Chapter 11

Exploratory Multivariate Analysis

Multivariate analysis is concerned with datasets that have more than one response variable for each observational or experimental unit. The datasets can be summarized by data matrices X with n rows and p columns, the rows representing the observations or cases, and the columns the variables. The matrix can be viewed either way, depending on whether the main interest is in the relationships between the cases or between the variables. Note that for consistency we represent the variables of a case by the *row* vector $\boldsymbol{x}$.

The main division in multivariate methods is between those methods that assume a given structure, for example, dividing the cases into groups, and those that seek to discover structure from the evidence of the data matrix alone (nowadays often called *data mining*, see for example Hand *et al.*, 2001). Methods for known structure are considered in Chapter 12.

In pattern-recognition terminology the distinction is between *supervised* and *unsupervised* methods. One of our examples is the (in)famous iris data collected by Anderson (1935) and given and analysed by Fisher (1936). This has 150 cases, which are stated to be 50 of each of the three species *Iris setosa*, *I. virginica* and *I. versicolor*. Each case has four measurements on the length and width of its petals and sepals. *A priori* this seems a supervised problem, and the obvious questions are to use measurements on a future case to classify it, and perhaps to ask how the variables vary among the species. (In fact, Fisher used these data to test a genetic hypothesis which placed *I. versicolor* as a hybrid two-thirds of the way from *I. setosa* to *I. virginica*.) However, the classification of species is uncertain, and similar data have been used to identify species by grouping the cases. (Indeed, Wilson (1982) and McLachlan (1992, §6.9) consider whether the iris data can be split into subspecies.)

Krzanowski (1988) and Mardia, Kent and Bibby (1979) are two general references on multivariate analysis. For pattern recognition we follow Ripley (1996), which also has a computationally-informed account of multivariate analysis.

Most of the emphasis in the literature and in this chapter is on continuous measurements, but we do look briefly at multi-way discrete data in Section 11.4.

Colour can be used very effectively to differentiate groups in the plots of this chapter, on screen if not on paper. The code given here uses both colours and symbols, but you may prefer to use only one of these to differentiate groups. (The colours used are chosen for use on a trellis device.)

R In R `library(mva)` is needed for most of the material in this chapter.

Running example: *Leptograpsus variegatus* crabs

Mahon (see Campbell and Mahon, 1974) recorded data on 200 specimens of *Leptograpsus variegatus* crabs on the shore in Western Australia. This occurs in two colour forms, blue and orange, and he collected 50 of each form of each sex and made five physical measurements. These were the carapace (shell) length `CL` and width `CW`, the size of the frontal lobe `FL` and rear width `RW`, and the body depth `BD`. Part of the authors' thesis was to establish that the two colour forms were clearly differentiated morphologically, to support classification as two separate species.

The data are physical measurements, so a sound initial strategy is to work on log scale. This has been done throughout.

11.1 Visualization Methods

The simplest way to examine multivariate data is via a *pairs* or *scatterplot matrix* plot, enhanced to show the groups as discussed in Section 4.5. Pairs plots are a set of two-dimensional projections of a high-dimension point cloud.

However, pairs plots can easily miss interesting structure in the data that depends on three or more of the variables, and genuinely multivariate methods explore the data in a less coordinate-dependent way. Many of the most effective routes to explore multivariate data use dynamic graphics such as exploratory projection pursuit (for example, Huber, 1985; Friedman, 1987; Jones and Sibson, 1987 and Ripley, 1996) which chooses 'interesting' rotations of the point cloud. These are available through interfaces to the package XGobi[1] for machines running `X11`.[2] A successor to XGobi, GGobi,[3] is under development.

Many of the other visualization methods can be viewed as projection methods for particular definitions of 'interestingness'.

Principal component analysis

Principal component analysis (PCA) has a number of different interpretations. The simplest is a projection method finding projections of maximal variability. That is, it seeks linear combinations of the columns of X with maximal (or minimal) variance. Because the variance can be scaled by rescaling the combination, we constrain the combinations to have unit length (which is true of projections).

Let S denote the covariance matrix of the data X, which is defined[4] by

$$nS = (X - n^{-1}\mathbf{1}\mathbf{1}^T X)^T (X - n^{-1}\mathbf{1}\mathbf{1}^T X) = (X^T X - n\overline{\boldsymbol{x}}\overline{\boldsymbol{x}}^T)$$

[1] `http://www.research.att.com/areas/stat/xgobi/`

[2] On UNIX *and* on Windows: a Windows port of XGobi is available at `http://www.stats.ox.ac.uk/pub/SWin`.

[3] `http://www.ggobi.org`.

[4] A divisor of $n - 1$ is more conventional, but `princomp` calls `cov.wt`, which uses n.

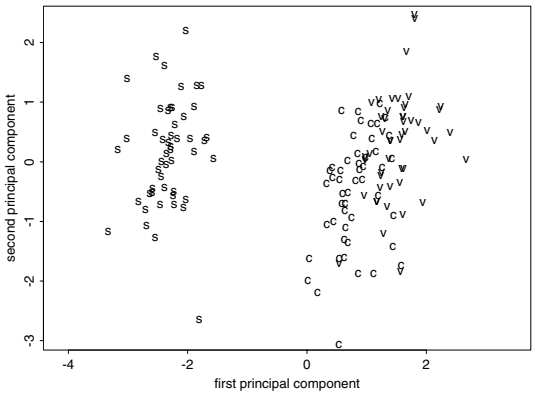

Figure 11.1: First two principal components for the log-transformed `iris` data.

where $\overline{\boldsymbol{x}} = \mathbf{1}^T X/n$ is the row vector of means of the variables. Then the sample variance of a linear combination $\boldsymbol{x}\boldsymbol{a}$ of a row vector $\boldsymbol{x}$ is $\boldsymbol{a}^T\Sigma\boldsymbol{a}$ and this is to be maximized (or minimized) subject to $\|\boldsymbol{a}\|^2 = \boldsymbol{a}^T\boldsymbol{a} = 1$. Since Σ is a non-negative definite matrix, it has an eigendecomposition

$$\Sigma = C^T\Lambda C$$

where Λ is a diagonal matrix of (non-negative) eigenvalues in decreasing order. Let $\boldsymbol{b} = C\boldsymbol{a}$, which has the same length as $\boldsymbol{a}$ (since C is orthogonal). The problem is then equivalent to maximizing $\boldsymbol{b}^T\Lambda\boldsymbol{b} = \sum \lambda_i b_i^2$ subject to $\sum b_i^2 = 1$. Clearly the variance is maximized by taking $\boldsymbol{b}$ to be the first unit vector, or equivalently taking $\boldsymbol{a}$ to be the column eigenvector corresponding to the largest eigenvalue of Σ. Taking subsequent eigenvectors gives combinations with as large as possible variance that are uncorrelated with those that have been taken earlier. The ith principal component is then the ith linear combination picked by this procedure. (It is only determined up to a change of sign; you may get different signs in different implementations of S.)

The first k principal components span a subspace containing the 'best' k-dimensional view of the data. It has a maximal covariance matrix (both in trace and determinant). It also best approximates the original points in the sense of minimizing the sum of squared distances from the points to their projections. The first few principal components are often useful to reveal structure in the data. The principal components corresponding to the smallest eigenvalues are the most nearly constant combinations of the variables, and can also be of interest.

Note that the principal components depend on the scaling of the original variables, and this will be undesirable except perhaps if (as in the `iris` data) they are in comparable units. (Even in this case, correlations would often be used.) Otherwise it is conventional to take the principal components of the *correlation* matrix, implicitly rescaling all the variables to have unit sample variance.

The function `princomp` computes principal components. The argument `cor` controls whether the covariance or correlation matrix is used (via rescaling the

variables).

```
> # S: ir <- rbind(iris[,,1], iris[,,2], iris[,,3])
> # R: data(iris3); ir <- rbind(iris3[,,1], iris3[,,2], iris3[,,3])
> ir.species <- factor(c(rep("s", 50), rep("c", 50), rep("v", 50)))
> (ir.pca <- princomp(log(ir), cor = T))
Standard deviations:
 Comp.1  Comp.2 Comp.3  Comp.4
 1.7125 0.95238 0.3647 0.16568
    ....
> summary(ir.pca)
Importance of components:
                        Comp.1  Comp.2   Comp.3    Comp.4
    Standard deviation 1.71246 0.95238 0.364703 0.1656840
Proportion of Variance 0.73313 0.22676 0.033252 0.0068628
 Cumulative Proportion 0.73313 0.95989 0.993137 1.0000000
> plot(ir.pca)
> loadings(ir.pca)
         Comp.1 Comp.2 Comp.3 Comp.4
Sepal L.  0.504  0.455  0.709  0.191
Sepal W. -0.302  0.889 -0.331
Petal L.  0.577        -0.219 -0.786
Petal W.  0.567        -0.583  0.580
> ir.pc <- predict(ir.pca)
> eqscplot(ir.pc[, 1:2], type = "n",
     xlab = "first principal component",
     ylab = "second principal component")
> text(ir.pc[, 1:2], labels = as.character(ir.species),
       col = 3 + codes(ir.species))
```

In the terminology of this function, the *loadings* are columns giving the linear combinations $\boldsymbol{a}$ for each principal component, and the *scores* are the data on the principal components. The plot (not shown) is the `screeplot`, a barplot of the variances of the principal components labelled by $\sum_{i=1}^{j} \lambda_i / \mathrm{trace}(\Sigma)$. The result of `loadings` is rather deceptive, as small entries are suppressed in printing but will be insignificant only if the correlation matrix is used, and that is *not* the default. The `predict` method rotates to the principal components.

As well as a data matrix `x`, the function `princomp` can accept data via a model formula with an empty left-hand side or as a variance or correlation matrix specified by argument `covlist`, of the form output by `cov.wt` and `cov.rob` (see page 336). Using the latter is one way to robustify principal component analysis. (S-PLUS has `princompRob` in library section `robust`, using `covRob`.)

S+

Figure 11.1 shows the first two principal components for the `iris` data based on the covariance matrix, revealing the group structure if it had not already been known. *A warning:* principal component analysis will reveal the gross features of the data, which may already be known, and is often best applied to residuals after the known structure has been removed. As we discovered in Figure 4.13 on page 96, animals come in varying sizes and two sexes!

```
> lcrabs <- log(crabs[, 4:8])
> crabs.grp <- factor(c("B", "b", "O", "o")[rep(1:4, each = 50)])
> (lcrabs.pca <- princomp(lcrabs))
Standard deviations:
  Comp.1   Comp.2   Comp.3   Comp.4    Comp.5
 0.51664 0.074654 0.047914 0.024804 0.0090522
> loadings(lcrabs.pca)
   Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
FL  0.452  0.157  0.438 -0.752  0.114
RW  0.387 -0.911
CL  0.453  0.204 -0.371        -0.784
CW  0.440        -0.672         0.591
BD  0.497  0.315  0.458  0.652  0.136
> lcrabs.pc <- predict(lcrabs.pca)
> dimnames(lcrabs.pc) <- list(NULL, paste("PC", 1:5, sep = ""))
```

(As the data on log scale *are* very comparable, we did not rescale the variables to unit variance.) The first principal component had by far the largest standard deviation, with coefficients that show it to be a 'size' effect. A plot of the second and third principal components shows an almost total separation into forms (Figure 4.13 and 4.14 on pages 96 and 97) on the third PC, the second PC distinguishing sex. The coefficients of the third PC show that it is contrasting overall size with FL and BD.

One ancillary use of principal component analysis is to *sphere* the data. After transformation to principal components, the coordinates are uncorrelated, but with different variances. Sphering the data amounts to rescaling each principal component to have unit variance, so the variance matrix becomes the identity. If the data were a sample from a multivariate normal distribution the point cloud would look spherical, and many measures of 'interestingness' in exploratory projection pursuit look for features in sphered data. Borrowing a term from time series, sphering is sometimes known as *pre-whitening*.

There are two books devoted solely to principal components, Jackson (1991) and Jolliffe (1986), which we think overstates its value as a technique.

Exploratory projection pursuit

Using projection pursuit in XGobi or GGobi allows us to examine the data much more thoroughly. Try one of

```
library(xgobi)
xgobi(lcrabs, colors = c("SkyBlue", "SlateBlue", "Orange",
      "Red")[rep(1:4, each = 50)])
xgobi(lcrabs, glyphs = 12 + 5*rep(0:3, each = 50))
```

A result of optimizing by the 'holes' index is shown in Figure 11.2.

Distance methods

This is a class of methods based on representing the cases in a low-dimensional Euclidean space so that their proximity reflects the similarity of their variables.

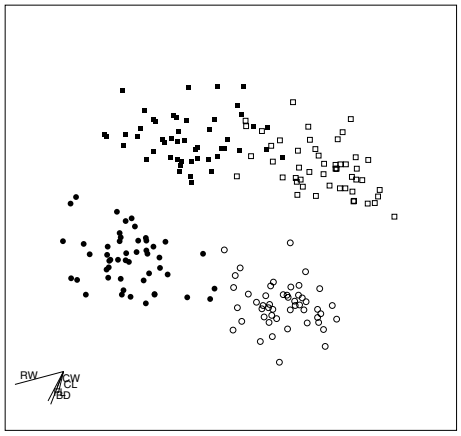

Figure 11.2: Projection pursuit view of the `crabs` data. Males are coded as filled symbols, females as open symbols, the blue colour form as squares and the orange form as circles.

We can think of 'squeezing' a high-dimensional point cloud into a small number of dimensions (2, perhaps 3) whilst preserving as well as possible the inter-point distances.

To do so we have to produce a measure of (dis)similarity. The function `dist` uses one of four distance measures between the points in the p-dimensional space of variables; the default is Euclidean distance. Distances are often called *dissimilarities*. Jardine and Sibson (1971) discuss several families of similarity and dissimilarity measures. For categorical variables most dissimilarities are measures of agreement. The *simple matching coefficient* is the proportion of categorical variables on which the cases differ. The *Jaccard coefficient* applies to categorical variables with a preferred level. It is the proportion of such variables with one of the cases at the preferred level in which the cases differ. The `binary` method of `dist` is of this family, being the Jaccard coefficient if all non-zero levels are preferred. Applied to logical variables on two cases it gives the proportion of variables in which only one is true among those that are true on at least one case.

R The function `daisy` (in package `cluster` in R) provides a more general way to compute dissimilarity matrices. The main extension is to variables that are not on interval scale, for example, ordinal, log-ratio and asymmetric binary variables. There are many variants of these coefficients; Kaufman and Rousseeuw (1990, §2.5) provide a readable summary and recommendations, and Cox and Cox (2001, Chapter 2) provide a more comprehensive catalogue.

The most obvious of the distance methods is *multidimensional scaling* (MDS), which seeks a configuration in $\mathbb{R}^d$ such that distances between the points best match (in a sense to be defined) those of the distance matrix. We start with the classical form of multidimensional scaling, which is also known as *principal coordinate analysis*. For the `iris` data we can use:

```
ir.scal <- cmdscale(dist(ir), k = 2, eig = T)
ir.scal$points[, 2] <- -ir.scal$points[, 2]
```

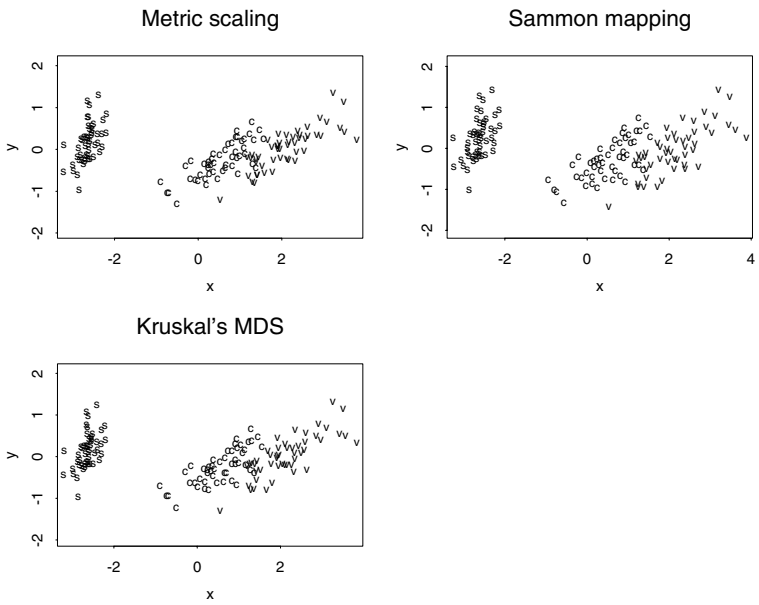

Figure 11.3: Distance-based representations of the `iris` data. The top left plot is by multidimensional scaling, the top right by Sammon's non-linear mapping, the bottom left by Kruskal's isotonic multidimensional scaling. Note that each is defined up to shifts, rotations and reflections.

```
eqscplot(ir.scal$points, type = "n")
text(ir.scal$points, labels = as.character(ir.species),
     col = 3 + codes(ir.species), cex = 0.8)
```

where care is taken to ensure correct scaling of the axes (see the top left plot of Figure 11.3). Note that a configuration can be determined only up to translation, rotation and reflection, since Euclidean distance is invariant under the group of rigid motions and reflections. (We chose to reflect this plot to match later ones.) An idea of how good the fit is can be obtained by calculating a measure[5] of 'stress':

```
> distp <- dist(ir)
> dist2 <- dist(ir.scal$points)
> sum((distp - dist2)^2)/sum(distp^2)
[1] 0.001747
```

which shows the fit is good. Using classical multidimensional scaling with a Euclidean distance as here is equivalent to plotting the first `k` principal components (without rescaling to correlations).

Another form of multidimensional scaling is Sammon's (1969) non-linear mapping, which given a dissimilarity d on n points constructs a k-dimensional

[5]There are many such measures.

configuration with distances $\widetilde{d}$ to minimize a weighted 'stress'

$$E_{\text{Sammon}}(d, \widetilde{d}) = \frac{1}{\sum_{i \neq j} d_{ij}} \sum_{i \neq j} \frac{(d_{ij} - \widetilde{d}_{ij})^2}{d_{ij}}$$

by an iterative algorithm implemented in our function `sammon`. We have to drop duplicate observations to make sense of $E(d, \widetilde{d})$; running `sammon` will report which observations are duplicates.[6] Figure 11.4 was produced by

```
ir.sam <- sammon(dist(ir[-143,]))
eqscplot(ir.sam$points, type = "n")
text(ir.sam$points, labels = as.character(ir.species[-143]),
     col = 3 + codes(ir.species), cex = 0.8)
```

Contrast this with the objective for classical MDS applied to a Euclidean configuration of points (but not in general), which minimizes

$$E_{\text{classical}}(d, \widetilde{d}) = \sum_{i \neq j} \left[d_{ij}^2 - \widetilde{d}_{ij}^2\right] \Big/ \sum_{i \neq j} d_{ij}^2$$

The Sammon function puts much more stress on reproducing small distances accurately, which is normally what is needed.

A more thoroughly non-metric version of multidimensional scaling goes back to Kruskal and Shepard in the 1960s (see Cox and Cox, 2001 and Ripley, 1996). The idea is to choose a configuration to minimize

$$STRESS^2 = \sum_{i \neq j} \left[\theta(d_{ij}) - \widetilde{d}_{ij}\right]^2 \Big/ \sum_{i \neq j} \widetilde{d}_{ij}^2$$

over both the configuration of points and an increasing function θ. Now the location, rotation, reflection and scale of the configuration are all indeterminate. This is implemented in function `isoMDS` which we can use by

```
ir.iso <- isoMDS(dist(ir[-143,]))
eqscplot(ir.iso$points, type = "n")
text(ir.iso$points, labels = as.character(ir.species[-143]),
     col = 3 + codes(ir.species), cex = 0.8)
```

The optimization task is difficult and this can be quite slow.

MDS plots of the `crabs` data tend to show just large and small crabs, so we have to remove the dominant effect of size. We used the carapace area as a good measure of size, and divided all measurements by the square root of the area. It is also necessary to account for the sex differences, which we can do by analysing each sex separately, or by subtracting the mean for each sex, which we did:

```
cr.scale <- 0.5 * log(crabs$CL * crabs$CW)
slcrabs <- lcrabs - cr.scale
cr.means <- matrix(0, 2, 5)
```

[6] In S we would use `(1:150)[duplicated(ir)]` .

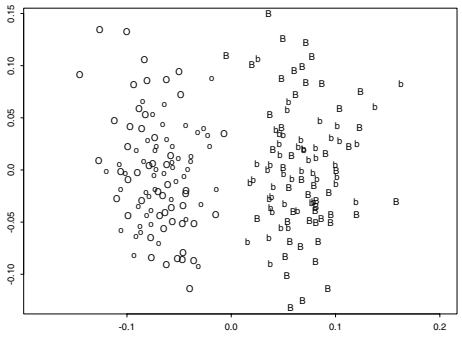

Figure 11.4: Sammon mapping of `crabs` data adjusted for size and sex. Males are coded as capitals, females as lower case, colours as the initial letter of blue or orange.

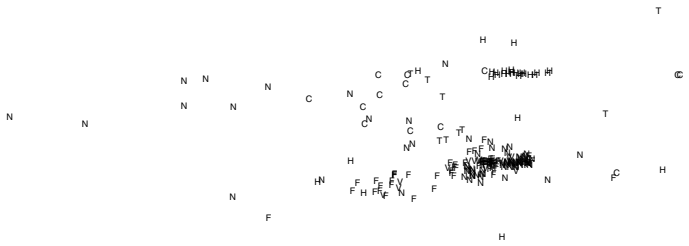

Figure 11.5: Isotonic multidimensional scaling representation of the `fgl` data. The groups are plotted by the initial letter, except `F` for window float glass, and `N` for window non-float glass. Small dissimilarities correspond to small distances on the plot and conversely.

```
cr.means[1,] <- colMeans(slcrabs[crabs$sex == "F", ])
cr.means[2,] <- colMeans(slcrabs[crabs$sex == "M", ])
dslcrabs <- slcrabs - cr.means[as.numeric(crabs$sex), ]
lcrabs.sam <- sammon(dist(dslcrabs))
eqscplot(lcrabs.sam$points, type = "n", xlab = "", ylab = "")
text(lcrabs.sam$points, labels = as.character(crabs.grp))
```

The MDS representations can be quite different in examples such as our dataset `fgl` that do not project well into a small number of dimensions; Figure 11.5 shows a non-metric MDS plot. (We omit one of an identical pair of fragments.)

```
fgl.iso <- isoMDS(dist(as.matrix(fgl[-40, -10])))
eqscplot(fgl.iso$points, type = "n", xlab = "", ylab = "", axes = F)
# either
for(i in seq(along = levels(fgl$type))) {
  set <- fgl$type[-40] == levels(fgl$type)[i]
  points(fgl.iso$points[set,], pch = 18, cex = 0.6, col = 2 + i)}
# S: key(text = list(levels(fgl$type), col = 3:8))
# or
text(fgl.iso$points,
```

```
     labels = c("F", "N", "V", "C", "T", "H")[fgl$type[-40]],
     cex = 0.6)
fgl.iso3 <- isoMDS(dist(as.matrix(fgl[-40, -10])), k = 3)
# S: brush(fgl.iso3$points)
fgl.col <- c("SkyBlue", "SlateBlue", "Orange", "Orchid",
             "Green", "HotPink")[fgl$type]
xgobi(fgl.iso3$points, colors = fgl.col)
```

This dataset fits much better into three dimensions, but that poses a challenge of viewing the results in some S environments. The optimization can be displayed dynamically in XGvis, part of XGobi.

Self-organizing maps

All multidimensional scaling algorithms are slow, not least because they work with all the distances between pairs of points and so scale at least as $O(n^2)$ and often worse. Engineers have looked for methods to find maps from many more than hundreds of points, of which the best known is 'Self-Organizing Maps' (Kohonen, 1995). Kohonen describes his own motivation as:

> 'I just wanted an algorithm that would effectively map similar patterns (pattern vectors close to each other in the input signal space) onto contiguous locations in the output space.' (p. VI)

which is the same aim as most variants of MDS. However, he interpreted 'contiguous' *via* a rectangular or hexagonal 2-D lattice of representatives[7] $\boldsymbol{m}_j$, with representatives at nearby points on the grid that are more similar than those that are widely separated. Data points are then assigned to the nearest representative (in Euclidean distance). Since Euclidean distance is used, pre-scaling of the data is important.

Kohonen's SOM is a family of algorithms with no well-defined objective to be optimized, and the results can be critically dependent on the initialization and the values of the tuning constants used. Despite this high degree of arbitrariness, the method scales well (it is at worst linear in n) and often produces useful insights in datasets whose size is way beyond MDS methods (for example, Roberts and Tarassenko, 1995).

If all the data are available at once (as will be the case in S applications), the preferred method is *batch SOM* (Kohonen, 1995, §3.14). For a single iteration, assign all the data points to representatives, and then update all the representatives by replacing each by the mean of all data points assigned to that representative or one of its neighbours (possibly using a distance-weighted mean). The algorithm proceeds iteratively, shrinking the neighbourhood radius to zero over a small number of iterations. Figure 11.6 shows the result of one run of the following code.

```
library(class)
gr <- somgrid(topo = "hexagonal")
crabs.som <- batchSOM(lcrabs, gr, c(4, 4, 2, 2, 1, 1, 1, 0, 0))
plot(crabs.som)
```

[7] Called 'codes' or a 'codebook' in some of the literature.

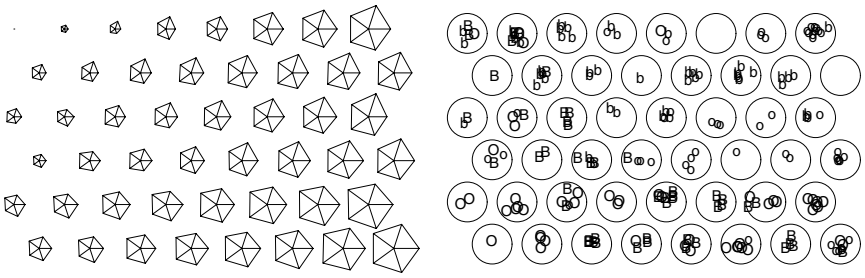

Figure 11.6: Batch SOM applied to the `crabs` dataset. The left plot is a `stars` plot of the representatives, and the right plot shows the assignments of the original points, coded as in 11.4 and placed randomly within the circle. (Plots from R.)

```
bins <- as.numeric(knn1(crabs.som$code, lcrabs, 0:47))
plot(crabs.som$grid, type = "n")
symbols(crabs.som$grid$pts[, 1], crabs.som$grid$pts[, 2],
        circles = rep(0.4, 48), inches = F, add = T)
text(crabs.som$grid$pts[bins, ] + rnorm(400, 0, 0.1),
     as.character(crabs.grp))
```

`batchSOM` The initialization used is to select a random subset of the data points. Different runs give different patterns but do generally show the gradation for small to large animals shown in the left panel[8] of Figure 11.6.

Traditional SOM uses an on-line algorithm, in which examples are presented in turn until convergence, usually by sampling from the dataset. Whenever an example $\boldsymbol{x}$ is presented, the closest representative $\boldsymbol{m}_j$ is found. Then

$$\boldsymbol{m}_i \leftarrow \boldsymbol{m}_i + \alpha[\boldsymbol{x} - \boldsymbol{m}_i] \qquad \text{for all neighbours } i \ .$$

Both the constant α and the definition of 'neighbour' change with time. This can be explored *via* function `SOM`, for example,

```
crabs.som2 <- SOM(lcrabs, gr); plot(crabs.som2)
```

See Murtagh and Hernández-Pajares (1995) for another statistical assessment.

Biplots

The biplot (Gabriel, 1971) is a method to represent both the cases and variables. We suppose that X has been centred to remove column means. The biplot represents X by two sets of vectors of dimensions n and p producing a rank-2 approximation to X. The best (in the sense of least squares) such approximation is given by replacing Λ in the singular value decomposition of X by D, a diagonal matrix setting $\lambda_3, \ldots$ to zero, so

$$X \approx \widetilde{X} = [\boldsymbol{u}_1\, \boldsymbol{u}_2] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^T \\ \boldsymbol{v}_2^T \end{bmatrix} = GH^T$$

[8] In S-PLUS the `stars` plot will be drawn on a rectangular grid.

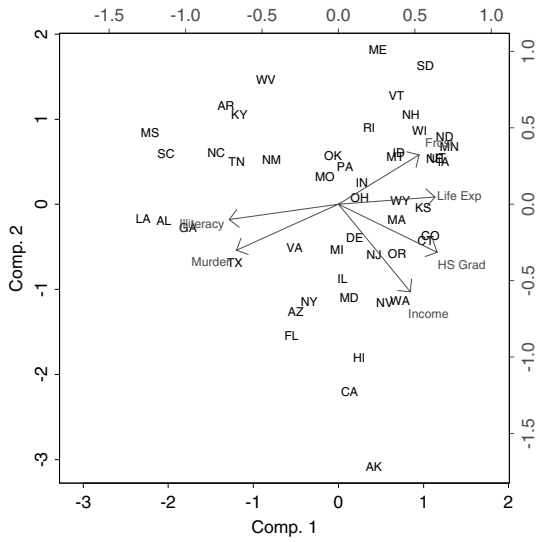

Figure 11.7: Principal component biplot of the part of the `state.x77` data. Distances between states represent Mahalanobis distance, and inner products between variables represent correlations. (The arrows extend 80% of the way along the variable's vector.)

where the diagonal scaling factors can be absorbed into G and H in a number of ways. For example, we could take

$$G = n^{a/2}\,[\boldsymbol{u}_1\,\boldsymbol{u}_2]\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}^{1-\lambda}, \qquad H = n^{-a/2}\,[\boldsymbol{v}_1\,\boldsymbol{v}_2]\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}^{\lambda}$$

The biplot then consists of plotting the $n + p$ two-dimensional vectors that form the rows of G and H. The interpretation is based on inner products between vectors from the two sets, which give the elements of $\widetilde{X}$. For $\lambda = a = 0$ this is just a plot of the first two principal components and the projections of the variable axes.

The most popular choice is $\lambda = a = 1$ (which Gabriel, 1971, calls the *principal component biplot*). Then G contains the first two principal components scaled to unit variance, so the Euclidean distances between the rows of G represent the Mahalanobis distances (page 334) between the observations and the inner products between the rows of H represent the covariances between the (possibly scaled) variables (Jolliffe, 1986, pp. 77–8); thus the lengths of the vectors represent the standard deviations.

Figure 11.7 shows a biplot with $\lambda = 1$, obtained by[9]

```
library(MASS, first = T)      # enhanced biplot.princomp
# R: data(state)
state <- state.x77[, 2:7]; row.names(state) <- state.abb
```

[9] An enhanced version of `biplot.princomp` from `MASS` is used.

```
biplot(princomp(state, cor = T), pc.biplot = T, cex = 0.7,
       expand = 0.8)
```

We specified a rescaling of the original variables to unit variance. (There are additional arguments `scale`, which specifies λ, and `expand`, which specifies a scaling of the rows of H relative to the rows of G, both of which default to 1.)

Gower and Hand (1996) in a book-length discussion of biplots criticize conventional plots such as Figure 11.7. In particular they point out that the axis scales are not at all helpful. Notice the two sets of scales. That on the lower and left axes refers to the values of the rows of G. The upper/right scale is for the values of the rows of H which are shown as arrows.

Independent component analysis

Independent component analysis (ICA) was named by Comon (1994), and has since become a 'hot' topic in data visualization; see the books Lee (1998); Hyvärinen *et al.* (2001) and the expositions by Hyvärinen and Oja (2000) and Hastie *et al.* (2001, §14.6).

ICA looks for rotations of sphered data that have approximately independent coordinates. This will be true (in theory) for all rotations of samples from multivariate normal distributions, so ICA is of most interest for distributions that are far from normal.

The original context for ICA was 'unmixing' of signals. Suppose there are $k \leqslant p$ independent sources in a data matrix S, and we observe the p linear combinations $X = SA$ with mixing matrix A. The 'unmixing' problem is to recover S. Clearly there are identifiability problems: we cannot recover the amplitudes or the labels of the signals, so we may as well suppose that the signals have unit variances. Unmixing is often illustrated by the problem of listening to just one speaker at a party. Note that this is a 'no noise' model: all the randomness is assumed to come from the signals.

Suppose the data X have been sphered; by assumption S is sphered and so X has variance A^TA and we look for an orthogonal matrix A. Thus ICA algorithms can be seen as exploratory projection pursuit in which the measure of interestingness emphasises independence (not just uncorrelatedness), say as the sum of the entropies of the projected coordinates. Like most projection pursuit indices, approximations are used for speed, and that proposed by Hyvärinen and Oja (2000) is implemented is the R package `fastICA`.[10] We can illustrate this for the `crabs` data, where the first and fourth signals shown in Figure 11.8 seem to pick out the two colour forms and two sexes respectively.

```
library(fastICA)
nICA <- 4
crabs.ica <- fastICA(crabs[, 4:8], nICA)
Z <- crabs.ica$S
par(mfrow = c(2, nICA))
for(i in 1:nICA) boxplot(split(Z[, i], crabs.grp))
```

[10] By Jonathan Marchini. Also ported to S-PLUS.

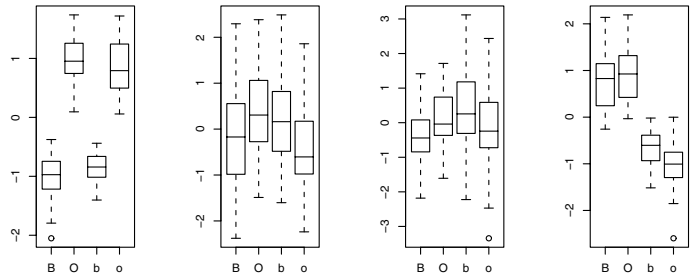

Figure 11.8: Boxplots of four 'signals' recovered by ICA from the `crabs` data.

There is a lot of arbitrariness in the use of ICA, in particular in choosing the number of signals. We might have expected to need two here, when the results are much less impressive.

Glyph representations

There is a wide range of ways to trigger multiple perceptions of a figure, and we can use these to represent each of a moderately large number of rows of a data matrix by an individual figure. Perhaps the best known of these are Chernoff's faces (Chernoff, 1973, implemented in the S-PLUS function `faces`; there are other versions by Bruckner, 1978 and Flury and Riedwyl, 1981) and the star plots as implemented in the function `stars` (see Figure 11.6), but Wilkinson (1999, Chapter 3) gives many more.

These glyph plots do depend on the ordering of variables and perhaps also their scaling, and they do rely on properties of human visual perception. So they have rightly been criticised as subject to manipulation, and one should be aware of the possibility that the effect may differ by viewer.[11] Nevertheless they can be very effective as tools for private exploration.

As an example, a stars plot for the `state.x77` dataset with variables in the order showing up in the biplot of Figure 11.7 can be drawn by

```
# S: stars(state.x77[, c(7, 4, 6, 2, 5, 3)], byrow = T)
# R: stars(state.x77[, c(7, 4, 6, 2, 5, 3)], full = FALSE,
          key.loc = c(10, 2))
```

Parallel coordinate plots

Parallel coordinates plots (Inselberg, 1984; Wegman, 1990) join the same points across a set of parallel axes. We can show the `state.x77` dataset in the order showing up in the biplot of Figure 11.7 by

```
parcoord(state.x77[, c(7, 4, 6, 2, 5, 3)])
```

[11] Especially if colour is involved; it is amazingly common to overlook the prevalence of red–green colour blindness.

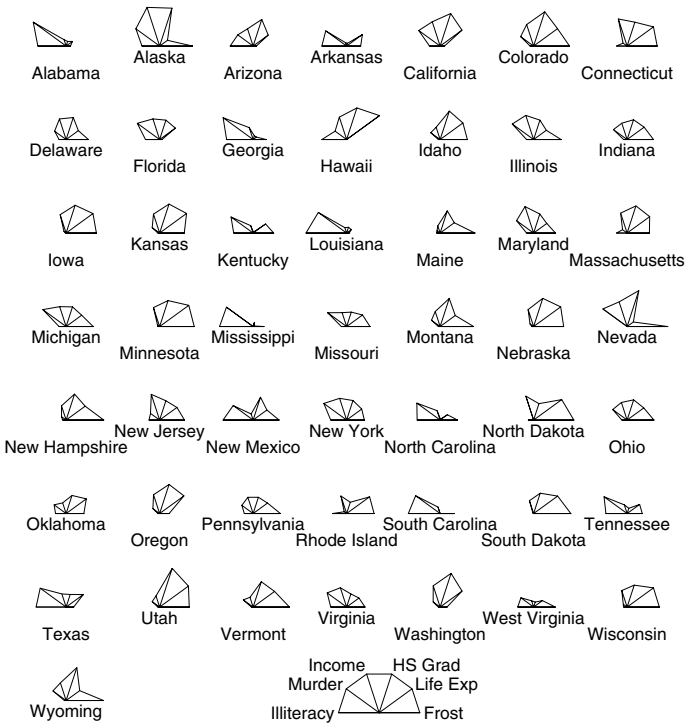

Figure 11.9: R version of `stars` plot of the `state.x77` dataset.

Such plots are often too 'busy' without a means of interaction to identify observations, sign-change and reorder variables, brush groups and so on (as is possible in XGobi and GGobi). As an example of a revealing parallel coordinate plot try

```
parcoord(log(ir)[, c(3, 4, 2, 1)], col = 1 + (0:149)%/%50)
```

on a device which can plot colour.

11.2 Cluster Analysis

Cluster analysis is concerned with discovering groupings among the cases of our n by p matrix. A comprehensive general reference is Gordon (1999); Kaufman and Rousseeuw (1990) give a good introduction and their methods are available in S-PLUS and in package `cluster` for R. Clustering methods can be clustered in many different ways; here is one.

- Agglomerative hierarchical methods (`hclust`, `agnes`, `mclust`).
  - Produces a set of clusterings, usually one with k clusters for each $k = n, \ldots, 2$, successively amalgamating groups.
  - Main differences are in calculating group–group dissimilarities from point–point dissimilarities.

  - Computationally easy.

- Optimal partitioning methods (`kmeans`, `pam`, `clara`, `fanny`).
  - Produces a clustering for fixed K.
  - Need an initial clustering.
  - Lots of different criteria to optimize, some based on probability models.
  - Can have distinct 'outlier' group(s).

- Divisive hierarchical methods (`diana`, `mona`).
  - Produces a set of clusterings, usually one for each $k = 2, \ldots, K \ll n$.
  - Computationally nigh-impossible to find optimal divisions (Gordon, 1999, p. 90).
  - Most available methods are *monothetic* (split on one variable at each stage).

Do not assume that 'clustering' methods are the best way to discover interesting groupings in the data; in our experience the visualization methods are often far more effective. There are many different clustering methods, often giving different answers, and so the danger of over-interpretation is high.

Many methods are based on a measure of the similarity or dissimilarity between cases, but some need the data matrix itself. A *dissimilarity coefficient* d is symmetric ($d(A, B) = d(B, A)$), non-negative and $d(A, A)$ is zero. A similarity coefficient has the scale reversed. Dissimilarities may be *metric*

$$d(A, C) \leqslant d(A, B) + d(B, C)$$

or *ultrametric*

$$d(A, B) \leqslant \max\big(d(A, C), d(B, C)\big)$$

but need not be either. We have already seen several dissimilarities calculated by `dist` and `daisy`.

Ultrametric dissimilarities have the appealing property that they can be represented by a *dendrogram* such as those shown in Figure 11.10, in which the dissimilarity between two cases can be read from the height at which they join a single group. Hierarchical clustering methods can be thought of as approximating a dissimilarity by an ultrametric dissimilarity. Jardine and Sibson (1971) argue that one method, single-link clustering, uniquely has all the desirable properties of a clustering method. This measures distances between clusters by the dissimilarity of the closest pair, and agglomerates by adding the shortest possible link (that is, joining the two closest clusters). Other authors disagree, and Kaufman and Rousseeuw (1990, §5.2) give a different set of desirable properties leading uniquely to their preferred method, which views the dissimilarity between clusters as the average of the dissimilarities between members of those clusters. Another popular method is complete-linkage, which views the dissimilarity between clusters as the maximum of the dissimilarities between members.

The function `hclust` implements these three choices, selected by its `method` argument which takes values `"compact"` (the default, for complete-linkage,

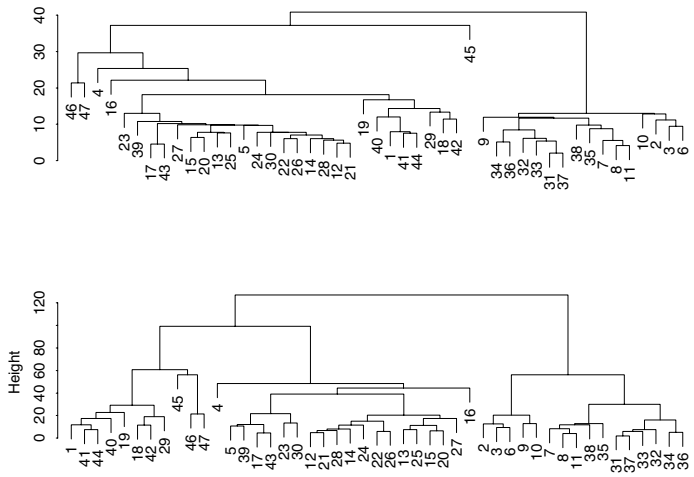

Figure 11.10: Dendrograms for the socio-economic data on Swiss provinces computed by single-link clustering (top) and divisive clustering (bottom).

R called `"complete"` in R), `"average"` and `"connected"` (for single-linkage, called `"single"` in R). Function `agnes` also has these (with the R names) and others.

The S dataset[12] `swiss.x` gives five measures of socio-economic data on Swiss provinces about 1888, given by Mosteller and Tukey (1977, pp. 549–551). The data are percentages, so Euclidean distance is a reasonable choice. We use single-link clustering:

```
# S: h <- hclust(dist(swiss.x), method = "connected")
# R: data(swiss); swiss.x <- as.matrix(swiss[, -1])
# R: h <- hclust(dist(swiss.x), method = "single")
plclust(h)
cutree(h, 3)
# S: plclust( clorder(h, cutree(h, 3) ))
```

The hierarchy of clusters in a dendrogram is obtained by cutting it at different heights. The first plot suggests three main clusters, and the remaining code reorders the dendrogram to display (see Figure 11.10) those clusters more clearly. Note that there appear to be two main groups, with the point 45 well separated from them.

Function `diana` performs *divisive* clustering, in which the clusters are repeatedly subdivided rather than joined, using the algorithm of Macnaughton-Smith *et al.* (1964). Divisive clustering is an attractive option when a grouping into a few large clusters is of interest. The lower panel of Figure 11.10 was produced by `pltree(diana(swiss.x))`.

[12] In R the numbers are slightly different, and the provinces has been given names.

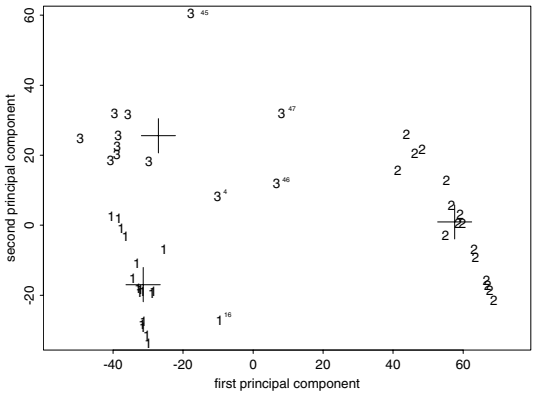

Figure 11.11: The Swiss provinces data plotted on its first two principal components. The labels are the groups assigned by K-means; the crosses denote the group means. Five points are labelled with smaller symbols.

Partitioning methods

The K-means clustering algorithm (MacQueen, 1967; Hartigan, 1975; Hartigan and Wong, 1979) chooses a pre-specified number of cluster centres to minimize the within-class sum of squares from those centres. As such it is most appropriate to continuous variables, suitably scaled. The algorithm needs a starting point, so we choose the means of the clusters identified by group-average clustering. The clusters *are* altered (cluster 3 contained just point 45), and are shown in principal-component space in Figure 11.11. (Its standard deviations show that a two-dimensional representation is reasonable.)

```
h <- hclust(dist(swiss.x), method = "average")
initial <- tapply(swiss.x, list(rep(cutree(h, 3),
   ncol(swiss.x)), col(swiss.x)), mean)
dimnames(initial) <- list(NULL, dimnames(swiss.x)[[2]])
km <- kmeans(swiss.x, initial)
(swiss.pca <- princomp(swiss.x))
Standard deviations:
  Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
  42.903 21.202  7.588 3.6879 2.7211
   ....
swiss.px <- predict(swiss.pca)
dimnames(km$centers)[[2]] <- dimnames(swiss.x)[[2]]
swiss.centers <- predict(swiss.pca, km$centers)
eqscplot(swiss.px[, 1:2], type = "n",
         xlab = "first principal component",
         ylab = "second principal component")
text(swiss.px[, 1:2], labels = km$cluster)
points(swiss.centers[,1:2], pch = 3, cex = 3)
identify(swiss.px[, 1:2], cex = 0.5)
```

By definition, K-means clustering needs access to the data matrix and uses Euclidean distance. We can apply a similar method using only dissimilarities if we confine the cluster centres to the set of given examples. This is known as the k-medoids criterion (of Vinod, 1969) implemented in `pam` and `clara`. Using `pam` picks provinces 29, 8 and 28 as cluster centres.

```
> library(cluster)          # needed in R only
> swiss.pam <- pam(swiss.px, 3)
> summary(swiss.pam)
Medoids:
     Comp. 1   Comp. 2 Comp. 3 Comp. 4  Comp. 5
[1,] -29.716  18.22162  1.4265 -1.3206  0.95201
[2,]  58.609   0.56211  2.2320 -4.1778  4.22828
[3,] -28.844 -19.54901  3.1506  2.3870 -2.46842
Clustering vector:
 [1] 1 2 2 1 3 2 2 2 2 2 2 3 3 3 3 3 1 1 1 3 3 3 3 3 3 3 3 3
[29] 1 3 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
    ....
> eqscplot(swiss.px[, 1:2], type = "n",
           xlab = "first principal component",
           ylab = "second principal component")
> text(swiss.px[,1:2], labels = swiss.pam$clustering)
> points(swiss.pam$medoid[,1:2], pch = 3, cex = 5)
```

The function `fanny` implements a 'fuzzy' version of the k-medoids criterion. Rather than point i having a membership of just one cluster v, its membership is partitioned among clusters as positive weights u_{iv} summing to one. The criterion then is

$$\min_{(u_{iv})} \sum_v \frac{\sum_{i,j} u_{iv}^2 u_{jv}^2 d_{ij}}{2 \sum_i u_{iv}^2}.$$

For our running example we find

```
> fanny(swiss.px, 3)
 iterations objective
         16    354.01
Membership coefficients:
          [,1]     [,2]     [,3]
 [1,] 0.725016 0.075485 0.199499
 [2,] 0.189978 0.643928 0.166094
 [3,] 0.191282 0.643596 0.165123
    ....
Closest hard clustering:
 [1] 1 2 2 1 3 2 2 2 2 2 2 3 3 3 3 3 1 1 1 3 3 3 3 3 3 3 3 3
[29] 1 3 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
```

The 'hard' clustering is formed by assigning each point to the cluster for which its membership coefficient is highest.

Other partitioning methods are based on the idea that the data are independent samples from a series of group populations, but the group labels have been lost, so the data can be regarded as from a mixture distribution. The idea is then to find the

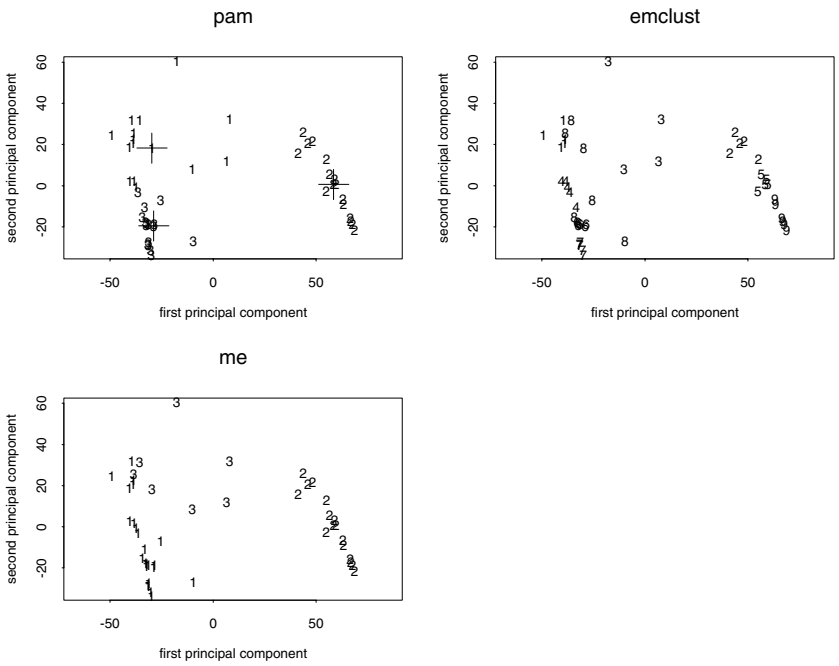

Figure 11.12: Clusterings of the Swiss provinces data by `pam` with three clusters (with the medoids marked by crosses), `me` with three clusters and `emclust` with up to nine clusters (it chose nine).

mixture distribution, usually as a mixture of multivariate normals, and to assign points to the component for which their posterior probability of membership is highest.

S+ S-PLUS has functions `mclust`, `mclass` and `mreloc` based on 'maximum-likelihood' clustering in which the mixture parameters and the classification are optimized simultaneously. Later work in the `mclust` library section[13] uses sounder methods in which the mixtures are fitted first. Nevertheless, fitting normal mixtures is a difficult problem, and the results obtained are often heavily dependent on the initial configuration supplied.

K-means clustering can be seen as 'maximum-likelihood' clustering where the clusters are assumed all to be spherically symmetric multivariate normals with the same spread. The `modelid` argument to the `mclust` functions allows a wider choice of normal mixture components, including `"EI"` (equal spherical) `"VI"` (spherical, differing by component), `"EEE"` (same elliptical), `"VEV"` (same shape elliptical, variable size and orientation) and `"VVV"` (arbitrary components).

Library section `mclust` provides hierarchical clustering via functions `mhtree` and `mhclass`. Then for a given number k of clusters the fitted mixture can be optimized by calling `me` (which here does not change the classification).

[13] Available at `http://www.stat.washington.edu/fraley/mclust/` and for R from CRAN.

```
library(mclust)
h <- mhtree(swiss.x, modelid = "VVV")
(mh <- as.vector(mhclass(h, 3)))
 [1] 1 2 2 3 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[29] 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 3 3 3
z <- me(swiss.x, modelid = "VVV", z = (ctoz(mh)+1/3)/2)
eqscplot(swiss.px[, 1:2], type = "n",
         xlab = "first principal component",
         ylab = "second principal component")
text(swiss.px[, 1:2], labels = max.col(z))
```

Function `mstep` can be used to extract the fitted components, and `mixproj` to plot them, but unfortunately only on a pair of the original variables.

Function `emclust` automates the whole cluster process, including choosing the number of clusters and between different `modelid`'s. One additional possibility controlled by argument `noise` is to include a background 'noise' term, that is a component that is a uniform Poisson process. It chooses lots of clusters (see Figure 11.12).

```
> vals <- emclust(swiss.x) # all possible models, 0:9 clusters.
> sm <- summary(vals, swiss.x)
> eqscplot(swiss.px[, 1:2], type = "n",
           xlab = "first principal component",
           ylab = "second principal component")
> text(swiss.px[, 1:2], labels = sm$classification)
```

11.3 Factor Analysis

Principal component analysis looks for linear combinations of the data matrix X that are uncorrelated and of high variance. Independent component analysis seeks linear combinations that are independent. *Factor analysis* seeks linear combinations of the variables, called *factors*, that represent underlying fundamental quantities of which the observed variables are expressions. The examples tend to be controversial ones such as 'intelligence' and 'social deprivation', the idea being that a small number of factors might explain a large number of measurements in an observational study. Such factors are to be inferred from the data.

We can think of both the factors of factor analysis and the signals of independent component analysis as *latent variables*, unobserved variables on each experimental unit that determine the patterns in the observations. The difference is that it is not the factors that are assumed to be independent, but rather the observations conditional on the factors.

The factor analysis model for a single common factor f is

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\lambda} f + \boldsymbol{u} \tag{11.1}$$

where $\boldsymbol{\lambda}$ is a vector known as the *loadings* and $\boldsymbol{u}$ is a vector of *unique* (or *specific*) factors for that observational unit. To help make the model identifiable, we

assume that the factor f has mean zero and variance one, and that $\boldsymbol{u}$ has mean zero and unknown *diagonal* covariance matrix Ψ. For $k < p$ common factors we have a vector $\boldsymbol{f}$ of common factors and a loadings matrix Λ, and

$$\boldsymbol{x} = \boldsymbol{\mu} + \Lambda\boldsymbol{f} + \boldsymbol{u} \tag{11.2}$$

where the components of $\boldsymbol{f}$ have unit variance and are uncorrelated and $\boldsymbol{f}$ and $\boldsymbol{u}$ are taken to be uncorrelated. Note that *all* the correlations amongst the variables in $\boldsymbol{x}$ must be explained by the common factors; if we assume joint normality the observed variables $\boldsymbol{x}$ will be conditionally independent given $\boldsymbol{f}$.

Principal component analysis also seeks a linear subspace like $\Lambda\boldsymbol{f}$ to explain the data, but measures the lack of fit by the sum of squares of the $\boldsymbol{u}_i$. Since factor analysis allows an arbitrary diagonal covariance matrix Ψ, its measure of fit of the $\boldsymbol{u}_i$ depends on the problem and should be independent of the units of measurement of the observed variables. (Changing the units of measurement of the observations does not change the common factors if the loadings and unique factors are re-expressed in the new units.)

Equation (11.2) and the conditions on $\boldsymbol{f}$ express the covariance matrix Σ of the data as

$$\Sigma = \Lambda\Lambda^T + \Psi \tag{11.3}$$

Conversely, if (11.3) holds, there is a k-factor model of the form (11.2). Note that the common factors $G^T\boldsymbol{f}$ and loadings matrix ΛG give rise to the same model for Σ, for any $k \times k$ orthogonal matrix G. Choosing an appropriate G is known as choosing a *rotation*. All we can achieve statistically is to fit the space spanned by the factors, so choosing a rotation is a way to choose an interpretable basis for that space. Note that if

$$s = \tfrac{1}{2}p(p+1) - [p(k+1) - \tfrac{1}{2}k(k-1)] = \tfrac{1}{2}(p-k)^2 - \tfrac{1}{2}(p+k) < 0$$

we would expect an infinity of solutions to (11.3). This value is known as the *degrees of freedom*, and comes from the number of elements in Σ minus the number of parameters in Ψ and Λ (taking account of the rotational freedom in Λ since only $\Lambda\Lambda^T$ is determined). Thus it is usual to assume $s \geqslant 0$; for $s = 0$ there may be a unique solution, no solution or an infinity of solutions (Lawley and Maxwell, 1971, pp. 10–11).

The variances of the original variables are decomposed into two parts, the *communality* $h_i^2 = \sum_j \lambda_{ij}^2$ and *uniqueness* ψ_{ii} which is thought of as the 'noise' variance.

Fitting the factor analysis model (11.2) is performed by the S function
S+ `factanal`. The default method in S-PLUS ('principal factor analysis') dates from the days of limited computational power, and is not intrinsically scale invariant—it should not be used. The preferred method is to maximize the likelihood over Λ and Ψ assuming multivariate normality of the factors $(\boldsymbol{f}, \boldsymbol{u})$, which depends only on the factor space and is scale-invariant. This likelihood can have multiple local maxima; this possibility is often ignored but `factanal` compares the fit found from several separate starting points. It is possible that the maximum likelihood solution will have some $\widehat{\psi}_{ii} = 0$, so the ith variable lies in the

estimated factor space. Opinions differ as to what to do in this case (sometimes known as a *Heywood case*), but often it indicates a lack of data or inadequacy of the factor analysis model. (Bartholomew and Knott, 1999, Section 3.18, discuss possible reasons and actions.)

It is hard to find examples in the literature for which a factor analysis model fits well; many do not give a measure of fit, or have failed to optimize the likelihood well enough and so failed to detect Heywood cases. We consider an example from Smith and Stanley (1983) as quoted by Bartholomew and Knott (1999, pp. 68–72).[14] Six tests were give to 112 individuals, with covariance matrix

```
         general picture  blocks   maze reading   vocab
general  24.641   5.991  33.520  6.023  20.755  29.701
picture   5.991   6.700  18.137  1.782   4.936   7.204
 blocks  33.520  18.137 149.831 19.424  31.430  50.753
   maze   6.023   1.782  19.424 12.711   4.757   9.075
reading  20.755   4.936  31.430  4.757  52.604  66.762
  vocab  29.701   7.204  50.753  9.075  66.762 135.292
```

The tests were of general intelligence, picture completion, block design, mazes, reading comprehension and vocabulary. The S-PLUS default in `factanal` is a single factor, but the fit is not good until we try two. The low uniqueness for reading ability suggests that this is close to a Heywood case, but it definitely is not one.

```
> S: ability.FA <- factanal(covlist = ability.cov, method = "mle")
> R: ability.FA <- factanal(covmat = ability.cov, factors = 1)
> ability.FA
    ....
The chi square statistic is 75.18 on 9 degrees of freedom.
    ....
> (ability.FA <- update(ability.FA, factors = 2))
    ....
The chi square statistic is 6.11 on 4 degrees of freedom.
The p-value is 0.191
    ....
> summary(ability.FA)
Uniquenesses:
 general picture  blocks    maze  reading   vocab
 0.45523 0.58933 0.21817 0.76942 0.052463 0.33358

Loadings:
        Factor1 Factor2
general 0.501   0.542
picture 0.158   0.621
 blocks 0.208   0.859
   maze 0.110   0.467
reading 0.957   0.179
  vocab 0.785   0.222
```

[14] Bartholomew & Knott give both covariance and correlation matrices, but these are inconsistent. Neither is in the original paper.

```
> round(loadings(ability.FA) %*% t(loadings(ability.FA)) +
            diag(ability.FA$uniq), 3)

         general picture blocks  maze reading vocab
general    1.000   0.416  0.570 0.308   0.577 0.514
picture    0.416   1.000  0.567 0.308   0.262 0.262
blocks     0.570   0.567  1.000 0.425   0.353 0.355
maze       0.308   0.308  0.425 1.000   0.189 0.190
reading    0.577   0.262  0.353 0.189   1.000 0.791
vocab      0.514   0.262  0.355 0.190   0.791 1.000
```

Remember that the first variable is a composite measure; it seems that the first factor reflects verbal ability, the second spatial reasoning. The main lack of fit is that the correlation 0.193 between `picture` and `maze` is fitted as 0.308.

Factor rotations

The usual aim of a rotation is to achieve 'simple structure', that is a pattern of loadings that is easy to interpret with a few large and many small coefficients.

There are many criteria for selecting rotations of the factors and loadings ma-
S+ trix; S-PLUS implements 12. There is an auxiliary function `rotate` that will rotate the fitted Λ according to one of these criteria, which is called via the `rotate` argument of `factanal`. The default `varimax` criterion is to maximize

$$\sum_{i,j}(d_{ij} - \overline{d}_{\cdot j})^2 \qquad \text{where} \qquad d_{ij} = \lambda_{ij}^2 / \textstyle\sum_j \lambda_{ij}^2 \tag{11.4}$$

and $\overline{d}_{\cdot j}$ is the mean of the d_{ij}. Thus the varimax criterion maximizes the sum over factors of the variances of the (normalized) squared loadings. The normalizing factors are the communalities that are invariant under orthogonal rotations.

Following Bartholomew & Knott, we illustrate the `oblimin` criterion[15] which minimizes the sum over all pairs of factors of the covariance between the squared loadings for those factors.

```
> loadings(rotate(ability.FA, rotation = "oblimin"))
        Factor1 Factor2
general  0.379   0.513
picture          0.640
 blocks          0.887
   maze          0.483
reading  0.946
  vocab  0.757   0.137

Component/Factor Correlations:
        Factor1 Factor2
Factor1 1.000   0.356
Factor2 0.356   1.000
```

[15] Not implemented in R at the time of writing.

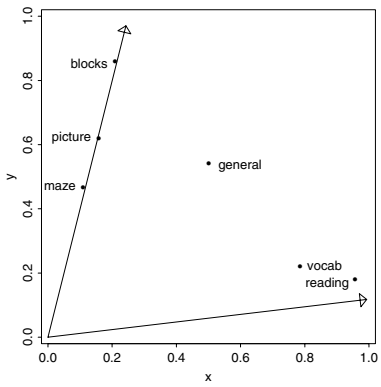

Figure 11.13: The loadings for the intelligence test data after varimax rotation, with the axes for the oblimin rotation shown as arrows.

```
> par(pty = "s")
> L <- loadings(ability.FA)
> eqscplot(L, xlim = c(0,1), ylim = c(0,1))
> identify(L, dimnames(L)[[1]])
> oblirot <- rotate(loadings(ability.FA), rotation = "oblimin")
> naxes <- solve(oblirot$tmat)
> arrows(rep(0, 2), rep(0, 2), naxes[,1], naxes[,2])
```

11.4 Discrete Multivariate Analysis

Most work on visualization and most texts on multivariate analysis implicitly assume continuous measurements. However, large-scale categorical datasets are becoming much more prevalent, often collected through surveys or 'CRM' (customer relationship management: that branch of data mining that collects information on buying habits, for example on shopping baskets) or insurance questionnaires.

There are some useful tools available for exploring categorical data, but it is often essential to use models to understand the data, most often log-linear models. Indeed, 'discrete multivariate analysis' is the title of an early influential book on log-linear models, Bishop *et al.* (1975).

Mosaic plots

There are a few ways to visualize low-dimensional contingency tables. *Mosaic plots* (Hartigan and Kleiner, 1981, 1984; Friendly, 1994; Emerson, 1998; Friendly, 2000) divide the plotting surface recursively according to the proportions of each factor in turn (so the order of the factors matters).

For an example, consider Fisher's (1940) data on colours of eyes and hair of people in Caithness, Scotland:

	fair	red	medium	dark	black
blue	326	38	241	110	3
light	688	116	584	188	4
medium	343	84	909	412	26
dark	98	48	403	681	85

in our dataset `caith`. Figure 11.14 shows mosaic plots for these data and for the `housing` data we used in Section 7.3, computed by

```
caith1 <- as.matrix(caith)
names(dimnames(caith1)) <- c("eyes", "hair")
mosaicplot(caith1, color = T)
# use xtabs in R
House <- crosstabs(Freq ~ Type + Infl + Cont + Sat, housing)
mosaicplot(House, color = T)
```

Correspondence analysis

Correspondence analysis is applied to two-way tables of counts.

Suppose we have an $r \times c$ table N of counts. Correspondence analysis seeks 'scores' f and g for the rows and columns which are maximally correlated. Clearly the maximum correlation is one, attained by constant scores, so we seek the largest non-trivial solution. Let R and C be matrices of the group indicators of the rows and columns, so $R^T C = N$. Consider the singular value decomposition of their correlation matrix

$$X_{ij} = \frac{n_{ij}/n - (n_{i\cdot}/n)(n_{\cdot j}/n)}{\sqrt{(n_{i\cdot}/n)(n_{\cdot j}/n)}} = \frac{n_{ij} - n\, r_i\, c_j}{n\sqrt{r_i\, c_j}}$$

where $r_i = n_{i\cdot}/n$ and $c_j = n_{\cdot j}/n$ are the proportions in each row and column. Let D_r and D_c be the diagonal matrices of r and c. Correspondence analysis corresponds to selecting the first singular value and left and right singular vectors of X_{ij} and rescaling by $D_r^{-1/2}$ and $D_c^{-1/2}$, respectively. This is done by our function `corresp`:

```
> corresp(caith)
First canonical correlation(s): 0.44637

 eyes scores:
    blue    light   medium   dark
-0.89679 -0.98732 0.075306 1.5743

 hair scores:
   fair      red    medium   dark  black
-1.2187 -0.52258 -0.094147 1.3189 2.4518
```

Can we make use of the subsequent singular values? In what Gower and Hand (1996) call 'classical CA' we consider $A = D_r^{-1/2} U\Lambda$ and $B = D_c^{-1/2} V\Lambda$. Then the first columns of A and B are what we have termed the row and column scores *scaled by* ρ, the first canonical correlation. More generally, we can see

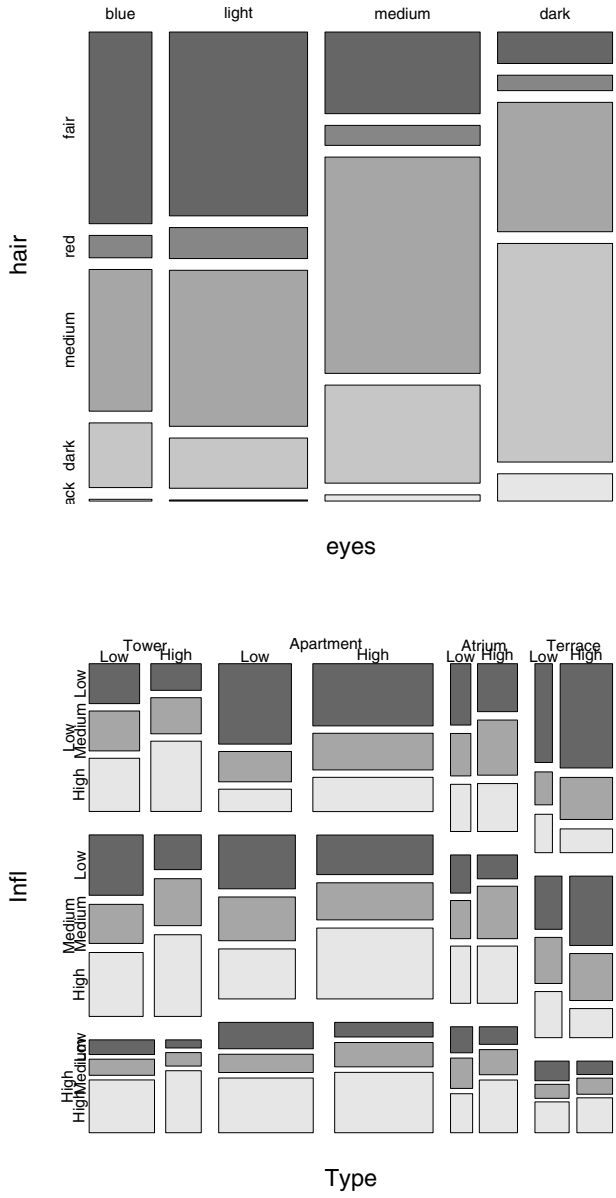

Figure 11.14: Mosaic plots for (top) Fisher's data on people from Caithness and (bottom) Copenhagen housing satisfaction data.

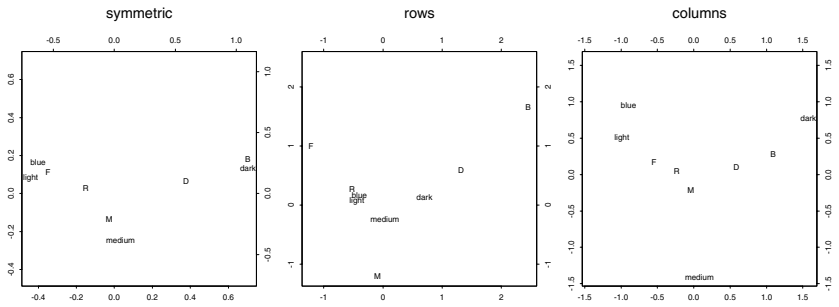

Figure 11.15: Three variants of correspondence analysis plots from Fisher's data on people in Caithness: (left) 'symmetric", (middle) 'row asymmetric' and (right) 'column asymmetric'.

distances between the rows of A as approximating the distances between the row profiles (rows rescaled to unit sum) of the table N, and analogously for the rows of B and the column profiles.

Classical CA plots the first two columns of A and B on the same figure. This is a form of a biplot and is obtained with our software by plotting a correspondence analysis object with `nf` $\geqslant 2$ or as the default for the method `biplot.correspondence`. This is sometimes known as a 'symmetric' plot. Other authors (for example, Greenacre, 1992) advocate 'asymmetric' plots. The asymmetric plot for the rows is a plot of the first two columns of A with the column labels plotted at the first two columns of $\Gamma = D_c^{-1/2}V$; the corresponding plot for the columns has columns plotted at B and row labels at $\Phi = D_r^{-1/2}U$. The most direct interpretation for the row plot is that

$$A = D_r^{-1}N\Gamma$$

so A is a plot of the *row profiles* (the rows normalized to sum to one) as convex combinations of the column vertices given by Γ.

By default `corresp` only retains one-dimensional row and column scores; then `plot.corresp` plots these scores and indicates the size of the entries in the table by the area of circles. The two-dimensional forms of the plot are shown in Figure 11.15 for Fisher's data on people from Caithness. These were produced by

```
# R: library(mva)
caith2 <- caith
dimnames(caith2)[[2]] <- c("F", "R", "M", "D", "B")
par(mfcol = c(1, 3))
plot(corresp(caith2, nf = 2)); title("symmetric")
plot(corresp(caith2, nf = 2), type = "rows"); title("rows")
plot(corresp(caith2, nf = 2), type = "col"); title("columns")
```

Note that the symmetric plot (left) has the row points from the asymmetric row plot (middle) and the column points from the asymmetric column plot (right) superimposed on the same plot (but with different scales).

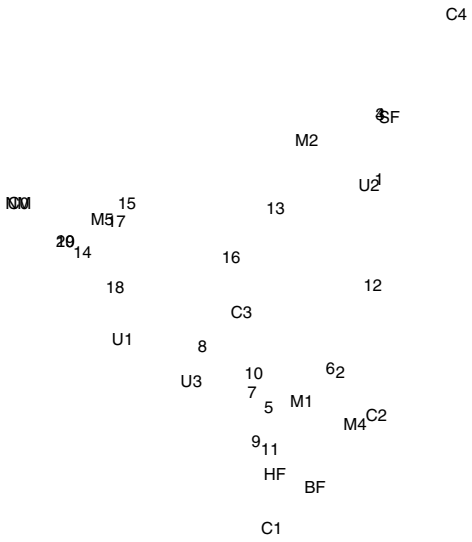

Figure 11.16: Multiple correspondence analysis plot of dataset `farms` on 20 farms on the Dutch island of Terschelling. Numbers represent the farms and labels levels of moisture (`M1`, `M2`, `M4` and `M5`), grassland usage (`U1`, `U2` and `U3`), manure usage (`C0` to `C4`) and type of grassland management (`SF`: standard, `BF`: biological, `HF`: hobby farming, `NM`: nature conservation). Levels `C0` and `NM` are coincident (on the extreme left), as are the pairs of farms 3 & 4 and 19 & 20.

Multiple correspondence analysis

Multiple correspondence analysis (MCA) is (confusingly!) a method for visualizing the joint properties of $p \geqslant 2$ categorical variables that does *not* reduce to correspondence analysis (CA) for $p = 2$, although the methods are closely related (see, for example, Gower and Hand, 1996, §10.2).

Suppose we have n observations on the p factors with ℓ total levels. Consider G, the $n \times \ell$ indicator matrix whose rows give the levels of each factor for each observation. Then all the row sums are p. MCA is often (Greenacre, 1992) defined as CA applied to the table G, that is the singular-value decomposition of $D_r^{-1/2}(G/\sum_{ij} g_{ij})D_c^{-1/2} = U\Lambda V^T$. Note that $D_r = pI$ since all the row sums are p, and $\sum_{ij} g_{ij} = np$, so this amounts to the SVD of $p^{-1/2}GD_c^{-1/2}/pn$.[16]

An alternative point of view is that MCA is a principal components analysis of the data matrix $X = G(pD_c)^{-1/2}$; with PCA it is usual to centre the data, but it transpires that the largest singular value is one and the corresponding singular vectors account for the means of the variables. A simple plot for MCA is to plot the first two principal components of X (which correspond to the second and third singular vectors of X). This is a form of biplot, but it will not be appropriate to add axes for the columns of X as the possible values are only $\{0, 1\}$, but it is usual to add the positions of 1 on each of these axes, and label these by the factor

[16] Gower and Hand (1996) omit the divisor pn.

level. (The 'axis' points are plotted at the appropriate row of $(pD_c)^{-1/2}V$.) The point plotted for each observation is the vector sum of the 'axis' points for the levels taken of each of the factors. Gower and Hand seem to prefer (e.g., their Figure 4.2) to rescale the plotted points by p, so they are plotted at the centroid of their levels. This is exactly the asymmetric row plot of the CA of G, apart from an overall scale factor of $p\sqrt{n}$.

We can apply this to the example of Gower and Hand (1996, p. 75) by

```
farms.mca <- mca(farms, abbrev = T)  # Use levels as names
plot(farms.mca, cex = rep(0.7, 2), axes = F)
```

shown in Figure 11.16

Sometimes it is desired to add rows or factors to an MCA plot. Adding rows is easy; the observations are placed at the centroid of the 'axis' points for levels that are observed. Adding factors (so-called *supplementary variables*) is less obvious. The 'axis' points are plotted at the rows of $(pD_c)^{-1/2}V$. Since $U\Lambda V^T = X = G(pD_c)^{-1/2}$, $V = (pD_c)^{-1/2}G^TU\Lambda^{-1}$ and

$$(pD_c)^{-1/2}V = (pD_c)^{-1}G^TU\Lambda^{-1}$$

This tells us that the 'axis' points can be found by taking the appropriate column of G, scaling to total $1/p$ and then taking inner products with the second and third columns of $U\Lambda^{-1}$. This procedure can be applied to supplementary variables and so provides a way to add them to the plot. The `predict` method for class `"mca"` allows rows or supplementary variables to be added to an MCA plot.

http://www.springer.com/978-0-387-95457-8

Modern Applied Statistics with S
Venables, W.N.; Ripley, B.D.
2002, XII, 498 p., Hardcover
ISBN: 978-0-387-95457-8