Statistical Data Mining

B. D. Ripley

April 2003

Introduction

This material is partly based on Ripley (1996), Venables & Ripley (1999, 2002) and the on-line complements available at

`http://www.stats.ox.ac.uk/pub/MASS4/`

My copyright agreements allow me to use the material on courses, but no further distribution is allowed.

The S code in this version of the notes was tested with S-PLUS 6.1 for Unix/Linux and Windows, and S-PLUS 2000 release 3. With minor changes it works with R version 1.6.2.

The specific add-ons for the material in this course are available at

`http://www.stats.ox.ac.uk/pub/bdr/SDM2001/`

All the other add-on libraries mentioned are available for Unix and for Windows. Compiled versions for S-PLUS 2000 are available from

`http://www.stats.ox.ac.uk/pub/SWin/`

and for S-PLUS 6.x from

`http://www.stats.ox.ac.uk/pub/MASS4/Winlibs/`

Contents

Chapter 1

Overview of Data Mining

Fifteen years ago *data mining* was a pejorative phrase amongst statisticians, but the English language evolves and that sense is now encapsulated in the phrase *data dredging*. In its current sense *data mining* means finding structure in large-scale databases. It is one of many newly-popular terms for this activity, another being *KDD* (Knowledge Discovery in Databases), and is a subject at the boundaries of statistics, engineering, machine learning and computer science.

Such phrases are to a large extent fashion, and finding structure in datasets is emphatically *not* a new activity. In the words of Witten & Franke (2000, p. 26)

> What's the difference between machine learning and statistics? Cynics, looking wryly at the explosion of commercial interest (and hype) in this area, equate data mining to statistics plus marketing.

What is new is the scale of databases that are becoming available through the computer-based acquisition of data, either through new instrumentation (fMRI machines can collect 100Mb of images in a hour's session) or through the by-product of computerised accounting records (for example, spotting fraudulent use of credit cards or telephones, linking sales to customers through 'loyalty' cards).

This is a short course in *statistical* data mining. As such we will not cover the aspects of data mining that are concerned with querying very large databases, although building efficient database interfaces to statistical software is becoming an important area in statistical computing. Indeed, many of the problems arise with quite modest datasets with a thousand or so examples, but even those were not common a decade or two ago.

We will always need to bear in mind the 'data dredging' aspect of the term. When (literally) mining or dredging, the proportion of good material to dross is usually very low, and when mining for minerals can often be too low to cover the costs of extraction. Exactly the same issues occur in looking for structure in data: it is all too easy to find structure that is only characteristic of the particular set of data to hand. We want *generalization* in the terminology of the psychologists, that is to find structure that will help with future examples too.

Statistics has been concerned with detecting structure in data under uncertainty for many years: that is what the design of experiments developed in the inter-war years had as its aims. Generally that gave a single outcome ('yield') on a hundred or so experimental points. *Multivariate analysis* was concerned with multiple (usually more than two and often fewer than twenty) measurements on different cases (often subjects).

In engineering, very similar (often identical) methods were being developed under the heading of *pattern recognition*. Engineers tend to distinguish between

statistical pattern recognition where everything is 'learnt from examples'

structural pattern recognition where most of the structure is imposed from *a priori* knowledge. This used to be called *syntactic pattern recognition*, in which the structure was imposed by a formal grammar, but that has proved to be pretty unsuccessful.

Note that structure is imposed in *statistical pattern recognition* via prior assumptions on the difference between signal and noise, but that structure is not deterministic as in structural pattern recognition. It is the inability to cope with exceptions that has bedevilled structural pattern recognition (and much of the research on expert systems).

However, a practically much more important distinction is between

unsupervised methods in which there is no known grouping of the examples

supervised methods in which the examples are known to be grouped in advance, or ordered by some response, and the task is to group future examples or predict which are going to be give a 'good' response.

It is important to bear in mind that unsupervised pattern recognition is like *looking for needles in haystacks*. It covers the formation of good scientific theories and the search for therapeutically useful pharmaceutical compounds. It is best thought of as hypothesis formation, and independent confirmation will be needed.

There are a large number of books in this area, including Duda *et al.* (2001); Hand *et al.* (2001); Hastie *et al.* (2001); Ripley (1996); Webb (1999); Witten & Frank (2000).

1.1 Multivariate analysis

Multivariate analysis is concerned with datasets which 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 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 unsupervised and supervised 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 is 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 between the species. (In fact, Fisher (1936) 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 sub-species.)

1.2 Graphical methods

The simplest way to examine multivariate data is via a pairs plot, enhanced to show the groups. More dynamic versions are available in XGobi, GGobi and S-PLUS's `brush`.

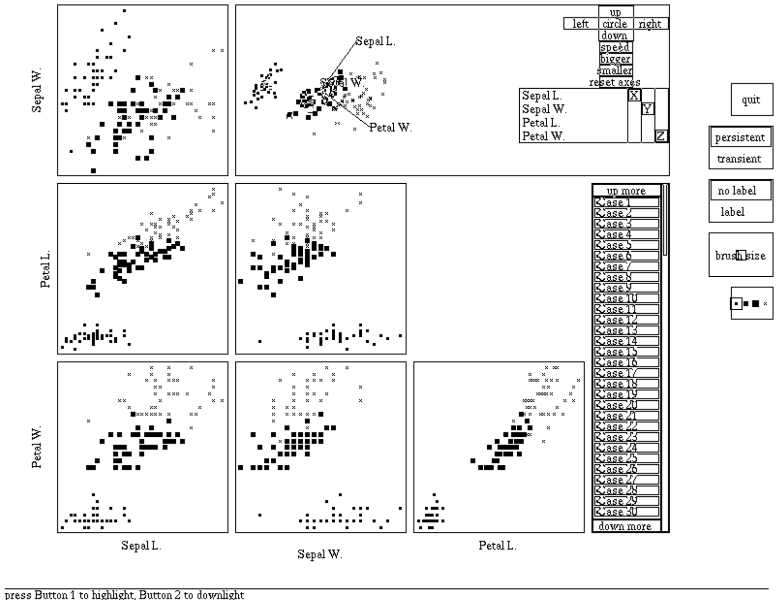

Figure 1.1: S-PLUS `brush` plot of the `iris` data.

Principal component analysis

Linear methods are the heart of classical multivariate analysis, and depend on seeking linear combinations of the variables with desirable properties. For the unsupervised case the main method is *principal component* analysis, which 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.

Let S denote the covariance matrix of the data X, which is defined[1] by

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

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 $\boldsymbol{xa}$ 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 which are uncorrelated with those which 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, even on different platforms.)

Another point of view is to seek new variables y_j which are rotations of the old variables to explain best the variation in the dataset. Clearly these new variables should be taken to be the principal components, in order. Suppose we use the first k principal components. Then the subspace they span contains the 'best' k-dimensional view of the data. It both has maximal covariance matrix (both in trace and determinant) and best approximates the original points in the sense of minimizing the sum of squared distance 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 re-scaling the variables).

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

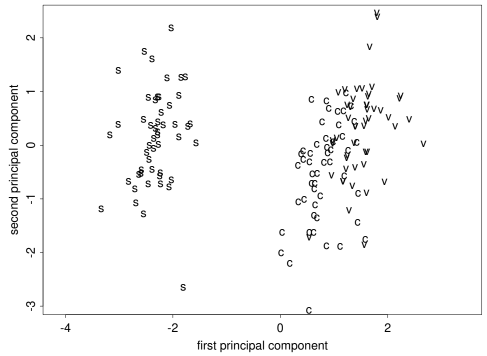

Figure 1.2: First two principal components for the log-transformed `iris` data.

```
> ir.pca <- princomp(log(ir), cor=T)
> ir.pca
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],  cex=0.8, labels = as.character(ir.sp),
       col = 1+as.numeric(ir.sp))
```

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.

Figure 1.2 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.

There are two books devoted solely to principal components, Jackson (1991) and Jolliffe (1986), which over-states its value as a technique. Other projection techniques such as *projection pursuit* discussed below choose rotations based on more revealing criteria than variances.

Distance methods

There 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. 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 & 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. 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 & Rousseeuw (1990, §2.5) provide a readable summary and recommendations, and Cox & 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]
eqscplot(ir.scal$points, type="n")
text(ir.scal$points, cex=0.8, labels = as.character(ir.sp),
     col = 1+as.numeric(ir.sp))
```


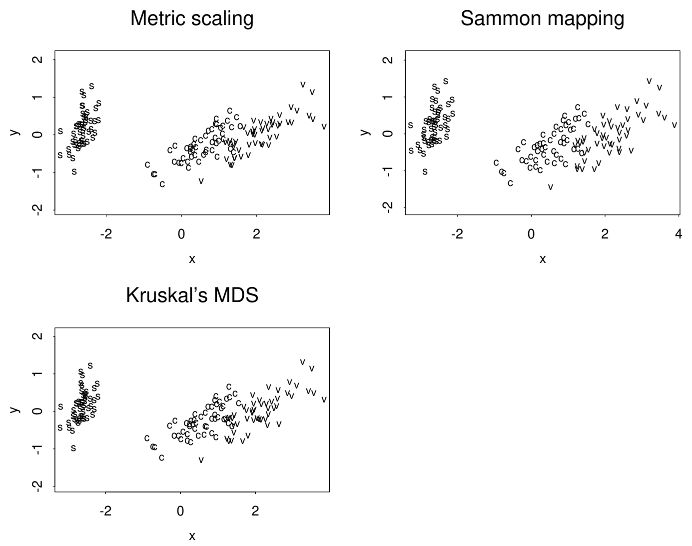

Figure 1.3: Distance-based representations of the `iris` data. The top left plot is by multidimensional scaling, the top right by Sammon's non-linear mapping, the bottom left by Kruskal's isotonic multidimensional scaling. Note that each is defined up to shifts, rotations and reflections.

where care is taken to ensure correct scaling of the axes (see the top left plot of Figure 1.3). 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.) 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 configuration with distances $\widetilde{d}$ to minimize a weighted measure of fit,

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

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` would report which observations are duplicates.

```
ir.sam <- sammon(dist(ir[-143,]))
eqscplot(ir.sam$points, type="n")
text(ir.sam$points, cex=0.8,  labels = as.character(ir.sp[-143]),
     col = 1+as.numeric(ir.sp[-143]))
```

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 & 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, cex=0.8,  labels = as.character(ir.sp[-143]),
     col = 1+as.numeric(ir.sp[-143]))
```

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

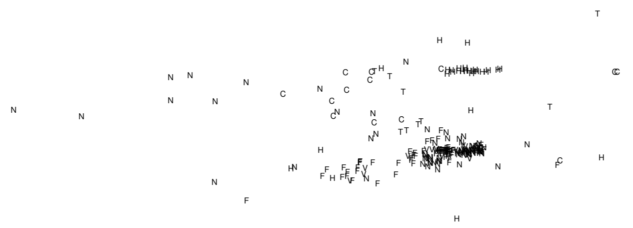

Figure 1.4: Isotonic multidimensional scaling representation in two dimensions of the `fgl` data. The groups are plotted by the initial letter, except `F` for window float glass, and `N` for window non-float glass. Small dissimilarities correspond to small distances on the plot and conversely.

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 1.4 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$points, type="n", xlab="", ylab="")
text(fgl.iso$points, labels = c("F", "N", "V", "C", "T", "H")
   [fgl$type[-40]], cex=0.6)
```


There are 214 fragments of class taken from scenes of crimes, classified into 'window float', 'window non-float', !vehicle window', 'containers', 'tableware' and 'vehicle headlamp' (column 10) and measurements of the refractive index and the chemical composition (8 oxides).

Biplots

The biplot (Gabriel, 1971) is another 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

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

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

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

The biplot then consists of plotting the $n+p$ 2-dimensional vectors which 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 represents the Mahalanobis distances

$$(\boldsymbol{x} - \boldsymbol{\mu})\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu})^T$$

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 1.5 shows a biplot with $\lambda = 1$, obtained by

```
library(MASS, first=T) # R: library(MASS)
state <- state.x77[,2:7]; row.names(state) <- state.abb
biplot(princomp(state, cor=T), pc.biplot=T, cex = 0.7, ex=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 & Hand (1996) in a book-length discussion of biplots criticize conventional plots such as Figure 1.5. In particular they point out that the axis scales are

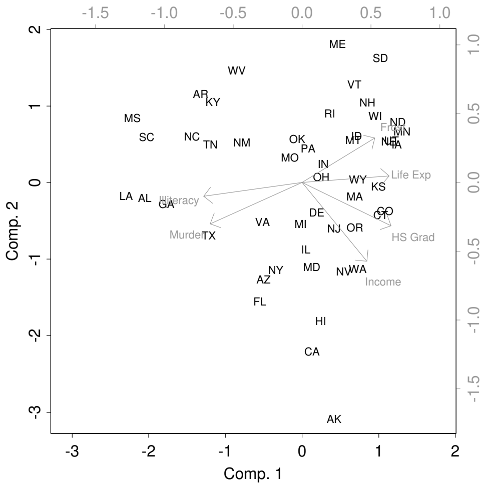

Figure 1.5: Principal component biplot of the part of the `state.x77` data. Distances between states represent Mahalanobis distance, and inner product between variables represent correlations. (The arrows extend 80% of the way along the variable's vector.)

not at all helpful. Notice that Figure 1.5 has 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. Gower & Hand's preferred style (for $\lambda = 0$) is to omit the external axes and to replace each arrow by a scale for that variable.

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 by Lee (1998); Hyvärinen *et al.* (2001) and the expositions by Hyvärinen & 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.

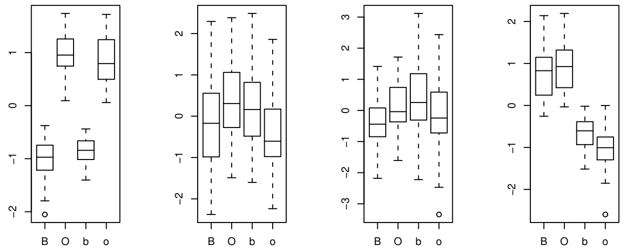

Figure 1.6: Boxplots of four 'signals' recovered by ICA from the `crabs` data.

Suppose the data X have been sphered; by assumption S is sphered and so X has variance $A^T A$ 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 & Oja (2000) is implemented is the R package `fastICA`.[2] We can illustrate this for the `crabs` data, where the first and fourth signals shown in Figure 1.6 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))
```

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 & Riedwyl, 1981) and the star plots as implemented in the function `stars` (see Figure 1.11), 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

[2] By Jonathan Marchini. Also ported to S-PLUS by BDR.

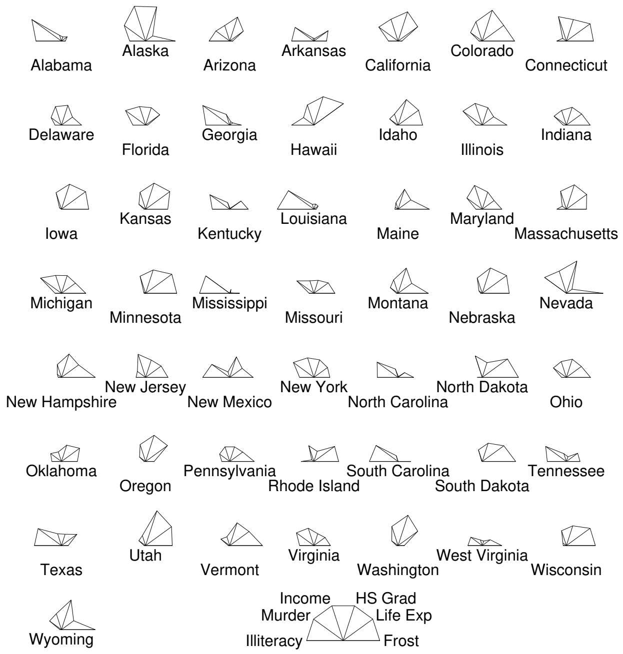

Figure 1.7: R version of `stars` plot of the `state.x77` dataset.

of the possibility that the effect may differ by viewer.[3] 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 1.5 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 1.5 by

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

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

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


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

on a device which can plot colour.

1.3 Cluster analysis

Cluster analysis is concerned with discovering group structure amongst the cases of our n by p matrix.

A comprehensive general reference is Gordon (1999); Kaufman & 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*

$$d(A,B) \leqslant \max\bigl(d(A,C), d(B,C)\bigr)$$

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 1.8, 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 & 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 & 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, 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[4] `swiss.x` gives five measures of socio-economic data on Swiss provinces about 1888, given by Mosteller & 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 1.8) 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 1.8 was produced by `pltree(diana(swiss.x))`.

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

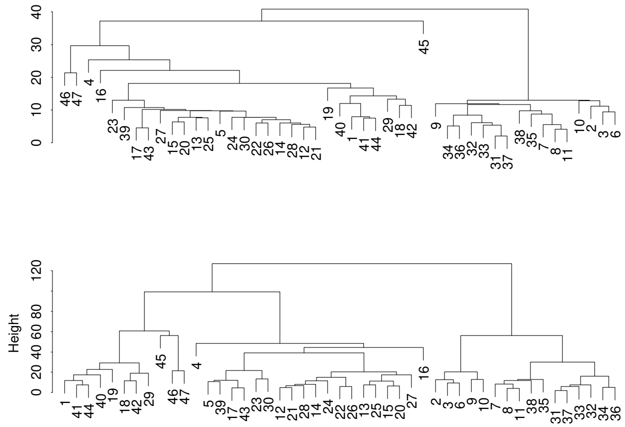

Figure 1.8: Dendrograms for the socio-economic data on Swiss provinces computed by single-link clustering (top) and divisive clustering (bottom).

Partitioning methods

The K-means clustering algorithm (MacQueen, 1967; Hartigan, 1975; Hartigan & 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 1.9. (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)
```

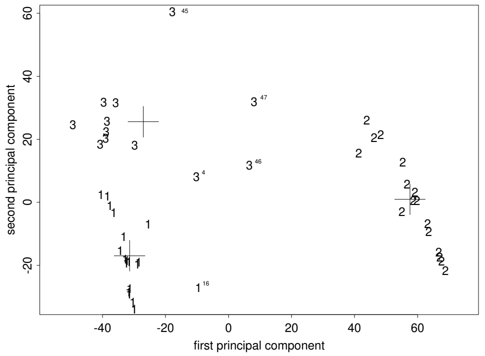

Figure 1.9: The Swiss provinces data plotted on its first two principal components. The labels are the groups assigned by K-means; the crosses denote the group means. Five points are labelled with smaller symbols.

```
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 2 1 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

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

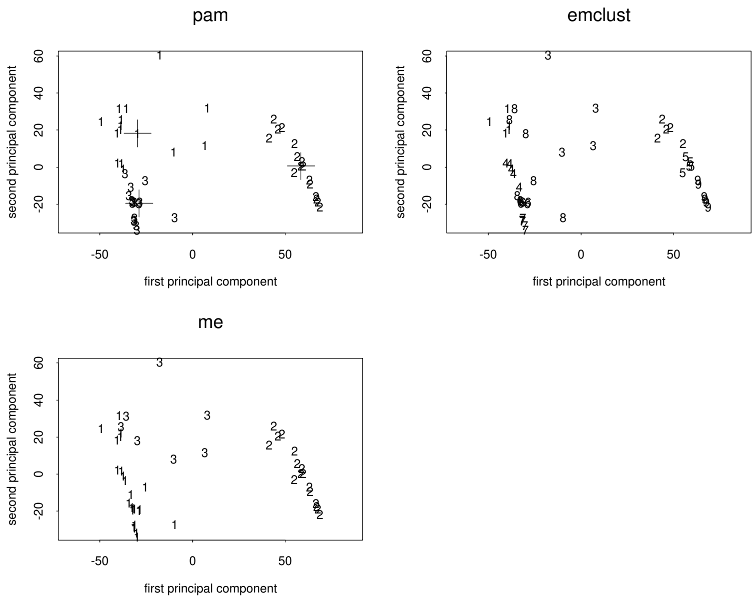

Figure 1.10: Clusterings of the Swiss provinces data by `pam` with three clusters (with the medoids marked by crosses), `me` with three clusters and `emclust` with up to nine clusters (it chose nine).

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 2 1 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 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-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[5] 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).

```
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
[29] 1 1 2 2 2 2 2 2 2 2 2 1 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 1.10).

```
> 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)
```

[5] Available at `http://www.stat.washington.edu/fraley/mclust/` and for R from CRAN. There are '1998' and '2002' versions available with different interfaces (but often the same function names) and different results. The '1998' versions are described here, as the '2002' versions do not return under Windows.

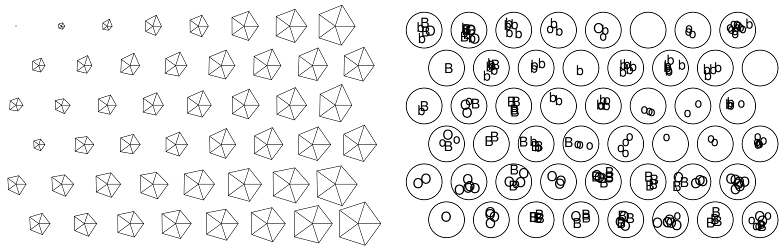

Figure 1.11: Batch SOM applied to the `crabs` dataset. The left plot is a `stars` plot of the representatives, and the right plot shows the assignments of the original points, placed randomly within the circle. (Plots from R.)

1.4 Kohonen's 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[6] $\boldsymbol{m}_j$, 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 & 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

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

number of iterations. Figure 1.11 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)

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[7] of Figure 1.11.

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 & Hernández-Pajares (1995) for another statistical assessment.

1.5 Exploratory projection pursuit

Projection pursuit methods seek a q-dimensional projection of the data that maximizes some measure of 'interestingness', usually for $q = 1$ or 2 so that it can be visualized. This measure would not be the variance, and would normally be scale-free. Indeed, most proposals are also affine invariant, so they do not depend on the correlations in the data either.

The methodology was named by Friedman & Tukey (1974),[8] who sought a measure which would reveal groupings in the data. Later reviews (Huber, 1985; Friedman, 1987; Jones & Sibson, 1987) have used the result of Diaconis & Freedman (1984) that a randomly selected projection of a high-dimensional dataset will appear similar to a sample from a multivariate normal distribution to stress that 'interestingness' has to mean departures from multivariate normality. Another argument is that the multivariate normal distribution is elliptically symmetrical,

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

[8] The *idea* goes back to Kruskal (1969, 1972). Kruskal (1969) needed a snappier title!

and cannot show clustering or non-linear structure, so all elliptically symmetrical distributions should be uninteresting.

The simplest way to achieve affine invariance is to 'sphere' the data before computing the index of 'interestingness'. Since a spherically symmetric point distribution has covariance matrix proportional to the identity, we transform the data to have identity covariance matrix. This can be done by transforming to principal components, discarding any components of zero variance (hence constant) and then rescaling each component to unit variance. As principal components are uncorrelated, the data are sphered. Of course, this process is susceptible to outliers and it may be wise to use a robust version of principal components. The discussion of Jones & Sibson (1987) included several powerful arguments against sphering, but as in principal component analysis something of this sort is needed unless a particular common scale for the features can be justified.

Specific examples of projection pursuit indices are given below. Once an index is chosen, a projection is chosen by numerically maximizing the index over the choice of projection. A q-dimensional projection is determined by a $p \times q$ orthogonal matrix and q will be small, so this may seem like a simple optimization task. One difficulty is that the index is often very sensitive to the projection directions, and good views may occur within sharp and well-separated peaks in the optimization space. Another is that the index may be very sensitive to small changes in the data configuration and so have very many local maxima. Rather than use a method which optimizes locally (such as quasi-Newton methods) it will be better to use a method which is designed to search for isolated peaks and so makes large steps. In the discussion of Jones & Sibson (1987), Friedman says

> 'It has been my experience that finding the substantive minima of a projection index is a difficult problem, and that simple gradient-guided methods (such as steepest ascent) are generally inadequate. The power of a projection pursuit procedure depends crucially on the reliability and thoroughness of the numerical optimizer.'

and our experience supports Friedman's wisdom. It will normally be necessary to try many different starting points, some of which may reveal projections with large values of the projection index.

Once an interesting projection is found, it is important to remove the structure it reveals to allow other interesting views to be found more easily. If clusters (or outliers) are revealed, these can be removed, and both the clusters and the remainder investigated for further structure. If non-linear structures are found, Friedman (1987) suggests non-linearly transforming the current view towards joint normality, but leaving the orthogonal subspace unchanged. This is easy for $q = 1$; any random variable with cumulative distribution function F can be transformed to a normal distribution by $\Phi^{-1}(F(X))$. For $q = 2$ Friedman suggests doing this for randomly selected directions until the two-dimensional projection index is small.

Projection indices

A very wide variety of indices have been proposed, as might be expected from the many ways a distribution can look non-normal. A projection index will be called repeatedly, so needs to be fast to compute. Recent attention has shifted towards indices which are rather crude approximations to desirable ones, but very fast to compute (being based on moments).

For simplicity, most of our discussion will be for one-dimensional projections; we return to two-dimensional versions at the end. Thus we seek a measure of the non-normality of a univariate random variable X. Our discussion will be in terms of the density f even though the index will have to be estimated from a finite sample. (This can be done by replacing population moments by sample moments or using some density estimate for f.)

The original Friedman–Tukey index had two parts, a 'spread' term and a 'local density' term. Once a scale has been established for X (including protecting against outliers), the local density term can be seen as a kernel estimator of $\int f^2(x)\,\mathrm{d}x$. The choice of bandwidth is crucial in any kernel estimation problem; as Friedman & Tukey were looking for compact non-linear features (cross-sections of 'rods'—see Tukey's contribution to the discussion of Jones & Sibson, 1987) they chose a small bandwidth. Even with efficient approximate methods to compute kernel estimates, this index remains one of the slowest to compute.

Jones & Sibson (1987) introduced an entropy index $\int f \log f$ (which is also very slow to compute) and indices based on moments such as $[\kappa_3^2 + 1/4\kappa_4^2]/12$, where the κ's are cumulants, the skewness and kurtosis here. These are fast to compute but sensitive to outliers (Best & Rayner, 1988).

Friedman (1987) motivated an index by first transforming normality to uniformity on $[-1, 1]$ by $Y = 2\Phi(X) - 1$ and using a moment measure of non-uniformity, specifically $\int (f_Y - 1/2)^2$. This can be transformed back to the original scale to give the index

$$I^L = \int \frac{\big[f(x) - \phi(x)\big]^2}{2\phi(x)}\,\mathrm{d}x.$$

This has to be estimated from a sample, and lends itself naturally to an orthogonal series expansion, the Legendre series for the transformed density.

The index I^L has the unfortunate effect of giving large weight to fluctuations in the density f in its tails (where ϕ is small), and so will display sensitivity to outliers and the precise scaling used for the density. This motivated Hall (1989) to propose the index

$$I^H = \int \big[f(x) - \phi(x)\big]^2\,\mathrm{d}x$$

and Cook *et al.* (1993) to propose

$$I^N = \int \big[f(x) - \phi(x)\big]^2 \phi(x)\,\mathrm{d}x.$$

Both of these are naturally computed via an orthogonal series estimator of f using Hermite polynomials. Note that all three indices reduce to $\sum_0^\infty w_i(a_i - b_i)^2$, where a_i are the coefficients in the orthogonal series estimator, and b_i are constants arising from the expansion for a standard normal distribution.

To make use of these indices, the series expansions have to be truncated, and possibly tapered as well. Cook *et al.* (1993) make the much more extreme suggestion of keeping only a very few terms, maybe the first one or two. These still give indices which are zero for the normal distribution, but which are much more attuned to large-scale departures from normality. For example, I_0^N is formed by keeping the first term of the expansion of I^N, $(a_0 - 1/2\sqrt{\pi})^2$ where $a_0 = \int \phi(x) f(x)\,\mathrm{d}x = E\phi(X)$, and this is maximized when a_0 is maximal. In this case the most 'interesting' distribution has all its mass at 0. The minimal value of a_0 gives a local maximum, attained by giving equal weight to ± 1. Now of course a point mass at the origin will not meet our scaling conditions, but this indicates that I_0^N is likely to respond to distributions with a central clump or a central hole. To distinguish between them we can maximize a_0 (for a clump) or $-a_0$ (for a central hole). These heuristics are borne out by experiment.

In principle the extension of these indices to two dimensions is simple. Those indices based on density estimation just need a two-dimensional density estimate and integration (and so are likely to be even slower to compute). Those based on moments use bivariate moments. For example, the index I^N becomes

$$I^N = \iint \big[f(x,y) - \phi(x)\phi(y)\big]^2 \phi(x)\phi(y)\,\mathrm{d}x\,\mathrm{d}y$$

and bivariate Hermite polynomials are used. To maintain rotational invariance in the index, the truncation has to include all terms up to a given degree of polynomial.

There is no unanimity over the merits of these indices (except the moment index, which seems universally poor). Some workers have reported that the Legendre index is very sensitive to outliers, and this is our experience. Yet Posse (1995) found it to work well, in a study that appears to contain no outliers. The natural Hermite index is particularly sensitive to a central hole and hence clustering. The best advice is to try a variety of indices.

1.6 An example of visualization

This is a dataset on 61 viruses with rod-shaped particles affecting various crops (tobacco, tomato, cucumber and others) described by Fauquet *et al.* (1988) and analysed by Eslava-Gómez (1989). There are 18 measurements on each virus, the number of amino acid residues per molecule of coat protein; the data come from a total of 26 sources. There is an existing classification by the number of RNA molecules and mode of transmission, into

39 *Tobamoviruses* with monopartite genomes spread by contact,
6 *Tobraviruses* with bipartite genomes spread by nematodes,

3 *Hordeiviruses* with tripartite genomes, transmission modeunknown and
13 'furoviruses', 12 of which are known to be spread fungally.

The question of interest to Fauquet *et al.* was whether the furoviruses form a distinct group, and they performed various multivariate analyses.

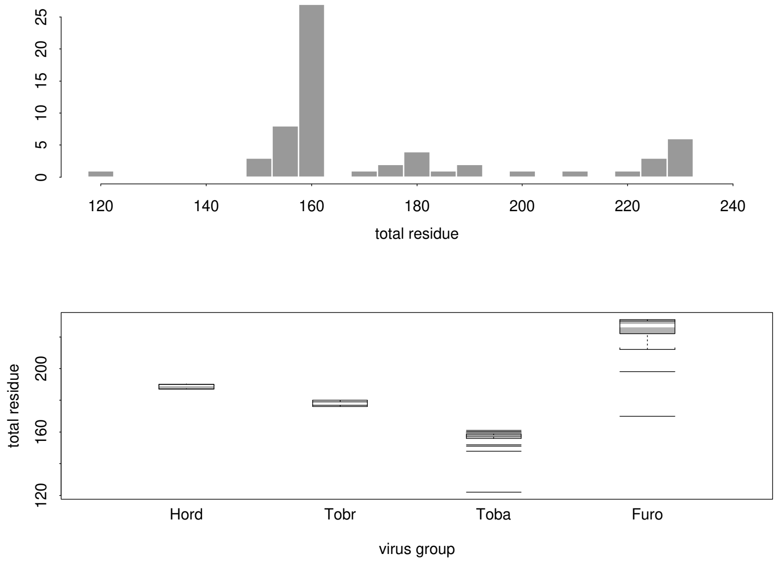

Figure 1.12: Histogram and boxplot by group of the viruses dataset. A boxplot is a representation of the distribution; the central grey box shows the middle 50% of the data, with median as a white bar. 'Whiskers' go out to plausible extremes, with outliers marked by bars.

One initial question with this dataset is whether the numbers of residues are absolute or relative. The data are counts from 0 to 32, with the totals per virus varying from 122 to 231. The average numbers for each amino acid range from 1.4 to 20.3. As a classification problem, this is very easy as Figure 1.12 shows. The histogram shows a multimodal distribution, and the boxplots show an almost complete separation by virus type. The only exceptional value is one virus in the furovirus group with a total of 170; this is the only virus in that group whose mode of transmission is unknown and Fauquet *et al.* (1988) suggest it has been tentatively classified as a *Tobamovirus*. The other outlier in that group (with a total of 198) is the only beet virus. The conclusions of Fauquet *et al.* may be drawn from the totals alone.

It is interesting to see if there are subgroups within the groups, so we will use this dataset to investigate further the largest group (the *Tobamoviruses*). There are

two viruses with identical scores, of which only one is included in the analyses. (No analysis of these data could differentiate between the two.)

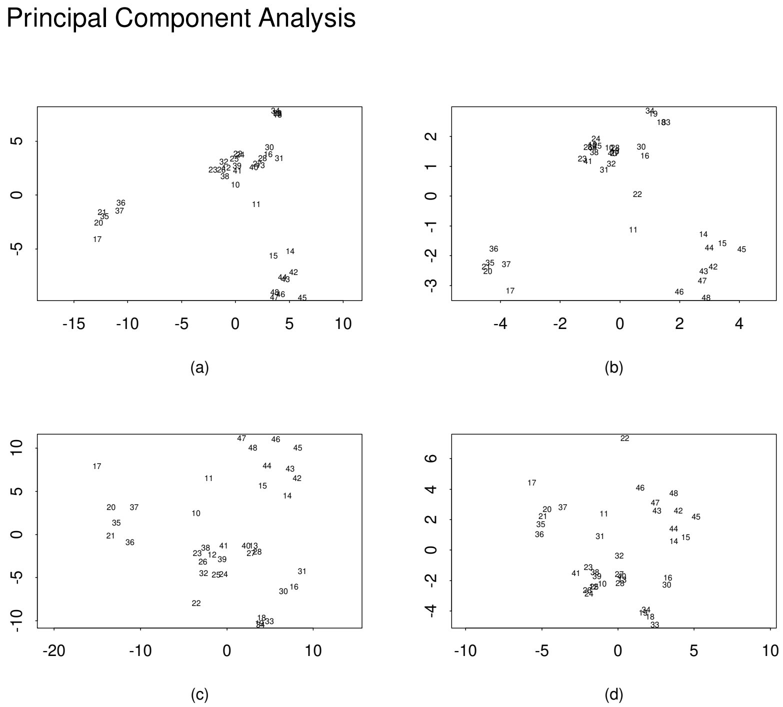

Figure 1.13: Principal component (top row) and Sammon mapping (bottom row) plots of the *Tobamovirus* group of the viruses example. The plots in the left column are of the raw data, those in the right column with variables rescaled to have unit variance. The points are labelled by the index number of the virus.

We consider the *Tobamovirus* group of the viruses example, which has $n = 38$ examples with $p = 18$ features. Figure 1.13 shows plots of the first two principal components with 'raw' and scaled variables. As the data here are counts, there are arguments for both, but principally for scaling as the counts vary in range quite considerably between variables. Virus 11 (Sammon's opuntia virus) stands out on both plots: this is the one virus with a much lower total (122 rather than 157–161). Both plots suggest three subgroupings.

In both cases the first two principal components have about equal variances, and together contain about 69% and 52% of the variance in the raw and scaled plots respectively.

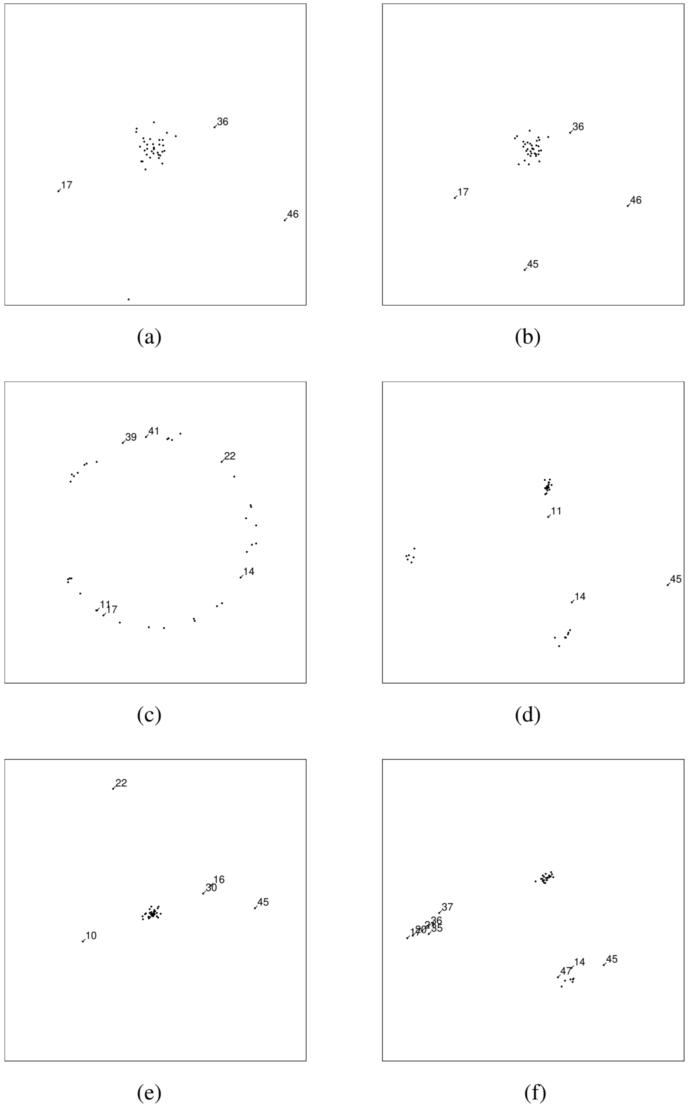

Figure 1.14: Projections of the *Tobamovirus* group of the viruses data found by projection pursuit. Views (a) and (b) were found using the natural Hermite index, view (c) by minimizing a_0, and views (d, e, f) were found by the Friedman–Tukey index $\int f^2$ with different choices of bandwidth for the kernel density estimator.

Projection Pursuit

Figure 1.14 shows six views of the main group of the viruses dataset obtained by (locally) optimizing various projection indices; this is a small subset of hundreds of views obtained in interactive experimentation in XGobi. With only 38 points in 18 dimensions, there is a lot of scope for finding a view in which an arbitrarily selected point appears as an outlier, and there is no clear sense in which this dataset contains outliers (except point 11, whose total residue is very much less than the others). When viewing rotating views of multidimensional datasets (a *grand tour* in the terminology of Asimov, 1985) true outliers are sometimes revealed by the differences in speed and direction which they display—certainly point 11 stands out in this dataset.

Not many views showed clear groupings rather than isolated points. The Friedman–Tukey index was most successful in this example. Eslava-Gómez (1989) studied all three groups (which violates the principle of removing known structure).

This example illustrates a point made by Huber (1985, §21); we need a *very* large number of points in 18 dimensions to be sure that we are not finding quirks of the sample but real features of the generating process. Thus projection pursuit may be used for hypothesis formation, but we will need independent evidence of the validity of the structure suggested by the plots.

Multidimensional Scaling

Figure 1.15 shows a local minimum for ordinal multidimensional scaling for the scaled viruses data. (The fit is poor, with $STRESS \approx 17\%$, and we found several local minima differing in where the outliers were placed.) This fit is similar to that by Sammon mapping in Figure 1.13, but the subgroups are more clearly separated, and viruses 10 (frangipani mosaic virus), 17 (cucumber green mottle mosaic virus) and 31 (pepper mild mottle virus) have been clearly separated. Figure 1.16 shows the distortions of the distances produced by the Sammon and ordinal scaling methods. Both show a tendency to increase large distances relative to short ones for this dataset, and both have considerable scatter.

Figure 1.15 shows some interpretable groupings. That on the upper left is the cucumber green mottle virus, the upper right group is the ribgrass mosaic virus and two others, and a group at bottom centre-right (16, 18, 19, 30, 33, 34) are the tobacco mild green mosaic and odontoglossum ringspot viruses.

Cluster Analysis

Figure 1.17 shows the clusters for 6-means for the virus data. The iterative process has to be started somewhere, and in this case was initialized from a hierarchical clustering discussed below. The choice of 6 clusters was by inspection of the visualization plots discussed above and the dendrograms shown in Figure 1.18.

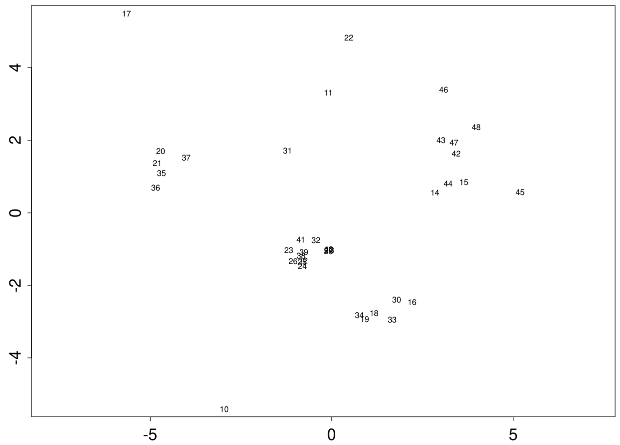

Figure 1.15: Non-metric multidimensional scaling plot of the *Tobamovirus* group of the viruses example. The variables were scaled before Euclidean distance was used. The points are labelled by the index number of the virus.

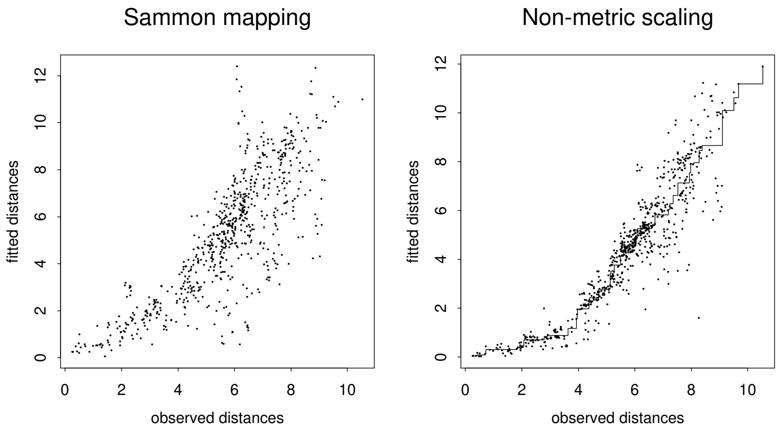

Figure 1.16: Distortion plots of Sammon mapping and non-metric multidimensional scaling for the viruses data. For the right-hand plot the fitted isotonic regression is shown as a step function.

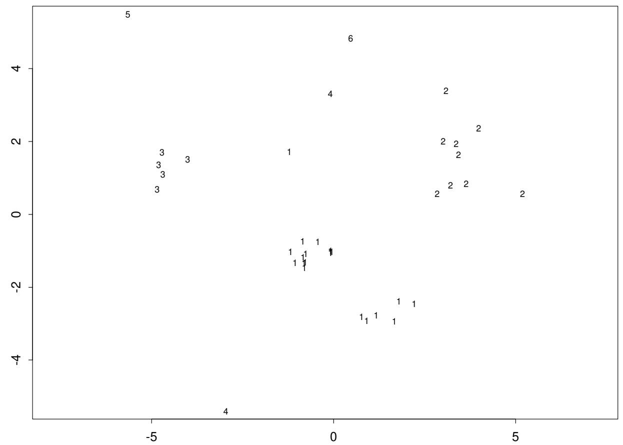

Figure 1.17: The clusters suggested by k-means for $k = 6$ for the virus data displayed on the ordinal multidimensional scaling plot.

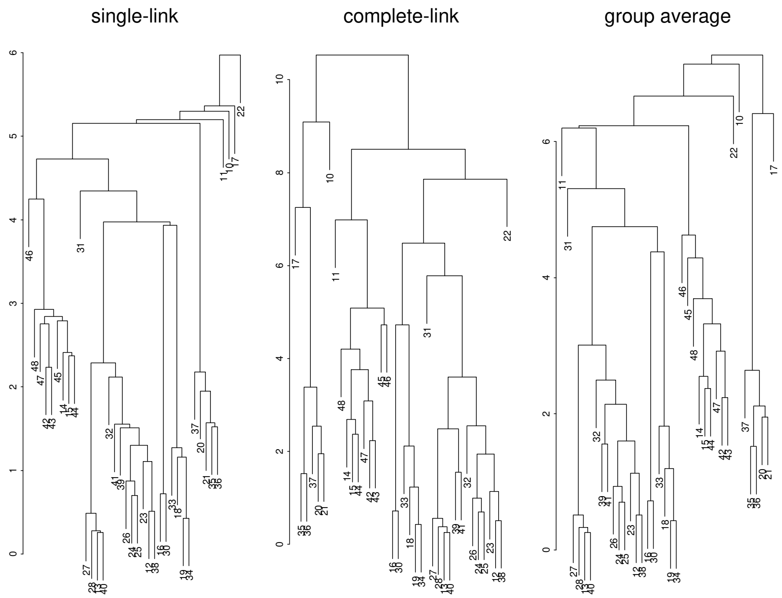

Figure 1.18: Dendrograms from three common hierarchical clustering techniques applied to the scaled viruses data. Such plots show the dissimilarity at which clusters are merged on the vertical scale and so show the construction process from bottom to top.

Figure 1.18 shows dendrograms produced by single-link, complete-link and group-average clustering for the viruses data. All identify viruses 10, 11, 17, 22 and 31 as loosely connected to the rest, and single-link also highlights virus 46.

(We note that 10, 11, 17, 31, 46 and 48 are called 'miscellaneous' in the original source.) Nevertheless, each graph gives the impression of three or four major groupings of viruses.

1.7 Categorical data

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 & 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 1.19 shows mosaic plots for these data and for the Copenhagen `housing` data, 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)
```


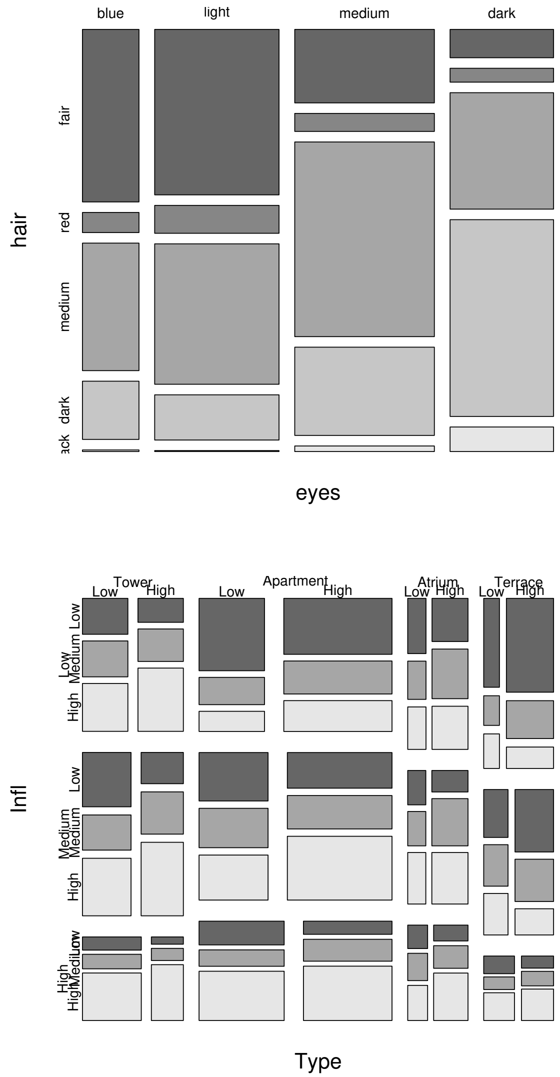

Figure 1.19: Mosaic plots for (top) Fisher's data on people from Caithness and (bottom) Copenhagen housing satisfaction data.

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 & 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 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

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

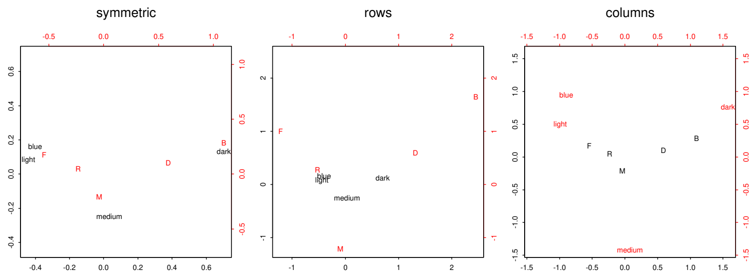

Figure 1.20: Three variants of correspondence analysis plots from Fisher's data on people in Caithness: (left) 'symmetric", (middle) 'row asymmetric' and (right) 'column asymmetric'.

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 1.20 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).

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 & 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 $p^{-1/2} G D_c^{-1/2}/pn$.[9]

[9] Gower & Hand (1996) omit the divisor pn.

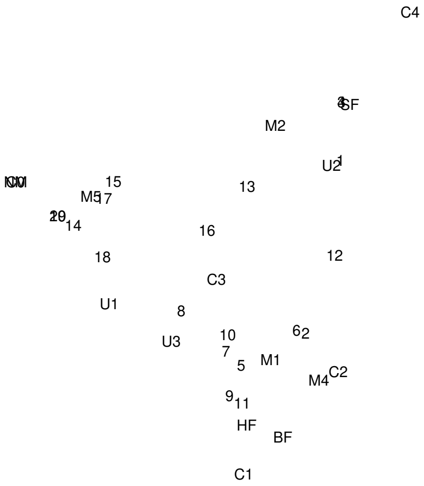

Figure 1.21: Multiple correspondence analysis plot of dataset `farms` on 20 farms on the Dutch island of Terschelling. Numbers represent the farms and labels levels of moisture (`M1`, `M2`, `M4` and `M5`), grassland usage (`U1`, `U2` and `U3`), manure usage (`C0` to `C4`) and type of grassland management (`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.

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 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 & 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 1.21

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 $U\Lambda V^T = X = G(pD_c)^{-1/2}$, $V = (pD_c)^{-1/2}G^T U\Lambda^{-1}$ and

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

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.

Chapter 2

Tree-based Methods

The use of tree-based models will be relatively unfamiliar to statisticians, although researchers in other fields have found trees to be an attractive way to express knowledge and aid decision-making. Keys such as Figure 2.1 are common in botany and in medical decision-making, and provide a way to encapsulate and structure the knowledge of experts to be used by less-experienced users. Notice how this tree uses both categorical variables and splits on continuous variables. (It is a tree, and readers are encouraged to draw it.)

The automatic construction of decision trees dates from work in the social sciences by Morgan & Sonquist (1963) and Morgan & Messenger (1973). In statistics Breiman *et al.* (1984) had a seminal influence both in bringing the work to the attention of statisticians and in proposing new algorithms for constructing trees. At around the same time decision tree induction was beginning to be used in the field of *machine learning*, notably by Quinlan (1979, 1983, 1986, 1993), and in engineering (Henrichon & Fu, 1969; Sethi & Sarvarayudu, 1982). Whereas there is now an extensive literature in machine learning, further statistical contributions are still sparse. The introduction within S of tree-based models described by Clark & Pregibon (1992) made the methods much more freely available. The library `rpart` (Therneau & Atkinson, 1997) provides a faster and more tightly-packaged set of S functions for fitting trees to data, which we describe here.

Ripley (1996, Chapter 7) gives a comprehensive survey of the subject, with proofs of the theoretical results.

Constructing trees may be seen as a type of variable selection. Questions of interaction between variables are handled automatically, and to a large extent so is monotonic transformation of both the x and y variables. These issues are reduced to which variables to divide on, and how to achieve the split.

Figure 2.1 is a *classification* tree since its endpoint is a factor giving the species. Although this is the most common use, it is also possible to have *regression* trees in which each terminal node gives a predicted value, as shown in Figure 2.2 for our dataset `cpus`.

Much of the machine learning literature is concerned with logical variables and correct decisions. The end point of a tree is a (labelled) partition of the space $\mathcal{X}$ of possible observations. In logical problems it is assumed that there *is* a partition of the space $\mathcal{X}$ that will correctly classify all observations, and the task is to find a

1.	Leaves subterete to slightly flattened, plant with bulb	*2.*
	Leaves flat, plant with rhizome	*4.*
2.	Perianth-tube > 10 mm	**I. × hollandica**
	Perianth-tube < 10 mm	*3.*
3.	Leaves evergreen	**I. xiphium**
	Leaves dying in winter	**I. latifolia**
4.	Outer tepals bearded	**I. germanica**
	Outer tepals not bearded	*5.*
5.	Tepals predominately yellow	*6.*
	Tepals blue, purple, mauve or violet	*8.*
6.	Leaves evergreen	**I. foetidissima**
	Leaves dying in winter	*7.*
7.	Inner tepals white	**I. orientalis**
	Tepals yellow all over	**I. pseudocorus**
8.	Leaves evergreen	**I. foetidissima**
	Leaves dying in winter	*9.*
9.	Stems hollow, perianth-tube 4–7mm	**I. sibirica**
	Stems solid, perianth-tube 7–20mm	*10.*
10.	Upper part of ovary sterile	*11.*
	Ovary without sterile apical part	*12.*
11.	Capsule beak 5–8mm, 1 rib	**I. enstata**
	Capsule beak 8–16mm, 2 ridges	**I. spuria**
12.	Outer tepals glabrous, many seeds	**I. versicolor**
	Outer tepals pubescent, 0–few seeds	**I. × robusta**

Figure 2.1: Key to British species of the genus *Iris*. Simplified from Stace (1991, p. 1140), by omitting parts of his descriptions.

tree to describe it succinctly. A famous example of Donald Michie (for example, Michie, 1989) is whether the space shuttle pilot should use the autolander or land manually (Table 2.1). Some enumeration will show that the decision has been specified for 253 out of the 256 possible observations. Some cases have been specified twice. This body of expert opinion needed to be reduced to a simple decision aid, as shown in Figure 2.3. (Table 2.1 appears to result from a decision tree that differs from Figure 2.3 in reversing the order of two pairs of splits.)

Note that the botanical problem is treated as if it were a logical problem, although there will be occasional specimens that do not meet the specification for their species.

2.1 Partitioning methods

The ideas for classification and regression trees are quite similar, but the terminology differs, so we consider classification first. Classification trees are more familiar and it is a little easier to justify the tree-construction procedure, so we consider them first.

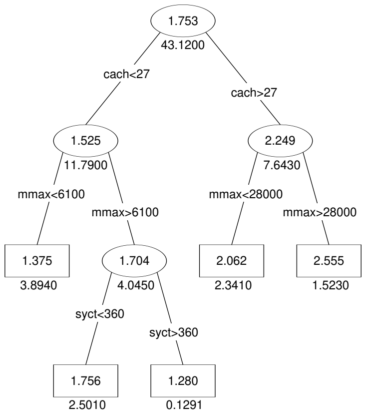

Figure 2.2: A regression tree for the cpu performance data on $\log_{10}$ scale. The value in each node is the prediction for the node; those underneath the nodes indicate the deviance contributions D_i.

Table 2.1: Example decisions for the space shuttle autolander problem.

stability	**error**	**sign**	**wind**	**magnitude**	**visibility**	**decision**
any	any	any	any	any	no	auto
xstab	any	any	any	any	yes	noauto
stab	LX	any	any	any	yes	noauto
stab	XL	any	any	any	yes	noauto
stab	MM	nn	tail	any	yes	noauto
any	any	any	any	Out of range	yes	noauto
stab	SS	any	any	Light	yes	auto
stab	SS	any	any	Medium	yes	auto
stab	SS	any	any	Strong	yes	auto
stab	MM	pp	head	Light	yes	auto
stab	MM	pp	head	Medium	yes	auto
stab	MM	pp	tail	Light	yes	auto
stab	MM	pp	tail	Medium	yes	auto
stab	MM	pp	head	Strong	yes	noauto
stab	MM	pp	tail	Strong	yes	auto

Classification trees

We have already noted that the endpoint for a tree is a partition of the space $\mathcal{X}$, and we compare trees by how well that partition corresponds to the correct decision rule for the problem. In logical problems the easiest way to compare partitions is to count the number of errors, or, if we have a prior over the space $\mathcal{X}$, to compute the probability of error.

In statistical problems the distributions of the classes over $\mathcal{X}$ usually overlap,

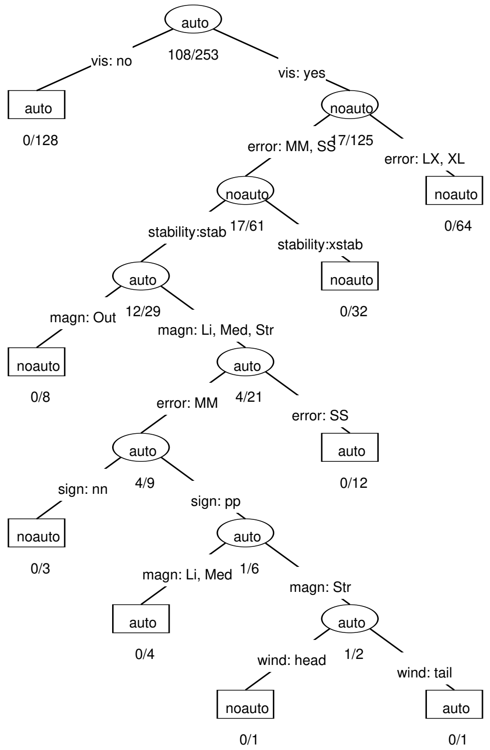

Figure 2.3: Decision tree for shuttle autolander problem. The numbers m/n denote the proportion of training cases reaching that node wrongly classified by the label.

so there is no partition that completely describes the classes. Then for each cell of the partition there will be a probability distribution over the classes, and the Bayes decision rule will choose the class with highest probability. This corresponds to assessing partitions by the overall probability of misclassification. Of course, in practice we do not have the whole probability structure, but a training set of n classified examples that we assume are an independent random sample. Then we can estimate the misclassification rate by the proportion of the training set that is misclassified.

Almost all current tree-construction methods use a one-step lookahead. That is, they choose the next split in an optimal way, without attempting to optimize the performance of the whole tree. (This avoids a combinatorial explosion over future choices, and is akin to a very simple strategy for playing a game such as chess.) However, by choosing the right measure to optimize at each split, we can ease future splits. It does not seem appropriate to use the misclassification rate to choose the splits.

What class of splits should we allow? Both Breiman *et al.*'s CART methodology and the `rpart` functions only allow binary splits, which avoids one difficulty in comparing splits, that of normalization by size. For a continuous variable x_j the allowed splits are of the form $x_j < t$ versus $x_j \geqslant t$. For ordered factors the splits are of the same type. For general factors the levels are divided into two classes. (Note that for L levels there are 2^L possible splits, and if we disallow the empty split and ignore the order, there are still $2^{L-1} - 1$. For ordered factors there are only $L - 1$ possible splits.) Some algorithms, including CART but excluding S, allow linear combination of continuous variables to be split, and Boolean combinations to be formed of binary variables.

The justification for the S methodology is to view the tree as providing a probability model (hence the title 'tree-based models' of Clark & Pregibon, 1992). At each node i of a classification tree we have a probability distribution p_{ik} over the classes. The partition is given by the *leaves* of the tree (also known as terminal nodes). Each case in the training set is assigned to a leaf, and so at each leaf we have a random sample n_{ik} from the multinomial distribution specified by p_{ik}.

We now condition on the observed variables $\boldsymbol{x}_i$ in the training set, and hence we know the numbers n_i of cases assigned to every node of the tree, in particular to the leaves. The conditional likelihood is then proportional to

$$\prod_{\text{cases } j} p_{[j]y_j} = \prod_{\text{leaves } i} \prod_{\text{classes } k} p_{ik}^{n_{ik}}$$

where $[j]$ denotes the leaf assigned to case j. This allows us to define a deviance for the tree as

$$D = \sum_i D_i, \qquad D_i = -2\sum_k n_{ik} \log p_{ik}$$

as a sum over leaves.

Now consider splitting node s into nodes t and u. This changes the probability model within node s, so the reduction in deviance for the tree is

$$D_s - D_t - D_u = 2\sum_k \left[n_{tk} \log \frac{p_{tk}}{p_{sk}} + n_{uk} \log \frac{p_{uk}}{p_{sk}} \right]$$

Since we do not know the probabilities, we estimate them from the proportions in the split node, obtaining

$$\hat{p}_{tk} = \frac{n_{tk}}{n_t}, \qquad \hat{p}_{uk} = \frac{n_{uk}}{n_u}, \qquad \hat{p}_{sk} = \frac{n_t \hat{p}_{tk} + n_u \hat{p}_{uk}}{n_s} = \frac{n_{sk}}{n_s}$$

so the reduction in deviance is

$$\begin{aligned} D_s - D_t - D_u &= 2\sum_k \left[n_{tk} \log \frac{n_{tk} n_s}{n_{sk} n_t} + n_{uk} \log \frac{n_{uk} n_s}{n_{sk} n_u} \right] \\ &= 2\Big[\sum_k n_{tk} \log n_{tk} + n_{uk} \log n_{uk} - n_{sk} \log n_{sk} \\ &\qquad + n_s \log n_s - n_t \log n_t - n_u \log n_u \Big] \end{aligned}$$

This gives a measure of the value of a split. Note that it is size-biased; there is more value in splitting leaves with large numbers of cases.

The tree construction process takes the maximum reduction in deviance over all allowed splits of all leaves, to choose the next split. (Note that for continuous variates the value depends only on the split of the ranks of the observed values, so we may take a finite set of splits.) The tree construction continues until the number of cases reaching each leaf is small (by default $n_i < 10$ in S) or the leaf is homogeneous enough (by default its deviance is less than 1% of the deviance of the root node in S, which is a size-biased measure). Note that as all leaves not meeting the stopping criterion will eventually be split, an alternative view is to consider splitting any leaf and choose the best allowed split (if any) for that leaf, proceeding until no further splits are allowable.

This justification for the value of a split follows Ciampi *et al.* (1987) and Clark & Pregibon, but differs from most of the literature on tree construction. The more common approach is to define a measure of the impurity of the distribution at a node, and choose the split that most reduces the average impurity. Two common measures are the *entropy* or *information* $\sum p_{ik} \log p_{ik}$ and the *Gini index*

$$\sum_{j \neq k} p_{ij} p_{ik} = 1 - \sum_k p_{ik}^2$$

As the probabilities are unknown, they are estimated from the node proportions. With the entropy measure, the average impurity differs from D by a constant factor, so the tree construction process is the same, except perhaps for the stopping rule. Breiman *et al.* preferred the Gini index.

Regression trees

The prediction for a regression tree is constant over each cell of the partition of $\mathcal{X}$ induced by the leaves of the tree. The deviance is defined as

$$D = \sum_{\text{cases } j} (y_j - \mu_{[j]})^2$$

and so clearly we should estimate the constant μ_i for leaf i by the mean of the values of the training-set cases assigned to that node. Then the deviance is the sum over leaves of D_i, the corrected sum of squares for cases within that node, and the value of a split is the reduction in the residual sum of squares.

The obvious probability model (and that proposed by Clark & Pregibon) is to take a normal $N(\mu_i, \sigma^2)$ distribution within each leaf. Then D is the usual scaled deviance for a Gaussian GLM. However, the distribution at internal nodes of the tree is then a mixture of normal distributions, and so D_i is only appropriate at the leaves. The tree-construction process has to be seen as a hierarchical refinement of probability models, very similar to forward variable selection in regression. In contrast, for a classification tree, one probability model can be used throughout the tree-construction process.

Missing values

One attraction of tree-based methods is the ease with which missing values can be handled. Consider the botanical key of Figure 2.1. We only need to know about a small subset of the 10 observations to classify any case, and part of the art of constructing such trees is to avoid observations that will be difficult or missing in some of the species (or as in capsules, for some of the cases). A general strategy is to 'drop' a case down the tree as far as it will go. If it reaches a leaf we can predict y for it. Otherwise we use the distribution at the node reached to predict y, as shown in Figure 2.2, which has predictions at all nodes.

An alternative strategy is used by many botanical keys and can be seen at nodes 9 and 12 of Figure 2.1. A list of characteristics is given, the most important first, and a decision made from those observations that are available. This is codified in the method of *surrogate splits* in which surrogate rules are available at non-terminal nodes to be used if the splitting variable is unobserved. Another attractive strategy is to split cases with missing values, and pass part of the case down each branch of the tree (Ripley, 1996, p. 232).

Cutting trees down to size

With 'noisy' data, that is when the distributions for the classes overlap, it is quite possible to grow a tree which fits the training set well, but which has adapted too well to features of that subset of $\mathcal{X}$. Similarly, regression trees can be too elaborate and over-fit the training data. We need an analogue of variable selection in regression.

The established methodology is cost-complexity *pruning*, first introduced by Breiman *et al.* (1984). They considered rooted subtrees of the tree $\mathcal{T}$ grown by the construction algorithm, that is the possible results of snipping off terminal subtrees on $\mathcal{T}$. The pruning process chooses one of the rooted subtrees. Let R_i be a measure evaluated at the leaves, such as the deviance or the number of errors, and let R be the value for the tree, the sum over the leaves of R_i. Let the size of the tree be the number of leaves. Then Breiman *et al.* showed that the set of rooted subtrees of $\mathcal{T}$ which minimize the cost-complexity measure

$$R_\alpha = R + \alpha\,\mathrm{size}$$

is itself nested. That is, as we increase α we can find the optimal trees by a sequence of snip operations on the current tree (just like pruning a real tree). This produces a sequence of trees from the size of $\mathcal{T}$ down to just the root node, but it may prune more than one node at a time. (Short proofs of these assertions are given by Ripley, 1996, Chapter 7. The tree $\mathcal{T}$ is not necessarily optimal for $\alpha = 0$, as we illustrate.)

We need a good way to choose the degree of pruning. If a separate validation set is available, we can predict on that set, and compute the deviance versus α for the pruned trees. This will often have a minimum, and we can choose the smallest tree whose deviance is close to the minimum.

If no validation set is available we can make one by splitting the training set. Suppose we split the training set into 10 (roughly) equally sized parts. We can then use 9 to grow the tree and test it on the tenth. This can be done in 10 ways, and we can average the results.

Examples of classification trees

Forensic Glass

Our first example comes from forensic testing of glass collected by B. German on 214 fragments of glass. Each case has a measured refractive index and composition (weight percent of oxides of Na, Mg, Al, Si, K, Ca, Ba and Fe). The fragments were originally classed as seven types, one of which was absent in this dataset. The categories which occur are window float glass (70), window non-float glass (76), vehicle window glass (17), containers (13), tableware (9) and vehicle headlamps (29). The composition sums to around 100%; what is not anything else is sand. The full tree is shown in figure 2.4.

For this tree the pruning sequence

$$(\alpha_i) = (-\infty, 0, 0.5, 1, 2, 2.5, 4.67, 7, 8, 11, 27, \infty)$$

Figure 2.5 shows four separate 10-fold cross-validation experiments, differing only in the random partition into groups. There is a fairly consistent pattern prefering $\alpha \approx 5$, but also some suggestion that $\alpha \approx 2$ might also be supportable.

The '1-SE' rule says to choose the smallest tree for which the cross-validated number of errors is within about one standard error of the minimum. Here the minimum is about error, so a Poisson distribution suggests the standard error is around 9.

Pima Indians

A population of women who were at least 21 years old, of Pima Indian heritage and living near Phoenix, Arizona, was tested for diabetes according to World Health Organization criteria. The data were collected by the US National Institute of Diabetes and Digestive and Kidney Diseases. The reported variables are

number of pregnancies
plasma glucose concentration in an oral glucose tolerance test
diastolic blood pressure (mm Hg)
triceps skin fold thickness (mm)
serum insulin (μ U/ml)
body mass index (weight in kg/(height in m)^2)
diabetes pedigree function
age in years

Of the 768 records, 376 were incomplete (most prevalently in serum insulin). Most of our illustrations omit serum insulin and use the 532 complete records on the remaining variables. These were randomly split into a training set of size 200 and a validation set of size 332.

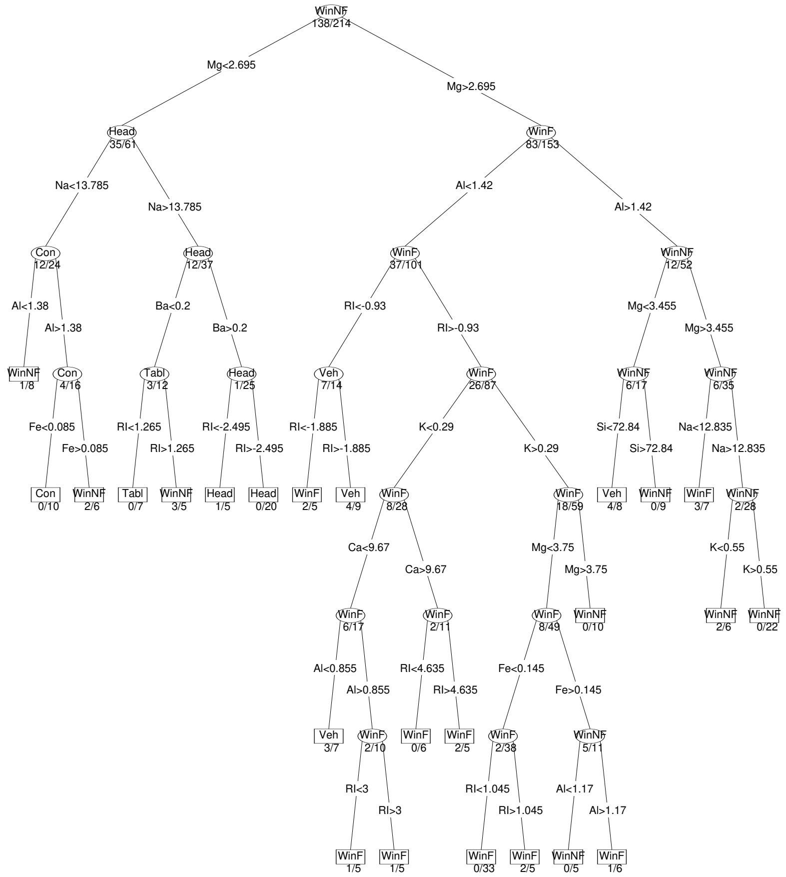

Figure 2.4: The full classification tree for the forensic glass data. Note that a number of final splits do not change the predicted class.

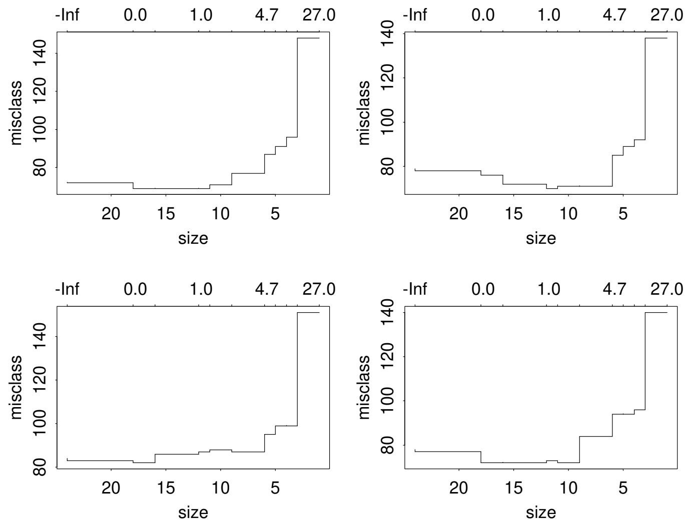

Figure 2.5: Four 10-fold cross-validation tests for choosing α (the top axis). The y-axis is the number of errors in the cross-validation.

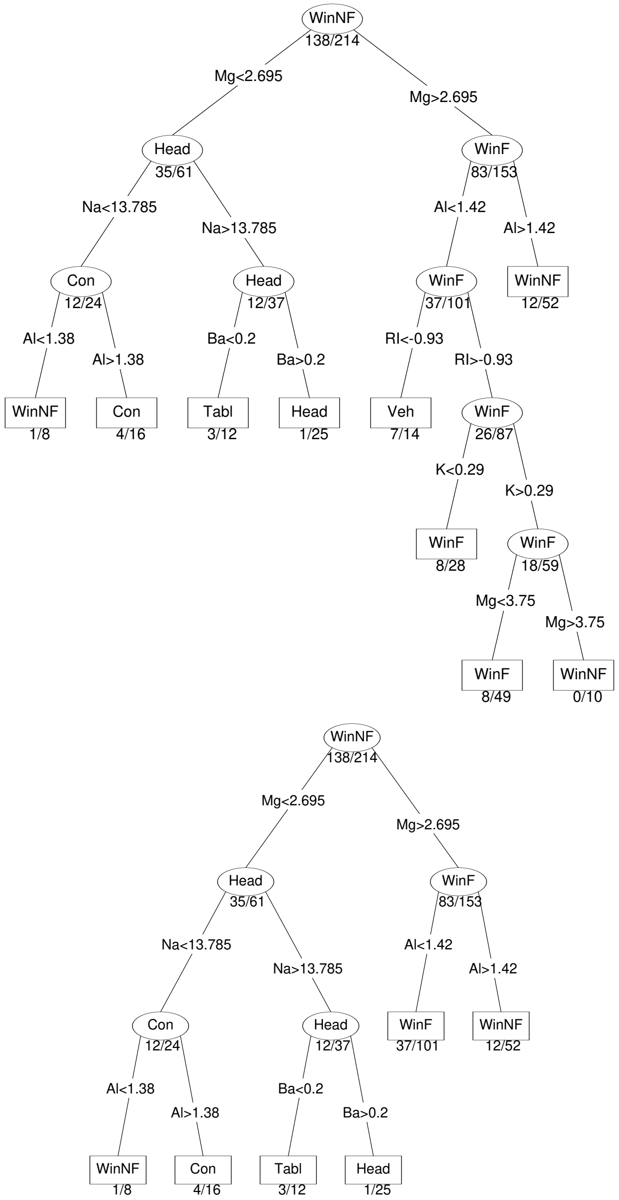

Figure 2.6: Prunings of figure 2.4 by $\alpha = 2$ (top) and $\alpha = 5$ (bottom).

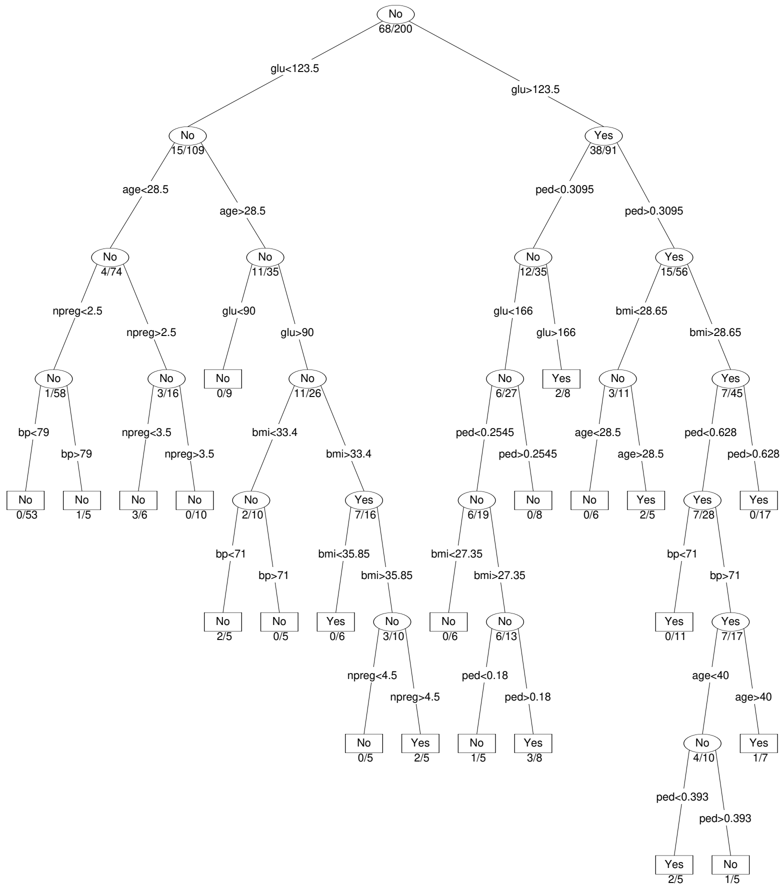

Figure 2.7: The full classification tree for the Pima Indians data.

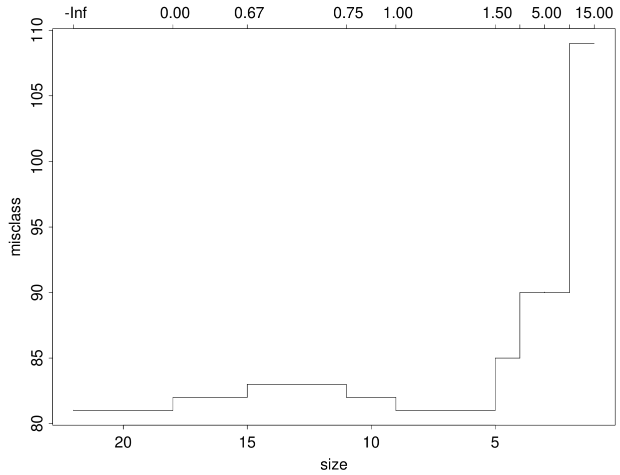

Figure 2.8: The prune sequence for figure 2.7 on the validation set.

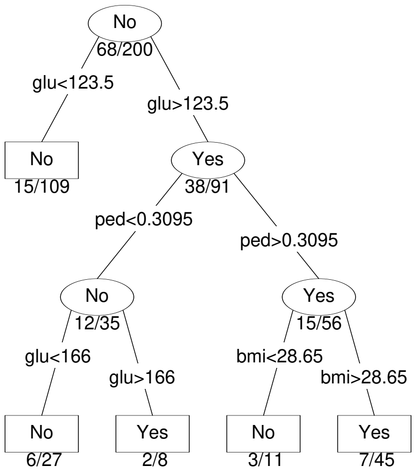

Figure 2.9: Figure 2.7 pruned at $\alpha = 1.5$.

2.2 Implementation in `rpart`

The simplest way to use tree-based methods is via the library section `rpart` by Terry Therneau and Beth Atkinson (Therneau & Atkinson, 1997). The underlying philosophy is of one function, `rpart`, that both grows and computes where to prune a tree; although there is a function `prune.rpart` it merely further prunes the tree at points already determined by the call to `rpart`, which has itself done some pruning. It is also possible to print a pruned tree by giving a pruning parameter to `print.rpart`. By default `rpart` runs a 10-fold cross-validation and the results are stored in the `rpart` object to allow the user to choose the degree of pruning at a later stage. Since all the work is done in a C function the calculations are quite fast.

The `rpart` system was designed to be easily extended to new types of responses. We only consider the following types, selected by the argument `method`.

`"anova"` A regression tree, with the impurity criterion the reduction in sum of squares on creating a binary split of the data at that node. The criterion $R(T)$ used for pruning is the mean square error of the predictions of the tree on the current dataset (that is, the residual mean square).

`"class"` A classification tree, with a categorical or factor response and default impurity criterion the Gini index. The deviance-based approach corresponds to the entropy index, selected by the argument setting `parms` to `list(split="information")`. The pruning criterion $R(T)$ is the predicted loss, normally the error rate.

If the `method` argument is missing an appropriate type is inferred from the response variable in the formula.

It is helpful to consider a few examples. First we consider a regression tree for our `cpus` data, then a classification tree for the `iris` data. The model is specified by a model formula with terms separated by `+`; interactions make no sense for trees, and `-` terms are ignored. The precise meaning of the argument `cp` is explained later; it is proportional to α in the cost-complexity measure.

```
> library(rpart)
> set.seed(123)
> cpus.rp <- rpart(log10(perf) ~ ., cpus[ , 2:8], cp = 1e-3)
> cpus.rp  # gives a large tree not show here.
> print(cpus.rp, cp = 0.01)
node), split, n, deviance, yval
      * denotes terminal node

 1) root 209 43.116000 1.7533
   2) cach<27 143 11.791000 1.5246
     4) mmax<6100 78  3.893700 1.3748
       8) mmax<1750 12  0.784250 1.0887 *
       9) mmax>=1750 66  1.948700 1.4268 *
     5) mmax>=6100 65  4.045200 1.7044
      10) syct>=360 7  0.129080 1.2797 *
```

```
        11) syct<360 58  2.501200 1.7557
          22) chmin<5.5 46  1.226200 1.6986 *
          23) chmin>=5.5 12  0.550710 1.9745 *
     3) cach>=27 66  7.642600 2.2488
       6) mmax<28000 41  2.341400 2.0620
        12) cach<96.5 34  1.592000 2.0081
          24) mmax<11240 14  0.424620 1.8266 *
          25) mmax>=11240 20  0.383400 2.1352 *
        13) cach>=96.5 7  0.171730 2.3236 *
       7) mmax>=28000 25  1.522900 2.5552
        14) cach<56 7  0.069294 2.2684 *
        15) cach>=56 18  0.653510 2.6668 *
```

This shows the predicted value (`yval`) and deviance within each node. We can plot the full tree by

```
> plot(cpus.rp, uniform = T); text(cpus.rp, digits = 3)
```

```
> ird <- data.frame(rbind(iris[,,1], iris[,,2],iris[,,3]),
         Species = c(rep("s",50), rep("c",50), rep("v",50)))
> ir.rp <- rpart(Species ~ ., data = ird, cp = 1e-3)
> ir.rp
node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 c (0.33333 0.33333 0.33333)
  2) Petal.L.>=2.45 100  50 c (0.50000 0.00000 0.50000)
    4) Petal.W.<1.75 54   5 c (0.90741 0.00000 0.09259) *
    5) Petal.W.>=1.75 46   1 v (0.02173 0.00000 0.97826) *
  3) Petal.L.<2.45 50   0 s (0.00000 1.00000 0.00000) *
```

The `(yprob)` give the distribution by class within each node.

Note that neither tree has yet been pruned to final size. We can now consider pruning by using `printcp` to print out the information stored in the `rpart` object.

```
> printcp(cpus.rp)

Regression tree:
rpart(formula = log10(perf) ~ ., data = cpus[, 2:8], cp = 0.001)

Variables actually used in tree construction:
[1] cach  chmax chmin mmax  syct

Root node error: 43.1/209 = 0.206

        CP nsplit rel error xerror   xstd
 1 0.54927      0     1.000  1.005 0.0972
 2 0.08934      1     0.451  0.480 0.0487
 3 0.08763      2     0.361  0.427 0.0433
 4 0.03282      3     0.274  0.322 0.0322
```



```
 5 0.02692      4     0.241  0.306 0.0306
 6 0.01856      5     0.214  0.278 0.0294
 7 0.01680      6     0.195  0.281 0.0292
 8 0.01579      7     0.179  0.279 0.0289
 9 0.00946      9     0.147  0.281 0.0322
10 0.00548     10     0.138  0.247 0.0289
11 0.00523     11     0.132  0.250 0.0289
12 0.00440     12     0.127  0.245 0.0287
13 0.00229     13     0.123  0.242 0.0284
14 0.00227     14     0.120  0.241 0.0282
15 0.00141     15     0.118  0.240 0.0282
16 0.00100     16     0.117  0.238 0.0279
```

The columns `xerror` and `xstd` are random, depending on the random partition used in the cross-validation. We can see the same output graphically (Figure 2.10) by a call to `plotcp`.

```
> plotcp(cpus.rp)
```

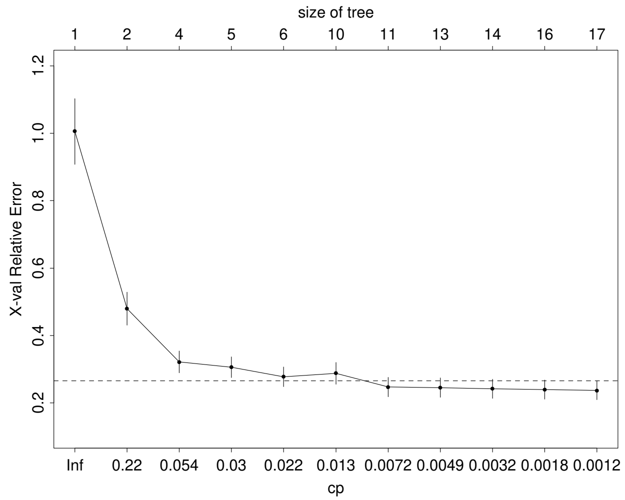

Figure 2.10: Plot by `plotcp` of the `rpart` object `cpus.rp1`.

We need to explain the *complexity parameter* `cp`; this is just the parameter α divided by the number $R(T_\emptyset)$ for the root tree.[1] A 10-fold cross-validation has been done within `rpart` to compute the entries[2] `xerror` and `xstd`; the complexity parameter may then be chosen to minimize `xerror`. An alternative procedure is to use the 1-SE rule, the largest value with `xerror` within one standard deviation of the minimum. In this case the 1-SE rule gives $0.238 +$

[1] Thus for most measures of fit the complexity parameter lies in $[0, 1]$.
[2] All the errors are scaled so the root tree has error $R(T_\emptyset)$ scaled to one.

0.0279 , so we choose line 7, a tree with 10 splits and hence 11 leaves.[3] We can examine this by

```
> print(cpus.rp, cp = 0.006, digits = 3)
node), split, n, deviance, yval
      * denotes terminal node

 1) root 209 43.1000 1.75
   2) cach<27 143 11.8000 1.52
     4) mmax<6.1e+03 78  3.8900 1.37
       8) mmax<1.75e+03 12  0.7840 1.09 *
       9) mmax>=1.75e+03 66  1.9500 1.43 *
     5) mmax>=6.1e+03 65  4.0500 1.70
      10) syct>=360 7  0.1290 1.28 *
      11) syct<360 58  2.5000 1.76
        22) chmin<5.5 46  1.2300 1.70
          44) cach<0.5 11  0.2020 1.53 *
          45) cach>=0.5 35  0.6160 1.75 *
        23) chmin>=5.5 12  0.5510 1.97 *
   3) cach>=27 66  7.6400 2.25
     6) mmax<2.8e+04 41  2.3400 2.06
      12) cach<96.5 34  1.5900 2.01
        24) mmax<1.12e+04 14  0.4250 1.83 *
        25) mmax>=1.12e+04 20  0.3830 2.14 *
      13) cach>=96.5 7  0.1720 2.32 *
     7) mmax>=2.8e+04 25  1.5200 2.56
      14) cach<56 7  0.0693 2.27 *
      15) cach>=56 18  0.6540 2.67 *
```

or

```
> cpus.rp1 <- prune(cpus.rp, cp = 0.006)
> plot(cpus.rp1, branch = 0.4, uniform = T)
> text(cpus.rp1, digits = 3)
```

The plot is shown in Figure 2.11.

For the `iris` data we have

```
> printcp(ir.rp)
    ....
Variables actually used in tree construction:
[1] Petal.L. Petal.W.

Root node error: 100/150 = 0.667

     CP nsplit rel error xerror   xstd
1 0.500      0      1.00   1.18 0.0502
2 0.440      1      0.50   0.60 0.0600
3 0.001      2      0.06   0.10 0.0306
```

[3] The number of leaves is always one more than the number of splits.

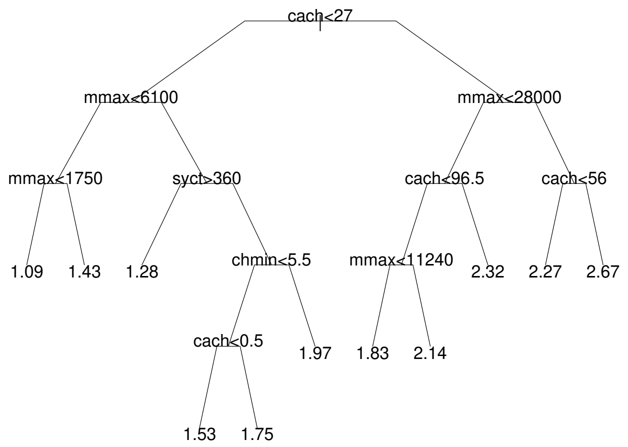

Figure 2.11: Plot of the `rpart` object `cpus.rp1`.

which suggests no pruning, but that too small a tree has been grown since `xerror` may not have reached its minimum.

The `summary` method, `summary.rpart`, produces voluminous output:

```
> summary(ir.rp)
Call:
rpart(formula = Species ~ ., data = ird, method = "class",
      cp = 0.001)

     CP nsplit rel error xerror     xstd
1 0.500      0      1.00   1.18 0.050173
2 0.440      1      0.50   0.60 0.060000
3 0.001      2      0.06   0.10 0.030551

Node number 1: 150 observations,    complexity param=0.5
  predicted class=c  expected loss=0.66667
    class counts:    50    50    50
   probabilities: 0.333 0.333 0.333
  left son=2 (100 obs) right son=3 (50 obs)
  Primary splits:
      Petal.L. < 2.45 to the right, improve=50.000, (0 missing)
      Petal.W. < 0.8  to the right, improve=50.000, (0 missing)
      Sepal.L. < 5.45 to the left,  improve=34.164, (0 missing)
      Sepal.W. < 3.35 to the left,  improve=19.039, (0 missing)
  Surrogate splits:
      Petal.W. < 0.8  to the right, agree=1.000, adj=1.00, (0 split)
      Sepal.L. < 5.45 to the right, agree=0.920, adj=0.76, (0 split)
      Sepal.W. < 3.35 to the left,  agree=0.833, adj=0.50, (0 split)
```

```
Node number 2: 100 observations,    complexity param=0.44
  predicted class=c  expected loss=0.5
    class counts:    50     0    50
   probabilities: 0.500 0.000 0.500
  left son=4 (54 obs) right son=5 (46 obs)
  Primary splits:
      Petal.W. < 1.75 to the left,  improve=38.9690, (0 missing)
      Petal.L. < 4.75 to the left,  improve=37.3540, (0 missing)
      Sepal.L. < 6.15 to the left,  improve=10.6870, (0 missing)
      Sepal.W. < 2.45 to the left,  improve= 3.5556, (0 missing)
  Surrogate splits:
      Petal.L. < 4.75 to the left,  agree=0.91, adj=0.804, (0 split)
      Sepal.L. < 6.15 to the left,  agree=0.73, adj=0.413, (0 split)
      Sepal.W. < 2.95 to the left,  agree=0.67, adj=0.283, (0 split)

Node number 3: 50 observations
  predicted class=s  expected loss=0
    class counts:     0    50     0
   probabilities: 0.000 1.000 0.000

Node number 4: 54 observations
  predicted class=c  expected loss=0.092593
    class counts:    49     0     5
   probabilities: 0.907 0.000 0.093

Node number 5: 46 observations
  predicted class=v  expected loss=0.021739
    class counts:     1     0    45
   probabilities: 0.022 0.000 0.978
```

The initial table is that given by `printcp`. The summary method gives the top few (default up to five) splits and their reduction in impurity, plus up to five surrogates, splits on other variables with a high agreement with the chosen split. In this case the limit on tree growth is the restriction on the size of child nodes (which by default must cover at least seven cases).

Two arguments to `summary.rpart` can help with the volume of output: as with `print.rpart` the argument `cp` effectively prunes the tree before analysis, and the argument `file` allows the output to be redirected to a file (via `sink`).

Forensic glass

For the forensic glass dataset `fgl` which has six classes we can use

```
> set.seed(123)
> fgl.rp <- rpart(type ~ ., fgl, cp = 0.001)
> plotcp(fgl.rp)
> printcp(fgl.rp)

Classification tree:
```



```
rpart(formula = type ~ ., data = fgl, cp = 0.001)

Variables actually used in tree construction:
[1] Al Ba Ca Fe Mg Na RI

Root node error: 138/214 = 0.645

      CP nsplit rel error xerror   xstd
1 0.2065      0     1.000  1.000 0.0507
2 0.0725      2     0.587  0.594 0.0515
3 0.0580      3     0.514  0.587 0.0514
4 0.0362      4     0.457  0.551 0.0507
5 0.0326      5     0.420  0.536 0.0504
6 0.0109      7     0.355  0.478 0.0490
7 0.0010      9     0.333  0.500 0.0495
```

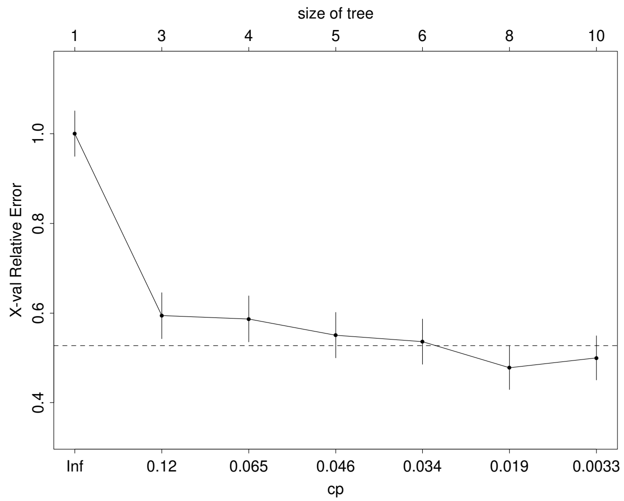

Figure 2.12: Plot by `plotcp` of the `rpart` object `fgl.rp`.

```
> print(fgl.rp, cp=0.02)
node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 214 138 WinNF (0.33 0.36 0.079 0.061 0.042 0.14)
   2) Ba<0.335 185 110 WinNF (0.37 0.41 0.092 0.065 0.049 0.016)
     4) Al<1.42 113  50 WinF (0.56 0.27 0.12 0.0088 0.027 0.018)
       8) Ca<10.48 101  38 WinF (0.62 0.21 0.13 0 0.02 0.02)
        16) RI>=-0.93 85  25 WinF (0.71 0.2 0.071 0 0.012 0.012)
          32) Mg<3.865 77  18 WinF (0.77 0.14 0.065 0 0.013 0.013) *
          33) Mg>=3.865 8   2 WinNF (0.12 0.75 0.12 0 0 0) *
```

```
        17) RI<-0.93 16   9 Veh (0.19 0.25 0.44 0 0.062 0.062) *
       9) Ca>=10.48 12   2 WinNF (0 0.83 0 0.083 0.083 0) *
     5) Al>=1.42 72  28 WinNF (0.083 0.61 0.056 0.15 0.083 0.014)
      10) Mg>=2.26 52  11 WinNF (0.12 0.79 0.077 0 0.019 0) *
      11) Mg<2.26 20   9 Con (0 0.15 0 0.55 0.25 0.05)
        22) Na<13.495 12   1 Con (0 0.083 0 0.92 0 0) *
        23) Na>=13.495 8   3 Tabl (0 0.25 0 0 0.62 0.12) *
   3) Ba>=0.335 29   3 Head (0.034 0.034 0 0.034 0 0.9) *
```

This suggests (Figure 2.12) a tree of size 8, plotted in Figure 2.13 by

```
> fgl.rp2 <- prune(fgl.rp, cp = 0.02)
> plot(fgl.rp2, uniform = T); text(fgl.rp2, use.n = T)
```

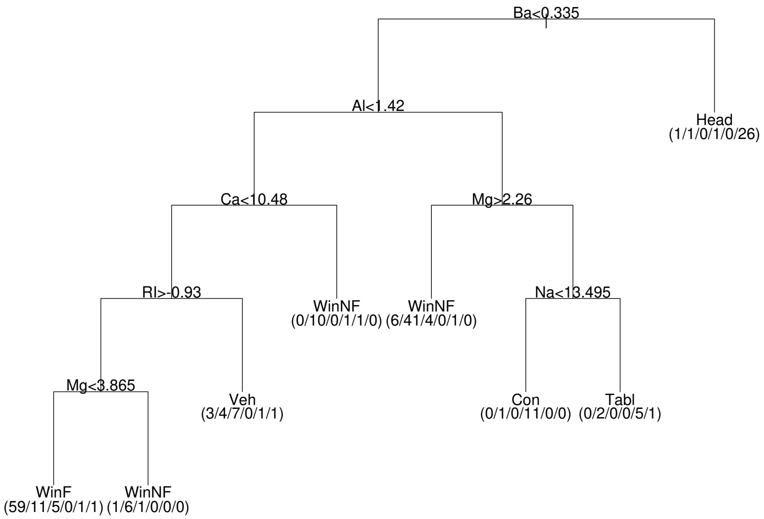

Figure 2.13: Plot of the `rpart` object `fgl.rp`.

Missing data

If the control parameter `maxsurrogate` is positive (without altering the parameter `usesurrogate`), the surrogates are used to handle missing cases both in training and in prediction (including cross-validation to choose the complexity). Each of the surrogate splits is examined in turn, and if the variable is available that split is used to decide whether to send the case left or right. If no surrogate is available or none can be used, the case is sent with the majority unless `usesurrogate` < 2 when it is left at the node.

The default `na.action` during training is `na.rpart`, which excludes cases only if the response or *all* the explanatory variables are missing. (This looks like a sub-optimal choice, as cases with missing response are useful for finding surrogate variables.)

When missing values are encountered in considering a split they are ignored and the probabilities and impurity measures are calculated from the non-missing values of that variable. Surrogate splits are then used to allocate the missing cases to the daughter nodes.

Surrogate splits are chosen to match as well as possible the primary split (viewed as a binary classification), and retained provided they send at least two cases down each branch, and agree as well as the rule of following the majority. The measure of agreement is the number of cases that are sent the same way, possibly after swapping 'left' and 'right' for the surrogate. (As far as we can tell, missing values on the surrogate are ignored, so this measure is biased towards surrogate variables with few missing values.)

Chapter 3

Neural Networks

Assertions are often made that neural networks provide a new approach to computing, involving analog (real-valued) rather than digital signals and massively parallel computation. For example, Haykin (1994, p. 2) offers a definition of a neural network adapted from Aleksander & Morton (1990):

> 'A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
>
> 1. Knowledge is acquired by the network through a learning process.
> 2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.'

In practice the vast majority of neural network applications are run on single-processor digital computers, although specialist parallel hardware is being developed (if not yet massively parallel). However, all the other methods we consider use real signals and can be parallelized to a considerable extent; it is far from clear that neural network methods will have an advantage as parallel computation becomes common, although they are frequently so slow that they need a speed-up.

The traditional methods of statistics and pattern recognition are either *parametric* based on a family of models with a small number of parameters, or *non-parametric* in which the models used are totally flexible. One of the impacts of neural network methods on pattern recognition has been to emphasize the need in large-scale practical problems for something in between, families of models with large but not unlimited flexibility given by a large number of parameters. The two most widely used neural network architectures, *multi-layer perceptrons* and *radial basis functions* (RBFs), provide two such families (and several others already existed in statistics).

Another difference in emphasis is on *'on-line'* methods, in which the data are not stored except through the changes the learning algorithm has made. The theory of such algorithms is studied for a very long stream of examples, but the practical distinction is less clear, as this stream is made up either by repeatedly cycling through the training set or by sampling the training examples (with replacement). In contrast, methods which use all the examples together are called *'batch'* methods. It is often forgotten that there are intermediate positions, such as using small batches chosen from the training set.

3.1 Feed-forward neural networks

Feed-forward neural networks provide a flexible way to generalize linear regression functions. General references are Bishop (1995); Hertz, Krogh & Palmer (1991) and Ripley (1993, 1996).

We start with the simplest but most common form with one hidden layer as shown in Figure 3.1. The input units just provide a 'fan-out' and distribute the inputs to the 'hidden' units in the second layer. These units sum their inputs, add a constant (the 'bias') and take a fixed function ϕ_h of the result. The output units are of the same form, but with output function ϕ_o. Thus

$$y_k = \phi_o\left(\alpha_k + \sum_h w_{hk}\,\phi_h\Big(\alpha_h + \sum_i w_{ih}\,x_i\Big)\right) \tag{3.1}$$

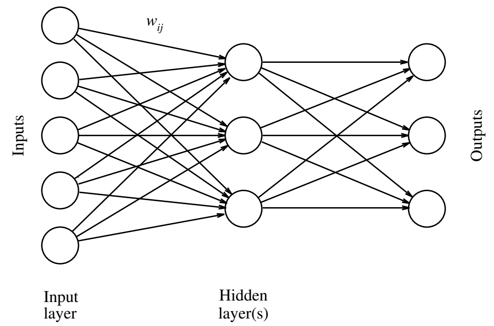

Figure 3.1: A generic feed-forward neural network.

The 'activation function' ϕ_h of the hidden layer units is almost always taken to be the logistic function

$$\ell(z) = \frac{\exp(z)}{1+\exp(z)}$$

and the output units are linear, logistic or threshold units. (The latter have $\phi_o(x) = I(x > 0)$.)

The general definition allows more than one hidden layer, and also allows 'skip-layer' connections from input to output when we have

$$y_k = \phi_o\left(\alpha_k + \sum_{i\to k} w_{ik}x_i + \sum_{j\to k} w_{jk}\phi_h\Big(\alpha_j + \sum_{i\to j} w_{ij}x_i\Big)\right) \tag{3.2}$$

which allows the non-linear units to perturb a linear functional form.

We can eliminate the biases α_i by introducing an input unit 0 which is permanently at $+1$ and feeds every other unit. The regression function f is then parametrized by the set of weights w_{ij}, one for every link in the network (or zero for links which are absent).

The original biological motivation for such networks stems from McCulloch & Pitts (1943) who published a seminal model of a neuron as a binary thresholding device in discrete time, specifically that

$$n_i(t) = H\Big(\sum_{j\to i} w_{ji} n_j(t-1) - \theta_i\Big)$$

the sum being over neurons j connected to neuron i. Here H denotes the Heaviside or threshold function $H(x) = I(x > 0)$, $n_i(t)$ is the output of neuron i at time t, and $0 < w_{ij} < 1$ are attenuation weights. Thus the effect is to threshold a weighted sum of the inputs at value θ_i. Real neurons are now known to be more complicated; they have a graded response rather than the simple thresholding of the McCulloch–Pitts model, work in continuous time, and can perform more general non-linear functions of their inputs, for example logical functions. Nevertheless, the McCulloch–Pitts model has been extremely influential in the development of artificial neural networks.

Feed-forward neural networks can equally be seen as a way to parametrize a fairly general non-linear function. Such networks *are* rather general: Cybenko (1989), Funahashi (1989), Hornik, Stinchcombe & White (1989) and later authors have shown that neural networks with linear output units can approximate any continuous function f uniformly on compact sets, by increasing the size of the hidden layer.

The approximation results are non-constructive, and in practice the weights have to be chosen to minimize some fitting criterion, for example least squares

$$E = \sum_p \|t^p - y^p\|^2$$

where t^p is the target and y^p the output for the pth example pattern. Other measures have been proposed, including for $y \in [0,1]$ 'maximum likelihood' (in fact minus the logarithm of a conditional likelihood) or equivalently the Kullback–Leibler distance, which amount to minimizing

$$E = \sum_p \sum_k \left[t_k^p \log \frac{t_k^p}{y_k^p} + (1 - t_k^p) \log \frac{1 - t_k^p}{1 - y_k^p} \right] \tag{3.3}$$

This is half the deviance for a logistic model with linear predictor given by (3.1) or (3.2).

One way to ensure that f is smooth is to restrict the class of estimates, for example by using a limited number of spline knots. Another way is *regularization* in which the fit criterion is altered to

$$E + \lambda C(f)$$

with a penalty C on the 'roughness' of f. *Weight decay*, specific to neural networks, uses as penalty the sum of squares of the weights w_{ij}. (This only makes sense if the inputs are rescaled to range about $[0,1]$ to be comparable with the outputs of internal units.) The use of weight decay seems both to help the optimization process and to avoid over-fitting. Arguments in Ripley (1993, 1994a) based on a Bayesian interpretation suggest $\lambda \approx 10^{-4} - 10^{-2}$ depending on the degree of fit expected, for least-squares fitting to variables of range one and $\lambda \approx 0.01 - 0.1$ for the entropy fit.

Regression examples

Figure 3.2 showed a simulation of an example taken from Wahba & Wold (1975) where it illustrates smoothing splines. From now on we consider just 100 points and assume that the true noise variance, $\sigma^2 = (0.2)^2$, is known.

Model selection amounts to choosing the number of hidden units. Figure 3.4 shows neural-net fitting with 8 hidden units treated as a non-linear regression problem, with the standard confidence limits produced by linearization (Bates & Watts, 1988). Figure 3.5 shows that there are quite a number of different local minima; all these solutions fit about equally well.

We clearly have over-fitting; there are many ways to avoid this, but weight decay (figure 3.6) is perhaps the simplest.

We now have two ways to control the fit, the number of hidden units and λ, the degree of regularization. The true curve is not exactly representable by our model, so we have some bias (figure 3.7). Choosing fewer hidden units leads to more bias, as does adding a regularizer. But it also reduces their variability, so reduces the mean-square error. Note that the error bands in figure 3.4 are smaller than those in figure 3.6; this is misleading as the conventional local linearization used for figure 3.4 is not sufficiently accurate.

Software

Software to fit feed-forward neural networks with a single hidden layer but allowing skip-layer connections (as in (3.2)) is provided in my library `nnet`. The format of the call is to the fitting function `nnet` is

```
nnet(formula, data, weights, size, Wts, linout=F, entropy=F,
   softmax=F, skip=F, rang=0.7, decay=0, maxit=100, trace=T)
```

The non-standard arguments are

`size`	number of units in the hidden layer.
`Wts`	optional initial vector for w_{ij}.
`linout`	logical for linear output units.
`entropy`	logical for entropy rather than least-squares fit.
`softmax`	logical for log-probability models.
`skip`	logical for links from inputs to outputs.
`rang`	if `Wts` is missing, use random weights from `runif(n,-rang, rang)`.
`decay`	parameter λ.

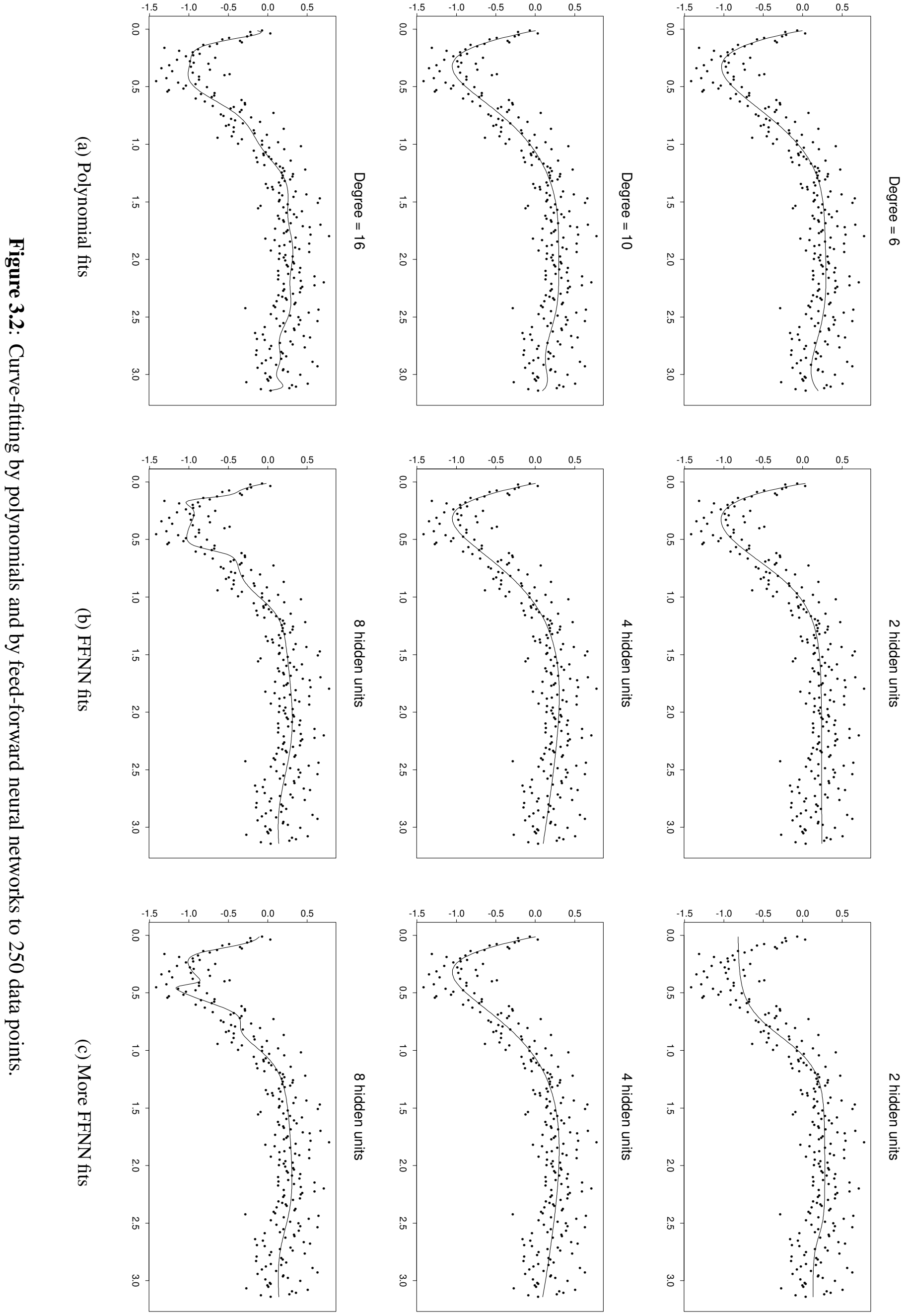

Figure 3.2: Curve-fitting by polynomials and by feed-forward neural networks to 250 data points.

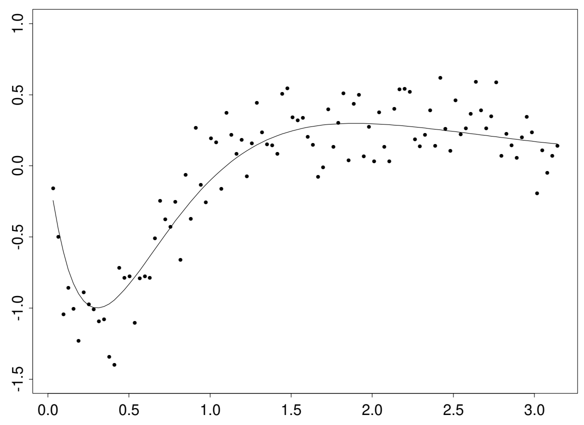

Figure 3.3: 100 data points from a curve-fitting problem, with the true curve.

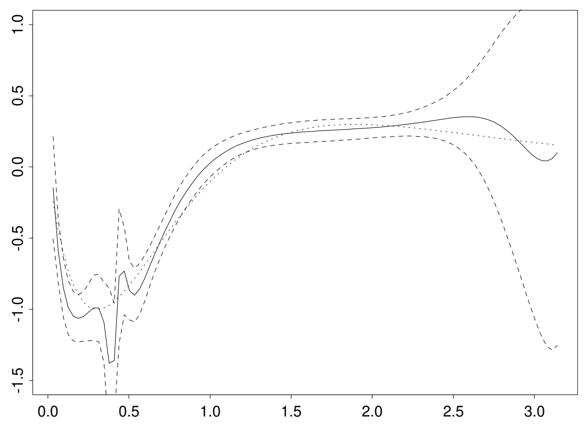

Figure 3.4: Curve fitted by 1-8-1 neural network, with ± 2 standard error bands (computed as a non-linear regression problem) and true curve (dotted).

`maxit`	maximum of iterations for the optimizer.
`Hess`	should be Hessian matrix at the solution be returned?
`trace`	logical for output from the optimizer. Very reassuring!

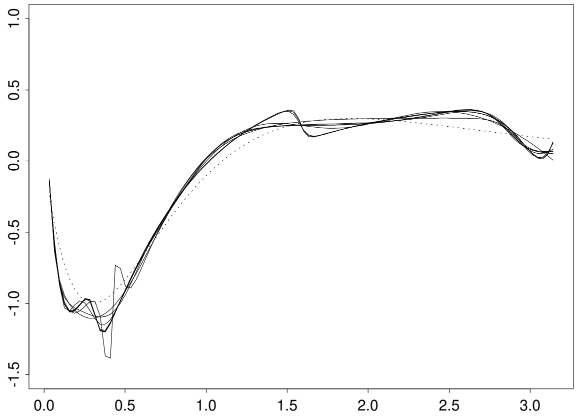

Figure 3.5: 1-8-1 neural networks with 25 different sets of starting values for the optimization.

There are `predict`, `print` and `summary` methods for neural networks, and a function `nnet.Hess` to compute the Hessian with respect to the weight parameters and so check if a secure local minimum has been found. For our `rock` example we have

```
> attach(rock)
> area1 <- area/10000; peri1 <- peri/10000
> rock1 <- data.frame(perm, area=area1, peri=peri1, shape)
> rock.nn <- nnet(log(perm) ~ area + peri + shape, data=rock1,
     size=3, decay=1e-3, linout=T, skip=T, maxit=1000, Hess=T)
# weights:  19
initial  value 1092.816748
iter  10 value 32.272454
    ....
final  value 14.069537
converged
> summary(rock.nn)
a 3-3-1 network with 19 weights
options were - skip-layer connections  linear output units
  decay=0.001
  b->h1 i1->h1 i2->h1 i3->h1
   1.21   8.74 -15.00  -3.45
  b->h2 i1->h2 i2->h2 i3->h2
   9.50  -4.34 -12.66   2.48
  b->h3 i1->h3 i2->h3 i3->h3
   6.20  -7.63 -10.97   3.12
   b->o  h1->o  h2->o  h3->o  i1->o  i2->o  i3->o
```


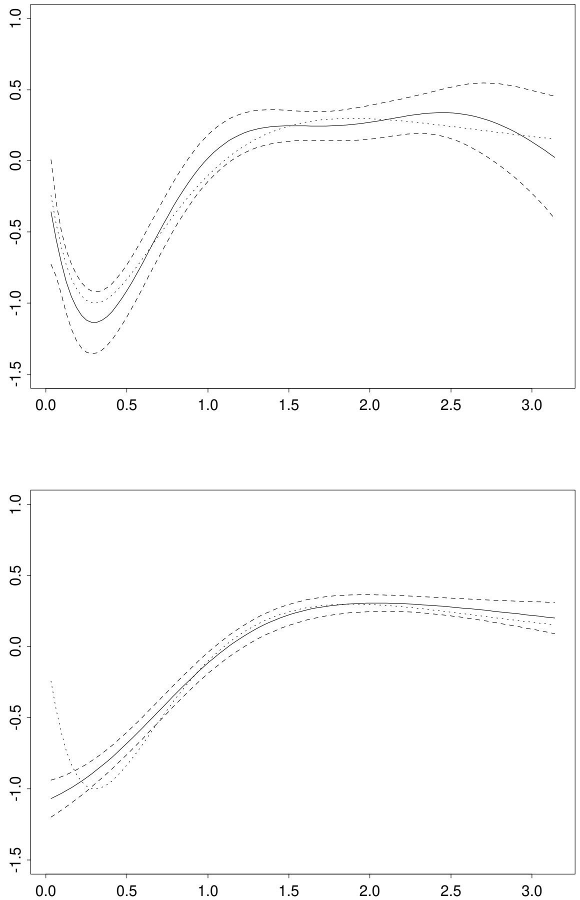

Figure 3.6: The effect of adding weight decay. Curves fitted by 1-8-1 neural network, with ± 2 standard error bands and true curve $\lambda = 10^{-3}$ (top) and $\lambda = 0.1$ (bottom).

```
   7.74  20.17  -7.68  -7.02 -12.95 -15.19   6.92
> sum((log(perm) - predict(rock.nn))^2)
[1] 12.231
> detach(rock)
> eigen(rock.nn@Hess, T)$values  #  $ in S+2000
 [1] 9.1533e+02 1.6346e+02 1.3521e+02 3.0368e+01 7.3914e+00
 [6] 3.4012e+00 2.2879e+00 1.0917e+00 3.9823e-01 2.7867e-01
```

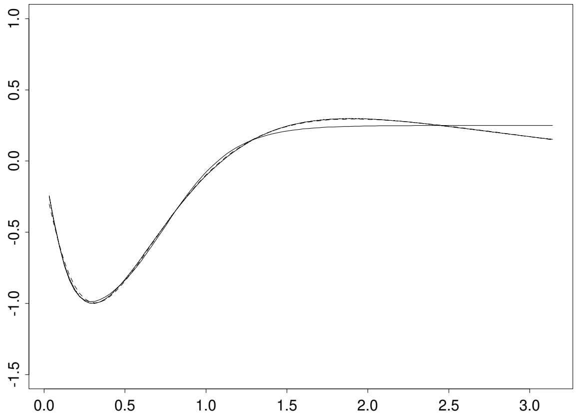

Figure 3.7: Fits to the *true curve* with 2 and 3 hidden units, and 8 units with $\lambda = 10^{-3}$.

```
[11] 1.9953e-01 7.5159e-02 3.2513e-02 2.5950e-02 1.9077e-02
[16] 1.0834e-02 6.8937e-03 3.7671e-03 2.6974e-03
```

(There are several solutions and a random starting point, so your results may well differ.) The quoted values include the weight decay term. The eigenvalues of the Hessian suggest that a secure local minimum has been achieved. In the summary the `b` refers to the 'bias' unit (input unit 0), and `i`, `h` and `o` to input, hidden and bias units.

To view the fitted surface for the `rock` dataset we can use

```
Xp <- expand.grid(area=seq(0.1,1.2,0.05),
                  peri=seq(0,0.5,0.02), shape=0.2)
rock.grid <- cbind(Xp,fit=predict(rock.nn, Xp))
trellis.device()
wireframe(fit ~ area + peri, rock.grid, screen=list(z=160,x=-60),
          aspect=c(1,0.5), drape=T)
```

An example: the `cpus` data

To use the `nnet` software effectively it is essential to scale the problem. A preliminary run with a linear model demonstrates that we get essentially the same results as the conventional approach to linear models.

```
cpus0 <- cpus[, 2:8]
for(i in 1:3) cpus0[,i] <- log10(cpus0[,i])
set.seed(123); samp <- sample(1:209, 100)
attach(cpus0)
```



```
cpus1 <- data.frame(syct=syct-2, mmin=mmin-3, mmax=mmax-4,
cach=cach/256, chmin=chmin/100, chmax=chmax/100, perf=perf)
detach()

test <- function(fit)
  sqrt(sum((log10(cpus1[-samp, "perf"]) -
          predict(fit, cpus1[-samp,]))^2)/109)
cpus.nn1 <- nnet(log10(perf) ~ ., data=cpus1[samp,], linout=T,
                 skip=T, size=0)
test(cpus.nn1)
[1] 0.21295
```

We now consider adding non-linear terms to the model.

```
cpus.nn2 <- nnet(log10(perf) ~ ., data=cpus1[samp,], linout=T,
                 skip=T, size=4, decay=0.01, maxit=1000)
final  value 2.369581
test(cpus.nn2)
[1] 0.21132
cpus.nn3 <- nnet(log10(perf) ~ ., data=cpus1[samp,], linout=T,
                 skip=T, size=10, decay=0.01, maxit=1000)
final  value 2.338387
test(cpus.nn3)
[1] 0.21068
cpus.nn4 <- nnet(log10(perf) ~ ., data=cpus1[samp,], linout=T,
                 skip=T, size=25, decay=0.01, maxit=1000)
final  value 2.339850
test(cpus.nn4)
[1] 0.23
```

This demonstrates that the degree of fit is almost completely controlled by the amount of weight decay rather than the number of hidden units (provided there are sufficient). We have to be able to choose the amount of weight decay *without* looking at the test set. To do so we use cross-validation and by averaging across multiple fits (see later).

```
CVnn.cpus <- function(formula, data=cpus1[samp, ],
     size = c(0, 4, 4, 10, 10),
     lambda = c(0, rep(c(0.003, 0.01), 2)),
     nreps = 5, nifold = 10, ...)
{
  CVnn1 <- function(formula, data, nreps=1, ri,  ...)
  {
    truth <- log10(data$perf)
    res <- numeric(length(truth))
    cat("  fold")
    for (i in sort(unique(ri))) {
      cat(" ", i,  sep="")
      for(rep in 1:nreps) {
        learn <- nnet(formula, data[ri !=i,], trace=F, ...)
        res[ri == i] <- res[ri == i] +
```

```
                              predict(learn, data[ri == i,])
            }
          }
          cat("\n")
          sum((truth - res/nreps)^2)
        }
        choice <- numeric(length(lambda))
        ri <- sample(nifold, nrow(data), replace=T)
        for(j in seq(along=lambda)) {
          cat("  size =", size[j], "decay =", lambda[j], "\n")
          choice[j] <- CVnn1(formula, data, nreps=nreps, ri=ri,
                             size=size[j], decay=lambda[j], ...)
          }
        cbind(size=size, decay=lambda, fit=sqrt(choice/100))
      }
      CVnn.cpus(log10(perf) ~ ., data=cpus1[samp,],
                linout=T, skip=T, maxit=1000)
           size decay     fit
      [1,]    0 0.000 0.19746
      [2,]    4 0.003 0.23297
      [3,]    4 0.010 0.20404
      [4,]   10 0.003 0.22803
      [5,]   10 0.010 0.20130
```

This took around 6 Mb and 15 minutes on the PC. The cross-validated results seem rather insensitive to the choice of model. The non-linearity does not seem justified.

3.2 Multiple logistic regression and discrimination

The function `multinom` is a wrapper function that uses `nnet` to fit a multiple logistic regression.

The model is specified by a formula. The response can be either a matrix giving the number of occurrences of each class at that particular `x` value, or (more commonly) a factor giving the observed class. The right-hand side specifies the design matrix in the usual way. If the response Y is a factor with just two levels, the model fitted is

$$\operatorname{logit} p(Y = 1 \mid X = \boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{x}$$

This is a logistic regression, and is fitted as a neural network with skip-layer connections and no units in the hidden layer. There is a potential problem in that both the bias unit and an intercept in $\boldsymbol{x}$ may provide an intercept term: this is avoided by constraining the bias coefficient to be zero. The entropy measure of fit is used; this is equivalent to maximizing the likelihood.

For a factor response with more that two levels or a matrix response the model fitted is

$$\log \frac{p(Y = c \mid X = \boldsymbol{x})}{p(Y = 1 \mid X = \boldsymbol{x})} = \boldsymbol{\beta}_c^T \boldsymbol{x} \tag{3.4}$$

where $\boldsymbol{\beta}_1 \equiv 0$. Once again the parameters are chosen by maximum likelihood. Approximate standard errors of the coefficients are found for `vcov.multinom` and `summary.multinom` by inverting the Hessian of the (negative) log-likelihood at the maximum likelihood estimator.

It is possible to add weight decay by setting a non-zero value for `decay` on the call to `multinom`. Beware that because the coefficients for class one are constrained to be zero, this has a rather asymmetric effect and that the quoted standard errors are no longer appropriate. Using weight decay has an effect closely analogous to ridge regression, and will often produce better predictions than using stepwise selection of the variables.

In all these problems the measure of fit is convex, so there is a unique global minimum. This is attained at a single point unless there is collinearity in the explanatory variables or the minimum occurs at infinity (which can occur if the classes are partially or completely linearly separable).

3.3 Neural networks in classification

A simple example

We will illustrate these methods by a small example taken from Aitchison & Dunsmore (1975, Tables 11.1–3) and used for the same purpose by Ripley (1996). The data are on diagnostic tests on patients with Cushing's syndrome, a hypersensitive disorder associated with over-secretion of cortisol by the adrenal gland. This dataset has three recognized types of the syndrome represented as a, b, c. (These encode 'adenoma', 'bilateral hyperplasia' and 'carcinoma', and represent the underlying cause of over-secretion. This can only be determined histopathologically.) The observations are urinary excretion rates (mg/24h) of the steroid metabolites tetrahydrocortisone and pregnanetriol, and are considered on log scale.

There are six patients of unknown type (marked u), one of whom was later found to be of a fourth type, and another was measured faultily.

Figure 3.8 shows the classifications produced by `lda` and the various options of quadratic discriminant analysis. This was produced by

```
predplot <- function(object, main="", len=100, ...)
{
   plot(Cushings[,1], Cushings[,2], log="xy", type="n",
     xlab="Tetrahydrocortisone", ylab = "Pregnanetriol", main)
   text(Cushings[1:21,1], Cushings[1:21,2],
      as.character(tp))
   text(Cushings[22:27,1], Cushings[22:27,2], "u")
   xp <- seq(0.6, 4.0, length=len)
   yp <- seq(-3.25, 2.45, length=len)
   cushT <- expand.grid(Tetrahydrocortisone=xp,
     Pregnanetriol=yp)
   Z <- predict(object, cushT, ...); zp <- unclass(Z$class)
   zp <- Z$post[,3] - pmax(Z$post[,2], Z$post[,1])
```

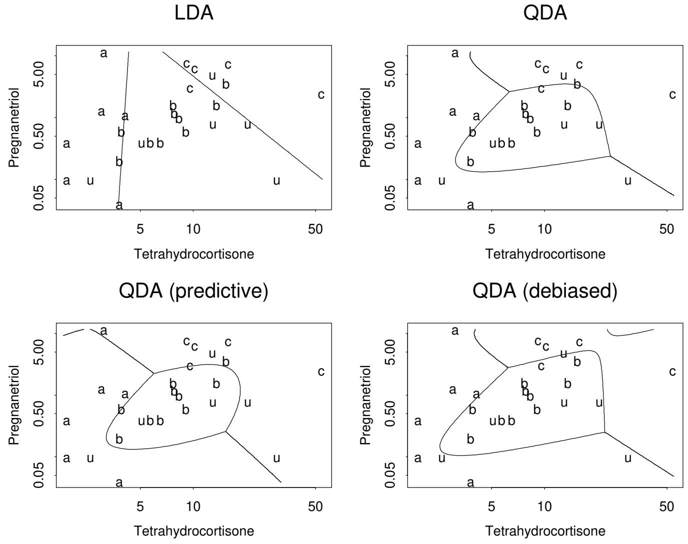

Figure 3.8: Linear and quadratic discriminant analysis applied to the Cushing's syndrome data.

```
    contour(xp/log(10), yp/log(10), matrix(zp, len),
      add=T, levels=0, labex=0)
    zp <- Z$post[,1] - pmax(Z$post[,2], Z$post[,3])
    contour(xp/log(10), yp/log(10), matrix(zp, len),
      add=T, levels=0, labex=0)
    invisible()
}
cush <- log(as.matrix(Cushings[, -3]))
tp <- factor(Cushings$Type[1:21])
cush.lda <- lda(cush[1:21,], tp); predplot(cush.lda, "LDA")
cush.qda <- qda(cush[1:21,], tp); predplot(cush.qda, "QDA")
```

We can contrast these with logistic discrimination performed by

```
library(nnet)
Cf <- data.frame(tp = tp,
   Tetrahydrocortisone = log(Cushings[1:21,1]),
   Pregnanetriol = log(Cushings[1:21,2]) )
cush.multinom <- multinom(tp ~ Tetrahydrocortisone
   + Pregnanetriol, Cf, maxit=250)
xp <- seq(0.6, 4.0, length=100); np <- length(xp)
yp <- seq(-3.25, 2.45, length=100)
cushT <- expand.grid(Tetrahydrocortisone=xp,
     Pregnanetriol=yp)
Z <- predict(cush.multinom, cushT, type="probs")
```



```
plot(Cushings[,1], Cushings[,2], log="xy", type="n",
  xlab="Tetrahydrocortisone", ylab = "Pregnanetriol")
text(Cushings[1:21,1], Cushings[1:21,2],
     labels = as.character(tp))
text(Cushings[22:27,1], Cushings[22:27,2], labels = "u")
zp <- Z[,3] - pmax(Z[,2], Z[,1])
contour(xp/log(10), yp/log(10), matrix(zp, np),
   add=T, levels=0, labex=0)
zp <- Z[,1] - pmax(Z[,2], Z[,3])
contour(xp/log(10), yp/log(10), matrix(zp, np),
   add=T, levels=0, labex=0)
```

When, as here, the classes have quite different variance matrices, linear and logistic discrimination can give quite different answers (compare Figures 3.8 and 3.9).

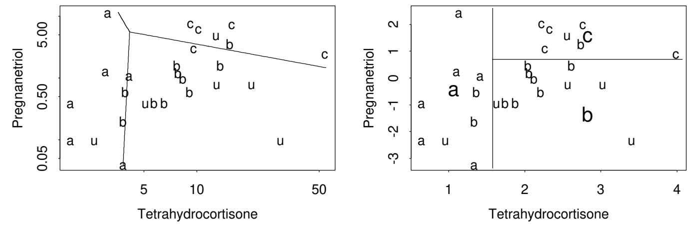

Figure 3.9: Logistic regression and classification trees applied to the Cushing's syndrome data. The classification trees are discussed on page 71.

Neural networks provide a flexible non-linear extension of multiple logistic regression, as we saw in Section 3.1. We can consider them for this example by the following code.

```
library(nnet)
cush <- cush[1:21,]; tpi <- class.ind(tp)
par(mfrow=c(2,2))
pltnn(main = "Size = 2")
set.seed(1); plt.bndry(size=2, col=2)
set.seed(3); plt.bndry(size=2, col=3); plt.bndry(size=2, col=4)

pltnn(main = "Size = 2, lambda = 0.001")
set.seed(1); plt.bndry(size=2, decay=0.001, col=2)
set.seed(2); plt.bndry(size=0, decay=0.001, col=4)

pltnn(main = "Size = 2, lambda = 0.01")
set.seed(1); plt.bndry(size=2, decay=0.01, col=2)
set.seed(2); plt.bndry(size=2, decay=0.01, col=4)

pltnn(main = "Size = 5, 20  lambda = 0.01")
set.seed(2); plt.bndry(size=5, decay=0.01, col=1)
set.seed(2); plt.bndry(size=20, decay=0.01, col=2)
```

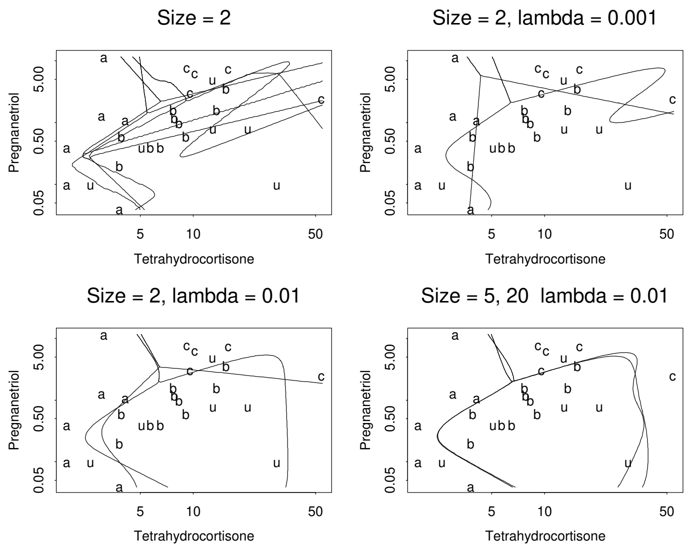

Figure 3.10: Neural networks applied to the Cushing's syndrome data. Each panel shows the fits from two or three local maxima of the (penalized) log-likelihood.

The results are shown in Figure 3.10. We see that in all cases there are multiple local maxima of the likelihood.

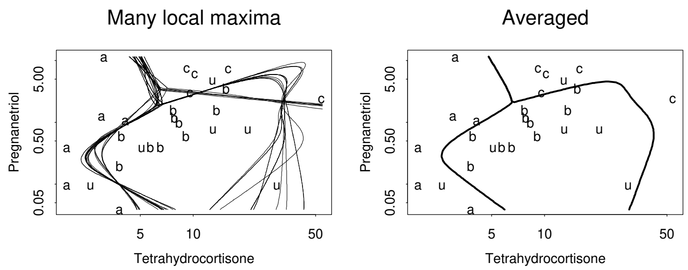

Figure 3.11: Neural networks applied to the Cushing's syndrome data. The right panel shows many local minima and their average (as the thick line).

Once we have a penalty, the choice of the number of hidden units is often not critical (see Figure 3.10). The spirit of the predictive approach is to average the predicted $p(c \mid \boldsymbol{x})$ over the local maxima. A simple average will often suffice:

```
# functions pltnn and b1 are in the scripts
pltnn("Many local maxima")
Z <- matrix(0, nrow(cushT), ncol(tpi))
```



```
for(iter in 1:20) {
    set.seed(iter)
    cush.nn <- nnet(cush, tpi,  skip=T, softmax=T, size=3,
        decay=0.01, maxit=1000, trace=F)
    Z <- Z + predict(cush.nn, cushT)
    # change @ to $ for S+2000
    cat("final value", format(round(cush.nn@value,3)), "\n")
    b1(predict(cush.nn, cushT), col=2, lwd=0.5)
}
pltnn(main = "Averaged")
b1(Z, lwd=3)
```

but more sophisticated integration can be done (Ripley, 1996, §5.5). Note that there are two quite different types of local maxima occurring here, and some local maxima occur several times (up to convergence tolerances).

Forensic glass

The forensic glass dataset `fgl` has 214 points from six classes with nine measurements, and provides a fairly stiff test of classification methods. The types of glass do not form compact well-separated groupings, and the marginal distributions are far from normal. There are some small classes (with 9, 13 and 17 examples), so we cannot use quadratic discriminant analysis.

We will assess their performance by 10-fold cross-validation, using the same random partition for all the methods. Logistic regression provides a suitable benchmark (as is often the case).

```
set.seed(123); rand <- sample (10, 214, replace=T)
con<- function(x,y)
{
   tab <- table(x,y)
   print(tab)
   diag(tab) <- 0
   cat("error rate = ", round(100*sum(tab)/length(x),2),"%\n")
   invisible()
}
CVtest <- function(fitfn, predfn, ...)
{
  res <- fgl$type
  for (i in sort(unique(rand))) {
     cat("fold ",i,"\n", sep="")
     learn <- fitfn(rand != i, ...)
     res[rand == i] <- predfn(learn, rand==i)
  }
  res
}
res.multinom <- CVtest(
   function(x, ...) multinom(type ~ ., fgl[x,], ...),
   function(obj, x) predict(obj, fgl[x, ],type="class"),
```

```
   maxit=1000, trace=F )

> con(fgl$type, res.multinom)
      WinF WinNF Veh Con Tabl Head
 WinF   44    20   4   0    2    0
WinNF   20    50   0   3    2    1
  Veh    9     5   3   0    0    0
  Con    0     4   0   8    0    1
 Tabl    0     2   0   0    4    3
 Head    1     1   0   3    1   23
error rate =  38.79 %
```

We will write some general functions for testing neural network models by V -fold cross-validation. First we re-scale the dataset so the inputs have range $[0,1]$.

```
fgl1 <- lapply(fgl[, 1:9], function(x)
               {r <- range(x); (x-r[1])/diff(r)})
fgl1 <- data.frame(fgl1, type=fgl$type)
```

Then we can experiment with multiple logistic regressions.

```
res.multinom <- CVtest(
   function(x, ...) multinom(type ~ ., fgl1[x,], ...),
   function(obj, x) predict(obj, fgl1[x, ],type="class"),
   maxit=1000, trace=F)
con(fgl$type, res.multinom)

res.mult2 <- CVtest(
   function(x, ...) multinom(type ~ ., fgl1[x,], ...),
   function(obj, x) predict(obj, fgl1[x, ], type="class"),
   maxit=1000, trace=F, decay=1e-3)
> con(fgl$type, res.mult2)
    ....
error rate =  36.45 %
```

It is straightforward to fit a fully specified neural network in the same way. We will, however, want to average across several fits and to choose the number of hidden units and the amount of weight decay by an inner cross-validation. To do so we wrote fairly general function that can easily be used or modified to suit other problems.

```
CVnn <- function(nreps=1, ...)
{
  res <-  matrix(0, 214, 6)
  dimnames(res) <- list(NULL, levels(fgl$type))
  for (i in sort(unique(rand))) {
    cat("fold ",i,"\n", sep="")
    for(rep in 1:nreps) {
      learn <- nnet(type ~ ., fgl1[rand !=i,], trace=F, ...)
      res[rand == i,] <- res[rand == i,] +
        predict(learn, fgl1[rand==i,])
```



```
    }
  }
  max.col(res/nreps)
}
> res.nn <- CVnn(maxit=1000, size=6, decay=0.01)
> con(fgl$type, res.nn)
    ....
error rate =  29.44 %

CVnn2 <- function(formula, data,
                  size = rep(6,2), lambda = c(0.001, 0.01),
                  nreps = 1, nifold = 5, verbose = 99, ...)
{
  CVnn1 <- function(formula, data, nreps=1, ri, verbose,  ...)
  {
    truth <- data[,deparse(formula[[2]])]
    res <-  matrix(0, nrow(data), length(levels(truth)))
    if(verbose > 20) cat("  inner fold")
    for (i in sort(unique(ri))) {
      if(verbose > 20) cat(" ", i,  sep="")
      for(rep in 1:nreps) {
        learn <- nnet(formula, data[ri !=i,], trace=F, ...)
        res[ri == i,] <- res[ri == i,] +
          predict(learn, data[ri == i,])
      }
   }
    if(verbose > 20) cat("\n")
    sum(unclass(truth) != max.col(res/nreps))
  }
  truth <- data[,deparse(formula[[2]])]
  res <-  matrix(0, nrow(data), length(levels(truth)))
  choice <- numeric(length(lambda))
  for (i in sort(unique(rand))) {
    if(verbose > 0) cat("fold ", i,"\n", sep="")
    ri <- sample(nifold, sum(rand!=i), replace=T)
    for(j in seq(along=lambda)) {
      if(verbose > 10)
        cat("  size =", size[j], "decay =", lambda[j], "\n")
      choice[j] <- CVnn1(formula, data[rand != i,], nreps=nreps,
                         ri=ri, size=size[j], decay=lambda[j],
                         verbose=verbose, ...)
    }
    decay <- lambda[which.is.max(-choice)]
    csize <- size[which.is.max(-choice)]
    if(verbose > 5) cat("  #errors:", choice, "  ")
    if(verbose > 1) cat("chosen size = ", csize,
                        " decay = ", decay, "\n", sep="")
    for(rep in 1:nreps) {
      learn <- nnet(formula, data[rand != i,], trace=F,
                    size=csize, decay=decay, ...)
```

```
      res[rand == i,] <- res[rand == i,] +
          predict(learn, data[rand == i,])
    }
  }
  factor(levels(truth)[max.col(res/nreps)],
         levels = levels(truth))
}
> res.nn2 <- CVnn2(type ~ ., fgl1, skip=T, maxit=500, nreps=10)
> con(fgl$type, res.nn2)
       WinF WinNF Veh Con Tabl Head
 WinF    57    10   3   0    0    0
WinNF    16    51   3   4    2    0
  Veh     8     3   6   0    0    0
  Con     0     3   0   9    0    1
 Tabl     0     1   0   1    5    2
 Head     0     3   0   1    0   25
error rate =  28.5 %
```

This fits a neural network 1000 times, and so is fairly slow (hours).

This code chooses between neural nets on the basis of their cross-validated error rate. An alternative is to use logarithmic scoring, which is equivalent to finding the deviance on the validation set. Rather than count 0 if the predicted class is correct and 1 otherwise, we count $-\log p(c \,|\, x)$ for the true class c. We can easily code this variant by replacing the line

```
sum(unclass(truth) != max.col(res/nreps))
```

by

```
sum(-log(res[cbind(seq(along=truth),unclass(truth))]/nreps))
```

in `CVnn2`.

3.4 A look at support vector machines

Support vector machines (SVMs) are the latest set of methods within this field. They have been promoted enthusiastically, but with little respect to the selection effects of choosing the test problem and the member of the large class of classifiers to present. The original ideas are in Boser *et al.* (1992); Cortes & Vapnik (1995); Vapnik (1995, 1998); the books by Cristianini & Shawe-Taylor (2000) and Hastie *et al.* (2001, §4.5, 12.2, 12.3) present the underlying theory.

The method for $g = 2$ classes is fairly simple to describe. Logistic regression will fit exactly in separable cases where there is a hyperplane that has all class-one points on one side and all class-two points on the other. It would be a coincidence for there to be only one such hyperplane, and fitting a logistic regression will tend to fit a decision surface $p(2 \,|\, \boldsymbol{x}) = 0.5$ in the middle of the 'gap' between the groups. Support vector methods attempt directly to find a hyperplane in the middle of the gap, that is with maximal margin (the distance from the hyperplane

to the nearest point). This is quadratic programming problem that can be solved by standard methods. Such a hyperplane has *support vectors*, data points that are exactly the margin distance away from the hyperplane. It will typically be a very good classifier.

The problem is that usually no separating hyperplane will exist. This difficulty is tackled in two ways. First, we can allow some points to be on the wrong side of their margin (and for some on the wrong side of the hyperplane) subject to a constraint on the total of the 'mis-fit' distances being less than some constant, with Lagrange multiplier $C > 0$. This is still a quadratic programming problem, because of the rather arbitrary use of sum of distances.

Second, the set of variables is expanded greatly by taking non-linear functions of the original set of variables. Thus rather than seeking a classifying hyperplane $f(x) = \boldsymbol{x}^T\boldsymbol{\beta} + \beta_0 = 0$, we seek $f(x) = h(\boldsymbol{x})^T\boldsymbol{\beta} + \beta_0 = 0$ for a vector of $M \gg p$ functions h_i. Then finding a optimal separating hyperplane is equivalent to solving

$$\min_{\beta_0, \boldsymbol{\beta}} \sum_{i=1}^{n} [1 - y_i f(\boldsymbol{x}_i)]_+ + \frac{1}{2C}\|\boldsymbol{\beta}\|^2$$

where $y_i = \pm 1$ for the two classes. This is yet another penalized fitting problem, not dissimilar (Hastie *et al.*, 2001, p. 380) to a logistic regression with weight decay (which can be fitted by `multinom`). The claimed advantage of SVMs is that because we only have to find the support vectors, the family of functions h can be large, even infinite-dimensional.

There is an implementation of SVMs for R in function `svm` in package `e1071`.[1] The default values do not do well, but after some tuning for the `crabs` data we can get a good discriminant with 21 support vectors. Here `cost` is C and `gamma` is a coefficient of the kernel used to form h.

```
> # R: library(e1071)
> # S: library(libsvm)
> crabs.svm <- svm(crabs$sp ~ ., data = lcrabs, cost = 100,
                   gamma = 1)
> table(true = crabs$sp, predicted = predict(crabs.svm))
    predicted
true  B   O
B   100   0
O     0 100
```

We can try a 10-fold cross-validation by

```
> svm(crabs$sp ~ ., data = lcrabs, cost = 100, gamma = 1,
      cross = 10)
    ....
Total Accuracy: 100
Single Accuracies:
 100 100 100 100 100 100 100 100 100 100
```

[1] Code by David Meyer based on C++ code by Chih-Chung Chang and Chih-Jen Lin. A port to S-PLUS is available for machines with a C++ compiler.

The extension to $g > 2$ classes is much less elegant, and several ideas have been used. The `svm` function uses one attributed to Knerr *et al.* (1990) in which classifiers are built comparing each pair of classes, and the majority vote amongst the resulting $g(g-1)/2$ classifiers determines the predicted class.

Forensic glass

```
res.svm <- CVtest(
  function(x, ...) svm(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ]),
  cost = 100, gamma = 1 )

con(true = fgl$type, predicted = res.svm)
 ....
error rate =  28.04 %
```

The following is faster, but not strictly comparable with the results above, as a different random partition will be used.

```
> svm(type ~ ., data = fgl, cost = 100, gamma = 1, cross = 10)
    ....
Total Accuracy: 71.03
Single Accuracies:
 66.67 61.90 68.18 76.19 77.27 85.71 76.19 72.73 57.14 68.18
```


Chapter 4

Near-neighbour Methods

There are a number of non-parametric classifiers based on non-parametric estimates of the class densities or of the log posterior. Library `class` implements the k–nearest neighbour classifier and related methods (Devijver & Kittler, 1982; Ripley, 1996) and learning vector quantization (Kohonen, 1990b, 1995; Ripley, 1996). These are all based on finding the k nearest examples in some reference set, and taking a majority vote amongst the classes of these k examples, or, equivalently, estimating the posterior probabilities $p(c\,|\,\mathbf{x})$ by the proportions of the classes amongst the k examples.

The methods differ in their choice of reference set. The k–nearest neighbour methods use the whole training set or an edited subset. Learning vector quantization is similar to K-means in selecting points in the space other than the training set examples to summarize the training set, but unlike K-means it takes the classes of the examples into account.

These methods almost always measure 'nearest' by Euclidean distance.

4.1 Nearest neighbour methods

A simple estimate of the posterior distribution is the proportions of the classes amongst the nearest k data points. This is a piecewise constant function and gives a classifier known as the k*-nearest neighbour rule*. If the prior probabilities are known and the proportions of the classes in the training set are not proportional to π_k, the proportions amongst the neighbours need to be weighted.

The version with $k = 1$ is often rather successful. This divides the space $\mathcal{X}$ into the cells of the Dirichlet tessellation[1] of the data points, and labels each by the class of the data point it contains.

Nearest neighbour rules can readily be extended to allow a 'doubt' option by the so-called (k, ℓ)-rules, called in this field a 'reject option'. These take a vote amongst the classes of the k nearest patterns in $\mathcal{X}$, but only declare the class with the majority if it has ℓ or more votes, otherwise declare 'doubt'. Indeed, if there

[1] Given a set of points in $\mathbb{R}^p$, associate with each those points of $\mathbb{R}^p$ to which it is nearest. This defines a *tile*, and the tiles partition the space. Also known as Voronoi or Thiessen polygons in $\mathbb{R}^2$.

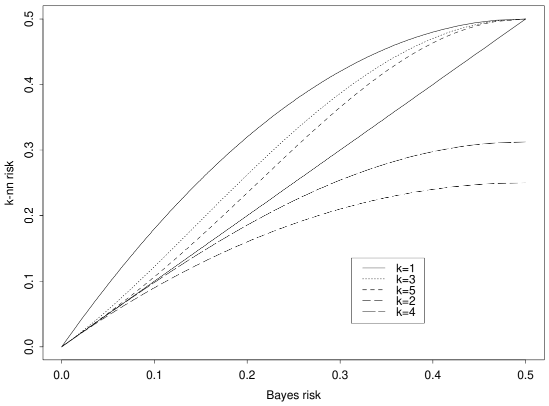

Figure 4.1: Large-sample risk r_k (k odd) or r'_k (k even) of k-nn rules against the Bayes risk r^* in a two-class problem.

are different error costs, we may want to allow the minimum majority to depend on the class to be declared.

The k-nn rule can be critically dependent on the distance used in the space $\mathcal{X}$, especially if there are few examples or k is large (Figure 4.5).

Large-sample results

Cover & Hart (1967) gave a famous result on the large-sample behaviour of the nearest neighbour rule. Note that the expected error rate is always bounded below by E^*, by the optimality of the Bayes rule.

Proposition 4.1 *Let E^* denote the error rate of the Bayes rule in a K-class problem. Then the error rate of the nearest neighbour rule averaged over training sets converges in L_1 as the size of the training set increases, to a value E_1 bounded above by*

$$E^*\left(2 - \frac{K}{K-1}E^*\right).$$

It is easy to see that the upper bound is attained if the densities $p_k(\mathbf{x})$ are identical and so the conditional risks are independent of $\mathbf{x}$.

For the k-th nearest neighbour rule detailed results are only available for two classes. Intuitively one would expect the 2-nn rule to be no improvement over the 1-nn rule, since it will achieve either a majority of two or a tie, which we will suppose is broken at random. The following result supports that intuition. On the other hand, we could report 'doubt' in the case of ties (the $(2,2)$-rule).

Proposition 4.2 *Suppose there are two classes, and let E_k denote the asymptotic error rate of the k-nn rule with ties broken at random and E'_k if ties are reported as 'doubt'. Then*

$$E'_2 \leqslant E'_4 \leqslant \cdots \leqslant E'_{2k} \nearrow E^* \swarrow E_{2k} = E_{2k-1} \leqslant \cdots \leqslant E_2 = E_1 = 2E'_2$$

Figure 4.1 shows $r_k(\mathbf{x})$ as a function of $r^*(\mathbf{x})$; this shows the agreement is excellent for moderate $r^*(\mathbf{x})$ even for small k (but not $k = 1$).

Proposition 4.3 *In the large-sample theory the means of the risk-averaged $(3,2)$-nn rule and the error rate of the $(2,2)$-nn rule are equal and provide a lower bound for the Bayes risk. The risk-averaged estimator has smaller variance.*

This suggests estimating a lower bound for the Bayes risk by running the 3-nn classifier on the training set and reporting 1/3 the number of occasions on which the neighbours are two of one class, one of another (and of course one of the neighbours will be the training-set example itself). If the distances are tied, we can average over ways of breaking the tie, since this will be equivalent to averaging over perturbations of the points.

Data editing

One common complaint about both kernel and k-nn methods is that they can take too long to compute and need too much storage for the whole training set. The difficulties are sometimes exaggerated, as there are fast ways to find near neighbours. However, in many problems it is only necessary to retain a small proportion of the training set to approximate very well the decision boundary of the k-nn classifier. This concept is known as *data editing*. It can also be used to improve the performance of the classifier by removing apparent outliers.

There are many editing algorithms: the literature on data editing is extensive but contains few comparisons. The *multiedit* algorithm of Devijver & Kittler (1982) can be specified as follows (with parameters I and V):

1. Put all patterns in the current set.
2. Divide the current set more or less equally into $V \geqslant 3$ sets. Use pairs cyclically as test and training sets.
3. For each pair classify the test set using the k-nn rule from the training set.
4. Delete from the current set all those patterns in the test set which were incorrectly classified.
5. If any patterns were deleted in the last I passes return to step 2.

The edited set is then used with the 1-nn rule (not the original value of k). Devijver & Kittler indicate that (for two classes) asymptotically the 1-nn rule on the edited set out-performs the k-nn rule on the original set and approaches the performance of the Bayes rule. (The idea is that each edit biases the retained

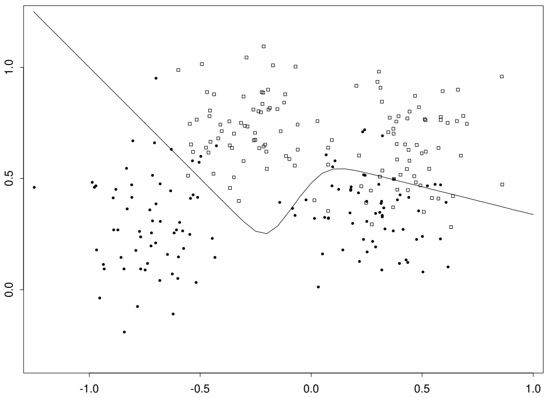

Figure 4.2: Two-class synthetic data from Ripley (1994b). The two classes are shown by solid circles and open squares: there are 125 points in each class.

points near $\mathbf{x}$ in favour of the class given by the Bayes rule at $\mathbf{x}$, so eventually this class dominates the nearest neighbours. This applies to any number of classes.)

Figure 4.3(a) illustrates the *multiedit* algorithm applied to the synthetic dataset shown in Figure 4.2 on page 82. The Bayes rule is known in this example (since it is synthetic). In practice *multiediting* can perform much less well and drop whole classes when applied to moderately sized training sets with more dimensions and classes. Another idea (Hand & Batchelor, 1978) is to retain only points whose likelihood ratio $p_y(\mathbf{x})/p_i(\mathbf{x})$ against every class $i \neq y$ exceeds some threshold t. (The densities are estimated non-parametrically.) It make more sense to retain points for which $p(y\,|\,\mathbf{x})$ is high, for example those which attain a majority ℓ in a (k, ℓ)-rule for a larger value of k. This is illustrated in Figure 4.3(b) for the synthetic example using the (10,9)-nn.

The *multiedit* algorithm aims to form homogeneous clusters in $\mathcal{X}$. However, only the points on the boundaries of the clusters are really effective in defining the classifier boundaries. *Condensing* algorithms aim to retain only the crucial exterior points in the clusters. For example, Hart (1968) gives:

1. Divide the current patterns into a store and a grabbag. One possible partition is to put the first point in the store, the rest in the grabbag.
2. Classify each sample in the grabbag by the 1-nn rule using the store as training set. If the sample is incorrectly classified transfer it to the store.
3. Return to 2 unless no transfers occurred or the grabbag is empty.
4. Return the store.

This is illustrated in Figure 4.3(c).

A refinement, the *reduced nearest neighbour rule* of Gates (1972), is to go back over the condensed training set and drop any patterns (one at a time) which are not

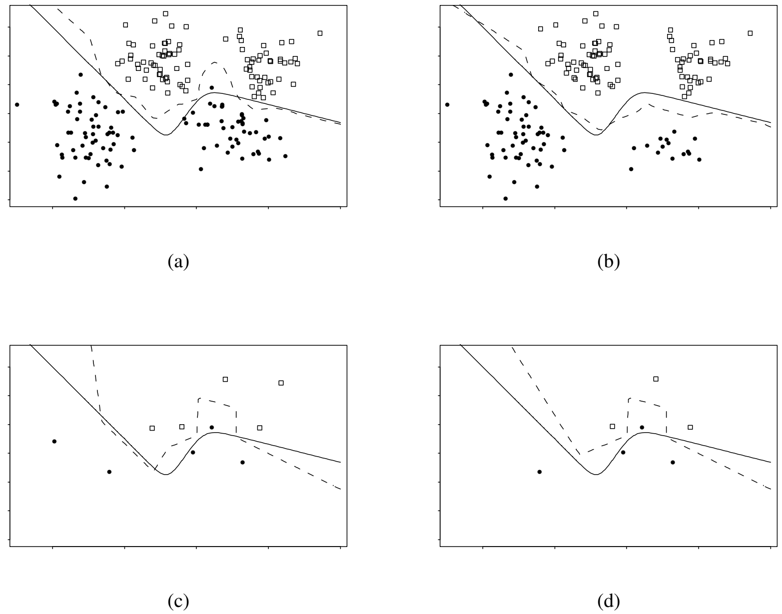

Figure 4.3: Reduction algorithms applied to Figure 4.2. The known decision boundary of the Bayes rule is shown with a solid line; the decision boundary for the 1-nn rule is shown dashed.
(a) *multiedit*.
(b) The result of retaining only those points whose posterior probability of the actual class exceeds 90% when estimated from the remaining points.
(c) *condense* after *multiedit*.
(d) *reduced nn* applied after *condense* to (a).

needed to correctly classify the rest of the (edited) training set. As Figure 4.3(d) shows, this can easily go too far and drop whole regions of a class.

A simple example

The simplest nonparametric method is k-nearest neighbours. We use Euclidean distance on the logged covariates, rather arbitrarily treating them equally.

```
library(class)
cush <- log(as.matrix(Cushings[1:21, -3]))
tp <- factor(Cushings$Type[1:21])
xp <- seq(0.6, 4.0, length=100); np <- length(xp)
yp <- seq(-3.25, 2.45, length=100)
cushT <- expand.grid(Tetrahydrocortisone=xp,
     Pregnanetriol=yp)
```

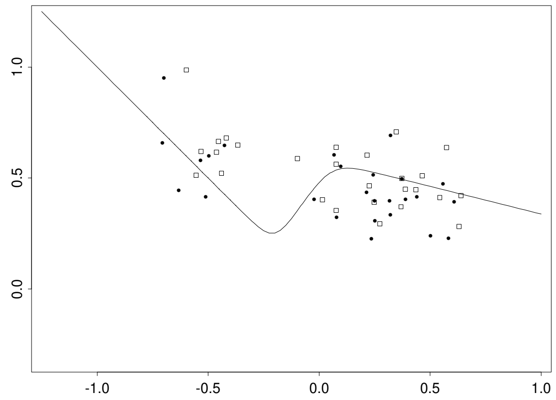

Figure 4.4: The result of the *reduced nearest neighbour rule* of Gates (1972) applied after *condense* to the unedited data of Figure 4.2.

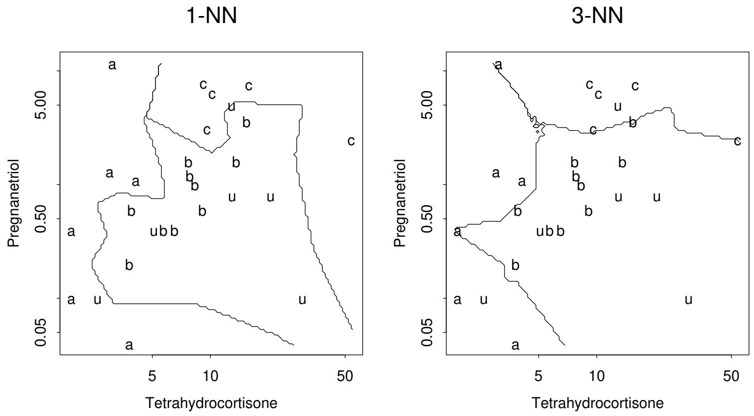

Figure 4.5: k-nearest neighbours applied to the Cushing's syndrome data.

```
par(pty="s", mfrow=c(1,2))
plot(Cushings[,1], Cushings[,2], log="xy", type="n",
     xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol",
     main = "1-NN")
text(Cushings[1:21,1], Cushings[1:21,2],
     labels = as.character(tp))
text(Cushings[22:27,1], Cushings[22:27,2], labels = "u")
Z <- knn(scale(cush, F, c(3.4, 5.7)),
         scale(cushT, F, c(3.4, 5.7)), tp)
contour(xp/log(10), yp/log(10), matrix(as.numeric(Z=="a"), np),
      add=T, levels=0.5, labex=0)
```



```
contour(xp/log(10), yp/log(10), matrix(as.numeric(Z=="c"), np),
      add=T, levels=0.5, labex=0)

plot(Cushings[,1], Cushings[,2], log="xy", type="n",
     xlab="Tetrahydrocortisone", ylab = "Pregnanetriol",
     main = "3-NN")
text(Cushings[1:21,1], Cushings[1:21,2],
     labels = as.character(tp))
text(Cushings[22:27,1], Cushings[22:27,2], labels = "u")
Z <- knn(scale(cush, F, c(3.4, 5.7)),
         scale(cushT, F, c(3.4, 5.7)), tp, k=3)
contour(xp/log(10), yp/log(10), matrix(as.numeric(Z=="a"), np),
      add=T, levels=0.5, labex=0)
contour(xp/log(10), yp/log(10), matrix(as.numeric(Z=="c"), np),
      add=T, levels=0.5, labex=0)
```

This dataset is too small to try the editing methods.

4.2 Learning vector quantization

The refinements of the k-nn rule aim to choose a subset of the training set in such a way that the 1-nn rule based on this subset approximates the Bayes classifier. It is not necessary that the modified training set is a subset of the original and an early step to combine examples to form prototypes was taken by Chang (1974). The approach taken in Kohonen's (1995) *learning vector quantization* is to construct a modified training set iteratively. Following Kohonen, we call the modified training set the *codebook*. This procedure tries to represent the decision boundaries rather than the class distributions. Once again the metric in the space $\mathcal{X}$ is crucial, so we assume the variables have been scaled in such a way that Euclidean distance is appropriate (at least locally).

Vector quantization

The use of 'vector quantization' is potentially misleading, since it has a different aim, but as it motivated Kohonen's algorithm we will digress for a brief description.

Vector quantization is a classical method in signal processing to produce an approximation to the distribution of a single class by a codebook. Each incoming signal is mapped to the nearest codebook vector, and that vector sent instead of the original signal. Of course, this can be coded more compactly by first sending the codebook, then just the indices in the codebook rather than the whole vectors. One way to choose the codebook is to minimize some measure of the approximation error averaged over the distribution of the signals (and in practice over the training patterns of that class). Taking the measure as the squared distance from the signal to the nearest codebook vector leads to the k-means algorithm which aims to minimize the sum-of-squares of distances within clusters. An 'on-line' iterative algorithm for this criterion is to present each pattern $\mathbf{x}$ in turn, and update the codebook by

$$\begin{aligned} \mathbf{m}_c &\leftarrow \mathbf{m}_c + \alpha(t)[\mathbf{x} - \mathbf{m}_c] \qquad \text{if } \mathbf{m}_c \text{ is closest to } \mathbf{x} \\ \mathbf{m}_i &\leftarrow \mathbf{m}_i \qquad \text{for the rest of the codebook.} \end{aligned} \tag{4.1}$$

Update rule (4.1) motivated Kohonen's iterative algorithms. Note that this is not a good algorithm for k-means.

Iterative algorithms

Kohonen (1990c) advocated a series of iterative procedures which has since been modified; our description follows the implementation known as `LVQ_PAK` documented in Kohonen *et al.* (1992). A initial set of codebook vectors is chosen from the training set. (We discuss later precisely how this might be done.) Each of the procedures moves codebook vectors to try to achieve better classification of the training set by the 1-nn rule based on the codebook. The examples from the training set are presented one at a time, and the codebook is updated after each presentation. In our experiments the examples were chosen randomly from the training set, but one might cycle through the training set in some pre-specified order.

The original procedure LVQ1 uses the following update rule. A example $\mathbf{x}$ is presented. The nearest codebook vector to $\mathbf{x}$, $\mathbf{m}_c$, is updated by

$$\begin{aligned} \mathbf{m}_c &\leftarrow \mathbf{m}_c + \alpha(t)[\mathbf{x} - \mathbf{m}_c] \qquad \text{if } \mathbf{x} \text{ is classified correctly by } \mathbf{m}_c \\ \mathbf{m}_c &\leftarrow \mathbf{m}_c - \alpha(t)[\mathbf{x} - \mathbf{m}_c] \qquad \text{if } \mathbf{x} \text{ is classified incorrectly} \end{aligned} \tag{4.2}$$

and all other codebook vectors are unchanged. Initially $\alpha(t)$ is chosen smaller than 0.1 (0.03 by default in `LVQ_PAK`) and it is reduced linearly to zero during the fixed number of iterations. The effect of the updating rule is to move a codebook vector towards nearby examples of its own class, and away from ones of other classes. 'Nearby' here can cover quite large regions, as the codebook will typically be small and in any case will cover $\mathcal{X}$ rather sparsely. Kohonen (1990c) motivates this as applying vector quantization to the function $\left|\pi_1 p_1(\mathbf{x}) - \pi_2 p_2(\mathbf{x})\right|$ for two classes (or the two classes which are locally most relevant).

A variant, OLVQ1, provides learning rates $\alpha_c(t)$ for each codebook vector, with an updating rule for the learning rates of

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 + (-1)^{I(\text{classification is incorrect})}\,\alpha_c(t-1)}. \tag{4.3}$$

This decreases the learning rate if the example is correctly classified, and increases it otherwise. Thus codebook vectors in the centre of classes will have rapidly decreasing learning rates, and those near class boundaries will have increasing rates (and so be moved away from the boundary quite rapidly). As the learning rates may increase, they are constrained not to exceed an upper bound, often 0.3.

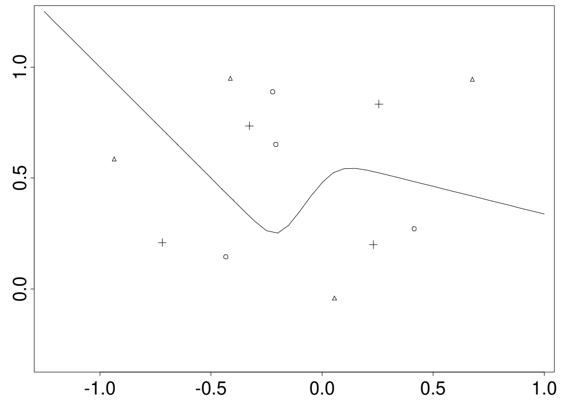

Figure 4.6: Results of learning vector quantization applied to Figure 4.2. The initially chosen codebook is shown by small circles, the result of OLVQ1 by + and subsequently applying 25,000 passes of LVQ2.1 by triangles. The known decision boundary of the Bayes rule is also shown.

Practical experience shows that the convergence is usually rapid, and `LVQ_PAK` uses 40 times as many iterations as codebook vectors.

An explanation of this rule is given by Kohonen *et al.* (1992) and Kohonen (1995, p.180) which we interpret as follows. At all times the codebook vectors are a linear combination of the training set vectors (and their initializers, if these are not in the training set). Let $s(t) = (-1)^{I(\text{classification is incorrect})}$, so we can rewrite (4.2) as

$$\begin{aligned} \mathbf{m}_c(t+1) &= \mathbf{m}_c(t) + s(t)\alpha(t)[\mathbf{x}(t) - \mathbf{m}_c] \\ &= [1 - s(t)\alpha(t)]\mathbf{m}_c(t) + s(t)\alpha(t)\mathbf{x}(t) \\ &= [1 - s(t)\alpha(t)][1 - s(t-1)\alpha(t-1)]\mathbf{m}_c(t-1) \\ &\qquad + [1 - s(t)\alpha(t)]s(t-1)\alpha(t-1)\mathbf{x}(t-1) + s(t)\alpha(t)\mathbf{x}(t). \end{aligned}$$

Now suppose $\mathbf{x}(t-1) = \mathbf{x}(t)$ and the same codebook vector is closest at both times (so $s(t-1) = s(t)$). If we ask that the multiplier of $\mathbf{x}(t)$ is the same in both terms, we find

$$[1 - s(t)\alpha(t)]\alpha(t-1) = \alpha(t)$$

which gives (4.3). This adaptive choice of rate seems to work well, as in our examples.

The procedure LVQ2.1 (Kohonen, 1990a) tries harder to approximate the Bayes rule by pairwise adjustments of the codebook vectors. Suppose $\mathbf{m}_s$, $\mathbf{m}_t$ are the two nearest neighbours to $\mathbf{x}$. They are updated simultaneously provided that $\mathbf{m}_s$ is of the same class as $\mathbf{x}$ and the class of $\mathbf{m}_t$ is different, and $\mathbf{x}$ falls into a ‘window’ near the mid-point of $\mathbf{m}_s$ and $\mathbf{m}_t$. Specifically, we must have

$$\min\left(\frac{d(\mathbf{x},\mathbf{m}_s)}{d(\mathbf{x},\mathbf{m}_t)}, \frac{d(\mathbf{x},\mathbf{m}_t)}{d(\mathbf{x},\mathbf{m}_s)}\right) > \frac{1-w}{1+w}$$

for $w \approx 0.25$. (We can interpret this condition geometrically. If $\mathbf{x}$ is projected onto the vector joining $\mathbf{m}_s$ and $\mathbf{m}_t$, it must fall at least $(1-w)/2$ of the distance from each end.) If all these conditions are satisfied the two vectors are updated by

$$\begin{aligned} \mathbf{m}_s &\leftarrow \mathbf{m}_s + \alpha(t)[\mathbf{x} - \mathbf{m}_s], \\ \mathbf{m}_t &\leftarrow \mathbf{m}_t - \alpha(t)[\mathbf{x} - \mathbf{m}_t]. \end{aligned} \tag{4.4}$$

This rule may update the codebook only infrequently. It tends to over-correct, as can be seen in Figure 4.6, where the result of iterating LVQ2.1 is to push the codebook vectors away from the decision boundary, and eventually off the figure. Thus it is recommended that LVQ2.1 only be used for a small number of iterations (30–200 times the number of codebook vectors).

The rule LVQ3 tries to overcome over-correction by using LVQ2.1 if the two closest codebook vectors to $\mathbf{x}$ are of different classes, and

$$\mathbf{m}_i \leftarrow \mathbf{m}_i + \epsilon\alpha(t)[\mathbf{x} - \mathbf{m}_i] \tag{4.5}$$

for ϵ around 0.1–0.5, for each of the two nearest codebook vectors if they are of the same class as $\mathbf{x}$. (The window is only used if the two codebook vectors are of different classes.) This introduces a second element into the iteration, of ensuring that the codebook vectors do not become too unrepresentative of their class distribution. It does still allow the codebooks to drift to the centre of the class distributions and even beyond, as Figure 4.7 shows.

The recommended procedure is to run OLVQ1 until convergence (usually rapid) and then a moderate number of further steps of LVQ1 and/or LVQ3.

4.3 Forensic glass

Figure 1.4 suggests that nearest neighbour methods might work well, and the 1–nearest neighbour classifier is (to date) unbeatable in this problem. We can estimate a lower bound for the Bayes risk as 10% by the method of proposition 4.3.

```
library(class)
fgl0 <- fgl[ ,-10] # drop type
{ res <- fgl$type
  for (i in sort(unique(rand))) {
     cat("fold ",i,"\n", sep="")
     sub <- rand == i
     res[sub] <- knn(fgl0[!sub, ], fgl0[sub,], fgl$type[!sub],
                     k=1)
  }
  res } -> res.knn1
```


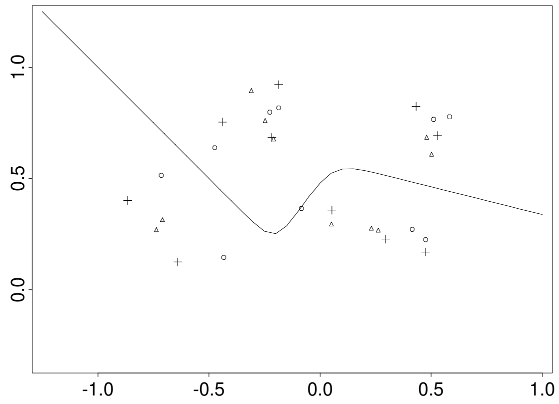

Figure 4.7: Further results of LVQ with a larger codebook. This time the triangles show the results from LVQ3.

```
> con(fgl$type, res.knn1)
      WinF WinNF Veh Con Tabl Head
 WinF   59     6   5   0    0    0
WinNF   12    57   3   3    1    0
  Veh    2     4  11   0    0    0
  Con    0     2   0   8    1    2
 Tabl    1     0   0   1    6    1
 Head    0     4   1   1    1   22
error rate =  23.83 %
> res.lb <- knn(fgl0, fgl0, fgl$type, k=3, prob=T, use.all=F)
> table(attr(res.lb, "prob"))
 0.333333 0.666667   1
       10       64 140
1/3 * (64/214) = 0.099688
```

Learning vector quantization

For LVQ as for k-nearest neighbour methods we have to select a suitable metric. The following experiments used Euclidean distance on the original variables, but the rescaled variables or Mahalanobis distance could also be tried.

```
cd0 <- lvqinit(fgl0, fgl$type, prior=rep(1,6)/6,k=3)
cd1 <- olvq1(fgl0, fgl$type, cd0)
con(fgl$type, lvqtest(cd1, fgl0))
```

We set an even prior over the classes as otherwise there are too few representatives of the smaller classes. Our initialization code follows Kohonen’s in selecting the number of representatives: in this problem 24 points are selected, four from each class.

```
CV.lvq <- function()
{
  res <- fgl$type
  for(i in sort(unique(rand))) {
    cat("doing fold",i,"\n")
    cd0 <- lvqinit(fgl0[rand != i,], fgl$type[rand != i],
                   prior=rep(1,6)/6, k=3)
    cd1 <- olvq1(fgl0[rand != i,], fgl$type[rand != i], cd0)
    cd1 <- lvq3(fgl0[rand != i,], fgl$type[rand != i],
                cd1, niter=10000)
    res[rand == i] <- lvqtest(cd1, fgl0[rand == i,])
  }
  res
}
con(fgl$type, CV.lvq())
      WinF WinNF Veh Con Tabl Head
 WinF   59    10   1   0    0    0
WinNF   10    61   1   2    2    0
  Veh    6     8   3   0    0    0
  Con    0     2   0   6    2    3
 Tabl    0     0   0   2    7    0
 Head    3     2   0   1    1   22
error rate =  26.17 %

# Try Mahalanobis distance
fgl0 <- scale(princomp(fgl[,-10])$scores)
con(fgl$type, CV.lvq())
    ....
error rate =  35.05 %
```

The initialization is random, so your results are likely to differ.

Chapter 5

Assessing Performance

The background notes show that *if* we can find the posterior probabilities $p(c\,|\,\mathbf{x})$ we can construct the best possible classifier. So performance assessment can be split into determining if we have a good estimate of the posterior probabilities, and direct assessment of performance.

5.1 Practical ways of performance assessment

The only really convincing way we have to compare generalization ability is to try out various classifiers. We need a *test* set unused for any other purpose.

- This is an experiment, so it should be designed and analysed. Even specialist statisticians forget this. Cohen (1995), Kleijnen (1987) and Kleijnen & van Groenendaal (1992) discuss these issues in a way that may be accessible to the neural network community.
- Comparisons between classifiers will normally be more precise that assessments of the true risk.
- Standard errors of estimates decrease at $O(n^{-1/2})$ so we often need a very large test set to measure subtle differences in performance. For example, if we count errors, we need differences of at least 5 and often 15 or more for statistical significance (Ripley, 1994a; Ripley, 1996, p. 77).

Validation

The same ideas can be used to choose within classes of classifiers (for example the number of hidden units). This is called *validation* and needs a separate validation set. A validation set can also be used to optimize over the weights in model averaging (suggested as long ago as Stone, 1974).

Cross-validation and cross-testing

With small datasets, separate training, validation and test sets seem wasteful. Can we reuse just one set? In V-fold cross validation we divide the data into V roughly equal pieces (how?). Use one for validation, $V-1$ for training, and rotate. Similarly for testing: if we need both we nest or rotate.

What size should V be? The original proposal (Lunts & Brailovsky, 1967; Lachenbruch, 1967; Stone, 1974) was $V=n$, leave-one-out CV (often confused with the distinct concept of *jackknifing*). This is often too expensive, and also too variable. The V-fold variant was proposed by Toussaint & Donaldson (1970) and more generally by Breiman *et al.* (1984). I usually use $V=5$ or 10.

How Do We Estimate Performance?

In supervised classification we should be interested in the achieved risk, as that was the objective we set out to minimize. To estimate this on a test set we count errors or add up losses. That can be a rather crude measure with high variance.

For validation we can choose whatever measure we wish. There is a lot to be said for *log-probability scoring*, that is

$$-\sum_i \log \widehat{p}(c_i \,|\, \mathbf{x}_i)$$

summed over the test set. This is estimating the crucial part of the Kullback-Leibler divergence $d(p, \widehat{p})$ and has a lower variability than error-counting.

Smoothed or risk-averaged estimates

Suppose we see an example with features $\mathbf{x}$. Then we only observe on the test set one of the possible classes c. Our expected loss given $X = \mathbf{x}$ is

$$e(\mathbf{x}) = \sum_c L\big(c, \widehat{c}(\mathbf{x})\big)\, p(c \,|\, \mathbf{x})$$

Thus if we knew the posterior probabilities, we could estimate the risk by averaging $e(\mathbf{x})$ over the test set. As we do not know them, we could use our best estimates to give

$$\widetilde{e}(\mathbf{x}) = \sum_c L\big(c, \widehat{c}(\mathbf{x})\big)\, \widetilde{p}(c \,|\, \mathbf{x})$$

and the resulting estimate of the risk is known as an *smoothed* or *risk-averaged* estimate. The idea goes back to Glick (1978); for more details see Ripley (1996, pp. 75–6). The loss $L\big(c, \widehat{c}(\mathbf{x})\big)$ takes a small set of values; $e(\mathbf{x})$ is much less variable (often with variance reduced by a factor of 10). So we get a more precise estimator of risk, and we do not even need to know the true classification!

Where is the catch? $\widetilde{p}(c \,|\, \mathbf{x})$ has to be a good estimate of $p(c \,|\, \mathbf{x})$. Can we check that? That is our next topic.

5.2 Calibration plots

One measure that a suitable model for $p(c \,|\, \mathbf{x})$ has been found is that the predicted probabilities are *well calibrated*, that is that a fraction of about p of the events we predict with probability p actually occur. Methods for testing calibration of probability forecasts have been developed in connection with weather forecasts (Dawid, 1982, 1986).

For the forensic glass example we are making six probability forecasts for each case, one for each class. To ensure that they are genuine forecasts, we should use the cross-validation procedure. A minor change to the code gives the probability predictions:

```
CVprobs <- function(fitfn, predfn, ...)
{
  res <- matrix(, 214, 6)
  for (i in sort(unique(rand))) {
     cat("fold ",i,"\n", sep="")
     learn <- fitfn(rand != i, ...)
     res[rand == i,] <- predfn(learn, rand==i)
  }
  res
}
probs.multinom <- CVprobs(
   function(x, ...) multinom(type ~ ., fgl[x,], ...),
   function(obj, x) predict(obj, fgl[x, ],type="probs"),
   maxit=1000, trace=F )
```

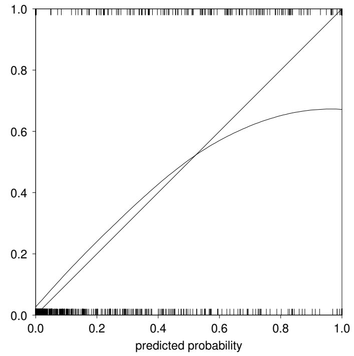

Figure 5.1: Calibration plot for multiple logistic fit to the `fgl` data.

We can plot these and smooth them by

```
probs.yes <- as.vector(class.ind(fgl$type))
probs <- as.vector(probs.multinom)
par(pty="s")
```

```
plot(c(0,1), c(0,1), type="n", xlab="predicted probability",
   ylab="", xaxs="i", yaxs="i", las=1)
rug(probs[probs.yes==0], 0.02, side=1, lwd=0.5)
rug(probs[probs.yes==1], 0.02, side=3, lwd=0.5)
abline(0,1)
newp <- seq(0, 1, length=100)
lines(newp, predict(loess(probs.yes ~ probs, span=1), newp))
```

A method with an adaptive bandwidth such as `loess` is needed here, as the distribution of points along the x axis can be very much more uneven than in this example. The result is shown in Figure 5.1. This plot does show substantial overconfidence in the predictions, especially at probabilities close to one. Indeed, only 22/64 of the events predicted with probability greater than 0.9 occurred. (The underlying cause is the multimodal nature of some of the underlying class distributions.)

Where calibration plots are not straight, the best solution is to find a better model. Sometimes the overconfidence is minor, and mainly attributable to the use of plug-in rather than predictive estimates. Then the plot can be used to adjust the probabilities (which may need then further adjustment to sum to one for more than two classes).

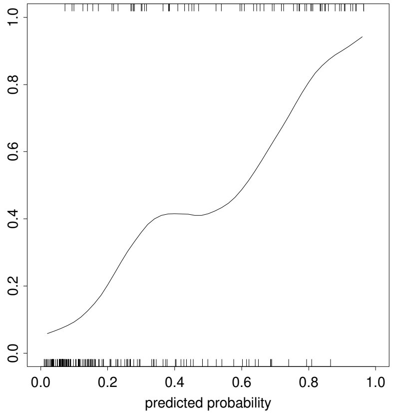

Figure 5.2: Calibration plot for a two-class medical diagnosis problem.

More often we see something like figure 5.3 in which the extreme probabilities (near zero or one) are estimated as too extreme. The reason is that we are not using $\widetilde{p}(k \mid \mathbf{x})$ but $\widehat{p}(k, \mid \mathbf{x}; \widehat{\theta})$, a *plug-in* estimate in which we assume that the fitted parameters $\widehat{\theta}$ are the true weights (and that the neural network represents the true posterior probabilities). We are ignoring the uncertainty in the $\widehat{\theta}$.

If necessary, we can use such graphs to re-calibrate, but usually they indicate a deeper problem. Calibration plots can help detect over-fitting: an example from Mark Mathieson is shown in figures 5.4 and 5.5.

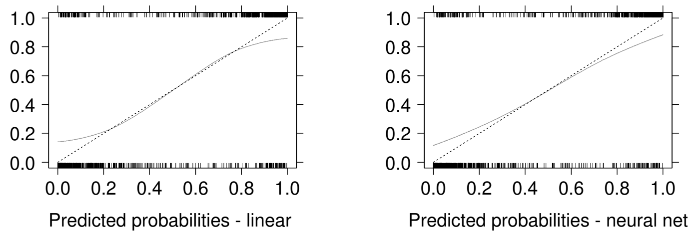

Figure 5.3: Calibration plots from a breast cancer prognosis study—the task is to predict years to relapse as one of the classes 0, 1, 2, 3, 4+.

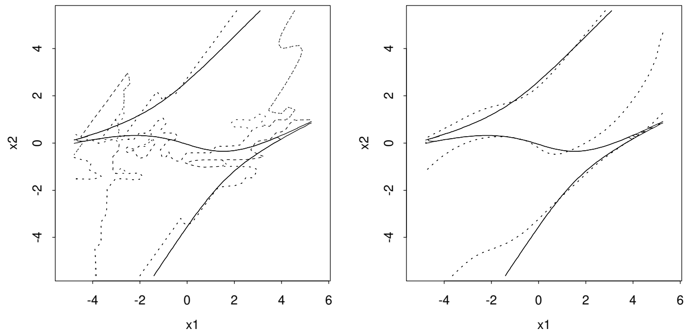

Figure 5.4: Three-class fits with the Bayes rule and fits from an ordinal neural network with weight decay parameter λ. $\lambda = 0$ (left) and $\lambda = 0.1$ (right). The true class boundaries are shown as solid lines, the fitted ones as dashed lines.

There is an extensive literature on probability forecasting: see Dawid (1986) and Cooke (1991). This has usually been for the assessment of human experts, but applies equally in machine learning. There are various aggregate scores that say how well 'experts' are doing on a variety of predictions. *Proper* scoring rules reward 'experts' who truly are expert. The most popular of these is the (half-)Brier score, the sum of squares of one minus the predicted probability of the event that happened, and especially *log-probability scoring*, the sum of minus the natural logarithms of those probabilities.

5.3 Performance summaries and ROC curves

Suppose we have just two classes, in medical diagnosis normal or diseased. Then class-conditional error rates have conventional names

$$\text{specificity} = \Pr(\text{predict normal} \mid \text{is normal})$$

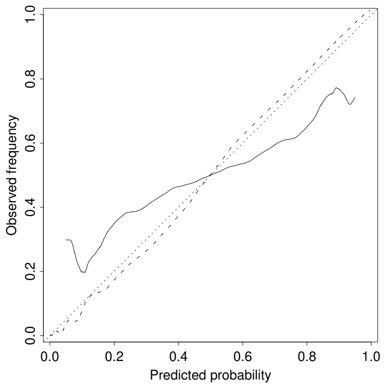

Figure 5.5: Calibration plots for the fits in figure 5.4; the line for $\lambda = 0.1$ is dashed.

$$\text{sensitivity} = \Pr(\text{predict diseased} \mid \text{is diseased})$$

Considering these separately is usually an admission that losses other than zero-one are appropriate.

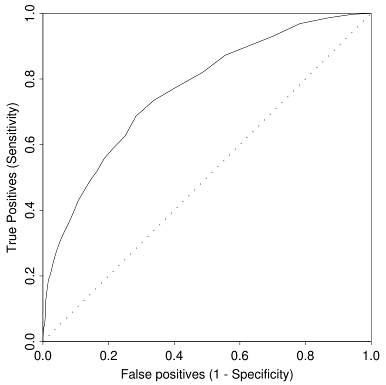

Figure 5.6: An ROC curve for the breast-cancer prognosis problem of figure 5.3.

Suppose our classifier outputs an approximation to $p(\text{diseased} \mid \mathbf{x})$. Then we get a family of classifiers by declaring 'diseased' if $p(\text{diseased} \mid \mathbf{x}) > \alpha$ and varying $\alpha \in (0, 1)$. The ROC (receiver operating curve) is a plot of sensitivity against $1 - \text{specificity}$ (Hand, 1997, Chapter 7). Figure 5.6 shows an example ROC. For equal losses we want to minimize the sum of sensitivity and $1 - \text{specificity}$, or $\text{sensitivity} - \text{specificity}$. The best choice of α in that case corresponds to the point at which the ROC has slope 1 (and we know corresponds to $\alpha = 0.5$); in this problem the sensitivity and specificity would be both about 0.8. As the ratio

of losses changes the optimal operating point moves along the curve, jumping along any linear sections of the curve.

It is common in medicine to quote the area under the ROC curve. This is a compromise between losses that may be irrelevant to the real problem, where for example high sensitivity may be much more important than high specificity.

5.4 Assessing generalization

It can be very helpful to have some idea of the *Bayes risk*, the smallest possible expected loss. (Too many people assume that it is zero!) This is a measure of the 'overlap' of the classes.

We obviously need a very flexible classifier. We can do something with k-nearest neighbour rules. A (k, ℓ)-nn rule declares a class if it has at least ℓ representatives amongst the k nearest points in $\mathcal{T}$, otherwise it declares 'doubt' $\mathcal{D}$. We can use proposition 4.3 on page 81.

In the theory this result does not depend on the metric used to measure the distances; in practice it does. With small samples this method tends to underestimate the lower bound on the Bayes risk (which is the safe direction).

Uniform Bounds

The theory pioneered by Vapnik and represented by Devroye *et al.* (1996) concerns universal procedures, which will with enough data get arbitrarily close to the Bayes risk. Universal strong consistency (risk converging to the Bayes risk with probability one over sequences of independent identically distributed training sets) can be achieved by k-nn rules for $k = o(n)$ and various other simple rules.

These limit results are proved by proving uniform bounds, and it has been suggested that these be used directly. From a published paper:

> 'If our network can be trained to classify correctly a fraction $1 - (1 - \beta)\epsilon$ of the n training examples, the probability that its error—a measure of its ability to generalize—is less than ϵ is at least $1 - \delta$.'

(authorship suppressed to protect the guilty).

Let pmc denote the true error rate of a classifier, and $\widehat{\mathsf{pmc}}$ the error rate on a training set of size n. The theorems are of the form

$$\Pr\bigl\{\widehat{\mathsf{pmc}}(g) < (1 - \beta)\mathsf{pmc}(g) \textit{ and } \mathsf{pmc}(g) > \epsilon \text{ for any } g \in \mathcal{F}\bigr\} < \delta \qquad (5.1)$$

for $n > n_0(\epsilon, \delta, \mathcal{F})$ for a class of classifiers $\mathcal{F}$. The probability is over the random choice of training set of size n. This says that however we choose the classifier in the class, our observed error rate on the training set will be close to true error rate of the classifier, *for most training sets*.

The authors of the quote have (like many lawyers and judges) confused $\Pr(A \text{ and } B)$ with $\Pr(B \mid A)$, for their quote means

$$\Pr\bigl\{\mathsf{pmc}(\widehat{c}) > \epsilon \;\big|\; \widehat{\mathsf{pmc}}(\widehat{c}) < (1 - \beta)\mathsf{pmc}(\widehat{c})\bigr\} < \delta$$

This is only implied by (5.1) (with a different δ) if we can bound below the probability of the conditioning term. We only have one training set, so how could we possibly do that? Sadly, this is not an isolated example.

These bounds apply to any classifier: we also need results that relate $\mathsf{pmc}(\widehat{c})$ to the Bayes risk. That is harder: for example Devroye *et al.* prove consistency for a projection pursuit regression trained by empirical risk minimization — and we have no idea how to do that in practice.

References

Aitchison, J. and Dunsmore, I. R. (1975) *Statistical Prediction Analysis*. Cambridge: Cambridge University Press. pages 69

Aleksander, I. and Morton, H. (1990) *An Introduction to Neural Computing*. London: Chapman & Hall. pages 58

Anderson, E. (1935) The irises of the Gaspe peninsula. *Bulletin of the American Iris Society* **59**, 2–5. pages 2

Asimov, D. (1985) The grand tour: a tool for viewing multidimensional data. *SIAM Journal on Scientific and Statistical Computing* **6**, 128–143. pages 27

Bates, D. M. and Watts, D. G. (1988) *Nonlinear Regression Analysis and Its Applications*. New York: John Wiley and Sons. pages 61

Best, D. J. and Rayner, J. C. W. (1988) A test for bivariate normality. *Statistics and Probability Letters* **6**, 407–412. pages 22

Bishop, C. M. (1995) *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press. pages 59

Bishop, Y. M. M., Fienberg, S. E. and Holland, P. W. (1975) *Discrete Multivariate Analysis*. Cambridge, MA: MIT Press. pages 30

Boser, B. E., Guyon, I. M. and Vapnik, V. N. (1992) A training algorithm for optimal margin classifiers. In *Proceeedings of the 5th Annual ACM Workshop on Computational Learning Theory.*, ed. D. Haussler, pp. 144–152. ACM Press. pages 76

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. New York: Chapman & Hall / CRC Press. (Formerly Monterey: Wadsworth and Brooks/Cole.). pages 36, 42, 92

Bruckner, L. A. (1978) On Chernoff faces. In *Graphical Representation of Multivariate Data*, ed. P. C. C. Wang, pp. 93–121. New York: Academic Press. pages 11

Chang, C. L. (1974) Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers* **23**, 1179–1184. pages 85

Chernoff, H. (1973) The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association* **68**, 361–368. pages 11

Ciampi, A., Chang, C.-H., Hogg, S. and McKinney, S. (1987) Recursive partitioning: A versatile method for exploratory data analysis in biostatistics. In *Biostatistics*, eds I. B. McNeil and G. J. Umphrey, pp. 23–50. New York: Reidel. pages 41

Clark, L. A. and Pregibon, D. (1992) Tree-based models. In *Statistical Models in S*, eds J. M. Chambers and T. J. Hastie, Chapter 9. New York: Chapman & Hall. pages 36, 40

Cohen, P. R. (1995) *Empirical Methods for Artificial Intelligence*. Cambridge, MA: The MIT Press. pages 91

Comon, P. (1994) Independent component analysis — a new concept? *Signal Processing* **36**, 287–314. pages 10

Cook, D., Buja, A. and Cabrera, J. (1993) Projection pursuit indices based on orthonormal function expansions. *Journal of Computational and Graphical Statistics* **2**, 225–250. pages 22, 23

Cooke, R. M. (1991) *Experts in Uncertainty. Opinion and Subjective Probability in Science*. New York: Oxford University Press. pages 95

Cortes, C. and Vapnik, V. (1995) Support-vector networks. *Machine Learning* **20**, 273–297. pages 76

Cover, T. M. and Hart, P. E. (1967) Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* **13**, 21–27. pages 80

Cox, T. F. and Cox, M. A. A. (2001) *Multidimensional Scaling*. Second Edition. Chapman & Hall / CRC. pages 6, 8

Cristianini, N. and Shawe-Taylor, J. (2000) *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge: Cambridge University Press. pages 76

Cybenko, G. (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems* **2**, 303–314. pages 60

Dawid, A. P. (1982) The well-calibrated Bayesian (with discussion). *Journal of the American Statistical Association* **77**, 605–613. pages 93

Dawid, A. P. (1986) Probability forecasting. In *Encyclopedia of Statistical Sciences*, eds S. Kotz, N. L. Johnson and C. B. Read, volume 7, pp. 210–218. New York: John Wiley and Sons. pages 93, 95

Devijver, P. A. and Kittler, J. V. (1982) *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice-Hall. pages 79, 81

Devroye, L., Györfi, L. and Lugosi, G. (1996) *A Probabilistic Theory of Pattern Recognition*. New York: Springer. pages 97, 98

Diaconis, P. and Freedman, D. (1984) Asymptotics of graphical projection pursuit. *Annals of Statistics* **12**, 793–815. pages 20

Duda, R. O., Hart, P. E. and Stork, D. G. (2001) *Pattern Classification*. Second Edition. New York: John Wiley and Sons. pages 2

Emerson, J. W. (1998) Mosaic displays in S-PLUS: a general implementation and a case study. *Statistical Computing and Graphics Newsletter* **9**(1), 17–23. pages 30

Eslava-Gómez, G. (1989) *Projection Pursuit and Other Graphical Methods for Multivariate Data*. D. Phil. thesis, University of Oxford. pages 23, 27

Fauquet, C., Desbois, D., Fargette, D. and Vidal, G. (1988) Classification of furoviruses based on the amino acid composition of their coat proteins. In *Viruses with Fungal Vectors*, eds J. I. Cooper and M. J. C. Asher, pp. 19–38. Edinburgh: Association of Applied Biologists. pages 23, 24

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics (London)* **7**, 179–188. pages 2, 3

Fisher, R. A. (1940) The precision of discriminant functions. *Annals of Eugenics (London)* **10**, 422–429. pages 30, 30

Flury, B. and Riedwyl, H. (1981) Graphical representation of multivariate data by means of asymmetrical faces. *Journal of the American Statistical Association* **76**, 757–765. pages 11

Friedman, J. H. (1987) Exploratory projection pursuit. *Journal of the American Statistical Association* **82**, 249–266. pages 20, 21, 22

Friedman, J. H. and Tukey, J. W. (1974) A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers* **23**, 881–890. pages 20

Friendly, M. (1994) Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association* **89**, 190–200. pages 30

Friendly, M. (2000) *Visualizing Categorical Data*. Cary, NC: SAS Institute. pages 30

Funahashi, K. (1989) On the approximate realization of continuous mappings by neural networks. *Neural Networks* **2**, 183–192. pages 60

Gabriel, K. R. (1971) The biplot graphical display of matrices with application to principal component analysis. *Biometrika* **58**, 453–467. pages 9, 9

Gates, G. W. (1972) The reduced nearest neighbor rule. *IEEE Transactions on Information Theory* **18**, 431–433. pages 82, 84

Glick, N. (1978) Additive estimators for probabilities of complete classification. *Pattern Recognition* **10**, 211–222. pages 92

Gordon, A. D. (1999) *Classification*. Second Edition. London: Chapman & Hall / CRC. pages 13, 13

Gower, J. C. and Hand, D. J. (1996) *Biplots*. London: Chapman & Hall. pages 9, 10, 32, 33, 33, 34

Greenacre, M. (1992) Correspondence analysis in medical research. *Statistical Methods in Medical Research* **1**, 97–117. pages 32, 33

Hall, P. (1989) On polynomial-based projection indices for exploratory projection pursuit. *Annals of Statistics* **17**, 589–605. pages 22

Hand, D., Mannila, H. and Smyth, P. (2001) *Principles of Data Mining*. Cambridge, MA: The MIT Press. pages 2

Hand, D. J. (1997) *Construction and Assessment of Classification Rules*. Chichester: Wiley. pages 96

Hand, D. J. and Batchelor, B. G. (1978) An edited condensed nearest neighbor rule. *Information Sciences* **14**, 171–180. pages 82

Hart, P. E. (1968) The condensed nearest neighbor rule. *IEEE Transactions on Information Theory* **14**, 515–516. pages 82

Hartigan, J. A. (1975) *Clustering Algorithms*. New York: John Wiley and Sons. pages 15

Hartigan, J. A. and Kleiner, B. (1981) Mosaics for contingency tables. In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, ed. W. F. Eddy, pp. 268–273. New York: Springer-Verlag. pages 30

Hartigan, J. A. and Kleiner, B. (1984) A mosaic of television ratings. *American Statistician* **38**, 32–35. pages 30

Hartigan, J. A. and Wong, M. A. (1979) A K-means clustering algorithm. *Applied Statistics* **28**, 100–108. pages 15

Hastie, T. J., Tibshirani, R. J. and Friedman, J. (2001) *The Elements of Statistical Learning. Data Mining Inference and Prediction*. New York: Springer-Verlag. pages 2, 10, 76, 77

Haykin, S. (1994) *Neural Networks. A Comprehensive Foundation*. New York: Macmillan College Publishing. pages 58

Henrichon, Jr., E. G. and Fu, K.-S. (1969) A nonparametric partitioning procedure for pattern classification. *IEEE Transactions on Computers* **18**, 614–624. pages 36

Hertz, J., Krogh, A. and Palmer, R. G. (1991) *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley. pages 59

Hornik, K., Stinchcombe, M. and White, H. (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359–366. pages 60

Huber, P. J. (1985) Projection pursuit (with discussion). *Annals of Statistics* **13**, 435–525. pages 20, 27

Hyvärinen, A., Karhunen, J. and Oja, E. (2001) *Independent Component Analysis*. New York: John Wiley and Sons. pages 10

Hyvärinen, A. and Oja, E. (2000) Independent component analysis. algorithms and applications. *Neural Networks* **13**, 411–430. pages 10, 11

Inselberg, A. (1984) The plane with parallel coordinates. *The Visual Computer* **1**, 69–91. pages 12

Jackson, J. E. (1991) *A User's Guide to Principal Components*. New York: John Wiley and Sons. pages 6

Jardine, N. and Sibson, R. (1971) *Mathematical Taxonomy*. London: John Wiley and Sons. pages 6, 14

Jolliffe, I. T. (1986) *Principal Component Analysis*. New York: Springer-Verlag. pages 6, 9

Jones, M. C. and Sibson, R. (1987) What is projection pursuit? (with discussion). *Journal of the Royal Statistical Society A* **150**, 1–36. pages 20, 21, 21, 22, 22

Kaufman, L. and Rousseeuw, P. J. (1990) *Finding Groups in Data. An Introduction to Cluster Analysis*. New York: John Wiley and Sons. pages 6, 13, 14

Kleijnen, J. P. C. (1987) *Statistical Tools for Simulation Practitioners*. New York: Marcel Dekker. pages 91

Kleijnen, J. P. C. and van Groenendaal, W. (1992) *Simulation: A Statistical Perspective*. Chichester: Wiley. pages 91

Knerr, S., Personnaz, L. and Dreyfus, G. (1990) Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neuro-computing: Algorithms, Architectures and Applications*, eds F. Fogelman Soulié and J. Hérault. Berlin: Springer-Verlag. pages 78

Kohonen, T. (1990a) Improved versions of learning vector quantization. In *Proceedings of the IEEE International Conference on Neural Networks, San Diego*, volume I, pp. 545–550. New York: IEEE Press. pages 87

Kohonen, T. (1990b) The self-organizing map. *Proceedings IEEE* **78**, 1464–1480. pages 79

Kohonen, T. (1990c) The self-organizing map. *Proceedings of the IEEE* **78**, 1464–1480. pages 86, 86

Kohonen, T. (1995) *Self-Organizing Maps*. Berlin: Springer-Verlag. pages 19, 19, 79, 85, 85, 87

Kohonen, T., Kangas, T., Laaksonen, J. and Torkkola, K. (1992) LVQ_PAK. *The learning vector quantization program package version 2.1*. Laboratory of Computer and Information Science, Helsinki University of Technology. [Version 3.1 became available in 1995]. pages 86, 87

Kruskal, J. B. (1969) Toward a practical method which helps uncover the structure of a set of multivariate observations by finding the linear transformation which optimizes a new 'index of condensation'. In *Statistical Computation*, eds R. C. Milton and J. A. Nelder, pp. 427–440. New York: Academic Press. pages 20, 20

Kruskal, J. B. (1972) Linear transformation of multivariate data to reveal clustering. In *Multidimensional Scaling: Theory and Application in the Behavioural Sciences*, eds R. N. Shephard, A. K. Romney and S. K. Nerlove, pp. 179–191. New York: Seminar Press. pages 20

Lachenbruch, P. (1967) An almost unbiased method of obtaining confidence intervals for the probability of misclassification in discriminant analysis. *Biometrics* **23**, 639–645. pages 92

Lee, T. W. (1998) *Independent Component Analysis: Theory and Applications*. Dordrecht: Kluwer Academic Publishers. pages 10

Lunts, A. L. and Brailovsky, V. L. (1967) Evaluation of attributes obtained in statistical decision rules. *Engineering Cybernetics* **3**, 98–109. pages 92

Macnaughton-Smith, P., Williams, W. T., Dale, M. B. and Mockett, L. G. (1964) Dissimilarity analysis: A new technique of hierarchical sub-division. *Nature* **202**, 1034–1035. pages 14

MacQueen, J. (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, eds L. M. Le Cam and J. Neyman, volume 1, pp. 281–297. Berkeley, CA: University of California Press. pages 15

McCulloch, W. S. and Pitts, W. (1943) A logical calculus of ideas immanent in neural activity. *Bulletin of Mathematical Biophysics* **5**, 115–133. pages 60

McLachlan, G. J. (1992) *Discriminant Analysis and Statistical Pattern Recognition*. New York: John Wiley and Sons. pages 3

Michie, D. (1989) Problems of computer-aided concept formation. In *Applications of Expert Systems 2*, ed. J. R. Quinlan, pp. 310–333. Glasgow: Turing Institute Press / Addison-Wesley. pages 37

Morgan, J. N. and Messenger, R. C. (1973) THAID*: a Sequential Search Program for the Analysis of Nominal Scale Dependent Variables*. Survey Research Center, Institute for Social Research, University of Michigan. pages 36

Morgan, J. N. and Sonquist, J. A. (1963) Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association* **58**, 415–434. pages 36

Mosteller, F. and Tukey, J. W. (1977) *Data Analysis and Regression*. Reading, MA: Addison-Wesley. pages 14

Murtagh, F. and Hernández-Pajares, M. (1995) The Kohonen self-organizing map method: An assessment. *Journal of Classification* **12**, 165–190. pages 20

Posse, C. (1995) Tools for two-dimensional exploratory projection pursuit. *Journal of Computational and Graphical Statistics* **4**, 83–100. pages 23

Quinlan, J. R. (1979) Discovering rules by induction from large collections of examples. In *Expert Systems in the Microelectronic Age*, ed. D. Michie. Edinburgh: Edinburgh University Press. pages 36

Quinlan, J. R. (1983) Learning efficient classification procedures and their application to chess end-games. In *Machine Learning*, eds R. S. Michalski, J. G. Carbonell and T. M. Mitchell, pp. 463–482. Palo Alto: Tioga. pages 36

Quinlan, J. R. (1986) Induction of decision trees. *Machine Learning* **1**, 81–106. pages 36

Quinlan, J. R. (1993) *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann. pages 36

Ripley, B. D. (1993) Statistical aspects of neural networks. In *Networks and Chaos — Statistical and Probabilistic Aspects*, eds O. E. Barndorff-Nielsen, J. L. Jensen and W. S. Kendall, pp. 40–123. London: Chapman & Hall. pages 59, 61

Ripley, B. D. (1994a) Neural networks and flexible regression and discrimination. In *Statistics and Images 2*, ed. K. V. Mardia, volume 2 of *Advances in Applied Statistics*, pp. 39–57. Abingdon: Carfax. pages 61, 91

Ripley, B. D. (1994b) Neural networks and related methods for classification (with discussion). *Journal of the Royal Statistical Society series B* **56**, 409–456. pages 82

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press. pages i, 2, 8, 36, 42, 42, 59, 69, 73, 79, 79, 91, 92

Roberts, S. and Tarassenko, L. (1995) Automated sleep EEG analysis using an RBF network. In *Neural Network Applications*, ed. A. F. Murray, pp. 305–322. Dordrecht: Kluwer Academic Publishers. pages 19

Sammon, J. W. (1969) A non-linear mapping for data structure analysis. *IEEE Transactions on Computers* **C-18**, 401–409. pages 7, 7

Sethi, I. K. and Sarvarayudu, G. P. R. (1982) Hierarchical classifier design using mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **4**, 441–445. pages 36

Stace, C. (1991) *New Flora of the British Isles*. Cambridge: Cambridge University Press. pages 37

Stone, M. (1974) Cross-validatory choice and assessment of statistical predictions (with discussion). *Journal of the Royal Statistical Society B* **36**, 111–147. pages 91, 92

Therneau, T. M. and Atkinson, E. J. (1997) An introduction to recursive partitioning using the RPART routines. Technical report, Mayo Foundation. pages 36, 49

Toussaint, G. and Donaldson, R. (1970) Algorithms for recognizing contour-traced hand-printed characters. *IEEE Transactions on Computers* **19**, 541–546. pages 92

Vapnik, V. N. (1995) *The Nature of Statistical Learning Theory*. New York: Springer-Verlag. pages 76

Vapnik, V. N. (1998) *Statistical Learning Theory*. New York: John Wiley and Sons. pages 76

Venables, W. N. and Ripley, B. D. (1999) *Modern Applied Statistics with S-PLUS*. Third Edition. New York: Springer-Verlag. pages i

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth Edition. New York: Springer-Verlag. pages i

Vinod, H. (1969) Integer programming and the theory of grouping. *Journal of the American Statistical Association* **64**, 506–517. pages 16

Wahba, G. and Wold, S. (1975) A completely automatic french curve. *Communications in Statistics* **4**, 1–17. pages 61

Webb, A. (1999) *Statistical Pattern Recognition*. London: Arnold. pages 2

Wegman, E. J. (1990) Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association* **85**, 664–675. pages 12

Wilkinson, L. (1999) *The Grammar of Graphics*. New York: Springer-Verlag. pages 11

Wilson, S. R. (1982) Sound and exploratory data analysis. In *COMPSTAT 1982, Proceedings in Computational Statistics*, eds H. Caussinus, P. Ettinger and R. Tamassone, pp. 447–450. Vienna: Physica-Verlag. pages 3

Witten, I. H. and Frank, E. (2000) *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco: Morgan Kaufmann. pages 2

Index

Entries in `this font` are names of S objects.