# Part VII
# Discriminant Methods & Associated Ordinations

## 1 Discrimination with Multiple Groups

The package *mclust* has the data set `diabetes`. It does not automatically become available when the package is attached; instead, it is necessary to bring it into the workspace by typing

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

Here is a 3-dimensional cloud plot:

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

### 1.1 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

```
> diabetes.lda <- lda(class ~ insulin + glucose + sspg, data = diabetes,
+     CV = TRUE)
> diabetes.lda
> tab <- table(diabetes$class, diabetes.lda$class)
> 1 - sum(tab[row(tab) == col(tab)])/sum(tab)
> confusion.lda <- rbind(tab[1, ]/sum(tab[1, ]), tab[2, ]/sum(tab[2,
+     ]), tab[3, ]/sum(tab[3, ]))
> dimnames(confusion.lda) <- list(Actual = c("chemical", "normal",
+     "overt"), "Predicted (cv)" = c("chemical", "normal", "overt"))
> print(confusion.lda)
> hat.lda <- predict(lda(class ~ insulin + glucose + sspg, data = diabetes))
> plot(hat.lda$x, col = unclass(diabetes$class) + 1)
```

### 1.2 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)
```

## 1.3 Use of `randomForest()`

First, fit a random tree 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)
```

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. It is however broadly consistent with the OOB error rate of 2%.

### 1.3.1 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 8.004481
iter   5 value 5.261540
iter  10 value 4.948916
final  value 4.875034
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.