Part VII
Discriminant Methods & Associated Ordinations

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. Random forests generate a natural measure of the relative nearness, or proximity of points. Non-metric scaling can then be used to represent these in $g-1$ dimensional space. In favorable cases, metric scaling may yield a workable represention.

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 such a plane, determined by the linear discriminant scores. A plot of these scores is an accurate summary of the analysis.

With $g \geq 4$ groups, then if $p \geq g-1$ linear discrinimant analysis yields scores that require $g-1 \geq 3$ dimensions for their representation. 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.

How many dimensions are really necessary. To answer this question in the case of `lda()`, examine the "proportion of trace" in the output from printing the `lda` object, when it is called with `CV=FALSE`

which is the default. 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, os 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 not the same well-defined mechanism for deciding on the number of dimensions.

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.[3] Thus, it is necessary to bring the data set `diabetes`it into the workspace by typing

```
> library(mclust)

use of mclust requires a license agreement
see http://www.stat.washington.edu/mclust/license.txt

> data(diabetes)
```

Here is a 3-dimensional cloud plot:

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

1.2 Linear discriminant analysis

We will 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
> 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 variance-covariance structure is clearly 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.

[3] 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 will specify the use of tenfold cross-validation, in order to estimate the error rate. The "decision values" will be used to define co-ordinate axes, so that data can be plotted.

```
> 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 less 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 in obtaining an accuracy measure that is not biased on account of the tuning. 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.

Also possible is to use a scaled version of the decision values. For example, 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.

```
> sort(decision.dist)[1:4]

[1] 0.0000000000 0.0008654923 0.0034727787 0.0054139013

> decision.dist[decision.dist == 0] <- 0.00043
> diabetes.mds <- isoMDS(decision.dist, decision.cmd)

initial  value 0.452432
iter   5 value 0.385817
final  value 0.382896
converged

> 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:

```
> 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:

```
> 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 distance

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 in such a space. 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)

initial  value 8.375905
iter   5 value 6.033850
iter  10 value 5.751992
```

```
iter  15 value 5.595654
iter  20 value 5.543829
iter  20 value 5.540744
iter  20 value 5.540546
final  value 5.540546
converged
```

```
> 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 `colour`s are specified, data will be grouped according to those colours.

```
> library(ggplot2)
> 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"))
```

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"))
> 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?

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)
```