Data Analysis with R Laboratories – Sets of Exercises, with R Code

John Maindonald

August 24, 2007

Laboratory exercises make extensive use of datasets from the *DAAG* and *DAAGxtras* packages. Make sure that they are installed.

For background to laboratory exercises I – II, see the document: *The R System – An Introduction and Overview.*

Contents

I R Basics **5**

1 Data Input **5**

2 Data Input from a Web Page **5**

3 The `paste()` Function; Further Details **5**

4 Missing Values **6**

5 Subsets of Dataframes **6**

6 Scatterplots **6**

7 Factors **8**

8 Stripcharts (base graphics) and Stripplots (*lattice*) **9**

9 Tabulation **9**

10 Sorting **10**

11 Use of a For Loop **10**

12 A Function **10**

II Practice with R **11**

1 Information about the Columns of Data Frames **11**

2 Data Input **11**

3 A Tabulation Exercise **11**

4 A For Loop **12**

5 Data Exploration – Distributions of Data Values 12

6 Random Samples 13

7 Smooth Curves 14

8 Information on Workspace Objects 14

9 Different Ways to Do a Calculation – Timings 15

10 Functions – Making Sense of the Code 16

11 *Use of `sapply()` to Give Multiple Graphs 17

12 *The Internals of R – Functions are Pervasive 17

III Informal and Formal Data Exploration 19

1 Informal Data Exploration 19

2 Bootstrap sampling 21

IV Models and Model Accuracy 23

1 Straight Line Regression 23

2 Analysis Using *randomForest* 27

V Ordination 29

1 Australian road distances 29

2 If distances must first be calculated . . . 31

3 Genetic Distances 31

4 *Distances between fly species 33

5 *Rock Art 34

VI Discriminant Methods – Error Rate Estimation, & Low-Dimensional Views 35

1 rpart Analyses – the Combined `Pima` Dataset 35

2 rpart Analyses – `Pima.tr` and `Pima.te` 37

3 Analysis Using *svm* 40

4 Analysis Using *randomForest* 40

5 Class Weights 41

6 Plots that show the “distances” between points 41

Part I
R Basics

1 Data Input

Exercise 1
Ensure that the *DAAGxtras* package is installed. From the R command line, attach it, and type `dataFile(c("molclock1", "molclock2"))`, thus:

```
> library(DAAGxtras)
> dataFile(c("molclock1", "molclock2"))  # NB dataFile, not datafile
```

This places the files **molclock1.txt** and **molclock2.txt** in the working directory. Use `file.show()` to examine the contents of each of these files. Alternatively, you can use R's script editor (under Windows, go to File | Open script...), or use another editor such as the Windows tinn-R editor that is designed to interface to R.

Use `read.table()` to read each of them into R. Check carefully whether you need `header=TRUE`. Then display the data frame and check that the data have been input correctly.

Note: If the file is located in a directory other than the working directory a fully specified file name, including the path, is necessary. For example, to input a file **travelbooks.txt** that has been placed in the directory **c:/datafiles/**, type

```
> travelbooks <- read.table("c:/datafiles/travelbooks.txt")
```

For input to R functions, forward slashes replace backslashes.

2 Data Input from a Web Page

Exercise 2
Files can be read directly from a web page, providing that there is a live internet connection, Here are examples:

```
> webfolder <- "http://www.maths.anu.edu.au/~johnm/datasets/text/"
> webpage <- paste(webfolder, "molclock.txt", sep="")
> molclock <- read.table(url(webpage))
```

Use this approach to input the file **travelbooks.txt** that is available from this same web page.

3 The `paste()` Function; Further Details

Exercise 3
Here are further examples that illustrate the use of `paste()`:

```
> paste("Leo", "the", "lion")
> paste("a", "b")
> paste("a", "b", sep="")
> paste(1:5)
> paste(1:5, collapse="")
```

4 Missing Values

Exercise 4
The following counts, for each species, the number of missing values for the column `root` of the data frame `rainforest` (*DAAG*):

```
> library(DAAG)
> with(rainforest, table(complete.cases(root), species))
```

For each species, how many rows are "complete", i.e., have no values that are missing?

Exercise 5
For each column of the data frame `Pima.tr2` (*MASS*), determine the number of missing values.

5 Subsets of Dataframes

Exercise 6
Use `head()` to check the names of the columns, and the first few rows of data, in the data frame `rainforest` (*DAAG*). Use `table(rainforest$species)` to check the names and numbers of each species that are present in the data. The following extracts the rows for the species *Acmena smithii*

```
> library(DAAG)
> Acmena <- subset(rainforest, species=="Acmena smithii")
```

The following extracts the rows for the species `Acacia mabellae` and `Acmena smithii`

```
> AcSpecies <- subset(rainforest, species %in% c("Acacia mabellae",
+                                                "Acmena smithii"))
```

Now extract the rows for all species except `C. fraseri`.

Exercise 7
Extract the following subsets from the data frame `ais` (*DAAG*):

(a) Extract the data for the rowers.

(b) Extract the data for the rowers, the netballers and the tennis players.

(c) Extract the data for the female basketabllers and rowers.

6 Scatterplots

Exercise 8
Using the Acmena data from the data frame `rainforest`, plot `wood` (wood biomass) vs `dbh` (diameter at breast height), trying both untransformed scales and logarithmic scales. Here is suitable code:

Exercise 8, continued

```
> Acmena <- subset(rainforest, species=="Acmena smithii")
> plot(wood ~ dbh, data=Acmena)
> plot(wood ~ dbh, data=Acmena, log="xy")
```

Use of the argument `log="xy"` gives logarithmic scales on both the x and y axes. For purposes of adding additional features to the plot, note that logarithms to base 10 are used.
For the second plot, add a fitted line, thus:

```
> plot(wood~dbh, data=Acmena, log="xy")
> ## The lm() command will fit a line; more details later
> ## abline() then plots this line.
> Acmena10.lm <- lm(log10(wood) ~ log10(dbh), data=Acmena)
> abline(Acmena10.lm)

> ## Now print the coefficents, for a log10 scale
> coef(Acmena10.lm)
> ## For comparison, print the coefficients for a natural log scale
> Acmena.lm <- lm(log(wood) ~ log(dbh), data=Acmena)
> coef(Acmena.lm)
```

Write down the equation that gives the fitted relationship between `wood` and `dbh`.

Exercise 9
The `orings` data frame gives data on the damage that had occurred in US space shuttle launches prior to the disastrous Challenger launch of January 28, 1986. Only the observations in rows 1, 2, 4, 11, 13, and 18 were included in the pre-launch charts used in deciding whether to proceed with the launch. Add a new column to the data frame that identifies rows that were included in the pre-launch charts. Now make three plots of `Total` incidents against `Temperature`:

(a) Plot only the rows that were included in the pre-launch charts.

(b) Plot all rows.

(c) Plot all rows, using different symbols or colors to indicate whether or not points were included in the pre-launch charts.

Comment, for each of the first two graphs, whether and open or closed symbol is preferable. For the third graph, comment on the your reasons for choice of symbols.

Use the following to identify rows that hold the data that were presented in the pre-launch charts:

```
> orings$Included <- logical(23)  # orings has 23 rows
> orings$Included[c(1,2,4,11,13,18)] <- TRUE
```

The construct `logical(23)` creates a vector of length 23 in which all values are `FALSE`. The following are two possibilities for the third plot; can you improve on these choices of symbols and/or colors?

```
> plot(Total ~ Temperature, data=orings, pch=orings$included+1)
> plot(Total ~ Temperature, data=orings, col=orings$Included+1)
```

Exercise 10
Using the data frame `oddbooks`, use graphs to investigate the relationships between:

(a) weight and volume;

(b) density and volume;

(c) density and page area.

7 Factors

Exercise 11
Investigate the use of the functions `as.character()` and `unclass()` with a factor argument. Comment on their use in the following code:

```
> par(mfrow=c(1,2), pty="s")
> plot(weight ~ volume, pch=unclass(cover), data=allbacks)
> plot(weight ~ volume, data=allbacks, type="n")
> with(allbacks, text(weight ~ volume, labels=as.character(cover)))
> par(mfrow=c(1,1))
```

[The setting `mfrow=c(1,2)` gives side by side plots. The setting `pty="s"` gives a square plotting region.]

Exercise 12
Run the following code:

```
> gender <- factor(c(rep("female", 91), rep("male", 92)))
> table(gender)
> gender <- factor(gender, levels=c("male", "female"))
> table(gender)
> gender <- factor(gender, levels=c("Male", "female")) # Note the mistake
>                              # The level was "male", not "Male"
> table(gender)
> rm(gender)                   # Remove gender
```

Explain the output from the final `table(gender)`.
The output is

```
gender
female   male
    91     92

> table(gender)
> gender <- factor(gender, levels=c("Male", "female")) # Note the mistake
>                              # The level was "male", not "Male"
> table(gender)
> rm(gender)                   # Remove gender
```

8 Stripcharts (base graphics) and Stripplots (*lattice*)

Exercise 13
Look up the help for the lattice function `dotplot()`.
Compare the following:

```
> ## First, use the regular graphics function stripchart()
> with(ant111b, stripchart(harvwt ~ site))
> ## Next, use lattice graphics
> library(lattice)
> stripplot(site ~ harvwt, data=ant111b)
> ## Next, use lattice graphics, but switch the x and y axes
> library(lattice)
> stripplot(site ~ harvwt, data=ant111b)
```

Note the differences in syntax between the two graphics systems.

Exercise 14
Check the class of each of the columns of the data frame `cabbages` (*MASS*). Do side by side plots of `HeadWt` against `Date`, for each of the levels of `Cult`.

```
> stripplot(Date ~ HeadWt | Cult, data=cabbages)
```

As the lattice graphics `stripplot()` function allows you to do a lot more than `stripchart()`, and as the lattice syntax is highly consistent across the different *lattice* functions, it seems best to use `stripplot()`.

Exercise 15
In the data frame `nsw74psid3`, use `stripplot()` to compare, between levels of `trt`, the continuous variables `age`, `educ`, `re74` and `re75`
It is possible to generate all the plots at once, side by side. A simplified version of the plot is:

```
> stripplot(trt ~ age + educ, data=nsw74psid1, outer=T, scale="free")
```

What are the respective effects of `scale = "free"`, and `outer = TRUE`? (Try leaving these at their defaults.)

9 Tabulation

Exercise 16
In the data set `nsw74psid3`, compare for each of the two levels of `trt`:

(a) the relative numbers of `black`;

(b) the relative numbers of hispanics (`hisp`);

(c) the relative numbers of married (`marr`).

10 Sorting

Exercise 17
Sort the rows in the data frame `Acmena` in order of increasing values of `dbh`.
[Hint: Use the function `order()`, applied to `age` to determine the order of row numbers required to sort rows in increasing order of age. Reorder rows of `Acmena` to appear in this order.]

```
> Acmena <- subset(rainforest, species=="Acmena smithii")
> ord <- order(Acmena$dbh)
> acm <- Acmena[ord, ]
```

Sort the row names of `possumsites` (*DAAG*) into alphanumeric order. Reorder the rows of `possumsites` in order of the row names.

11 Use of a For Loop

Exercise 18

(a) Create a `for` loop that, given a numeric vector, prints out one number per line, with its square and cube alongside.

(b) Look up `help(while)`. Show how to use a `while` loop to achieve the same result.

(c) Show how to achieve the same result without the use of an explicit loop.

12 A Function

Exercise 19
The following function calculates the mean and standard deviation of a numeric vector.

```
> meanANDsd <- function(x){
+     av <- mean(x)
+     sdev <- sd(x)
+     c(mean=av, sd = sdev) # The function returns this vector
+ }
```

Modify the function so that: (a) the default is to use `rnorm()` to generate 20 random normal numbers, and return the standard deviation; (b) if there are missing values, the mean and standard deviation are calculated for the remaining values.

Part II
Practice with R

1 Information about the Columns of Data Frames

Exercise 1
Functions that may be used to get information about data frames include `str()`, `dim()`, `row.names()` and `names()`. Try each of these functions with the data frames `allbacks`, `ant111b` and `tinting` (all in *DAAG*).
For getting information about the class of each column use e.g.

```
> library(DAAG)
> sapply(ant111b, class)
```

or

```
> unlist(sapply(ant111b, class))
```

This applies the function `class()` to each column of the data frame.
For each of these data frames, use `table()` to tabulate the number of values for each level.

2 Data Input

Exercise 2
The function `read.csv()` is a variant of `read.table()` that is designed to read in comma delimited files such as may be obtained from Excel. Use this function to read in the file **crx.data** that is available from the web page `http://mlearn.ics.uci.edu/databases/credit-screening/`.
Check the file **crx.names** to see which columns should be numeric, which categorical and which logical. Make sure that the numbers of missing values in each column are the number given in the file **crx.names**

For a first pass at reading in the data, try:

```
> crxpage <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.data"
> crx <- read.csv(url(crxpage), header=TRUE)
```

Exercise 3
For a challenging data input task, input the data from the file **bostonc.txt**. You can create this by attaching the *DAAG* package and entering `datafile("bostonc")` thus:

```
> datafile("bostonc")
```

Examine the contents of the initial lines of the file carefully before trying to read it in. You will need to change the parameters `comment.char` and `skip` from their defaults.

3 A Tabulation Exercise

Exercise 4
Tabulate the number of observations in each of the different districts in the data frame `rockArt` (*DAAGxtras*). Create a factor `groupDis` in which all `Districts` with less than 5 observations are grouped together into the category `other`.

```
> library(DAAGxtras)
> groupDis <- as.character(rockArt$District)
> tab <- table(rockArt$District)
> le4 <- rockArt$District %in% names(tab)[tab <= 4]
> groupDis[le4] <- "other"
> groupDis <- factor(groupDis)
```

4 A For Loop

Exercise 5
The following code uses a `for` loop to plot graphs that compare the relative population growth (here, by the use of a logarithmic scale) for the Australian states and territories.

```
> library(DAAG)
> oldpar <- par(mfrow=c(2,4))
> for (i in 2:9){
+ plot(austpop[,1], log(austpop[, i]), xlab="Year", ylab=names(austpop)[i],
+      pch=16, ylim=c(0,10))}
> par(oldpar)
```

Which Australian adminstration(s) showed the most rapid increase in the early years? Which showed the most rapid increase in later years?

5 Data Exploration – Distributions of Data Values

Exercise 6
The data frame `rainforest` (*DAAG* package) has data on four different rainforest species. Use `table(rainforest$species)` to check the names and numbers of the species present. In the following, attention will be limited to the species Acmena *smithii*.
The following plots a histogram showing the distribution of the diameter at base height:

```
> library(DAAG)       # The data frame rainforest is from DAAG
> Acmena <- subset(rainforest, species=="Acmena smithii")
> hist(Acmena$dbh)
```

By default, frequencies are used to label the the vertical axis.

An alternative is to use a density scale (`prob=TRUE`). The histogram is interpreted as a crude density plot. The density, which estimates the number of values per unit interval, changes in discrete jumps at the breakpoints (= class boundaries). The histogram can then be directly overlaid with a density plot, thus:

```
> hist(Acmena$dbh, prob=TRUE, xlim=c(0,50))   # Use a density scale
> lines(density(Acmena$dbh, from=0))
```

Why use the argument `from=0`? What is the effect of omitting it?

[Density estimates, as given by R's function `density()`, change smoothly and do not depend on an arbitrary choice of breakpoints, making them generally preferable to histograms. They do sometimes require tuning to give a sensible result. Note especially the parameter `bw`, which determines how the bandwidth is chosen, and hence affects the smoothness of the density estimate.]

6 Random Samples

Exercise 7
Histograms and density plots are, for "small" samples, notoriously variable under repeated sampling. This is true even for sample sizes as large as 50 or 100.

By taking repeated random samples from the normal distribution, and plotting the distribution for each such sample, one can get an idea of the effect of sampling variation on the sample distribution.

A random sample of 100 values from a normal distribution (with mean 0 and standard deviation 1) can be obtained, and a histogram and overlaid density plot shown, thus:

```
> y <- rnorm(100)
> hist(y, probability=TRUE)  # probability=TRUE gives a y density scale
> lines(density(y))
```

(a) Take 5 samples of size 25, then showing the plots.

(b) Take 5 samples of size 100, then showing the plots.

(c) Take 5 samples of size 500, then showing the plots.

(d) Take 5 samples of size 2000, then showing the plots.

Note: By preceding the plots with `par(mfrow=c(4,5))`, all 20 plots can be displayed on the one graphics page. (To bunch the graphs up more closely, make the further settings `par(mar=c(3.1,3.1,0.6,0.6), mgp=c(2.25,0.5,0))`)

Comment on the usefulness of a sample histogram and/or density plot for judging whether the population distribution is likely to be close to normal.

Exercise 8
This explores the function `sample()`, used to take a sample of values that are stored or enumerated in a vector. Samples may be with or without replacement; specify `replace = FALSE` (the default) or `replace = TRUE`. The parameter `size` determines the size of the sample. By default the sample has the same size (length) as the vector from which samples are taken. Take several samples of size 5 from the vector `1:5`, with `replace=FALSE`. Then repeat the exercise, this time with `replace=TRUE`. Note how the two sets of samples differ.

Exercise 9
If in Exercise 6 above a new random sample of trees could be taken, the histogram and density plot would change. How much might we expect them to change?

The boostrap approach treats the one available sample as a microcosm of the population. Repeated with replacement samples are taken from the one available sample. This is equivalent to repeating each sample value and infinite number of times, then taking random samples from the population that is thus created. The expectation is that variation between those samples will be comparable to variation between samples from the original population.

(a) Take repeated (5 or more) bootstrap samples from Acmena dataset of Exercise 6, and show the density plots. [Use `sample(Acmena$dbh, replace=TRUE)`].

(b) Repeat, now with the `cerealsugar` data from *DAAG*.

7 Smooth Curves

Exercise 10
The following compares three different smoothing functions. Comment on the different syntax and, in the case of `lowess()`, the different default output that is returned. Why, for the smooth obtained using `lowess()`, is it necessary to sort data in order of values of `dbh`? (Try omitting the ordering, and observe the result.)

```
> Acmena <- subset(rainforest, species=="Acmena smithii")
> ## Use lowess()
> plot(wood ~ dbh, data=Acmena)
> ord <- order(Acmena$dbh)
> with(Acmena[ord, ], lines(predict(loess(wood ~ dbh)) ~ dbh))
> ## Now use panel.smooth()
> plot(wood ~ dbh, data=Acmena)
> with(Acmena, panel.smooth(dbh, wood))
```

For each of the functions just noted, what are the parameters that control the smoothness of the curve? What, in each case, is the default?

8 Information on Workspace Objects

Exercise 11
An R workspace includes objects `possum1`, `possum2`, ...`possum5`. The folowing shows how to get the size of one of these objects one at a time.

```
> possum1 <- rnorm(10)
> object.size(possum1)
```

The names of the objects can be obtained with

```
> nam <- ls(pattern="^possum")
```

To get the sizes from the names that are held in `nam`, do

```
> sapply(nam, function(x)object.size(get(x)))
```

Create objects `possum2`, ...`possum5`, and enter this command. Explain the successive steps in the computation.
[Hint: Compare `class(possum1)` with `class("possum1")`, and `object.size(possum1)` with `object.size("possum1")`]

*Exercise 12**
The function `ls()` lists, by default, the names of objects in the current environment. If used from the command line, it lists the objects in the workspace. If used in a function, it lists the names of the function's local variables. To get a listing of the contents of the workspace, do the following

```
> workls <- function()ls(name=".GlobalEnv")
> workls()
```

Exercise 12, continued*

(a) If `ls(name=".GlobalEnv")` is replaced by `ls()`, the function lists the names of its local variables. Modify `workls()` so that you can use it to demonstrate this.
[Hint: Consider adapting `if(is.null(name))ls()` for the purpose.]

(b) Write a function that calculates the sizes of all objects in the workspace, then listing the names and sizes of the largest ten objects.

9 Different Ways to Do a Calculation – Timings

Exercise 13
This exercise will investigate the relative times for alternative ways to do a calculation. First, we will create both matrix and data frame versions of a largish data set.

```
> xxMAT <- matrix(runif(480000), ncol=50)
> xxDF <- as.data.frame(xxMAT)
```

The function `system.time()` will provide timings. The first three numbers that are returned will be of interest; these are the user cpu time, the system cpu time, and the elapsed time.

Repeat each calculation several times, and note whether there is variation between repeats. If there is, make the setting `options(gcFirst=TRUE)`, and see whether this leads to more consistent timings. NB: If your computer chokes on these calculations, reduce the dimensions of `xxMAT` and `xxDF`

(a) The following compares the times taken to increase each element by 1:

```
> system.time(invisible(xxMAT+1))[1:3]
> system.time(invisible(xxDF+1))[1:3]
```

(b) Now compare the following alternative ways to calculate the means of the 50 columns:

```
> ## Use apply() [matrix argument], or sapply() [data frame argument]
> system.time(av1 <- apply(xxMAT, 2, mean))[1:3]
> system.time(av1 <- sapply(xxDF, mean))[1:3]
> ## Use a loop that does the calculation for each column separately
> system.time({av2 <- numeric(50);
+             for(i in 1:50)av[i] <- mean(xxMAT[,i])
+             })[1:3]
> system.time({av2 <- numeric(50);
+             for(i in 1:50)av[i] <- mean(xxDF[,i])
+             })[1:3]
> ## Matrix multiplication
> system.time({colOFones <- rep(1, dim(xxMAT)[2])
+              av3 <- xxMAT %*% colOFones / dim(xxMAT)[2]
+             })[1:3]
```

(c) Pick one of the above calculations. Vary the number of rows in the matrix, keeping the number of columns constant, and plot each of user CPU time and system CPU time against number of rows of data.

Suggest why the calculation that uses matrix multiplication is so efficient, relative to the other options.

10 Functions – Making Sense of the Code

*Exercise 14**
Data in the data frame `fumig` (*DAAGxtras*) are from a series of fumigation trials, in which produce was exposed to the fumigant over a 2-hour time period. Concentrations in the chamber were measured at times 5, 10, 30, 60, 90 and 120 minutes. Code given following this exercise calculates a concentration-time (c-t) product that measures exposure to the fumigant, leading to the measure `ctsum`.

Examine the code in the three alternative functions given below, and the data frame `fumig` (in the `DAAGxtras` package) that is given as the default argument for the parameter `df`. Do the following:

(a) Run all three functions, and check that they give the same result.

(b) Annotate the code for `calcCT1()` to explain what each line does.

(c) Are fumigant concentration measurements noticeably more variable at some times than at others?

(d) Which function is fastest? [In order to see much difference, it will be necessary to put the functions in loops that run perhaps 1000 or more times.]

Code for 3 functions that do equivalent calculations

```
> ## Function "calcCT1"
> "calcCT1" <-
+   function(df=fumig, times=c(5,10,30,60,90,120), ctcols=3:8){
+     multiplier <- c(7.5,12.5,25,30,30,15)
+     m <- dim(df)[1]
+     ctsum <- numeric(m)
+     for(i in 1:m){
+       y <- unlist(df[i, ctcols])
+       ctsum[i] <- sum(multiplier*y)/60
+     }
+     df <- cbind(ctsum=ctsum, df[,-ctcols])
+     df
+   }
> ##
> ## Function "calcCT2"
> "calcCT2" <-
+   function(df=fumig, times=c(5,10,30,60,90,120), ctcols=3:8){
+     multiplier <- c(7.5,12.5,25,30,30,15)
+     mat <- as.matrix(df[, ctcols])
+     ctsum <- mat%*%multiplier/60
+     cbind(ctsum=ctsum, df[,-ctcols])
+   }
> ##
> ## Function "calcCT3"
> "calcCT3" <-
+   function(df=fumig, times=c(5,10,30,60,90,120), ctcols=3:8){
+     multiplier <- c(7.5,12.5,25,30,30,15)
+     mat <- as.matrix(df[, ctcols])
+     ctsum <- apply(mat, 1, function(x)sum(x*multiplier))/60
+     cbind(ctsum=ctsum, df[,-ctcols])
+   }
```

11 *Use of `sapply()` to Give Multiple Graphs

Exercise 15 (Optional extra I)*
Here is code for the calculations that compare the relative population growth rates for the Australian states and territories, but avoiding the use of a loop:

```
> oldpar <- par(mfrow=c(2,4))
> invisible(
+ sapply(2:9, function(i, df)
+        plot(df[,1], log(df[, i]),
+             xlab="Year", ylab=names(df)[i], pch=16, ylim=c(0,10)),
+             df=austpop)
+ )
> par(oldpar)
```

Run the code, and check that it does indeed give the same result as the use of an explicit loop.
[By wrapping the code in the function `invisible()`, printed output that gives no useful information can be suppressed.]
Note that `lapply()` could be used in place of `sapply()`.

There are several subtleties here:

(i) The first argument to `sapply()` can be either a list (which is, technically, a non-atomic vector) or a vector.[1] Here, we have supplied the vector 2:9

(ii) The second argument is a function. Here we have supplied an anonymous function that has two arguments. The argument `i` takes as its values, in turn, the sucessive elements in the first argument to `sapply`

(iii) Where as here the anonymous function has further arguments, they are supplied as additional arguments to `sapply()`. Hence the parameter `df=austpop`.

12 *The Internals of R – Functions are Pervasive

Exercise 16 (Optional extra II)*
This exercise peeks into the internals of the way in which R structures arithmetic and related computations. Those internals are close enough to the surface that users can experiment with their use.

The binary arithmetic operators `+`, `-`, `*`, `/` and `^` are implemented as functions. (R is a functional language; albeit with features that compromise its purity as a member of this genre!) Try the following:

```
> "+"(2,5)
> "-"(10,3)
> "/"(2,5)
> "*"("+"(5,2), "-"(3,7))
```

[1]By "vector" we usually mean an atomic vector, with "atoms" that are of one of the modes ”logical”, ”integer”, ”numeric”, ”complex”, ”character”' or ”raw”. (Vectors of mode ”raw” can for our purposes be ignored.)

Exercise 16, continued*
There are two other binary arithmetic operators – `%%` and `%/%`. Look up the relevant help page, and explain, with examples, what they do. Try

```
> (0:25) %/% 5
> (0:25) %% 5
```

Of course, these are also implemented as functions. Write code that demonstrates this.

Note also that `[` is implemented as a function. Try

```
> z <- c(2, 6, -3, NA, 14, 19)
> "["(z, 5)
> heights <- c(Andreas=178, John=185, Jeff=183)
> "["(heights, c("Jeff", "John"))
```

Rewrite these using the usual syntax.

Use this syntax to extract, from the data frame `possumsites` (*DAAG*), the altitudes for Byrangery and Conondale.

Note: Expressions in which arithmetic operators appear as explicit functions with binary arguments translate directly into postfix reverse Polish notation, introduced in 1920 by the Polish logician and mathematician Jan ÅĄukasiewicz. Postfix notation is widely used in the interpreters and compilers that translate computer language code into machine or assembly language instructions. See the Wikipedia article “Reverse Polish Notation”.

Part III
Informal and Formal Data Exploration

1 Informal Data Exploration

These exercises explore data that will later be used for exercises in error rate estimation.

Exercise 1
Look up the help page for the data frame `Pima.tr2` (*MASS* package), and note the columns in the data frame. The eventual interest is in using use variables in the first seven column to classify diabetes according to `type`. Here, we explore the individual columns of the data frame.

(a) Several columns have missing values. Analysis methods inevitably ignore or handle in some special way rows that have one or moe missing values. It is therefore desirable to check whether rows with missing values seem to differ systematically from other rows.

Determine the number of missing values in each column, broken down by `type`, thus:

```
> library(MASS)
> ## Create a function that counts NAs
> count.na <- function(x)sum(is.na(x))
> ## Check function
> count.na(c(1, NA, 5, 4, NA))
> ## For each level of type, count the number of NAs in each column
> for(lev in levels(Pima.tr2$type))
+   print(sapply(subset(Pima.tr2, type==lev), count.na))
```

The function `by()` can be used to break the calculation down by levels of a factor, avoiding the use of the `for` loop, thus:

```
> by(Pima.tr2, Pima.tr2$type, function(x)sapply(x, count.na))
```

(b) Create a version of the data frame `Pima.tr2` that has `anymiss` as an additional column:

```
> missIND <- complete.cases(Pima.tr2)
> Pima.tr2$anymiss <- c("miss","nomiss")[missIND+1]
```

For remaining columns, compare the means for the two levels of `anymiss`, separately for each level of `type`. Compare also, for each level of `type`, the number of missing values.

Exercise 2

(a) Use strip plots to compare values of the various measures for the levels of `anymiss`, for each of the levels of `type`. Are there any columns where the distribution of differences seems shifted for the rows that have one or more missing values, relative to rows where there are no missing values?
Hint: The following indicates how this might be done efficiently:

```
> library(lattice)
> stripplot(anymiss ~ npreg + glu | type, data=Pima.tr2, outer=TRUE,
+           scales=list(relation="free"), xlab="Measure")
```

Exercise 2, continued

(b) Density plots are in general better than strip plots for comparing the distributions. Try the following, first with the variable `npreg` as shown, and then with each of the other columns except `type`. Note that for `skin`, the comparison makes sense only for `type=="No"`. Why?

```
> library(lattice)
> ## npreg & glu side by side (add other variables, as convenient)
> densityplot( ~ npreg + glu | type, groups=anymiss, data=Pima.tr2,
+             auto.key=list(columns=2), scales=list(relation="free"))
```

Exercise 3

Better than either strip plots or density plots may be Q-Q plots. Using `qq()` from *lattice*, investigate their use. In this exercise, we use random samples from normal distributions to help develop an intuitive understanding of Q-Q plots, as they compare with density plots.

(a) First consider comparison using (i) a density plot and (ii) a Q-Q plot when samples are from populations in which one of the means is shifted relative to the other. Repeat the following several times,

```
> y1 <- rnorm(100, mean=0)
> y2 <- rnorm(150, mean=0.5)  # NB, the samples can be of different sizes
> df <- data.frame(gp=rep(c("first","second"), c(100,150)), y=c(y1, y2))
> densityplot(~y, groups=gp, data=df)
> qq(gp ~ y, data=df)
```

(b) Now make the comparison, from populations that have different standard deviations. For this, try, e.g.

```
> y1 <- rnorm(100, sd=1)
> y2 <- rnorm(150, sd=1.5)
```

Again, make the comparisons using both density plots and Q-Q plots.

Exercise 4

Now consider the data set `Pima.tr2`, with the column `anymiss` added as above.

(a) First make the comparison for `type="No"`.

```
> qq(anymiss ~ npreg, data=Pima.tr2, subset=type=="No")
```

Compare this with the equivalent density plot, and explain how one translates into the other. Comment on what these graphs seem to say.

(b) The following places the comparisons for the two levels of `type` side by side:

```
> qq(anymiss ~ npreg | type, data=Pima.tr2)
```

Comment on what this graph seems to say.

NB: With `qq()`, use of "+" to get plots for the different columns all at once will not, in the current version of *lattice*, work.

2 Bootstrap sampling

Exercise 5
The following takes a with replacement sample of the rows of `Pima.tr2`.

```
> rows <- sample(1:dim(Pima.tr2)[1], replace=TRUE)
> densityplot(~ bmi, groups=type, data=Pima.tr2[rows, ],
+           scales=list(relation="free"), xlab="Measure")
```

Repeat, but using `anymiss` as the grouping factor, and with different panels for the two levels of `type`. Repeat for several different bootstrap samples. Are there differences between levels of `anymiss` that seem consistent over repeated bootstrap samples?

Exercise 6
Exercise 2 compared density plots, for several of the variables, between rows that had one or more missing values and those that had no missing values. We can use the bootstrapping idea to check the copnsistency of apparent differences across repeated bootstrap samples.

The distribution for `bmi` gives the impression that it has a different shape, between rows where one or more values was missing and rows where no values were missing, at least for `type=="Yes"`. One way to check whether this difference is consistent under repeated sampling is to treat the sample as representative of the population, and take repeated with replacement ("bootstrap") samples from it.

The following takes a bootstrap sample, then showing the Q-Q plot

```
> rownum <- 1:dim(Pima.tr2)[1]  # generate row numbers
> chooserows <- sample(rownum, replace=TRUE)
> qq(anymiss ~ bmi | type, data=Pima.tr2[chooserows, ],
+    auto.key=list(columns=2))
```

Wrap these lines of code in a function. Repeat the formation of the bootstrap samples and the plots several times. Does the shift in the distribution seem consistent under repeating sampling?

Judgements based on examination of graphs are inevitably subjective. They do however make it possible to compare differences in the shapes of distributions. Here, the shape difference is of more note than any difference in mean or median.

Exercise 7
In the data frame `nswdemo` (*DAAGxtras* package), compare the distribution of `re78` for those who received work training (`trt==1`) with controls (`trt==0`) who did not.

```
> library(DAAGxtras)
> densityplot(~ re78, groups=trt, data=nswdemo, from=0,
+             auto.key=list(columns=2))
```

The distributions are highly skew. A few very large values may unduly affect the comparison.

A reasonable alternative is to compare values of `log(re78+23)`. The value 23 is chosen between it is half the minimum non-zero value of `re78`. Here is the density plot.

```
> unique(sort(nswdemo$re78))[1:3]  # Examine the 3 smallest values
> densityplot(~ log(re78+23), groups=trt, data=nswdemo,
+             auto.key=list(columns=2))
```

Do the distribution for control and treated have similar shapes?

Exercise 8
Now examine the displacement, under repeated bootstrap sampling, of one mean relative to the other. Here is code for the calculation:

```
> twoBoot <- function(n=999, df=nswdemo, ynam="re78", gp="trt"){
+   d2 <- numeric(n+1)
+   fac <- df[, gp]
+   if(!is.factor(fac))fac <- factor(fac)
+   if(length(levels(fac)) != 2) stop(paste(gp, "must have 2 levels"))
+   y <- df[, ynam]
+   d2[1] <- diff(tapply(y, fac, mean))
+   for(i in 1:n){
+     chooserows <- sample(1:length(y), replace=TRUE)
+     faci <- fac[chooserows]
+     yi <- y[chooserows]
+     d2[i+1] <- diff(tapply(yi, faci, mean))
+ }
+   d2
+ }
> ##
> d2 <- twoBoot()
> quantile(d2, c(.025,.975))   # 95% confidence interval
```

Note that a confidence interval should not be interpreted as a probability statement. It takes no account of prior probability. Rather, 95% of intervals that are calculated in this way can be expected to contain the true probability.

Exercise 9
Another possibility is to work with a permutation distribution. If the difference between treated and controls is entirely due to sampling variation, then permuting the treatment labels will give another sample from this same distribution. Does the observed difference between treated and controls seem "extreme", relative to this permutation distribution? Here is code that may be used.

```
> dnsw <- numeric(1000)
> y <- nswdemo$re78
> treat <- nswdemo$trt
> dnsw[1] <- mean(y[treat==1]) - mean(y[treat==0])
> for(i in 2:1000){
+   trti <- sample(treat)
+   dnsw[i] <- mean(y[trti==1]) - mean(y[trti==0])
+ }
> sum(dnsw > dnsw[1])/length(dnsw)

[1] 0.029

> 2*min(sum(d2<0)/length(d2), sum(d2>0)/length(d2))  # 2-sided comparison

[1] 0.062
```

Compare the result with that from the bootstrap approach.

Replace `re78` with `log(re78+23)` and repeat the calculations.

Note: In formalizing the result for a test of hypothesis, note that the difference between `treat==1` and `treat==1` might go in either direction.

Part IV
Models and Model Accuracy

1 Straight Line Regression

Exercise 1
A plot of heart weight (`heart`) versus body weight (`weight`), for Cape Fur Seal data in the data set `cfseal` (*DAAG*) shows a relationship that is approximately linear. Check this. However variability about the line increases with increasaing weight. It is better to work with `log(heart)` and `log(weight)`, where the relationship is again close to linear, but variability about the line is more homogeneous. Such a linear relationship is consistent with biological allometry, here across different individuals. Allometric relationships are pairwise linear on a logarithmic scale.

Plot `log(heart)` against `log(weight)`, and fit the least squares regression line for `log(heart)` on `log(weight)`.

```
> library(DAAG)
> cflog <- log(cfseal[, c("heart", "weight")])
> names(cflog) <- c("logheart", "logweight")
> plot(logheart ~ logweight, data = cflog)
> cfseal.lm <- lm(logheart ~ logweight, data = cflog)
> abline(cfseal.lm)
```

Exercise 2
In the training/test sample appraoch, the data are split into two parts. The model is trained on one part (the training set), and tested on the other part. Run the following code:

```
> fold <- sample(1:2, dim(cflog)[1], replace = TRUE)
> train <- subset(cflog, fold == 1)
> test <- subset(cflog, fold == 2)
> cfseal.lm <- lm(logheart ~ logweight, data = train)
> hat <- predict(cfseal.lm, newdata = test)
> plot(hat, test$logheart)
> meansquare <- sum((hat - test$logheart)^2)/dim(test)[1]
> sd <- sqrt(meansquare)
> sd
```

Values for `sd` should be of the same order of magnitude as `summary(cfseal.lm)$sigma)` Repeat the calculations 100 times, and store the 100 values of `sd`

Exercise 3
We could interchange the training and test set, and repeat the calculations, averaging the two mean squares. But why limit ourselves to a split into two parts. Why not split the data into an arbitrary number of parts?

(a) In cross-validation, the data are split into perhaps 10 folds. Each of the ten parts is used in turn as the test data, with the remaining nine parts used as training data. Here is skeleton code for the calculations.

Exercise 3, continued

```
> cvskeleton <- function(data, nfolds = 3) {
+     n <- dim(data)[1]
+     folds <- sample(rep(1:nfolds, length.out = n))
+     ufold <- unique(folds)
+     for (i in ufold) {
+         trainrows <- (1:n)[i != folds]
+         testrows <- (1:n)[i == folds]
+         print(testrows)
+     }
+     invisible(split(1:n, folds))
+ }
```

Run the function and check that each row does indeed appear in exactly one of the test sets.

(b) Here is a function that implements cross-validation calculations. For the regression calculation, it will be necessary to supply it with functions that do the training and test calculations, thus:

```
> cftrainfun <- function(formula = logheart ~ logweight, traindata) lm(formula,
+     traindata)
> cftestfun <- function(obj, newdata) predict(obj, newdata)
```

Here then is the function:

```
> cvpred <- function(formula = type ~ ., data, trainfun, testfun,
+     nfolds = 2, balanced = TRUE) {
+     n <- dim(data)[1]
+     pred <- numeric(n)
+     if (balanced)
+         folds <- sample(rep(1:nfolds, length.out = n))
+     else folds <- sample(1:nfolds, n, replace = T)
+     ufold <- unique(folds)
+     for (i in ufold) {
+         testrows <- i == folds
+         traindata <- subset(data, !testrows)
+         trained.obj <- trainfun(formula, traindata = traindata)
+         testdata <- data[testrows, ]
+         pred[testrows] <- testfun(obj = trained.obj, newdata = testdata)
+     }
+     invisible(pred)
+ }
```

Now run the function, and compare predictions with observed values

```
> predobs <- cvpred(logheart ~ logweight, data = cflog, trainfun = cftrainfun,
+     testfun = cftestfun)
> sd <- sqrt(sum((predobs - cflog$logheart)^2)/dim(cflog)[1])
```

Run the calculation several times.

Exercise 4
For each of (1) `nfolds=2` and (2) `nfolds=10`, run the calculations of Exercise 3 100 times, calculating a standard deviation at each run. Compare the two sets of standard deviation estimates, both using a density plot and using a Q-Q plot. Which calculation would you expect to give the more consistent (less variable) results? Why? Do the graphs bear this out?

Exercise 5
The following fits a simple linear discriminant model to the data frame `Pima.tr`, then using `Pima.te` to test the predictions. We are using the training/test set approach.

```
> Pima.lda <- lda(type ~ ., data = Pima.tr)
> predclass <- predict(Pima.lda, newdata = Pima.te)$class
> tab12 <- table(Pima.te$type, predclass)
> tab12
> sum(tab12[row(tab12) == col(tab12)])/sum(tab12)
```

Note that we are regarding `Pima.tr` as data set 1, and texttttPima.te as data set 2. The confusion matrix when we train on data set 1 and test on data set 2 is therefore called `tab12`.
Now use `Pima.te` as the training data, and `Pima.tr` as the test data, and repeat the exercise, leading to the confusion matrix `tab21`.

You should find that the confusion matrices differ substantially, and that the overall accuracies differ. What might explain this?

Exercise 6
Now work with `Pima.tr` alone, and `Pima.te` alone, calculating cross-validated predictions and confusion matrices. As before functions will be be defined to handle the training and testing. Use is then made of the function `cvpred()` that was defined above.

(a) Calculate cross-validated estimates, with `Pima.tr`.

```
> Pimatrainfun <- function(formula, traindata) lda(formula, traindata)
> Pimatestfun <- function(obj, newdata) predict(obj, newdata)$class
> predclass <- cvpred(type ~ ., data = Pima.tr, nfolds = 10, trainfun = Pimatrainfun,
+     testfun = Pimatestfun)
> tab11 <- table(Pima.tr$type, predclass)
> sum(tab11[row(tab11) == col(tab11)])/sum(tab11)
```

(b) Now calculate cross-validated estimates for `Pima.te`, leading to the confusion matrix `tab22`.

Repeat each of these calculations several times, to get an idea of the statistical variation in the confusion matrices.
Compare `tab11`, `tab21`, `tab22` and `tab12`, and comment.

Exercise 7
The resubstitution measure of accuracy compares predictions for the training data with observations from the same training data. For example:

```
> Pima.lda <- lda(type ~ ., data = Pima.tr)
> predclass <- predict(Pima.lda)$class
> tab1resub <- table(Pima.tr$type, predclass)
> tab1resub
> sum(tab1resub[row(tab1resub) == col(tab1resub)])/sum(tab1resub)
```

Exercise 7, continued
Such resubstitution measures of accuracy can be grossly optimistic. Why? Do they seem to be optimistic for the models that have been fitted to these data.

Exercise 8
A feature of the resubstitution measure is that it cannot decrease as more terms are added to the model. For example, try adding in all second order interaction terms, i.e., we have all factors and interactions involving three or fewer columns.

```
> Pima.lda <- lda(type ~ .^3, data = Pima.tr)
> predclass <- predict(Pima.lda)$class
> tab1resub3 <- table(Pima.tr$type, predclass)
> tab1resub3
> sum(tab1resub3[row(tab1resub3) == col(tab1resub3)])/sum(tab1resub3)
```

There is a large increase in the resubstitution measure of predictive accuracy, but does it mean anything. In order to check, we do a cross-validation calculation.

```
> predclass <- cvpred(type ~ .^3, data = Pima.tr, nfolds = 10,
+     trainfun = Pimatrainfun, testfun = Pimatestfun)
> tab11cv3 <- table(Pima.tr$type, predclass)
> sum(tab11cv3[row(tab11cv3) == col(tab11cv3)])/sum(tab11cv3)
```

Compare the resubstitution measure with the cross-validation measure, and comment.

Exercise 9
Most of the rows with missing values for `Pima.tr2` arise from missing values for the column `skin`. Compare

```
> sum(!complete.cases(Pima.tr2[, -4]))
> sum(!complete.cases(Pima.tr2))
```

The following omits entirely rows where columns other than `skin` are missing. It then creates an outcome variable with four categories: `type=="No"` & `NA` or not for `skin`, `type=="Yes"` & `NA` or not for `skin`.

```
> mostlyOK <- complete.cases(Pima.tr2[, -4])
> Pima.tr2A <- Pima.tr2[mostlyOK, ]
> type4 <- with(Pima.tr2A, paste(type, complete.cases(skin), sep = ""))
> Pima.tr2A <- cbind(Pima.tr2A[, c(1:3, 5:7)], type4 = factor(type4))
```

Now try

```
> PimaA.lda <- lda(type4 ~ ., data = Pima.tr2A)
> PimaA.lda
```

Exercise 9, continued
We can get a plot of the results:

```
> plot(PimaA.lda, dimen = 2)
> scores <- predict(PimaA.lda)$x
> library(lattice)
> xyplot(scores[, 2] ~ scores[, 1], groups = type4, data = Pima.tr2A,
+     auto.key = list(columns = 2), par.settings = list(superpose.symbol = list(pch = 16)))
```

The function `cvpred()` can be used as before to estimate the confusion matrix. Or use the setting `CV=TRUE` when the function `lda` is called, which uses the simple leave-one-out form of cross-validation.

```
> table(Pima.tr2A$type4, lda(type4 ~ ., data = Pima.tr2A, CV = TRUE)$class)
```

2 Analysis Using *randomForest*

Exercise 10
Repeat exercise 9, but now using `randomForest()`. For now, we use this as a black box.

```
> library(randomForest)
> Pima.rf <- randomForest(type4 ~ ., data = Pima.tr2A)
> Pima.rf

Call:
 randomForest(formula = type4 ~ ., data = Pima.tr2A)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 53.52%
Confusion matrix:
         NoFALSE NoTRUE YesFALSE YesTRUE class.error
NoFALSE        2     46        1       6   0.9636364
NoTRUE        19     93        2      18   0.2954545
YesFALSE       3     15        0      11   1.0000000
YesTRUE        3     26        2      37   0.4558824
```

Note that OOB = Out of Bag Error rate, calculated using an approach that has much the same effect as cross-validation, applied to the data specified by the `data` parameter. Notice that `random-Forest()` will optionally give, following the one function call, an assessment of predictive error for a completely separate set of test data that has had no role in training the model.

```
> type4te <- with(Pima.te, paste(type, TRUE, sep = ""))
> type4te <- factor(type4te, levels = levels(Pima.tr2A$type4))
> randomForest(type4 ~ ., data = Pima.tr2A, xtest = Pima.te[c(1:3,
+     5:7)], ytest = type4te)
```

Where such a test set is available, this provides a reassuring check that `randomForest()` is not over-fitting.

Note: The function `tuneRF()` can be used for such limited tuning as `randomForest()` allows. Look up `help(tuneRF()`, and run this function in order to find an optimal value for the parameter `mtry`. Then repeat the above with this optimum value of `mtry`, and again compare the OOB error with the error on the test set.

Part V
Ordination

The package *oz*, used to draw a map of Australia, must be installed. In order to use the dataframe `diabetes` from the package *mclust*, that package must be installed. Or you can obtain the R image file from `"http://www.maths.anu.edu.au/~johnm/courses/dm/math3346/data/` Also required are the functions `read.dna()` and `dist.dna()` from the package *ape*; that package must be installed.

Where there is no "natural" measure of distance between points, and there are groups in the data that correspond to categorizations that will be of interest when the data are plotted, some form of discriminant analysis may be the preferred starting point for obtaining a low-dimensional representation. The low-dimensional representation that is likely to be superior to that obtained from ordination without regard to any such categorization.

Two of the possibilities that the ordination methods considered in this laboratory addresses are:

- Distances may be given, from which it is desired to recover a low-dimensional representation.
  - For example we may, as below, attempt to generate a map in which the distances on the map accurately reflect Australian road travel distances.
  - Or we may, based on genomic differences, derive genomic "distances " between, e.g., different insect species. The hope is that, with a judicious choice of the distance measure, the distances will be a monotone function of the time since the two species separated. We'd like to derive a 2 or 3-dimensional representation in which the distances accurately reflect the closeness of the evolutionary relationships.
- There may be no groupings of the data that are suitable for use in deriving a low-dimensional representation. Hence we calculate, using a metric that seems plausible, a matrix of distances between pairs of points, from which we in turn try to derive a low-dimensional representation.

1 Australian road distances

The distance matrix that will be used is in the matrix `audists`, in the image file **audists.Rdata**.

Here is how the data can be read in from the text file:

```
audists <- read.table("audists.txt", sep="\t")
audists[is.na(audists)] <- 0
## Also, we will later use the data frame aulatlong
aulatlong <- read.table("aulatlong.txt", row.names=1)[,2:1]
aulatlong[,2] <- -aulatlong[,2]
colnames(aulatlong) <- c("latitude", "longitude")
```

Consiser first the use of classical multi-dimensional scaling, as implemented in the function `cmdscale()`:

```
> library(DAAGxtras)
> aupoints <- cmdscale(audists)
> plot(aupoints)
> text(aupoints, labels = paste(rownames(aupoints)))
```

An alternative to `text(aupoints, labels=paste(rownames(aupoints)))`, allowing better placement of the labels, is `identify(aupoints, labels=rownames(aupoints))`. We can compare the distances in the 2-dimensional representation with the original road distances:

```
> origDists <- as.matrix(audists)
> audistfits <- as.matrix(dist(aupoints))
> misfit <- audistfits - origDists
> for (j in 1:9) for (i in (j + 1):10) {
```

```
+     lines(aupoints[c(i, j), 1], aupoints[c(i, j), 2], col = "gray")
+     midx <- mean(aupoints[c(i, j), 1])
+     midy <- mean(aupoints[c(i, j), 2])
+     text(midx, midy, paste(round(misfit[i, j])))
+ }
> print(round(misfit))
```

```
          Adelaide Alice Brisbane Broome Cairns Canberra Darwin Melbourne Perth
Adelaide         0   140     -792   -156    366       20     11        82   482
Alice          140     0    -1085   -175    -41       76   -118       106   -26
Brisbane      -792 -1085        0    198    319      -25   -233      -471   153
Broome        -156  -175      198      0    527       -7      6       -65   990
Cairns         366   -41      319    527      0      277    -31       178     8
Canberra        20    76      -25     -7    277        0     -1      -241   372
Darwin          11  -118     -233      6    -31       -1      0       -12    92
Melbourne       82   106     -471    -65    178     -241    -12         0   301
Perth          482   -26      153    990      8      372     92       301     0
Sydney        -273  -314      -56     70    251       -8    -58      -411   271
          Sydney
Adelaide    -273
Alice       -314
Brisbane     -56
Broome        70
Cairns       251
Canberra      -8
Darwin       -58
Melbourne   -411
Perth        271
Sydney         0
```

The graph is a tad crowded, and for detailed information it is necessary to examine the table.

It is interesting to overlay this "map" on a physical map of Australia.

```
> if (!exists("aulatlong")) load("aulatlong.RData")
> library(oz)
> oz()
> points(aulatlong, col = "red", pch = 16, cex = 1.5)
> comparePhysical <- function(lat = aulatlong$latitude, long = aulatlong$longitude,
+     x1 = aupoints[, 1], x2 = aupoints[, 2]) {
+     fitlat <- predict(lm(lat ~ x1 + x2))
+     fitlong <- predict(lm(long ~ x1 + x2))
+     x <- as.vector(rbind(lat, fitlat, rep(NA, 10)))
+     y <- as.vector(rbind(long, fitlong, rep(NA, 10)))
+     lines(x, y, col = 3, lwd = 2)
+ }
> comparePhysical()
```

An objection to `cmdscale()` is that it gives long distances the same weight as short distances. It is just as prepared to shift Canberra around relative to Melbourne and Sydney, as to move Perth. It makes more sense to give reduced weight to long distances, as is done by `sammon()` (*MASS*).

```
> library(MASS)
> aupoints.sam <- sammon(audists)
```

```
Initial stress        : 0.01573
stress after  10 iters: 0.00525, magic = 0.500
stress after  20 iters: 0.00525, magic = 0.500
```

```
> oz()
> points(aulatlong, col = "red", pch = 16, cex = 1.5)
> comparePhysical(x1 = aupoints.sam$points[, 1], x2 = aupoints.sam$points[,
+     2])
```

Notice how Brisbane, Sydney, Canberra and Melbourne now maintain their relative positions much better.

Now try full non-metric multi-dimensional scaling (MDS). THis preserves only, as far as possible, the relative distances. A starting configuration of points is required. This might come from the configuration used by `cmdscale()`. Here, however, we use the physical distances.

```
> oz()
> points(aulatlong, col = "red", pch = 16, cex = 1.5)
> aupoints.mds <- isoMDS(audists, as.matrix(aulatlong))

initial  value 11.875074
iter   5 value 5.677228
iter  10 value 4.010654
final  value 3.902515
converged

> comparePhysical(x1 = aupoints.mds$points[, 1], x2 = aupoints.mds$points[,
+     2])
```

Notice how the distance between Sydney and Canberra has been shrunk quite severely.

2 If distances must first be calculated . . .

There are two functions that can be used to find distances – `dist()` that is in the base *statistics* package, and `daisy()` that is in the *cluster* package. The function `daisy()` is the more flexible. It has a parameter `stand` that can be used to ensure standardization when distancces are calculated, and allows columns that are factor or ordinal. Unless measurements are comparable (e.g., relative growth, as measured perhaps on a logarithmic scale, for different body measurements), then it is usually desirable to standardize before using ordination methods to examine the data.

```
> library(cluster)
> library(mclust)
> data(diabetes)
> diadist <- daisy(diabetes[, -1], stand = TRUE)
> plot(density(diadist, from = 0))
```

3 Genetic Distances

Here, matching genetic DNA or RNA or protein or other sequences are available from each of the different species. Distances are based on probabilistic genetic models that describe how gene sequences change over time. The package *ape* implements a number of alternative measures. For details see `help(dist.dna)`.

3.1 Hasegawa's selected primate sequences

The sequences were selected to have as little variation in rate, along the sequence, as possible. The sequences are available from:
`http://evolution.genetics.washington.edu/book/primates.dna`. They can be read into R as:

```
> library(ape)
> webpage <- "http://evolution.genetics.washington.edu/book/primates.dna"
> primates.dna <- read.dna(url(webpage))
```

Now calculate distances, using Kimura's F84 model, thus

```
> primates.dist <- dist.dna(primates.dna, model = "F84")
```

We now try for a two-dimensional representation, using `cmdscale()`.

```
> primates.cmd <- cmdscale(primates.dist)
> eqscplot(primates.cmd)
> rtleft <- c(4, 2, 4, 2)[unclass(cut(primates.cmd[, 1], breaks = 4))]
> text(primates.cmd[, 1], primates.cmd[, 2], row.names(primates.cmd),
+     pos = rtleft)
```

Now see how well the distances are reproduced:

```
> d <- dist(primates.cmd)
> sum((d - primates.dist)^2)/sum(primates.dist^2)

[1] 0.1977138
```

This is large enough (20%, which is a fraction of the total sum of squares) that it may be worth examining a 3-dimensional representation.

```
> primates.cmd <- cmdscale(primates.dist, k = 3)
> cloud(primates.cmd[, 3] ~ primates.cmd[, 1] * primates.cmd[,
+     2])
> d <- dist(primates.cmd)
> sum((d - primates.dist)^2)/sum(primates.dist^2)

[1] 0.1045168
```

Now repeat the above with `sammon()` and `mds()`.

```
> primates.sam <- sammon(primates.dist, k = 3)

Initial stress        : 0.11291
stress after  10 iters: 0.04061, magic = 0.461
stress after  20 iters: 0.03429, magic = 0.500
stress after  30 iters: 0.03413, magic = 0.500
stress after  40 iters: 0.03409, magic = 0.500

> eqscplot(primates.sam$points)
> rtleft <- c(4, 2, 4, 2)[unclass(cut(primates.sam$points[, 1],
+     breaks = 4))]
> text(primates.sam$points[, 1], primates.sam$points[, 2], row.names(primates.sam$points),
+     pos = rtleft)
```

There is no harm in asking for three dimensions, even if only two of them will be plotted.

```
> primates.mds <- isoMDS(primates.dist, primates.cmd, k = 3)

initial  value 19.710924
iter   5 value 14.239565
iter  10 value 11.994621
iter  15 value 11.819528
iter  15 value 11.808785
iter  15 value 11.804569
final  value 11.804569
converged
```

```
> eqscplot(primates.mds$points)
> rtleft <- c(4, 2, 4, 2)[unclass(cut(primates.mds$points[, 1],
+     breaks = 4))]
> text(primates.mds$points[, 1], primates.mds$points[, 2], row.names(primates.mds$points),
+     pos = rtleft)
```

....
....
The two remaining examples are optional extras.

4 *Distances between fly species

These data have been obtained from databases on the web. We extract them from an Excel **.csv** file (**dipt.csv**) and store the result in an object of class `"dist"`. The file **dipt.csv** is available from `http://www.maths.anu.edu.au/~johnm/datasets/ordination`.

Only below diagonal elements are stored, in column dominant order, in the distance object `dipdist` that is created below. For manipulating such an object as a matrix, should this be required, `as.matrix()` can be used to turn it into a square symmetric matrix. Specify, e.g., `dipdistM <- as.matrix(dipdist)`. For the clustering and ordination methods that are used here, storing it as a distance object is fine.

```
> webfile <- "http://www.maths.anu.edu.au/~johnm/datasets/ordination/dipt.csv"
> diptera <- read.csv(url(webfile), comment = "", na.strings = c(NA,
+     ""))
> dim(diptera)

[1] 110 114

> species <- as.character(diptera[, 1])
> table(substring(species, 1, 1))

  #
110

> species <- substring(species, 2)
> genus <- as.character(diptera[, 2])
> dipdist <- as.dist(diptera[1:110, 5:114])
> attributes(dipdist)$Labels <- species
> length(dipdist)

[1] 5995
```

Two of the functions that will be used – `sammon()` and `isoMDS()` – require all distances to be positive. Each zero distance will be replaced by a small positive number.

```
> dipdist[dipdist == 0] <- 0.5 * min(dipdist[dipdist > 0])
```

Now use (i) classical metric scaling; (ii) sammon scaling; (iii) isometric multi-dimensional scaling, with i as the starting configuration; (iv) isometric multi-dimensional scaling, with ii as the starting configuration.

```
> dipt.cmd <- cmdscale(dipdist)
> genus <- sapply(strsplit(rownames(dipt.cmd), split = "_", fixed = TRUE),
+     function(x) x[1])
> nam <- names(sort(table(genus), decreasing = TRUE))
> genus <- factor(genus, levels = nam)
> plot(dipt.cmd, col = unclass(genus), pch = 16)
> dipt.sam <- sammon(dipdist)
> plot(dipt.sam$points, col = unclass(genus), pch = 16)
> dipt.mds <- isoMDS(dipdist, dipt.cmd)
> plot(dipt.mds$points, col = unclass(genus), pch = 16)
> dipt.mds2 <- isoMDS(dipdist, dipt.sam$points)
> plot(dipt.mds2$points, col = unclass(genus), pch = 16)
```

5 *Rock Art

Here, we use data that were collected by Meredith Wilson for her PhD thesis. The 614 features were all binary – the presence or absence of specific motifs in each of 103 Pacific sites. It is necessary to omit 5 of the 103 sites because they have no motifs in common with any of the other sites. Data are in the dataset `pacific.RData`.

The binary measure of distance was used – the number of locations in which only one of the sites had the marking, as a proportion of the sites where one or both had the marking. Here then is the calculation of distances:

```
> if (!exists("pacific")) load("pacific.Rdata")
> pacific.dist <- dist(x = as.matrix(pacific[-c(47, 54, 60, 63,
+     92), 28:641]), method = "binary")
> sum(pacific.dist == 1)/length(pacific.dist)

[1] 0.6311803

> plot(density(pacific.dist, to = 1))
> symmat <- as.matrix(pacific.dist)
> table(apply(symmat, 2, function(x) sum(x == 1)))

13 21 27 28 29 32 33 35 36 38 40 41 42 43 44 45 46 47 48 49 51 52 53 54 55 56
 1  1  1  1  2  1  2  1  2  2  1  2  4  3  1  3  1  2  1  1  2  2  3  2  2  2
57 58 61 62 64 65 66 67 68 69 70 71 73 75 76 77 79 81 83 84 85 90 91 92 93 94
 1  3  3  1  2  1  1  1  3  3  1  1  4  1  2  1  1  1  2  1  1  3  1  1  3  1
95 96 97
 1  3  4
```

It turns out that 63% of the distances were 1. This has interesting consequences, for the plots we now do.

```
> pacific.cmd <- cmdscale(pacific.dist)
> cmdplot()
> samplot()
```

Part VI
Discriminant Methods – Error Rate Estimation, & Low-Dimensional Views

1 rpart Analyses – the Combined `Pima` Dataset

Note the rpart terminology:

size	Number of leaves	Used in plots from `plotcp()`
`nsplit`	Number of splits = size - 1	Used in `printcp()` output
`cp`	Complexity parameter	Appears as CP in graphs and printed output. A smaller `cp` gives a more complex model, i.e., more splits.
rel error	Resubstitution error measure	Multiply by baseline error to get the corresponding absolute error measure. In general, treat this error measure with scepticism.
xerror	Crossvalidation error estimate	Multiply by baseline error to get the corresponding absolute error measure.

After attaching the *MASS* package, type `help(Pima.tr)` to get a description of these data. They are relevant to the investigation of conditions that may pre-dispose to diabetes.

The `Pima` data frame combines `Pima.tr` and `Pima.te` from the *MASS* package, thus:

```
> library(e1071)
> library(MASS)
> Pima <- rbind(Pima.tr, Pima.te)
```

1.1 Fitting the model

Fit an rpart model to the `Pima` data:

```
> library(rpart)
> Pima.rpart <- rpart(type ~ ., data = Pima, method = "class")
> plotcp(Pima.rpart)
```

The formula `type` $\sim$ `.` has the effect of using as explanatory variables all columns of `Pima` except `type`. The parameter cp is a complexity parameter; it penalizes models that are too complex. A small penalty leads to more splits. Note that cp, at this initial fit, has to be small enough so that the minimum of the cross-validated error is attained.

Try this several times. The result will vary somewhat from run to run. Why?

One approach is to choose the model that gives the absolute minimum of the cross-validated error. If the fitting has been repeated several times, the cross-validation errors can be averaged over the separate runs.

A more cautious approach is to choose a model where there is some modest certainty that the final split is giving a better than chance improvement. For this, the suggestion is to choose the smallest number of splits so that cross-validated error rate lies under the dotted line. This is at a height of (minimum cross-validated error rate) + 1 standard error.

The choice of 1 SE, rather than some other multiple of the SE, is somewhat arbitrary. The aim is to identify a model where the numbers of splits stays pretty much constant under successive runs of `rpart`. For this it is necessary to move back somewhat, on the curve that plots the cross-validated error rate against `cp`, from the flat part of the curve where the cross-validated error rate is a minimum. The 1 SE rule identifies a better-defined value of `cp` where the curve has a detectable negative slope.

For example, in one of my runs, the 1 SE rule gave `cp`=0.038, with size=4. The resulting tree can be cut back to this size with:

```
> Pima.rpart4 <- prune(Pima.rpart, cp = 0.037)
```

The value of cp is chosen to be less than 0.038, but more than the value that led to a further split.
Plot the tree, thus:

```
> plot(Pima.rpart4)
> text(Pima.rpart4)
```

Note also the printed output from

```
> printcp(Pima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima, method = "class")

Variables actually used in tree construction:
[1] age   bmi   glu   npreg ped

Root node error: 177/532 = 0.33271

n= 532

        CP nsplit rel error  xerror     xstd
1 0.265537      0   1.00000 1.00000 0.061400
2 0.056497      1   0.73446 0.77966 0.057116
3 0.025424      3   0.62147 0.74576 0.056284
4 0.020716      5   0.57062 0.76836 0.056844
5 0.014124      9   0.48588 0.71751 0.055552
6 0.011299     11   0.45763 0.67797 0.054464
7 0.010000     14   0.42373 0.69492 0.054940
```

Get the absolute cross-validated error by multiplying the root node error by xerror. With `nsplit=3` (a tree of size 4 leaves), this is, in the run I did, 0.3327*0.7006 = 0.233. (The accuracy is obtained by subtracting this from 1.0, i.e., about 77%.)

Where it is not clear from the graph where the minimum (or the minimum+SE, if that is used) lies, it will be necessary to resort to use this printed output for that purpose. It may be necessary to use a value of `cp` that is smaller than the default in the call to `rpart()`, in order to be reasonably sure that the optimum (according to one or other criterion) has been found.

Exercise 1: Repeat the above several times, i.e.

```
> library(rpart)
> Pima.rpart <- rpart(type ~ ., data = Pima, method = "class")
> plotcp(Pima.rpart)
```

(a) Overlay the several plots of the cross-validated error rate against the number of splits. Why does the cross=validated error rate vary somewhat from run to run? Average over the several runs and choose the optimum size of tree based on (i) the minimum cross-validated error rate, and (ii) the minimum cross-validated error rate, plus one SE.

(b) Show the relative error rate on the same graph.
NB: You can get the error rate estimates from:

```
> errmat <- printcp(Pima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima, method = "class")
```

```
Variables actually used in tree construction:
[1] age   bmi   glu   npreg ped

Root node error: 177/532 = 0.33271

n= 532

        CP nsplit rel error  xerror     xstd
1 0.265537      0   1.00000 1.00000 0.061400
2 0.056497      1   0.73446 0.77966 0.057116
3 0.025424      3   0.62147 0.70621 0.055249
4 0.020716      5   0.57062 0.72316 0.055701
5 0.014124      9   0.48588 0.71186 0.055401
6 0.011299     11   0.45763 0.68927 0.054783
7 0.010000     14   0.42373 0.67232 0.054302
```

```
> colnames(errmat)
```

```
[1] "CP"        "nsplit"    "rel error" "xerror"    "xstd"
```

```
> resub.err <- 1 - 0.3327 * errmat[, "rel error"]
> cv.err <- 1 - 0.3327 * errmat[, "xerror"]
```

]

Exercise 2: Prune the model back to give the optimum tree, as determined by the one SE rule. How does the error rate vary with the observed value of `type`? Examine the confusion matrix. This is most easily done using the function `xpred()`. (The following assumes that 3 leaves, i.e., `cp` less than about 0.038 and greater than 0.011, is optimal.)

```
> Pima.rpart <- prune(Pima.rpart, cp = 0.037)
> cvhat <- xpred.rpart(Pima.rpart4, cp = 0.037)
> tab <- table(Pima$type, cvhat)
> confusion <- rbind(tab[1, ]/sum(tab[1, ]), tab[2, ]/sum(tab[2,
+     ]))
> dimnames(confusion) <- list(ActualType = c("No", "Yes"), PredictedType = c("No",
+     "Yes"))
> print(confusion)
```

```
          PredictedType
ActualType        No       Yes
       No  0.8647887 0.1352113
       Yes 0.4802260 0.5197740
```

The table shows how the predicted accuracy changes, depending on whether the correct type is `Yes` or `No`.

How would you expect the overall estimate of predictive accuracy to change, using the same fitted rpart model for prediction:

- if 40% were in the `Yes` category?
- if 20% were in the `Yes` category?

2 rpart Analyses – `Pima.tr` and `Pima.te`

Exercise 3 These exercises will use the two data sets `Pima.tr` and `Pima.te`, and carrying out a variation on the calculations of Exercise 2.

What are the respective proportions of the two types in the two data sets?

Exercise 3a: Repeat Exercise 1, but now use the data frame `Pima.tr` to determine the model, and comparing cross-validation accuracies from calculations with `Pima.tr` with accuracies when `Pima.te` is used as the test data:

```
> trPima.rpart <- rpart(type ~ ., data = Pima.tr, method = "class")
> plotcp(trPima.rpart)
> printcp(trPima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 0.97059 0.097791
3 0.073529      2   0.61765 0.75000 0.090647
4 0.058824      3   0.54412 0.76471 0.091224
5 0.014706      4   0.48529 0.64706 0.086152
6 0.010000      7   0.44118 0.70588 0.088822
```

as above, e.g.

```
> trPima.rpart <- prune(trPima.rpart, cp = 0.02)
```

How does the error rate vary between the `No`'s and `Yes`'s. How does the expected error rate vary with the proportion of `No`'s in the target population?

Exercise 3b: Refit the model. Choose values of `cp` that will give the points on the graph obtained from `plotcp(trPima.rpart)`. Suitable values of `cp` are those that appear on the graph given by `plotcp()`. These (except for the first; what happens there?) are the geometric means of the successive pairs that are printed, and can be obtained thus:

```
> trPima.rpart <- rpart(type ~ ., data = Pima.tr, method = "class")
> cp.all <- printcp(trPima.rpart)[, "CP"]

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 1.05882 0.099827
3 0.073529      2   0.61765 0.83824 0.093882
4 0.058824      3   0.54412 0.83824 0.093882
5 0.014706      4   0.48529 0.69118 0.088180
6 0.010000      7   0.44118 0.76471 0.091224
```

```
> n <- length(cp.all)
> cp.all <- sqrt(cp.all * c(Inf, cp.all[-n]))
> nsize <- printcp(trPima.rpart)[, "nsplit"] + 1

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 1.05882 0.099827
3 0.073529      2   0.61765 0.83824 0.093882
4 0.058824      3   0.54412 0.83824 0.093882
5 0.014706      4   0.48529 0.69118 0.088180
6 0.010000      7   0.44118 0.76471 0.091224
```

Observe that `nsize` is one greater than the number of splits.

Prune back successively to these points. In each case determine the cross-validated error for the training data and the error for test data. Plot both these errors against the size of tree, on the same graph. Are they comparable? The following will get you started:

```
> tr.cverr <- printcp(trPima.rpart)[, "xerror"] * 0.34

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 1.05882 0.099827
3 0.073529      2   0.61765 0.83824 0.093882
4 0.058824      3   0.54412 0.83824 0.093882
5 0.014706      4   0.48529 0.69118 0.088180
6 0.010000      7   0.44118 0.76471 0.091224

> n <- length(cp.all)
> trPima0.rpart <- trPima.rpart
> te.cverr <- numeric(n)
> for (i in n:1) {
+     trPima0.rpart <- prune(trPima0.rpart, cp = cp.all[i])
+     hat <- predict(trPima0.rpart, newdata = Pima.te, type = "class")
+     tab <- table(hat, Pima.te$type)
+     te.cverr[i] <- 1 - sum(tab[row(tab) == col(tab)])/sum(tab)
+ }
```

Comment on the comparison, and also on the dependence on the number of splits.

3 Analysis Using *svm*

Exercise 4: Compare the result also with the result from an SVM (Support Vector Machine) model. For getting predictions for the current test data, it will be necessary, for models fitted using `svm()`, to use the `newdata` argument for `predict()`. Follow the prototype

```
> library(e1071)
> library(MASS)
> trPima.svm <- svm(type ~ ., data = Pima.tr)
> hat <- predict(trPima.svm, newdata = Pima.te)
> tab <- table(Pima.te$type, hat)
> 1 - sum(tab[row(tab) == col(tab)])/sum(tab)
> confusion.svm <- rbind(tab[1, ]/sum(tab[1, ]), tab[2, ]/sum(tab[2,
+     ]))
> print(confusion.svm)
```

4 Analysis Using *randomForest*

Exercise 5: Repeat the previous exercise, but now using `randomForest()`

```
> library(randomForest)
> Pima.rf <- randomForest(type ~ ., data = Pima.tr, xtest = Pima.te[,
+     -8], ytest = Pima.te$type)
> Pima.rf
```

```
Call:
 randomForest(formula = type ~ ., data = Pima.tr, xtest = Pima.te[,      -8], ytest = Pima.te$type)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 27.5%
Confusion matrix:
     No Yes class.error
No  110  22   0.1666667
Yes  33  35   0.4852941
                Test set error rate: 23.49%
Confusion matrix:
     No Yes class.error
No  191  32   0.1434978
Yes  46  63   0.4220183
```

Note that OOB = Out of Bag Error rate, calculated using an approach that has much the same effect as cross-validation, applied to the data specified by the `data` parameter. Notice that `randomForest()` will optionally give, following the one function call, an assessment of predictive error for a completely separate set of test data that has had no role in training the model. Where such a test set is available, this provides a reassuring check that `randomForest()` is not over-fitting.

Exercise 6: The function `tuneRF()` can be used for such limited tuning as `randomForest()` allows. Look up `help(tuneRF()`, and run this function in order to find an optimal value for the parameter `mtry`. Then repeat the above with this optimum value of `mtry`, and again compare the OOB error with the error on the test set.

Exercise 7: Comment on the difference between `rpart()` and `randomForest()`: (1) in the level of automation; (2) in error rate for the data sets for which you have a comparison; (3) in speed of execution.

5 Class Weights

Exercise 8: Analysis with and without specification of class weights Try the following:

```
> Pima.rf <- randomForest(type ~ ., data = Pima, method = "class")
> Pima.rf.4 <- randomForest(type ~ ., data = Pima, method = "class",
+     classwt = c(0.6, 0.4))
> Pima.rf.1 <- randomForest(type ~ ., data = Pima, method = "class",
+     classwt = c(0.9, 0.1))
```

What class weights have been implicitly assumed, in the first calculation?

Compare the three confusion matrices. The effect of the class weights is not entirely clear. They do not function as prior probabilites. Prior probabilities can be fudged by manipulating the sizes of the bootstrap samples from the different classes.

6 Plots that show the "distances" between points

In analyses with `randomForest`, the proportion of times (over all trees) that any pair of observations ("points") appears at the same terminal node can be use as a measure of proximity between the pair. An ordination method can then be used to find a low-dimensional representation of the points that as far as possible preserves the distances or (for non-metric scaling) preserves the ordering of the distances. Here is an example:

```
> Pima.rf <- randomForest(type ~ ., data = Pima, proximity = TRUE)
> Pima.prox <- predict(Pima.rf, proximity = TRUE)
> Pima.cmd <- cmdscale(1 - Pima.prox$proximity)
> Pima.cmd3 <- cmdscale(1 - Pima.prox$proximity, k = 3)
> library(lattice)
> cloud(Pima.cmd3[, 1] ~ Pima.cmd3[, 2] * Pima.cmd3[, 2], groups = Pima$type)
```

Exercise 9: What is the plot from `cloud()` saying? Why is this not overly surprising?:

Note: The function `randomForest()` has the parameter `classwt`, which is supposed to assign weights to the respective classes. There is apparently (at least in verson 4.5-16) a fault in its implementation, and in most cases it has almost no effect.

The required effect can be achieved by multiplying the default values of elements of the parameter `sampsize` by amounts that reflect the relative weights.

References

Andrews D F and Herzberg A M, 1985. *Data. A Collection of Problems from Many Fields for the Student and Research Worker.* Springer-Verlag. (pp. 339-353)

Charig, C. R., 1986. Comparison of treatment of renal calculi by operative surgery, percutaneous nephrolithotomy, and extracorporeal shock wave lithotripsy. *British Medical Journal*, 292:879–882.

Gordon, N. C. et al.(1995): 'Enhancement of Morphine Analgesia by the GABA$_B$ against Baclofen'. *Neuroscience 69: 345-349.*

Intersalt Cooperative Research Group. 1988. Intersalt: an international study of electrolyte excretion and blood pressure: results for 24 hour urinary sodium and potassium excretion. British Medical Journal 297: 319-328.

Maindonald, J. H.; Waddell, B. C.; and Petry, R. J., 2001. Apple cultivar effects on codling moth (Lepidoptera: Tortricidae) egg mortality following fumigation with methyl bromide. *Postharvest Biology and Technology*, 22:99–110.

Meyer, M.C. and Finney, T. (2005): 'Who wants airbags?'. *Chance* 18:3-16.

Wilkinson, G. N. & Rogers, C. E. 1973. *Symbolic description of models in analysis of variance.* Applied Statistics 22: 392-399.