classifying the flowers into on of the fours species. The rules will be of the form:

'if petal length>x then . . , but if petal length $\leq$x then something else'

i.e. the rules are based on the values of one variable at a time and gradually partition the data set into groups.

```
> data(iris)
> attach(iris)
> ir.tr<-tree(Species~.,iris)
> plot(ir.tr)
> summary(ir.tr)

Classification tree:
tree(formula = Species ~ ., data = iris)
Variables actually used in tree construction:
[1] "Petal.Length" "Petal.Width"  "Sepal.Length"
Number of terminal nodes:  6
Residual mean deviance:  0.1253 = 18.05 / 144
Misclassification error rate: 0.02667 = 4 / 150
> text(ir.tr,all=T,cex=0.5)
```

Now look at the graphical representation:

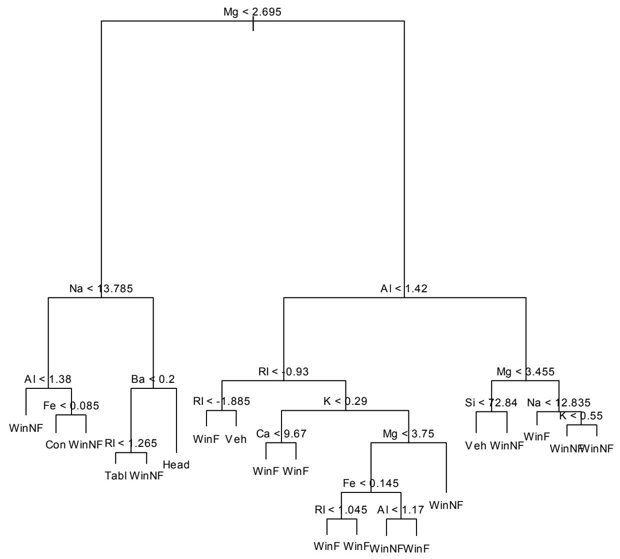


```
> ir.tr
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
   2) Petal.Length < 2.45 50   0.000 setosa ( 1.00000 0.00000 0.00000 ) *
   3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
     6) Petal.Width < 1.75 54  33.320 versicolor ( 0.00000 0.90741 0.09259 )
      12) Petal.Length < 4.95 48   9.721 versicolor ( 0.00000 0.97917 0.02083 )
```

```
        24) Sepal.Length < 5.15 5   5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
        25) Sepal.Length > 5.15 43   0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
      13) Petal.Length > 4.95 6   7.638 virginica ( 0.00000 0.33333 0.66667 ) *
     7) Petal.Width > 1.75 46   9.635 virginica ( 0.00000 0.02174 0.97826 )
      14) Petal.Length < 4.95 6   5.407 virginica ( 0.00000 0.16667 0.83333 ) *
      15) Petal.Length > 4.95 40   0.000 virginica ( 0.00000 0.00000 1.00000 ) *
>
```

Another example: the forensic glass data `fgl`. the data give the refractive index and oxide content of six types of glass.

```
> data(fgl)
> attach(fgl)
> summary(fgl)
       RI                 Na              Mg              Al
 Min.   :-6.8500   Min.   :10.73   Min.   :0.000   Min.   :0.290
 1st Qu.:-1.4775   1st Qu.:12.91   1st Qu.:2.115   1st Qu.:1.190
 Median :-0.3200   Median :13.30   Median :3.480   Median :1.360
 Mean   : 0.3654   Mean   :13.41   Mean   :2.685   Mean   :1.445
 3rd Qu.: 1.1575   3rd Qu.:13.82   3rd Qu.:3.600   3rd Qu.:1.630
 Max.   :15.9300   Max.   :17.38   Max.   :4.490   Max.   :3.500
       Si               K                Ca               Ba
 Min.   :69.81   Min.   :0.0000   Min.   : 5.430   Min.   :0.0000
 1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240   1st Qu.:0.0000
 Median :72.79   Median :0.5550   Median : 8.600   Median :0.0000
 Mean   :72.65   Mean   :0.4971   Mean   : 8.957   Mean   :0.1750
 3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172   3rd Qu.:0.0000
 Max.   :75.41   Max.   :6.2100   Max.   :16.190   Max.   :3.1500
       Fe              type
 Min.   :0.00000   WinF :70
 1st Qu.:0.00000   WinNF:76
 Median :0.00000   Veh  :17
 Mean   :0.05701   Con  :13
 3rd Qu.:0.10000   Tabl : 9
 Max.   :0.51000   Head :29
> fgl.tr<-tree(type~.,fgl)
> summary(fgl.tr)

Classification tree:
tree(formula = type ~ ., data = fgl)
Number of terminal nodes:  20
Residual mean deviance:  0.6853 = 133 / 194
Misclassification error rate: 0.1542 = 33 / 214
> plot(fgl.tr)
> text(fgl.tr,all=T,cex=0.5))
Error: syntax error
> text(fgl.tr,all=T,cex=0.5)
```



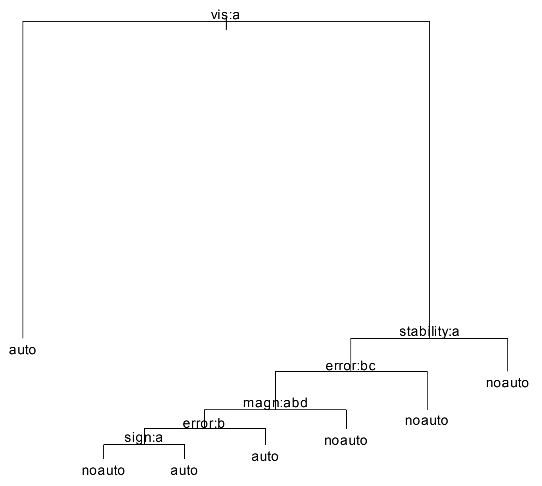
Decision Trees:

One common use of classification trees is as an aid to decision making — not really different from classification but sometimes distinguished.

Data shuttle gives guidance on whether to use autolander or manual control on landing the space shuttle under various conditions such as head or tail wind of various strengths, good or poor visibility (always use auto in poor visibility!) etc, 6 factors in all. There are potentially 256 combinations of conditions and these can be tabulated and completely enumerated but displaying the correct decision as a tree is convenient and attractive.

```
> data(shuttle)
> attach(shuttle)
> summary(shuttle)
 stability   error   sign      wind        magn       vis         use
 stab :128   LX:64   nn:128   head:128   Light :64   no :128   auto  :145
 xstab:128   MM:64   pp:128   tail:128   Medium:64   yes:128   noauto:111
             SS:64                       Out   :64
             XL:64                       Strong:64
> table(use,vis)
        vis
use       no yes
  auto   128  17
  noauto   0 111
> table(use,wind)
        wind
use       head tail
  auto      72   73
  noauto    56   55
```

```
> table(use,magn,wind)
, , wind = head

        magn
use       Light Medium Out Strong
  auto       19     19  16     18
  noauto     13     13  16     14

, , wind = tail

        magn
use       Light Medium Out Strong
  auto       19     19  16     19
  noauto     13     13  16     13

> shuttle
    stability error sign wind   magn vis    use
1       xstab    LX   pp head  Light  no   auto
2       xstab    LX   pp head Medium  no   auto
3       xstab    LX   pp head Strong  no   auto
…         …         …         …         …         …
…         …         …         …         …         …
…         …         …         …         …         …
…         …         …         …         …         …
255      stab    MM   nn head Medium yes noauto
256      stab    MM   nn head Strong yes noauto
>
> shuttle.tr<-tree(use~.,shuttle)
> plot(shuttle.tr)
> text(shuttle.tr)
```



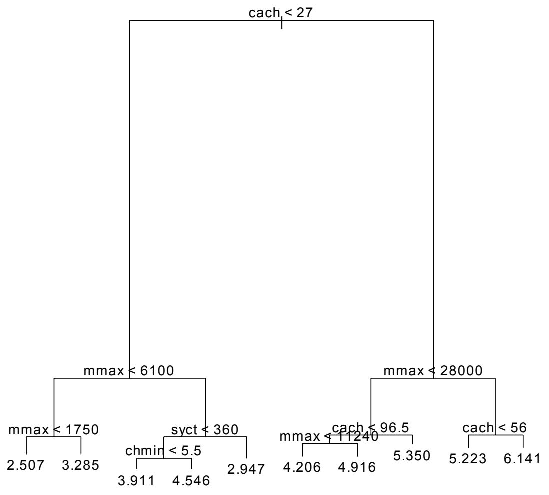
In this default display, the levels of the factors are indicated by a,b, . alphabetically and the tree is read so that levels indicated are to the left branch and others to the right, e.g. at the first branching `vis:a` indicates `no` for the left branch and `yes` for the right one. At the branch labelled `magn:abd` the right branch is for level c which is '`out of range`'; all other levels take the left branch. The plot can of course be enhanced with better labels.

Regression Trees:

We can think of classification trees as modelling a ***discrete*** factor or outcome as depending on various explanatory variables, either continuous or discrete**.** For example, the *iris species* depended upon values of the continuous variables giving the dimensions of the sepals and petals. In an analogous way we could model a ***continuous*** outcome on explanatory variables using tree-based methods, i.e. *regression trees*. The analysis can be thought of as categorizing the continuous outcome into discrete levels, i.e. turning the continuous outcome into a discrete factor. The number of distinct levels can be controlled by specifying the minimum number of observations (`minsize`) at a node that can be split and the reduction of variance produced by splitting a node (`mindev`). This is illustrated on the hills data, but first an example of data on c.p.u. performance of 209 different processors in data set `cpus` contained in the `MASS` library. The measure of performance is perf and we model the log of this variable.

```
> library(MASS)
> library(tree)
> data(cpus)
> attach(cpus)
> summary(cpus)
                    name           syct             mmin              mmax      
 WANG VS10             :  1   Min.   :  17.0   Min.   :   64   Min.   :   64  
 WANG VS 90            :  1   1st Qu.:  50.0   1st Qu.:  768   1st Qu.: 4000  
 STRATUS 32            :  1   Median : 110.0   Median : 2000   Median : 8000  
 SPERRY 90/80 MODEL 3:  1   Mean   : 203.8   Mean   : 2868   Mean   :11796  
 SPERRY 80/8           :  1   3rd Qu.: 225.0   3rd Qu.: 4000   3rd Qu.:16000  
 SPERRY 80/6           :  1   Max.   :1500.0   Max.   :32000   Max.   :64000  
 (Other)               :203                                                      
      cach            chmin            chmax             perf            estperf     
 Min.   :  0.00 Min.   : 0.000 Min.   :  0.00 Min.   :   6.0 Min.   :  15.0  
 1st Qu.:  0.00 1st Qu.: 1.000 1st Qu.:  5.00 1st Qu.:  27.0 1st Qu.:  28.0  
 Median :  8.00 Median : 2.000 Median :  8.00 Median :  50.0 Median :  45.0  
 Mean   : 25.21 Mean   : 4.699 Mean   : 18.27 Mean   : 105.6 Mean   :  99.3  
 3rd Qu.: 32.00 3rd Qu.: 6.000 3rd Qu.: 24.00 3rd Qu.: 113.0 3rd Qu.: 101.0  
 Max.   :256.00 Max.   :52.000 Max.   :176.00 Max.   :1150.0 Max.   :1238.0  
```

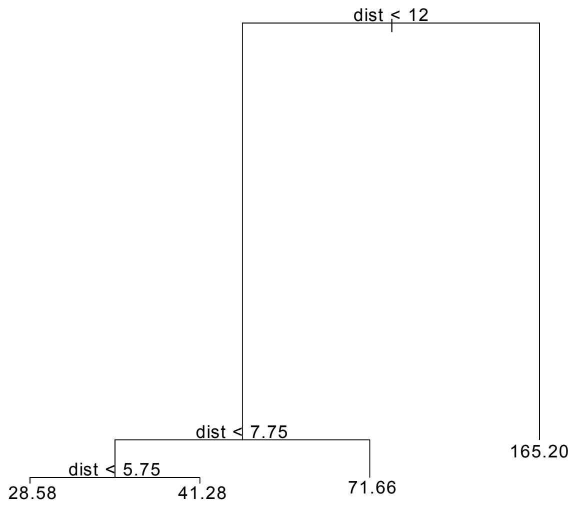
```
> cpus.tr<-tree(log(perf)~.,cpus[,2:8])
> plot(cpus.tr)
> text(cpus.tr)
```



The attraction of the display is that it gives a quick way of predicting cpu performance for a processor with specified characteristics. The accuracy of the predictions can be increased by increasing the number of terminal nodes (or leaves). However, this does not offer a substitute for more investigative modelling and outlier identification.

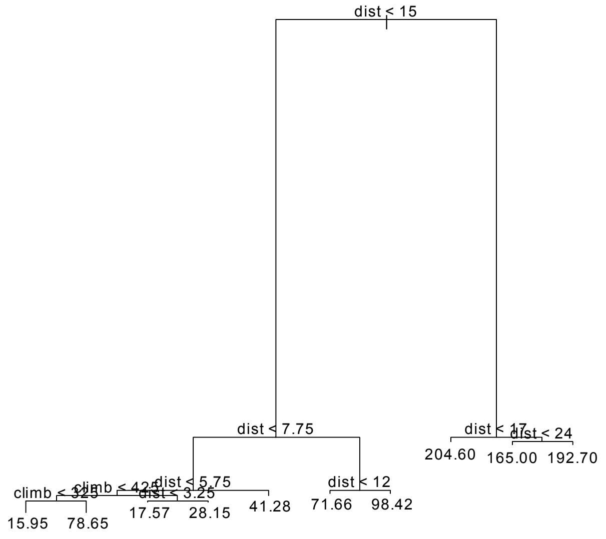
Now we illustrate some of the refinements with the more familiar Scottish Hill race data `hills`

```
> data(hills)
> hills.tr<-tree(time~.,hills)
> plot(hills.tr)
> text(hills.tr,cex=1.25)
```



Next, increase the number of terminal nodes, i.e. increase the resolution of forecasting.

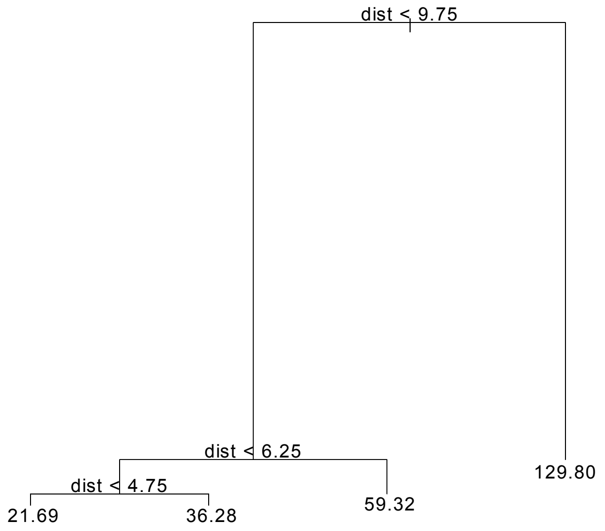
```
> hills.tr1<-tree(time~.,hills,control=tree.control(nobs=35,
+ minsize=2,mindev=.003))
> plot(hills.tr1)
> text(hills.tr1)
```



With function `tree`, the maximum number of terminal nodes is 32.

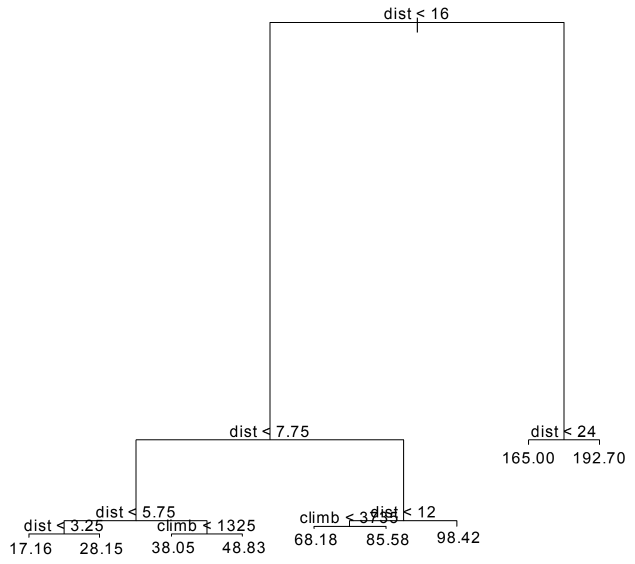
Next, for comparison we repeat the above analyses removing the known outliers observations 7 and 18.

```
> hills.tr2<-tree(time~.,hills[-c(7,18),])
> plot(hills.tr2)
> text(hills.tr2,cex=1.25)
```



And with more nodes:

```
> hills.tr3<-tree(time~., hills[-c(7,18),],
+ control=tree.control(nobs=33,
+ minsize=2,mindev=.003))
> plot(hills.tr3)
> text(hills.tr3)
```



It is clear that there differences in the trees after dropping the outliers but these are not quite so great as they might appear, taking into account the resolution of the end nodes.

More refined facilities are available in library `rpart` (pruning etc).

7 Neural Networks

7.1 Introduction

We have seen how we can consider classification and discrimination problems as a form of modelling the relationship between a categorical variable and various explanatory variables. We could make this more explicit and use, for example, logistic regression techniques. For example, suppose we have two categories, A and B, and explanatory variables x_1, ,x_k then we could model the probability that an object with values of x_1, ,x_k belongs to category A as a logistic function of the x_1, ,x_k:

$$\text{P[belongs to A]} = \frac{\exp\{\alpha + \beta_1 x_1 + \ldots + \beta_k x_k\}}{1 + \exp\{\alpha + \beta_1 x_1 + \ldots + \beta_k x_k\}}$$

and then estimate the unknown parameters β_i from training data on objects with known classifications. New observations would be classified by classifying them as of type A if the estimated probability of belonging to A is > 0.5, otherwise classify them as of type B. The technique is widely used and is very effective, it is known as *logistic discrimination*. It can readily handle cases where the x_i are a mixture of continuous and binary variables. If there is an explanatory variable whish is categorical with k>2 levels then it needs to be replaced by k–1 dummy binary variables (though this step can be avoided with tree-based methods).

The idea could be extended to discrimination and classification with several categories, *multiple logistic discrimination.*

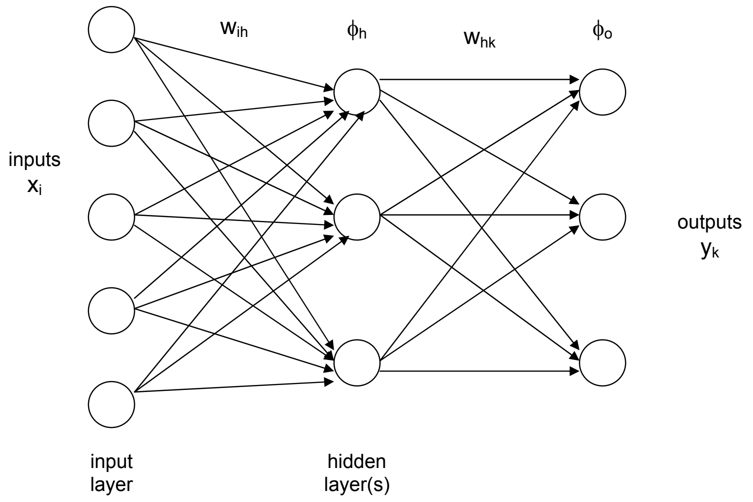
Neural networks, from a statistical point of view, can be thought of as a further extension of the idea and special cases of them are essentially non-linear logistic models. However, the technique is rather more general than just non-linear logistic modelling. It also has analogies with generalized additive modelling (mentioned very briefly on p97).

The full model for a feed-forward neural network with one hidden layer is

$$y_k = \phi_0\left(\alpha_k + \sum_h w_{hk}\phi_h(\alpha_h + \sum_i w_{ih}x_i)\right)$$

where the 'inputs' are x_i (i.e. values of explanatory variables), the 'outputs' are y_k (i.e. values of the dependent variable, and the α_j and w_{ij} are unknown parameters which have to be estimated (i.e. the network has to be 'trained') by minimising some fitting criterion, e.g. least squares or a measure of entropy. The functions ϕ_j are 'activation functions' and are often taken to be the logistic function $\phi(x)=\exp(x)/\{1+\exp(x)\}$. The w_{ij} are usually thought of as *weights* feeding forward input from the observations through a 'hidden layer' of units (ϕ_h) to output units which also consist of activation functions ϕ_o.

The model is often represented graphically as a set of inputs linked through a hidden layer to the outputs:



The number of inputs is the number of explanatory variables x_i, the number of outputs is the number of levels of y_k (if y_k is categorical), or the dimension of y_k (if y_k is continuous) and the number of 'hidden units' is open to choice. The greater the number of hidden units the larger the number of parameters to be estimated and (generally) the better will be the fit of the predicted y_k with the observed y_k.

7.2 Examples

These next two examples are taken from the `help(nnet)` output and are illustrations on the iris data yet again, this time the classification is based on (i.e. the neural network is trained on) a random 50% sample of the data and evaluated on the other 50%. In the first example the target values are taken to be the vectors (1,0,0), (0,1,0) and (0,0,1) for the three species (i.e. indicator variables) and we classify new data (i.e. with new values of the sepal and petal measurements) by which column has the maximum estimated value.

```
> library(nnet)
> data(iris3)
># use half the iris data
> ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])
> targets <-class.ind(c(rep("s",50),rep("c",50),rep("v",50)))
> samp<-c(sample(1:50,25),sample(51:100,25),
+ sample(101:150,25))
>ir1 <- nnet(ir[samp,], targets[samp,], size=2, rang=0.1,
+                    decay=5e-4, maxit=200)
# weights:  19
initial  value 54.827508
iter  10 value 30.105123
iter  20 value 18.718125
… … … … … … … … … … … …
… … … … … … … … … … … …
iter 200 value 0.532392
final  value 0.532392
stopped after 200 iterations
>       test.cl <- function(true, pred){
+               true <- max.col(true)
+               cres <- max.col(pred)
+               table(true, cres)
+       }
>       test.cl(targets[-samp,], predict(ir1, ir[-samp,]))
    cres
true  1  2  3
   1 24  0  1
   2  0 25  0
   3  2  0 23
```

Thus, the classification rule only misclassifies 3 out of the 75 flowers which were not used in the analysis. If we used a net with only 1 unit in the hidden layer:

```
> ir1 <- nnet(ir[samp,], targets[samp,], size=1, rang=0.1,
+             decay=5e-4, maxit=200)
# weights:  11
initial  value 57.220735
iter  10 value 35.168339
…    …    …    …    …    …
iter  60 value 17.184611
final  value 17.167133
converged
>      test.cl <- function(true, pred){
+              true <- max.col(true)
+              cres <- max.col(pred)
+              table(true, cres)
+      }
>      test.cl(targets[-samp,], predict(ir1, ir[-samp,]))
    cres
true  1  2  3
   1 22  0  3
   2  0 25  0
   3  0  0 25
>
```

then it is still only 3, though a different 3 clearly. To see what the actual values of the predictions are we can print the first five rows of the estimated target values:

```
> predict(ir1, ir[-samp,])[1:5,]
             c         s v
[1,] 0.1795149 0.9778684 0
[2,] 0.1822938 0.9747983 0
[3,] 0.1785939 0.9788104 0
[4,] 0.1758644 0.9813966 0
[5,] 0.1850007 0.9714523 0
```

and we see that although it does not estimate the values as precisely (0,1,0) (or (1,0,0) or (0,0,1)) they are close. Hence the use of the `mac.col` function above.

We can find out more about the actual fitted (or trained) network, including the estimated weights with `summary()` etc:

```
> ir1
a 4-1-3 network with 11 weights
options were - decay=5e-04
> summary(ir1)
a 4-1-3 network with 11 weights
options were - decay=5e-04
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 -0.15    0.41   0.74  -1.01  -1.18
 b->o1 h1->o1
 -0.06  -1.59
 b->o2 h1->o2
 -6.59  11.28
 b->o3 h1->o3
  3.75 -39.97
```

and we could draw a graphical representation putting in values of the weights along the arrows.

Another way of tackling the same problem is given by the following:

```
> ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
+          species=c(rep("s",50), rep("c", 50), rep("v", 50)))
>      ir.nn2 <- nnet(species ~ ., data=ird, subset=samp,
+ size=2, rang=0.1,  decay=5e-4, maxit=200)
# weights:  19
initial  value 82.614238
iter  10 value 27.381769
…    …    …    …    …
iter 200 value 0.481454
final  value 0.481454
stopped after 200 iterations
>table(ird$species[-samp], predict(ir.nn2, ird[-samp,],
type="class"))
     c  s  v
  c 24  0  1
  s  0 25  0
  v  2  0 23
```

again, 3 of the new data are misclassified. However, if try a net with only one hidden unit we actually succeed slightly better:

```
>  ir.nn2 <- nnet(species ~ ., data=ird, subset=samp, size=1,
+ rang=0.1, decay=5e-4, maxit=200)
# weights:  11
initial  value 82.400908
final  value 3.270152
converged
>    table(ird$species[-samp], predict(ir.nn2, ird[-samp,],
+ type="class"))

     c  s  v
  c 24  0  1
  s  0 25  0
  v  1  0 24


> summary(ir.nn2)
a 4-1-3 network with 11 weights
options were - softmax modelling  decay=5e-04
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 -1.79  -0.44  -0.91   1.05   1.65
 b->o1 h1->o1
  7.11  -0.99
 b->o2 h1->o2
 12.30 -36.31
 b->o3 h1->o3
-19.45  37.43
```

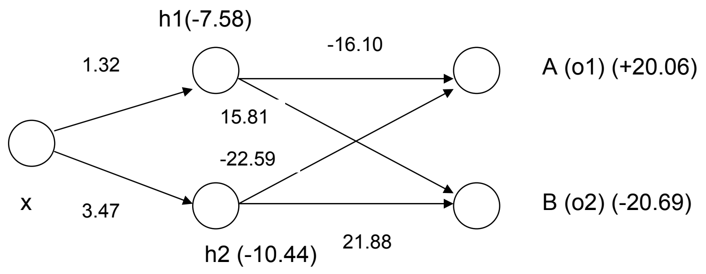
Exercise: Try modifying the **R** commands above to train a network on a much smaller sample, say 10 from each species, and the classifying the remainder. This can be done by changing the 25 to 10 in each of the three sample commands on P171. (I found that the misclassification rate on the new data was 6 out of 120 and even with training samples of 5 from each species it was 8 out of 135).

Simple Example:

This is an artificial example: the objective is to train a network with a hidden layer containing two units to return a value A for low numbers and a value B for high ones. The following code sets up a dataframe (`nick`) which has 8 rows and two columns. The first column has the values of `x` and the second the targets. The first five rows will be used for training the net and the last three will be fed into the trained net for classification, so the first 3 rows have low values of `x` and `target` value `A`, the next 2 rows have high values of `x` and the `target` value `B` and the final 3 rows have test values of `x` and unknown classifications.

```
> library(nnet) # open nnet library
> nick<-
+ data.frame(x=c(1.1,1.7,1.3,5.6,7.2,8.1,1.8,3.0),
+ targets=c(rep("A",3),rep("B",2),rep("U",3)))
> attach(nick)
# check dataframe is ok
> nick
    x   targets
1   1.1       A
2   1.7       A
3   1.3       A
4   5.6       B
5   7.2       B
6   8.1       U
7   1.8       U
8   3.0       U
> nick.net<-nnet(targets~.,data=nick[1:5,],size=2)
# weights:  10
initial  value 3.844981
final  value 0.039811
converged
Warning message:
group(s) U are empty in: nnet.formula(targets ~ .,
data = nick[1:5, ], size = 2)
```

```
# check predictions on training data
> predict(nick.net,nick[1:5,],type="class")
[1] "A" "A" "A" "B" "B"
# now classify new data
> predict(nick.net,nick[6:8,],type="class")
[1] "B" "A" "A"
# see what the predictions look like numerically
> predict(nick.net,nick[6:8,])
             A            B
6 1.364219e-15 1.000000e+00
7 1.000000e+00 4.659797e-18
8 1.000000e+00 1.769726e-08
> predict(nick.net,nick[1:5,])
             A            B
1 1.000000e+00 2.286416e-18
2 1.000000e+00 3.757951e-18
3 1.000000e+00 2.477393e-18
4 1.523690e-08 1.000000e+00
5 2.161339e-14 1.000000e+00
>
# look at estimates of weights.
> summary(nick.net)
a 1-2-2 network with 10 weights
options were - softmax modelling
 b->h1 i1->h1
 -7.58   1.32
 b->h2 i1->h2
-10.44   3.47
 b->o1 h1->o1 h2->o1
 20.06 -16.10 -22.59
 b->o2 h1->o2 h2->o2
-20.69  15.81  21.88
```



7.3 Extended example: Data set Book

Data set `book.txt` is available from the WebCT Datasets page. If you ***right-click*** on the filename then you can download the file onto your hard disk. Suppose you have downloaded the file to file `book.txt` in directory temp on your `C` drive. So its full pathname would be `C:\temp\book.txt.` Then to read it into **R** you need to do

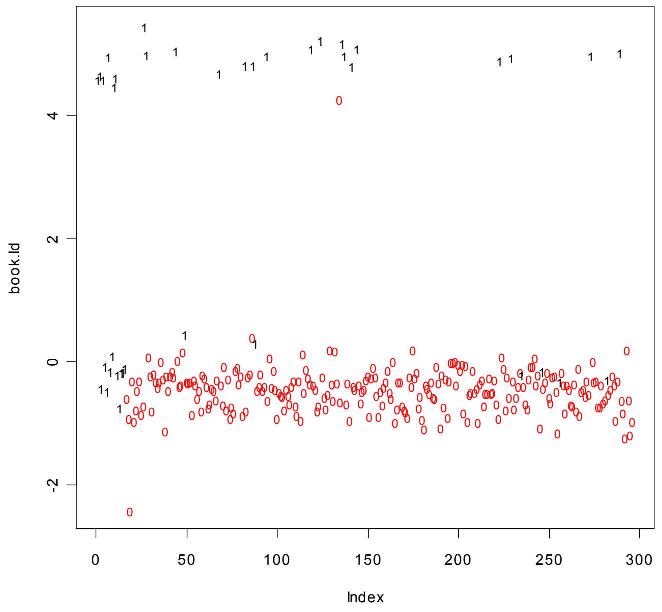
```
> book<- read.table("c:\\temp\\book.txt")
> attach(book)
```

which will make the data set and its variables accessible to the session.

NOTE the use of the double backslash \\ in the pathname.

This data set has 16 variables, plus a binary classification (QT). The variables are a mixture of continuous (5 variables), binary (8 vars) and ordinal (3). An exploratory PCA on the correlation matrix (not shewn here) on 'raw' variables (i.e. ordinal not transformed to dummy variables) indicates very high dimensionality, typical of such sets with a mixture of types. The first 6 PCs account for only 75% of variability, the first 9 for 90%. Plots on PCs indicate that there is some well-defined structure revealed on the mid-order PCs but strikingly the cases with QT=1 are clearly divided into two groups, one of which separates fairly well from cases with QT=0 but the other is interior to those with QT=0 from all perspectives.

A Linear Discriminant Analysis emphasizes that these latter points are consistently misclassified. The plot below (from R) shews the data on the first (and only) crimcoord against sequence number. There then follows various analyses using a random subset to classify the remainder using both LDA and various simple neural nets. In this exercise LDA appears to win. Again the ‘raw’ variables are used for illustration but recoding would be better.



Plot of Book data on first discriminant (*vs* index in data file)

```
         Predicted
             0   1
true   0  256   1
       1   16  23
         17
```

Raw misclassification rate 17/296

Next take random samples of 200 and then use the LDF obtained to classify the remaining 96 (details of code and some output suppressed):

```
> samp<- sample(1:296,200)
> books.lda<-lda(book[samp,],qt[samp])
> table(qt[-samp],predict(book.lda,book[-samp,])$class)
                    predicted
              0  1       0  1        0  1       0  1       0  1
   true     0 82  0    0 87  0     0 84  1    0 88  0    0 84  0
            1  7  7    1  2  7     1  3  8    1  4  4    1  6  6
misclassif
 rates        7         2          4          4          6

             0   1       0   1       0  1       0  1       0  1
   true    0 81  1     0 79  0     0 81  0    0 89  0    0 83  1
           1  4 10     1  6 11     1  7  8    1  2  5    1  4  8
misclassify
 rates        5         6           7          2          5
```

i.e. overall about 5%

Now try a neural net with 8 hidden units:

```
> book.net<-nnet(book,qt,size=8,rang=0.1,decay=5e-4,maxit=200)
> q<-class.ind(qt)
> book.net<-nnet(book,q,size=8,rang=0.1,decay=5e-4,maxit=200)
> book.net
a 16-8-2 network with 154 weights
options were - decay=5e-04
> test.cl(q,predict(book.net,book))
    pred
true    1   2
   1 257   0
   2  11  28
    11
```

Raw misclassification rate 11/96, now again with 15 hidden units:

```
> book.net<-nnet(book,q,size=15,rang=0.1,decay=5e-4,maxit=200)
# weights:  287
> test.cl(q,predict(book.net,book))
    pred
true   1  2
   1 257  0
   2   9 30
     9
```

Raw misclassification rate 9/96

Now again with 20 hidden units:

```
> book.net<-nnet(book,q,size=20,rang=0.1,decay=5e-4,maxit=200)
# weights:  382
> test.cl(q,predict(book.net,book))
    pred
true   1  2
   1 257  0
   2   4 35
      4
```

Raw misclassification rate 4/96

Now try training the net on 200 randomly selected cases and classify the remaining 96.

```
> book.net<-
nnet(book[samp,],q[samp,],size=20,rang=0.1,decay=5e-
4,maxit=200)
# weights:  382
> test.cl(q[-samp,],predict(book.net,book[-samp,]))
    pred         pred         pred          pred         pred
true  1 2    true  1 2    true  1  2    true  1 2    true  1 2
   1 82 2       1 77 9       1 73 10       1 79 9       1 77 5
   2  6 6       2  5 5       2  6  7       2  3 5       2  8 6
misclassif
 rates
      8          14          16            12           13
```

```
    pred         pred         pred         pred         pred
true  1 2    true  1 2    true  1 2    true  1 2    true  1 2
   1 81 4       1 81 1       1 88 2       1 81 9       1 85 5
   2  6 5       2  7 7       2  2 4       2  2 4       2  2 4

misclassif
 rates
    10           8            4            11           7

   pred         pred
true  1 2    true  1  2
   1 86 4       1 80 10
   2  2 4       2  2  4
misclassif
 rates
     6           12
```

i.e. overall about 10%

Next, try this again with only 5 hidden units:

```
> book.net<-
nnet(book[samp,],q[samp,],size=5,rang=0.1,decay=5e-
4,maxit=300)
# weights:  97
    pred         pred         pred         pred         pred
true  1 2    true  1 2    true  1 2    true  1 2    true  1 2
   1 85 5       1 82 4       1 77 7       1 82 2       1 77 8
   2  2 4       2  3 7       2  6 6       2  7 5       2  4 7
misclassif
 rates
     7           7            13           9            12
    pred         pred         pred         pred          pred
true  1 2    true  1 2    true  1 2    true  1  2    true  1 2
   1 76 5       1 78 5       1 79 5       1 75 10       1 78 6
   2  8 7       2  6 7       2  3 9       2  6  5       2  6 6
 misclassif
 rates
    13           11           8            16           12
>
```

i.e. overall about 11%

7.4 Summary

The above account is only a very brief introduction to simple neural networks, with particular reference to their use for classification. As with tree-based methods they can also be used for regression problems. Little has been said about the use and choice of activation functions, fitting criteria etc and examples have been given entirely in the context of the simple and basic facilities offered in the `nnet` library of **R**. To find out more then look at the reference Ripley (1996) given on p1, this is written with statistical terminology and largely from a statistical point of view. Another definitive reference is Chris Bishop (1995), *Neural Networks for Pattern Recognition,* Oxford, Clarendon Press.

8 Concluding Remarks

The sections above have been intended to give an introduction to and a flavour of the more practical and intuitive methods which are used by practicing applied statisticians in their day-to-day work. Some of the techniques are used only at early or intermediate stages of the investigation into obtaining understanding and insight into the data, some are used to provide the end-points of the analysis and would be used in the final report. The account given above is inevitably selective and incomplete, many important areas have not been mentioned (e.g. time series analysis, spatial statistics, survival data,) nor have all the available techniques been covered within those areas that have been touched upon. In part this is because the account is based around the facilities offered in one particular computer package or language **R**. What should have become apparent is that this package itself contains a great deal of instructional material and example data sets that should encourage you to try out new and unfamiliar methods. Use the code given on worked examples in the help system, try modifying it and seeing what happens. Remember the maxim (Aristotle) that:

‘for the things we have to know before we can do them, we learn by doing them’.

This is a quotation given in the start of a book *Applied Stochastic Modelling* by Byron Morgan (2000), another good book to read.

NRJF

n.fieller@shef.ac.uk
http://www.shef.ac.uk/nickfieller