Part XII
Linear Discriminant Analysis vs Random Forests

Package: `randomForest`

For linear discriminant analysis, we will use the function `lda()` (*MASS* package). Covariates are assumed to have a common multivariate normal distribution. It may have poor predictive power where there are complex forms of dependence on the explanatory factors and variables. Where it is effective, it has the virtue of simplicity.

The function `qda()` weakens the assumptions underlying `lda()` to allow different variance-covariance matrices for different groups within the data. This sets limits on the minimum group size.

Where there are two classes, generalized linear models (`glm()`) have very similar properties to linear discriminant analysis. It makes weaker assumptions. The trade-off is that estimated group membership probabilities are conditional on the observed matrix.

With all these "linear" methods, the model matrix can replace columns of covariate data by a set of spline (or other) basis columns that allow for effects that are nonlinear in the covariates. Use `termplot()` with a `glm` object, with the argument `smooth=panel.smooth`, to check for hints of nonlinear covariate effects. Detection of nonlinear effects typically requires very extensive data.

A good first check on whether these "linear" methods are adequate is comparison with the highly nonparametric analysis of the function `randomForest()` (*randomForest* package). Random Forests may do well when complex interactions are required to explain the dependence.

Here, attention will mostly be limited to data where there are two groups only.

1 Accuracy for Classification Models – the `Pima` Data

Type `help(Pima.tr2)` (*MASS* package) to get a description of these data. They are relevant to the investigation of conditions that may pre-dispose to diabetes. All the explanatory variables can be treated as continuous variables. There are no factor columns, or columns (e.g. of 0/1 data) that might be regarded as factors.

1.1 Fitting an lda model

First try linear discriminant analysis, using the model formula `type` $\sim$ `..` This takes as explanatory variables all columns of `Pima.tr` except `type`.

A first run of the calculations has `CV=TRUE`, thus using leave-one-out cross-validation to to predictions class membership and hence get an accuracy estimate. The second run of the calculations has `CV=FALSE` (the default), allowing the use of `predict()` to obtain (among other information) discriminant scores.

```
> library(MASS)
> PimaCV.lda <- lda(type ~ ., data=Pima.tr, CV=TRUE)
> tab <- table(Pima.tr$type, PimaCV.lda$class)
> ## Calculate confusion matrix from cross-validation
> conCV1 <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> dimnames(conCV1) <- list(Actual=c("No", "Yes"),
+                          "Predicted (cv)"=c("No","Yes"))
> print(round(conCV1,3))

      Predicted (cv)
Actual    No   Yes
   No  0.864 0.136
   Yes 0.456 0.544
```

```
> ## Now refit the model and get prediction scores
> Pima.lda <- lda(type ~ ., data=Pima.tr)
> ## Get a training set accuracy estimate; can be highly optimistic
> Pima.hat <- predict(Pima.lda)
> tabtrain <- table(Pima.tr$type, Pima.hat$class)
> conTrain <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> dimnames(conTrain) <- list(Actual=c("No", "Yes"),
+                       "Predicted (cv)"=c("No","Yes"))
> print(round(conTrain,3))

      Predicted (cv)
Actual    No   Yes
   No  0.864 0.136
   Yes 0.456 0.544
```

Notice that, here, the two accuracy measures are the same. In general, the training set accuracy can be optimistic.

Now plot the discriminant scores. As there are two groups only, there is just one set of scores.

```
> library(lattice)
> densityplot(~Pima.hat$x, groups=Pima.tr$type)
```

A function that calculates the confusion matrices and overall accuracy would be helpful:

```
> confusion <- function(actual, predicted, names=NULL,
+                       printit=TRUE, prior=NULL){
+   if(is.null(names))names <- levels(actual)
+   tab <- table(actual, predicted)
+   acctab <- t(apply(tab, 1, function(x)x/sum(x)))
+   dimnames(acctab) <- list(Actual=names,
+                       "Predicted (cv)"=names)
+   if(is.null(prior)){
+     relnum <- table(actual)
+     prior <- relnum/sum(relnum)
+     acc <- sum(tab[row(tab)==col(tab)])/sum(tab)
+   } else
+   {
+     acc <- sum(prior*diag(acctab))
+     names(prior) <- names
+ }
+   if(printit)print(round(c("Overall accuracy"=acc,
+                            "Prior frequency"=prior),4))
+   if(printit){    cat("\nConfusion matrix", "\n")
+     print(round(acctab,4))
+   }
+   invisible(acctab)
+ }
```

1.2 The model that includes first order interactions

Is the outcome influenced by the combined effect of covariates, e.g., by whether they increase or decrease together. A check is to include the effects of all products of variable values such as `npreg*glu`, `npreg*bp`, etc. In this instance, it will turn out that this leads to a model that is over-fitted.

The model formula `(a+b+c)^2` expands to `a+b+c+a:b+a:c+b:c`. Note the following:

- `a:a` is the same as `a`.

- If `a` and `b` are (different) factors, then `a:b` is the interaction between `a` and `b`, i.e., it allows for effects that are due to the specific combination of level of `a` with level of `b`.
- If `a` is a factor and `b` is a variable, the interaction `a:b` allows for different coefficients of the variable for different levels of the factor.
- If `a` and `b` are (different) variables, then `a:b` is the result of multiplying `a` by `b`, element by element.

Exercise 1
Try adding interaction terms to the model fitted above:

```
> ## Accuracy estimated by leave-one-out CV
> PimaCV2.lda <- lda(type ~ .^2, data=Pima.tr, CV=TRUE)
> confusion(Pima.tr$type, PimaCV2.lda$class)
> ## Now estimate "Accuracy" on training data
> Pima2.hat <- predict(lda(type ~ .^2, data=Pima.tr))$class
> confusion(Pima.tr$type, Pima2.hat)
```

Observe that the training set measure (*resubstitution* accuracy or *apparent* accuracy) now substantially exaggerates the accuracy. The model that includes all interactions terms is in truth giving lower predictive accuracy; it overfits.

1.3 Proportion correctly classified

Consider the fit

```
> PimaCV.lda <- lda(type ~ ., data=Pima.tr, CV=TRUE)
> confusion(Pima.tr$type, PimaCV.lda$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
              0.755               0.660               0.340

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.864 0.136
   Yes 0.456 0.544
```

The overall accuracy is estimated as 0.755. If however we keep the same rule, but change the prior proportions of the two classes, the overall accuracy will change. If for example, the two classes are in the ratio 0.9:0.1, the overall accuracy will be $0.9 \times 0.8636 + 0.1 \times 0.5441 \simeq 0.83$. The No's are easier to classify; with a higher proportion of No's the classification accuracy increases.

However the classification rule that is optimal also changes if the prior proportions change. The function `lda()` allows specification of a prior, thus:

```
> prior <- c(0.9, 0.1)
> PimaCVp.lda <- lda(type ~ ., data=Pima.tr, CV=TRUE, prior=prior)
> confusion(Pima.tr$type, PimaCVp.lda$class, prior=c(0.9, 0.1))

   Overall accuracy  Prior frequency.No Prior frequency.Yes
               0.91                0.90                0.10

Confusion matrix
      Predicted (cv)
```

```
Actual    No    Yes
   No  0.977 0.0227
   Yes 0.691 0.3088
```

If the rule is modified to be optimal relative to the new prior proportions, the accuracy thus increases to 0.91, approximately.

Exercise 2
Now assume prior proportions of 0.85 and 0.15. Repeat the above calculations, i.e.

- Estimate the accuracy using the rule that is designed to be optimal when the prior proportions are as in the sample.
- Estimate the accuracy using the rule that is designed to be optimal when the prior proportions are 0.85:0.15.

1.4 The ROC (receiver operating characteristic)

This introduces the terms *sensitivity* and *specificity*. With prior proportions as in the sample (0.755:0.245), the sensitivity (true positive rate) was estimated as 0.544; this is the probability of correctly identifying a person who is a diabetic as a diabetic. The false positive rate (1 - Specificity) was estimated as 0.136. There is a trade-off between sensitivity and specificity. The ROC curve, which is a plot of sensitivity against specificity, displays this trade-off graphically.

The analysis assumes that the cost of both types of mis-classification are equal. Varying the costs, while keeping the prior probabilities the same, is equivalent to keeping the costs equal, but varying the prior probabilities. The following calculation takes advantage of this equivalence.

Exercise 3
Run the following calculations:

```
> truepos <- numeric(19)
> falsepos <- numeric(19)
> p1 <- (1:19)/20
> for(i in 1:19){
+   p <- p1[i]
+   Pima.CV1p <- lda(type ~ ., data=Pima.tr, CV=TRUE, prior=c(p, 1-p))
+   confmat <- confusion(Pima.tr$type, Pima.CV1p$class, printit=FALSE)
+   falsepos[i] <- confmat[1,2]
+   truepos[i] <- confmat[2,2]
+ }
```

Now plot the curve.

```
> plot(truepos ~ falsepos, type="l", xlab="False positive rate",
+      ylab="True positive rate (Sensitivity)")
```

Read off the sensitivity at a low false positive rate (e.g., 0.1), and at a rate around the middle of the range, and comment on the tradeoff.

The ROC curve allows assessment of the effects of different trade-offs between the two types of cost.

1.5 Accuracy on test data

There is an additional data set – `Pima.te` – that has been set aside for testing. The following checks the accuracy on these "test" data.

```
> Pima.lda <- lda(type ~ ., data = Pima.tr)
> testhat <- predict(Pima.lda, newdata=Pima.te)
> confusion(Pima.te$type, testhat$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
              0.798               0.672                0.328

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.888 0.112
   Yes 0.385 0.615
```

This improves on the leave-one-out CV accuracy on `Pima.tr`. The difference in the prior proportions is too small to have much effect on the overall accuracy. The apparent improvement may be a chance effect. Another possibility is that the division of the data between `Pima.tr` and `Pima.te` may not have been totally random, and `Pima.te` may have fewer hard to classify points. There are two checks that may provide insight:

- Swap the roles of training and test data, and note whether the relative accuracies are similar.
- Repeat the calculations on a bootstrap sample of the training data, to get an indication of the uncertainty in the accuracy assessment.

Exercise 4
Try the effect of swapping the role of training and test data.

```
> swapCV.lda <- lda(type ~ ., data = Pima.te, CV=TRUE)
> confusion(Pima.te$type, swapCV.lda$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
              0.783               0.672                0.328

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.888 0.112
   Yes 0.431 0.569

> swap.lda <- lda(type ~ ., data = Pima.te)
> otherhat <- predict(Pima.lda, newdata=Pima.tr)
> confusion(Pima.tr$type, otherhat$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
               0.77                0.66                 0.34

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.871 0.129
   Yes 0.426 0.574
```

Note that, again, the accuracy is greater for `Pima.te` than for `Pima.tr`, but the difference is smaller.

Exercise 5
Now check the accuracy on a bootstrap sample:

```
> prior <- table(Pima.tr$type)
> prior <- prior/sum(prior)
> index <- sample(1:dim(Pima.tr)[1], replace=TRUE)
> boot.lda <- lda(type ~ ., data = Pima.tr[index, ], CV=TRUE)
> cmat <- confusion(Pima.tr[index, "type"], boot.lda$class, printit=FALSE)
> print(c(acc=round(prior[1]*cmat[1,1]+prior[2]*cmat[2,2],4)))

acc.No
 0.801
```

The calculations should be repeated several times. The changes in the predictive accuracy estimates are substantial.
Note the need to relate all accuracies back to the same prior probabilities, to ensure comparability.
Annotate the code to explain what it does.

(From running this code five times, I obtained results of 0.77, 0.74, 0.72, 0.71 and 0.82.)

2 Logistic regression – an alternative to `lda`

As the Pima data have only two classes (levels of `type`) the calculation can be handled as a regression problem, albeit with the reponse on a logit scale, i.e., the linear model predicts $\log(\frac{\pi}{1-\pi})$, where π is the probability of having diabetes.

Exercise 6
Fit a logistic regression model and check the accuracy.

```
> Pima.glm <- glm(I(unclass(type)-1) ~ ., data=Pima.tr, family=binomial)
> testhat <- round(predict(Pima.glm, newdata=Pima.te, type="response"))
> confusion(Pima.te$type, testhat)
```

Compare the accuracy with that obtained from `lda()`.
A cross-validation estimate of accuracy, based on the training data, can be obtained thus:

```
> library(DAAG)
> CVbinary(Pima.glm)

Fold:  6 1 4 7 9 5 3 10 2 8
Internal estimate of accuracy = 0.775
Cross-validation estimate of accuracy = 0.76
```

This should be repeated several times. How consistent are the results?

One advantage of `glm()` is that asymptotic standard error estimates are available for parameter estimates:

```
> round(summary(Pima.glm)$coef, 3)

            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -9.773      1.770  -5.520    0.000
npreg          0.103      0.065   1.595    0.111
glu            0.032      0.007   4.732    0.000
bp            -0.005      0.019  -0.257    0.797
skin          -0.002      0.022  -0.085    0.932
```

```
bmi          0.084      0.043   1.953    0.051
ped          1.820      0.666   2.735    0.006
age          0.041      0.022   1.864    0.062
```

These results suggest that `npreg`, `bp` and `skin` can be omitted without much change to predictive accuracy. Predictive accuracy may actually increase. There is however, no guarantee of this, and it is necessary to check. Even though there is individually no detectable effect, the combined effect of two or more of them may be of consequence.

Using this logistic regression approach, there is no built-in provision to adjust for prior probabilities. Users can however make their own adjustments.

One advantage of `glm()` is that the `termplot()` function is available to provide a coarse check on possible nonlinear effects of covariates. Use `termplot()` with a `glm` object as the first argument, and with the argument `smooth=panel.smooth`. The resulting graphs can be examined for hints of nonlinear covariate effects. Detection of nonlinear effects may require very extensive data.

3 Data that are More Challenging – the crx Dataset

The data can be copied from the web:

```
> webpage <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.data"
> webn <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.names"
> test <- try(readLines(webpage)[1])
> if (!inherits(test, "try-error")){
+   download.file(webpage, destfile="crx.data")
+   crx <- read.csv("crx.data", header=FALSE, na.strings="?")
+   download.file(webn, destfile="crx.names")
+   }
```

Column 16 is the outcome variable. Factors can be identified as follows:

```
> if(exists("crx"))
+   sapply(crx, function(x)if(is.factor(x))levels(x))
```

These data have a number of factor columns. It will be important to understand how they are handled.

3.1 Factor terms – contribution of the model matrix

As with normal theory linear models, the matrix has an initial column of ones that allows for a constant term. (In all rows, the relevant parameter is muitiplied by 1.0, so that the contribution to the fitted value is the same in all rows.) For terms that correspond directly to variables, the model matrix incoporates the variable directly as one of its columns. With the default handling of a factor term

- Implicitly there is a column that corresponds to the initial level of the factor, but as it has all elements 0 it can be omitted;
- For the third and any subsequent levels, the model matrix has a column that is zeros except for rows where the factor is at that level.

A factor that has only two levels will generate a single column, with 0s correponding to the first level, and 1s for the second level. The Pima data has, except for the response variable `type`, no binary variables.

3.2 Fitting the model

Exercise 7
Now fit a linear discriminant model:

```
> if(exists("crx")){
+   crxRed <- na.omit(crx)
+   crxCV.lda <- lda(V16 ~ ., data=crxRed, CV=TRUE)
+   confusion(crxRed$V16, crxCV.lda$class)
+ }
```

Note the message

```
Warning message:
In lda.default(x, grouping, ...) : variables are collinear
```

Now, for comparison, fit the model using `glm()`. This is one way to get details on the reasons for collinearity. Also, for using `glm()`, the argument `na.action=na.exclude` is available, which omits missing values when the model is fit, but then places `NA`s in those positions when fitted values, predicted values, etc., are calculated. This ensures that predicted values match up with the rows of the orginal data.

```
> if(exists("crx")){
+   crx.glm <- glm(V16 ~ ., data=crx, family=binomial, na.action=na.exclude)
+   alias(crx.glm)
+   confusion(crx$V16, round(fitted(crx.glm)))
+   summary(crx.glm)$coef
+ }
```

From the output from `alias(crx.glm)`, what can one say about the reasons for multi-collinearity?

Now display the scores from the linear discriminant calculations:

```
> if(exists("crx")){
+   crxRed <- na.omit(crx)
+   crx.lda <- lda(V16 ~ ., data=crxRed)
+   crx.hat <- predict(crx.lda)
+   densityplot(~crx.hat$x, groups=crxRed$V16)
+ }
```

This plot is actually quite interesting. What does it tell you?

4 Use of Random Forest Results for Comparison

A good strategy is to use results from the random forests method for comparison. The accuracy of this algorithm, when it does give a worthwhile improvement over `lda()`, is often hard to beat. This method has the advantage that it can be applied pretty much automatically. It is good at handling situations where explanatory variables and factors interact in a relatively complex manner.

Here are results for `Pima.tr` as training data, at the same time applying predictions to `Pima.te` as test data. Notice that there are two confusion matrices, one giving the OOB estaimtes for `Pima.tr`, and the other for `Pima.te`.

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

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

        OOB estimate of  error rate: 28.5%
Confusion matrix:
     No Yes class.error
No  109  23       0.174
Yes  34  34       0.500
                Test set error rate: 23.5%
Confusion matrix:
     No Yes class.error
No  192  31       0.139
Yes  47  62       0.431
```

Look at the OOB estimate of accuracy, which is pretty much equivalent to a cross-validation estimate of accuracy. This error will be similar to the error on test data that are randomly chosen from the same population.

The accuracy is poorer than for `lda()`. As before, the error rate is lower on `Pima.te` than on `Pima.te`. Note however the need to re-run the calculation several times, as the accuracy will vary from run to run.

Here are results for `crx`.

```
> if(exists("crxRed")){
+   crx.rf <- randomForest(V16 ~ ., data=crxRed)
+   crx.rf
+ }

Call:
 randomForest(formula = V16 ~ ., data = crxRed)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 11.9%
Confusion matrix:
    +   - class.error
+ 255  41       0.139
-  37 320       0.104
```

Accuracy is similar to that from use of `lda()`.

5 Note – The Handling of NAs

The assumption that underlies any analysis that omits missing values is that, for purposes of the analysis, missingness is uninformative. This may be incorrect, and it is necessary to ask: Are the subjects where there are missing values different in some way?

The missing value issue is pertinent both to the Pima data and to the `crx` data. There is a further dataset, `Pima.tr2`, that augments `Pima.tr` with 100 subjects that have missing values in one or more of the explanatory variables. The question then arises: Is the pattern of missingness the same for those without diabetes as for those with diabetes?

The following shows the numbers of missing values for each of the variables

```
> if(exists("Pima.tr2", where=".GlobalEnv", inherits=FALSE))
+   rm(Pima.tr2)
> sapply(Pima.tr2,function(x)sum(is.na(x)))

npreg   glu    bp  skin   bmi   ped   age  type
    0     0    13    98     3     0     0     0

> sum(!complete.cases(Pima.tr2))

[1] 100
```

Note that the variable `skin` accounts for 98 of the 100 subjects where there is one or more missing value.

A first step is to check whether the subjects with one or more missing values differ in some systematic manner from subjects with no missing values. The major issue is for values that are missing for `skin`. We start by creating a new variable – here named `complete` – that distinguishes subjects with missing values for `skin` from others. We omit observations that are missing on any of the other variables.

```
> newPima <- subset(Pima.tr2, complete.cases(bp) & complete.cases(bmi))
> newPima$NOskin <- factor(is.na(newPima$skin), labels=c("skin","NOskin"))
>   ## NB: FALSE (skin is not NA) precedes TRUE in alphanumeric order
> newPima$skin <- NULL   # Omit the column for skin
```

The argument `labels=c("skin","NOskin")` takes the values (here `FALSE` and `TRUE`) in alphanumeric order, then making `skin` and `NOskin` the levels. Omission of this argument would result in levels `FALSE` and `TRUE`.[5]

We now do a linear discriminant analysis in which variables other than `skin` are explanatory variables.

```
> completeCV.lda <- lda(NOskin ~ npreg+glu+bp+bmi+ped+age+type,
+                       data=newPima, CV=TRUE)
> confusion(newPima$NOskin, completeCV.lda$class)

      Overall accuracy   Prior frequency.skin Prior frequency.NOskin
                 0.694                  0.704                  0.296

Confusion matrix
        Predicted (cv)
Actual     skin NOskin
  skin    0.970 0.0300
  NOskin 0.964 0.0357
```

A linear discriminant analysis seems unable to distinguish the two groups. The overall accuracy does not reach what could be achieved by predicting all rows as complete.

5.1 Does the missingness give information on the diagnosis?

If there is a suspicion that it does, then a valid analysis may be possible as follows. Missing values of continuous variables are replaced by 0 (or some other arbitrary number). For each such variable, and each observation for which it is missing, there must be a factor, e.g. with levels `miss` and `nomiss`, that identifies the subject for whom the value is missing. Where values of several variables are missing for the one subject, the same factor may be used. This allows an analysis in which all variables, together with the newly created factors, are "present" for all subjects.

[5] NB also `factor(is.na(newPima$skin), levels=c(TRUE, FALSE))`; levels would then be `TRUE` and `FALSE`, in that order.

Part XIII
Discriminant Methods & Associated Ordinations

Packages: `e1071`, `ggplot2`, `mlbench` (this has the `Vehicle` dataset), `mclust`, textttrandomForest

These exercises will introduce classification into three or more groups. They examine and compare three methods – linear discriminant analysis, Support Vector machines, and random forests.

As noted in an earlier laboratory, linear discriminant analysis generates scores that can be plotted directly. Support Vector Machines generate decision scores, one set for each of the g groups; these can be approximated in $g-1$ dimensional and plotted. For random forests the pairwise proximity of points, calculated as the proportion of trees for whiah the two observations end up in the same terminal node, is a natural measure of the relative nearness. These can be subtracted from 1.0 to yield relative distances, with non-metric scaling then be used to obtain a representation in a low-dimensional space. In favorable cases, metric scaling may yield a workable represention, without going to further step to non-metric scaling.

Again, it will be handy to have the function `confusion()` available.

```
> confusion <- function(actual, predicted, names=NULL,
+                       printit=TRUE, prior=NULL){
+   if(is.null(names))names <- levels(actual)
+   tab <- table(actual, predicted)
+   acctab <- t(apply(tab, 1, function(x)x/sum(x)))
+   dimnames(acctab) <- list(Actual=names,
+                       "Predicted (cv)"=names)
+   if(is.null(prior)){
+     relnum <- table(actual)
+     prior <- relnum/sum(relnum)
+     acc <- sum(tab[row(tab)==col(tab)])/sum(tab)
+   } else
+   {
+     acc <- sum(prior*diag(acctab))
+     names(prior) <- names
+ }
+   if(printit)print(round(c("Overall accuracy"=acc,
+                            "Prior frequency"=prior),4))
+   if(printit){
+     cat("\nConfusion matrix", "\n")
+     print(round(acctab,4))
+   }
+   invisible(acctab)
+ }
```

1 Discrimination with Multiple Groups

With three groups, there are three group centroids. These centroids determine a plane, onto which data points can be projected. Linear discrinimant analysis assumes, effectively, that discriminants can be represented without loss of information in this plane. It yields two sets of linear discriminant scores that can be plotted, giving an accurate graphical summary of the analysis.

More generally, with $g \geq 4$ groups, there are $max(g-1,p)$ dimensions, and $max(g-1,p)$ sets of linear discriminant scores. With three (or more) sets of scores, the function `plot3d()` in the *rgl* package can be used to give a 3D scatterplot that rotates dynamically under user control.

The output from printing an `lda` object that is called with `CV=FALSE` (the default) includes "proportion of trace" information. The successive linear discriminants explain successively smaller proportions of the trace, i.e., of the ratio of the between to within group variance. It may turn out that the final discriminant or the final few discriminants explain a very small proportion of the trace, so that they can be dropped.

With methods other than `lda()`, representation in a low-dimensional space is still in principle possible. However any such representation arises less directly from the analysis, and there is nothing directly comparable to the "proportion of trace" information.

1.1 The `diabetes` dataset

The package *mclust* has the data set `diabetes`. Datasets in this package do not automatically become available when the package is attached.[6] Thus, it is necessary to bring the data set `diabetes`it into the workspace by typing

```
> library(mclust)
> data(diabetes)
```

Here is a 3-dimensional cloud plot:

```
> library(lattice)
> cloud(insulin ~ glucose+sspg, groups=class, data=diabetes)
```

1.2 Linear discriminant analysis

First try linear discriminant analysis. We specify `CV=TRUE`, in order to obtain predictions of class membership that are derived from leave-one-out cross-validation. We then run the calculations a second time, now with `CV=FALSE`, in order to obtain an object from which we can obtain the discriminant scores

```
> library(MASS)
> library(lattice)
> diabetesCV.lda <- lda(class ~ insulin+glucose+sspg, data=diabetes, CV=TRUE)
> confusion(diabetes$class, diabetesCV.lda$class)
> diabetes.lda <- lda(class ~ insulin+glucose+sspg, data=diabetes)
> diabetes.lda              # Linear discriminant 1 explains most of the variance
> hat.lda <- predict(diabetes.lda)$x
> xyplot(hat.lda[,2] ~ hat.lda[,1], groups=diabetes$class,
+        auto.key=list(columns=3), par.settings=simpleTheme(pch=16))
```

1.3 Quadratic discriminant analysis

The plot of linear discriminant scores makes it clear that the variance-covariance structure is very different between the three classes. It is therefore worthwhile to try `qda()`.

```
> diabetes.qda <- qda(class ~ insulin+glucose+sspg, data=diabetes, CV=TRUE)
> confusion(diabetes$class, diabetes.qda$class)
```

The prediction accuracy improves substantially.

[6] This package does not implement the lazy data mechanism.

Exercise 1
Suppose now that the model is used to make predictions for a similar population, but with a different mix of the different classes of diabetes.

(a) What would be the expected error rate if the three classes occur with equal frequency?

(b) What would be the expected error rate in a population with a mix of 50% chemical, 25% normal and 25% overt?

1.4 Support vector machines

This requires the *e1071* package. Here, we can coonveniently specify tenfold cross-validation, in order to estimate predictve accuracy. A plot of points can be based on a low-dimensional representation of the "decision values".

```
> library(e1071)
> diabetes.svm <- svm(class ~ insulin+glucose+sspg, data=diabetes, cross=10)
> summary(diabetes.svm)
> pred <- predict(diabetes.svm, diabetes[,-1], decision.values = TRUE)
> decision <- attributes(pred)$decision.values
> decision.dist <- dist(decision)
> decision.cmd <- cmdscale(decision.dist)
> library(lattice)
> xyplot(decision.cmd[,2] ~ decision.cmd[,1], groups=diabetes$class)
```

The cross-validated prediction accuracy is lower than using `lda()`. There is however substantial scope for using another kernel, and/or setting various tuning parameters. If there is such tuning, care is needed to ensure that the tuning does not bias the accuracy measure. Two possibilities are: (1) divide the data into training and test sets, and use the test set accuracy rather than the cross-validation accuracy; or (2) repeat the tuning at each cross-validation fold. Automation of option (2) will require the writing of special code. Depending on the nature of the tuning, it may not be straightforward.

1.4.1 A rubber sheet representation of the decision values

The plot given above projected the decision values on to a Euclidean space. A non-metric representation may be preferable. The following uses the *MASS* program `isoMDS()` to obtain a 2-dimensional representation. The function `isoMDS()` requires a starting configuration; we use the values from `cmdscale()` for that purpose:

```
> diabetes.mds <- isoMDS(decision.dist, decision.cmd)
```

It turns out that observations 4 and 80 have zero distance, which `isoMDS()` is unable to handle. We therefore add a small positive quantity to the distance between these two observations.

```
> eps <- min(decision.dist[decision.dist > 0])
> ## We add half the smallest non-zero distance
> decision.dist[decision.dist == 0] <- eps
> diabetes.mds <- isoMDS(decision.dist, decision.cmd)
> xyplot(diabetes.mds$points[,2] ~ diabetes.mds$points[,1],
+        groups=diabetes$class)
```

1.5 Use of `randomForest()`

First, fit a randomForest discriminant model, calculating at the same time the proximities. The proximity of any pair of points is the proportion of trees in which the two points appear in the same terminal node:

```
> ## Use randomForest(), obtain proximities
> library(randomForest)
> diabetes.rf <- randomForest(class~., data=diabetes, proximity=TRUE)
> print(diabetes.rf)
```

Note the overall error rate.

Exercise 2
Suppose that the model is used to make predictions for a similar population, but with a different mix of the different classes of diabetes.

(a) What would be the expected error rate if the three classes occur with equal frequency?

(b) What would be the expected error rate in a population with a mix of 50% chemical, 25% normal and 25% overt?

Points that in a large proportion of trees appear at the same terminal node are in some sense "close together", whereas points that rarely appear in the same terminal node are "far apart". This is the motivation for subtracting the proximities from 1.0, and treating the values obtained as distances in Euclidean space. Inititially, a two-dimensional representation will be tried. If this representation reproduces the distances effectively, the result will be a plot in which the visual separation of the points reflects the accuracy with which the algorithm has been able to separate the points:

```
> ## Euclidean metric scaling
> diabetes.rf.cmd <- cmdscale(1-diabetes.rf$proximity)
> plot(diabetes.rf.cmd, col=unclass(diabetes$class)+1)
```

The clear separation between the three groups, on this graph, is remarkable, though perhaps to be expected given that the OOB error rate is ∼2%.

1.5.1 A rubber sheet representation of the proximity-based "distances"

There is no necessary reason why distances that have been calculated as above should be Euclidean distances. It is more reasonable to treat them as relative distances. The `isoMDS()` function in the *MASS* package uses the ordinates that are given by `cmdscale()` as a starting point for Kruskal's "non-metric" multi-dimensional scaling. In effect distances are represented on a rubber sheet that can be stretched or shrunk in local parts of the sheet, providing only that relative distances are unchanged.

Almost certainly, some distances will turn out to be zero. In order to proceed, zero distances will be replaced by 0.5 divided by the number of points. (The minimum non-zero distance must be at least `1/dim(diabetes)[1]`. Why?)

```
> sum(1-diabetes.rf$proximity==0)
> distmat <- as.dist(1-diabetes.rf$proximity)
> distmat[distmat==0] <- 0.5/dim(diabetes)[1]
```

We now proceed with the plot.

```
> diabetes.rf.mds <- isoMDS(distmat, y=diabetes.rf.cmd)
> xyplot(diabetes.rf.mds$points[,2] ~ diabetes.rf.mds$points[,1],
+        groups=diabetes$class, auto.key=list(columns=3))
```

Note that these plots exaggerate the separation between the groups. They represent visually the effectiveness of the algorithm in classifying the training data. To get a fair assessment, the plot should show points for a separate set of test data.

Exercise 10: Make a table that compares `lda()`, `svm()` and `randomForest()`, with respect to error rate for the three classes separately.

1.6 Vehicle dataset

Repeat the above, now with the `Vehicle` data frame from the *mlbench* package. For the plots, ensure that the *ggplot2* package (and dependencies) is installed.

Plots will use `quickplot()`, from the *ggplot2* package. This makes it easy to get density plots. With respect to arguments to `quickplot()`, note that:

- `quickplot()` has the `geom` (not `geometry`) argument, where base and lattice graphics would use `type`;
- If different `colours` are specified, data will be grouped according to those colours.

```
> library(ggplot2)
> library(mlbench)
> data(Vehicle)
> VehicleCV.lda <- lda(Class ~ ., data=Vehicle, CV=TRUE)
> confusion(Vehicle$Class, VehicleCV.lda$class)
> Vehicle.lda <- lda(Class ~ ., data=Vehicle)
> Vehicle.lda
> hat.lda <- predict(Vehicle.lda)$x
> quickplot(hat.lda[,1], hat.lda[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
```

1.6.1 Vehicle dataset – Quadratic discriminant analysis

```
> Vehicle.qda <- qda(Class ~ ., data=Vehicle, CV=TRUE)
> confusion(Vehicle$Class, Vehicle.qda$class)
```

Support Vector Machines

```
> Vehicle.svm <- svm(Class ~ ., data=Vehicle, cross=10)
> summary(Vehicle.svm)
> pred <- predict(Vehicle.svm, Vehicle, decision.values = TRUE)
> decision <- attributes(pred)$decision.values
> decision.dist <- dist(decision)
> eps <- min(decision.dist[decision.dist>0])/2
> decision.cmd <- cmdscale(decision.dist)
> quickplot(decision.cmd[,1], decision.cmd[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
> ## Now try Kruskal's non-metric MDS
> decision.dist[decision.dist == 0] <- eps
> decision.mds <- isoMDS(decision.dist, decision.cmd)$points
> quickplot(decision.mds[,1], decision.mds[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
```

Compare the metric and non-metric plots. Do they tell the same story?

1.7 Vehicle dataset – random forests

```
> Vehicle.rf <- randomForest(Class~., data=Vehicle, proximity=TRUE)
> print(Vehicle.rf)
> distmat <- 1-Vehicle.rf$proximity
> Vehicle.rf.cmd <- cmdscale(distmat)
> quickplot(Vehicle.rf.cmd[,1], Vehicle.rf.cmd[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
> eps <- min(distmat[distmat>0])/2
> distmat[distmat == 0] <- eps
```

```
> Vehicle.rf.mds <- isoMDS(distmat, Vehicle.rf.cmd)$points
> quickplot(Vehicle.rf.mds[,1], Vehicle.rf.mds[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
```

Now calculate the distances for the points generated by `isoMDS()`, and plot these distances against distances (in `distmat`) generated by subtracting the proximities from 1.

```
> mdsdist <- dist(Vehicle.rf.mds)
> plot(as.dist(distmat), mdsdist)
```

Part XIV
Ordination

Packages: `DAAGxtras`, `ape`, `cluster`, `mclust`, `oz`

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])))
+ }
> colnames(misfit) <- abbreviate(colnames(misfit), 6)
> print(round(misfit))
```

	Adelad	Alice	Brisbn	Broome	Cairns	Canbrr	Darwin	Melbrn	Perth	Sydney
Adelaide	0	140	-792	-156	366	20	11	82	482	-273
Alice	140	0	-1085	-175	-41	76	-118	106	-26	-314
Brisbane	-792	-1085	0	198	319	-25	-233	-471	153	-56
Broome	-156	-175	198	0	527	-7	6	-65	990	70
Cairns	366	-41	319	527	0	277	-31	178	8	251
Canberra	20	76	-25	-7	277	0	-1	-241	372	-8
Darwin	11	-118	-233	6	-31	-1	0	-12	92	-58
Melbourne	82	106	-471	-65	178	-241	-12	0	301	-411
Perth	482	-26	153	990	8	372	92	301	0	271
Sydney	-273	-314	-56	70	251	-8	-58	-411	271	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]){
+ ## Get best fit in space of (latitude, longitude)
+ 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)
> ## Examine distribution of distances
> 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"
> test <- try(readLines(webpage)[1])
> if (!inherits(test, "try-error")){
+ primates.dna <- read.dna(con <- url(webpage))
+ close(con)
+ hasdata <- TRUE
+ } else hasdata <- FALSE
```

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

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

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

```
> if(hasdata){
+ 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:

```
> if(hasdata){
+ d <- dist(primates.cmd)
+ sum((d-primates.dist)^2)/sum(primates.dist^2)
+ }
```

```
[1] 0.1977
```

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.

```
> if(hasdata){
+ 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] 0.1045
```

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

```
> if(hasdata){
+ primates.sam <- sammon(primates.dist, k=3)
+ 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)
+ }
```

```
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
```

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

```
> if(hasdata){
+ primates.mds <- isoMDS(primates.dist, primates.cmd, k=3)
+ 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)
+ }
```

```
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
```

. . . . . .
. . . . . .

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.

```
> ## Diptera
> webpage <- "http://www.maths.anu.edu.au/~johnm/datasets/ordination/dipt.csv"
> test <- try(readLines(webpage)[1])
> if (!inherits(test, "try-error")){
+ diptera <- read.csv(webpage, comment="", na.strings=c(NA,""))
+ dim(diptera)
+ ## Now observe that the names all start with "#"
+ species <- as.character(diptera[,1])
+ table(substring(species, 1, 1))
+ ##  Now strip off the initial "#"
+ species <- substring(species,2)
+ genus <- as.character(diptera[,2])
+ ## Now store columns 5 to 114 of diptera as a distance object.
+ ## The distance matrix stores, in vector form, the elements of the
+ ## matrix that are below the diagonal.
+ dipdist <- as.dist(diptera[1:110,5:114])
+ attributes(dipdist)$Labels <- species
+ length(dipdist)   ## The length reflects the number of elements in
+                   ## the lower triangle, i.e., below the diagonal
+ hasdata <- TRUE
+      } else hasdata <- FALSE
```

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.

```
> if(hasdata)
+ 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.

```
> if(hasdata){
+ 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 *DAAGxtras* package.

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:

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

[1] 0.6312

> plot(density(pacific.dist, to = 1))
> ## Now check that all columns have some distances that are less than 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)
> eqscplot(pacific.cmd)
```

Part XV
Trees, SVM, and Random Forest Discriminants

Packages: `e1071`, `lattice`, `randomForest`

1 rpart Analyses – the `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.

1.1 Fitting the model

Fit an rpart model to the `Pima.tr` data:

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

The formula `type` $\sim$ `.` has the effect of using as explanatory variables all columns of `Pima.tr` 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)   # NB: plot, not plotcp()
> text(Pima.rpart4)      # Labels the tree
```

Note also the printed output from

```
> printcp(Pima.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 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.00 0.099
3 0.074      2      0.62   0.85 0.094
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.71 0.089
6 0.010      7      0.44   0.78 0.092
```

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.tr, 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.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 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.91 0.096
4 0.059      3      0.54   0.90 0.096
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.81 0.093

> colnames(errmat)        # Hints at what will come next

[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.tr$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  0.8636 0.1364
       Yes 0.4706 0.5294
```

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`.

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

Exercise 3a: Refit the model.

```
> 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 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   0.90 0.096
3 0.074      2      0.62   0.79 0.092
4 0.059      3      0.54   0.79 0.092
5 0.015      4      0.49   0.68 0.088
6 0.010      7      0.44   0.75 0.091
```

Choose values of `cp` that will give the points on the graph given by `plotcp(trPima.rpart)`. 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 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.87 0.095
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.85 0.094

> 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 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.87 0.095
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.85 0.094
```

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 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.87 0.095
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.85 0.094

> 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*

Random forests can be fit pretty much automatically, with little need or opportunity for tuning. For datasets where there is a relatively complex form of dependence on explanatory factors and variables, random forests may give unrivalled accuracy.

Fiting proceeds by fitting (in fact, overfitting) many (by default, 500) trees, with each tree fitted to a different bootstrap sample of the data, with a different random sample of variables also. Each tree is taken to its full extent. The predicted class is then determined ny a simple vote over all trees.

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$typ
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 29%
Confusion matrix:
     No Yes class.error
No  108  24      0.1818
Yes  34  34      0.5000
                Test set error rate: 23.49%
Confusion matrix:
     No Yes class.error
No  191  32      0.1435
Yes  46  63      0.4220
```

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.tr, method="class")
> Pima.rf.4 <- randomForest(type ~ ., data=Pima.tr, method="class", classwt=c(.6,.4))
> Pima.rf.1 <- randomForest(type ~ ., data=Pima.tr, method="class", classwt=c(.9,.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.tr, 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.tr$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 seems to have little effect.

6.1 Prior probabilities

The effect of assigning prior probabilities can be achieved by choosing the elements of the parameter `sampsize` (the bootstrap sample sizes) so that they are in the ratio of the required prior probabilities.

```
> ## Default
> randomForest(type ~ ., data=Pima.tr, sampsize=c(132,68))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(132,      68))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 29.5%
Confusion matrix:
     No Yes class.error
No  108  24      0.1818
Yes  35  33      0.5147
```

```
> ## Simulate a prior that is close to 0.8:0.2
> randomForest(type ~ ., data=Pima.tr, sampsize=c(132,33))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(132,      33))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 26%
Confusion matrix:
     No Yes class.error
No  122  10     0.07576
Yes  42  26     0.61765

>   # Notice the dramatically increased accuracy for the No's
> ## Simulate a prior that is close to 0.1:0.9
> randomForest(type ~ ., data=Pima.tr, sampsize=c(17,68))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(17,      68))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 40.5%
Confusion matrix:
    No Yes class.error
No  58  74      0.5606
Yes  7  61      0.1029
```

With small sample sizes, it may be beneficial to fit a greater number of trees.

7 Further Examples

(a) Look up the help page for the dataset `Vehicle` from the *mlbench* package:

   (a) Fit a linear discriminant analysis model (`lda()`). Plot the first two sets of discriminant scores, identifying the different vehicle types;

   (b) Try quadratic discriminant analysis (`qda()`) for comparison;

   (c) Compare the accuracy with that from fitting a random forest model. Use the proximities to derive a three-dimensional representation of the data. Examine both the projection onto two dimensions and (using `rgl()` from the *rgl* package) the three-dimensional representation.

   (d) Compare the two (or more; you might derive more than one plot from the proximities) low-dimensional plots. Do any of the plots offer any clue on why quadratic discriminant analysis is so effective?

(b) Repeat Exercise 1, now with the dataset `fgl` (forensic glass data) in the *MASS* package. Note that, for these data, `qda()` fails. Why does it fail?

Part XVI
Data Exploration and Discrimination – Largish Dataset

`DAAGxtras`, `randomForest`

Note: These computations are at the limit, or beyond, what a machine with 512MB memory and running Microsoft Windows is able to handle. A reboot may be necessary in order to get the calculations to run. If you cannot get calculations to run at all, try taking every second observation in the relevant data frames, and working with these reduced datasets.

One a 512MB Mac system running OS X, calculations have run without problem. The same is almost certainly true on a Unix or Linux system (I have not tested this),

The data used in this laboratory are derived from a dataset that gives forest cover type (seven different types), for 581,012 sites in the United States. There are 54 columns of explanatory features (variables or dummy variables that code for qualitative effects). The 55th column holds the cover type, given as an integer in the range 1 to 7. Data, stored in the R image file **covtype.RData**, are available from
`http://www.maths.anu.edu.au/~johnm/datasets/forestCover/`
[To obtain the data as a text file, go to the UCI Machine Learning Repository at
`http://www.ics.uci.edu/~mlearn/MLRepository.html`

1 Data Input and Exploration

As available from the repository, data are comma delimited, and (assuming that the file has been placed in the working directory; if accessible from another location, the path must be included) can be read in with

```
covtype <- read.csv("covtype.data", header=FALSE)
```

For purposes of this laboratory, data have been placed in the image file **covtype.RData**.

The image file `covtype.RData` holds all 581,012 records. The first 11340 records have been sampled in some systematic way from an original total data set, for use as a training set. The next 3780 records, again sampled in some systematic way from the total data, were used as test data. Below, the first 11340 records will be extracted into the data frame `covtrain`, while the next 3780 records will be extracted into the data frame `covtest`.

1.1 Extraction of subsets of interest

The following extracts these data, plus a systematic sample of every 50th record, taken right through the 565,982 records that remain after the training and test sets have been extracted. If these calculations prove troublesome on your system, skip them.[7] The datasets `covtrain`, `covtest` and `covsample` are included with the *DAAGxtras* package.

The following assumes that the file **covtype.RData** is in your working directory; if it is elsewhere you must of course include the path:

```
> attach("covtype.RData")
> covtrain <- covtype[1:11340,]
> covtest <- covtype[11340+(1:3780),]
> every50th <- seq(from=11340+3780+1, to=581012, by=50)
```

[7] On a 1.25GHz G4 Powerbook with 512MB of RAM, the elapsed time for extraction of `covtrain` was 28 seconds while for extraction of `covtest` the time was 16 seconds (try, e.g., `system.time(covtest <- covtype[11340+(1:3780),])` – the third of these numbers is the elapsed time). These times are however highly variable. On less well endowed systems, the calculations may take an unreasonable time, or may not run at all.

```
> covsample <-  covtype[every50th, ]
> tab.all <- table(covtype$V55)    # Keep for later
> detach("file:covtype.RData")
```

Because data are stored columnwise, extraction of subsets of rows that spread across all columns requires substantial manipulation within memory, which can be time-consuming.

An alternative, for the input of `covtrain` and `covtest`, is:

```
covtrain <- read.csv("covtype.data", header=FALSE, nrows=11340)
covtest <- read.csv("covtype.data", header=FALSE, skip=11340, nrows=3780)
```

(Use these on Windows machines with less than 512MB of random access memory.)

Question: Which of the following is preferable?

```
every50th <- seq(from=11340+3780+1, to=581012, by=50)
every50th <- seq(from=15121, to=581012, by=50)
every50th <- 15121+(0:((581012-15121) %/% 50))*50
```

Which is more consistent with notions of literate programming? Is computational efficiency a consideration?
(The symbol `%/%` is the integer division operator. Try, e.g., `11 %/% 3`, `12 %/% 3`, etc. Try also `11 %% 3`, `12 %% 3`, which give the remainders after division.)

1.2 Image files

Having extracted these data, they can be saved to image files, which can then be attached so that they are available as required:

```
> save(covtrain, file="covtrain.RData")
> rm(covtrain)
> save(covtest, file="covtest.RData")
> rm(covtest)
> save(covsample, file="covsample.RData")
> rm(covsample)
```

Now attach these image files, making `covtrain`, `covtest` and `covsample` available as required:

```
> attach("covtrain.RData")
> attach("covtest.RData")
> attach("covsample.RData")
```

Alternatively, they can be made available by typing

```
> library(DAAGxtras)
```

1.3 Data exploration

Next, we extract some basic statistical information about these data:

```
> options(digits=3)
> tab.train <- table(covtrain$V55)
> tab.test <- table(covtest$V55)
> tab.sample <- table(covsample$V55)
> tab.all/sum(tab.all)
```

```
      1       2       3       4       5       6       7
0.36461 0.48760 0.06154 0.00473 0.01634 0.02989 0.03530
```

```
> tab.sample/sum(tab.sample)

       1        2        3        4        5        6        7
0.371620 0.494169 0.059640 0.000707 0.014579 0.028185 0.031101

> tab.train/sum(tab.train)

    1     2     3     4     5     6     7
0.143 0.143 0.143 0.143 0.143 0.143 0.143

> tab.test/sum(tab.test)

    1     2     3     4     5     6     7
0.143 0.143 0.143 0.143 0.143 0.143 0.143
```

What is interesting is that the proportions of the different cover types, in both the training and the test data, are equal. That is not the case for the data as a whole, and in this respect the "training" and "test" data are untypical.

The above suggests that, in forming the training and test data, observations were taken from the original main body of data until cover types 3-7 were largely "used up". It might be suspected that the proportions of the different cover types will vary systematically as one moves through data. The following function, which models the occurrence of the specified forest cover as a function of distance (as a proportion) through the data, can be used to check this:

```
> library(splines)
> runningprops <- function(df=covtrain, type=1){
+   n <- dim(df)[1]
+   propthru <- (1:n)/(n+1)
+   occurs <- as.integer(df$V55==type)
+   print(table(occurs))
+   cov.glm <- glm(occurs ~ bs(propthru,6), family=binomial)
+   hat <- predict(cov.glm, type="response")
+   cbind(propthru, hat)
+ }
> hat.train <- runningprops()

occurs
   0    1
9720 1620

> hat.test <- runningprops(df=covtest)

occurs
   0    1
3240  540

> hat.sample <- runningprops(df=covsample)

occurs
   0    1
7112 4206

> print(range(c(hat.train[,2], hat.test[,2], hat.sample[,2])))

[1] 0.00264 0.64866
```

Next, plot this information:

```
> plot(hat.train[,1], hat.train[,2], ylim=c(0,0.65))
> lines(hat.test[,1], hat.test[,2], col=2)
> lines(hat.sample[,1], hat.sample[,2], col=3)
```

What does this plot suggest?

Exercise 1: Repeat the above plots, but now for forest cover type 2.

Exercise 2: Another way to estimate `hat` would be as a moving average of values of the variable `occurs` in the above function. Write a function to calculate moving averages, with the window `wid` as one of its parameters.

Exercise 3: What other preliminary explorations of these data might be useful?

2 Tree-Based Classification

Now fit a tree-based model for `covtrain`:

```
> library(rpart)
> train.rpart <- rpart(V55 ~ ., data=covtrain, cp=0.0001, method="class")
> train.rpart <- prune(train.rpart, cp=0.0048)
> trainhat.train <- xpred.rpart(train.rpart, cp=0.0048)
> testhat.train <- predict(train.rpart, newdata=covtest, type="class")
> samplehat.train <- predict(train.rpart, newdata=covsample, type="class")
```

Next, we will define a function that calculates error rates for each different cover type:

```
> errs.fun <- function(obs, predicted){
+ tab <- table(obs, predicted)
+ grosserr <- 1-sum(tab[row(tab)==col(tab)])/sum(tab)
+ errs <- 1-tab[row(tab)==col(tab)]/apply(tab,1,sum)
+ names(errs) <- paste(1:length(errs))
+ print("Overall error rate (%)")
+ print(round(100*grosserr,1))
+ print("Error rate (%), broken down by forest cover type")
+ print(round(100*errs,2))
+ cat("\n")
+ invisible(errs)
+ }
```

Now apply the function, first to `trainhat.train`, then to `testhat.train`, and finally to `samplehat.train`:

```
> errs.fun(covtrain$V55, trainhat.train)

[1] "Overall error rate (%)"
[1] 35.4
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
51.23 63.70 45.56 11.11 15.12 54.14  6.91

> errs.fun(covtest$V55, testhat.train)

[1] "Overall error rate (%)"
[1] 34.5
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
55.56 60.74 40.19 10.37 11.67 56.11  6.67

> errs.fun(covsample$V55, samplehat.train)

[1] "Overall error rate (%)"
[1] 55.6
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
54.11 62.40 43.41  0.00 16.97 56.11  6.25
```

Exercise 4: Explain: calculations:

- What data have been used, in each case, to test the model?
- Explain the notation used for the different sets of fitted values (`trainhat.train`, `testhat.train`, `samplehat.train`.)
- Why is it that, although the three sets of error rates are relatively similar when broken down by cover type, are there are major differences in the overall error rate?
- Which, if any, of these overall error rates is it reasonable to quote in practical application of the results? If none of them seem appropriate, what error rate do you consider should be quoted?

Exercise 5: Do the following calculation and comment on the results:

```
> sample.rpart <- rpart(V55 ~ ., data=covsample, cp=0.0001, method="class")
> sample.rpart <- prune(sample.rpart, cp=0.001)
> samplehat.sample <- xpred.rpart(sample.rpart, cp=0.001)
> samplehat.sample <- factor(samplehat.sample, levels=1:7)
> trainhat.sample <- predict(sample.rpart, newdata=covtrain, type="class")
> testhat.sample <- predict(sample.rpart, newdata=covtest, type="class")
> errs.fun(covtrain$V55, trainhat.sample)

[1] "Overall error rate (%)"
[1] 59.8
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
 30.4  17.9  12.2 100.0 100.0 100.0  58.0

> errs.fun(covtest$V55, testhat.sample)

[1] "Overall error rate (%)"
[1] 59.4
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
 30.9  17.6  12.2 100.0 100.0 100.0  54.8

> errs.fun(covsample$V55, samplehat.sample)

[1] "Overall error rate (%)"
[1] 26.9
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
 29.7  18.2  20.7 100.0  97.0  87.2  54.0
```

3 Use of `randomForest()`

For use of `randomForest()`, calculations speed up greatly if explanatory variables are input as columns of a matrix, while (for classification) the response variable is input as a vector of class factor.

Exercise 6: Run the following computations, and interpret the output, comparing it with the relevant output from `rpart()`. Suggest why the error rates may be so much lower than those from `rpart()`:

```
> library(randomForest)
> xcovtrain <- as(covtrain[,1:54], "matrix")
> ycovtrain <- covtrain[,55]
> xsampletrain <- as(covsample[,1:54], "matrix")
> ysampletrain <- covsample[,55]
> ycovtrain <- factor(covtrain[,55])
> ysampletrain <- factor(covsample[,55])
> covtrain.rf <- randomForest(x=xcovtrain, y=ycovtrain,
+                             xtest=xsampletrain, ytest=ysampletrain)
> covtrain.rf

Call:
 randomForest(x = xcovtrain, y = ycovtrain, xtest = xsampletrain,      ytest = ysampletrain)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 7

        OOB estimate of  error rate: 17.4%
Confusion matrix:
     1    2    3    4    5    6    7 class.error
1 1182  244    2    0   40    8  144      0.2704
2  310 1047   40    0  151   57   15      0.3537
3    0    2 1173  118   17  310    0      0.2759
4    0    0   25 1567    0   28    0      0.0327
5    2   63   31    0 1488   36    0      0.0815
6    0    6  183   61   13 1357    0      0.1623
7   69    0    1    0    1    0 1549      0.0438
                Test set error rate: 30.3%
Confusion matrix:
     1    2   3  4   5   6   7 class.error
1 3055  631   5  0 125  23 367      0.2737
2 1032 3575 128  3 579 232  44      0.3608
3    0    2 495 54   6 118   0      0.2667
4    0    0   0  8   0   0   0      0.0000
5    0   10   6  0 145   4   0      0.1212
6    0    1  35 10   4 269   0      0.1567
7   14    1   0  0   0   0 337      0.0426
```

4 Further comments

This has been a preliminary exploration of these data. The model that is optimal changes as one moves through the data. This can be verified by selecting successsive subsets of perhaps 10,000 successive observations at various points through the data (e.g.: 1-10,000, 101,000-110,000, . . .), and comparing: (1) gross error rate for that subset as calculated by using `xpred.rpart()` and `errs.fun()`, with (2) the gross error rate when `train.rpart()` or (more appropriately) `sample.rpart()` is used determine fitted values for that subset.

Thus, after the first 15,120 (11,340 + 3780) records, a reasonable guess is that the order of records reflects an ordering of the data according to geographical location. The ordering holds information that can be used, even without knowledge of more exact geographical locations, to get improved model predictions.