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.