.

Applied Multivariate Analysis

George Michailidis

Department of Statistics
University of Michigan

Table of Contents

Chapter 1: Principal Components Analysis

1. Motivation

Principal components analysis (PCA) belongs to the class of *projection methods*. Such methods choose one or more *linear combinations* of the original variables (features, attributes) to *maximize* some measure of *"interestingness"*. In PCA the goal is to reduce the *dimensionality* of a data set comprised of a large number of *interrelated* variables, while retaining as much as possible of the *variation* present in the data. PCA has also considerable theoretical importance because of its relationship to elliptical distributions and to standard distance.

2. Derivation of Principal Components

Let $\mathbf{x}$ be a vector of p real random variables and let Σ denote their covariance matrix; i.e. $\Sigma = \mathbb{E}(\mathbf{x} - \mu)(\mathbf{x} - \mu)'$. Without loss of generality, we set $\mu = \vec{0}$.

The covariance matrix summarizes all the *linear bivariate* associations in this set of random variables. In PCA, we are interested in reducing the dimensionality p of the original variables, by considering *linear combinations* of them and retaining the *"most important"* of these new variables.

Define a new variable

$$y_1 = \beta_1' \mathbf{x} = \sum_{j=1}^{p} \beta_{j1} x_j, \tag{1}$$

that is a linear combination of the x's with coefficients $\{\beta_{j1}\}$.

The question is how to choose the weights β_1. The choice depends on the measure of "interestingness" that we are going to use. For PCA, the measure of interestingness that we want to maximize is the *variance* of the new variable $\mathbf{y}_1$.

Criterion: $\max_{y_1} \text{Var}\,(y_1) = \max \text{Var}\left(\sum_{j=1}^{p} \beta_{j1} x_j\right) = \max \beta_1' \Sigma \beta_1$.

A moment of reflection shows that the maximum would be achieved for an infinite β_1; thus, a *normalization constraint* must be imposed. The standard constraint employed is $||\beta_1|| = 1$. Hence, the problem becomes

$$\max_{\beta_1} \beta_1' \Sigma \beta_1 \text{ s.t } ||\beta_1|| = 1. \tag{2}$$

The Rayleigh-Rietz theorem indicates that the solution is given by the eigenvector corresponding to the largest eigenvalue λ_1 of Σ.

If a second linear combination $y_2 = \beta_2' x$ is desired, then the problem becomes

$$\max_{\beta_2} \beta_2' \Sigma \beta_2 \text{ s.t } ||\beta_2|| = 1 \text{ and } < \beta_1, \beta_2 >= 0. \tag{3}$$

In general, we can construct p linear combinations $\mathbf{y} = B'\mathbf{x}$, B being a $p \times p$ matrix whose k-th column corresponds to β_k.

The optimal set of weights is given by the eigenvectors of Σ;

$$\Sigma = B\Lambda B'.$$

Notice that the *Principal Components* $\mathbf{y}$ are by construction an *orthonormal linear transformation* of $\mathbf{x}$.

2.1. **Properties of the Principal Components.** .

(1) $\mathbb{E}(\mathbf{y}) = \mathbb{E}(B'\mathbf{x}) = B'\mathbb{E}(\mathbf{x}) = 0$, ex hypothesis.

(2) $\text{Cov}(\mathbf{y}) \equiv \Sigma_y = B'\Sigma B = B'(B\Lambda) = \Lambda$. Hence, the principal components are uncorrelated and $\text{Var}(y_j) = \lambda_j$.

(3) Consider an orthonormal transformation $\mathbf{z} = C'\mathbf{x}$, where $\mathbf{z}$ is a q-element vector and C a $p \times q$ real matrix. Denote by $\Sigma_z = C'\Sigma C$ the variance covariance matrix of $\mathbf{z}$. Then, $\mathbf{trace}(\Sigma_z)$ is maximized by setting $C = B_q$, where B_q contains the first q columns of B.

Denote by $\mathbf{c}_k$ the $k-$th column of C. Notice that the columns of B form a basis in $p-$dimensional space. Therefore, we can write

$$\mathbf{c}_k = \sum_{j=1}^{p} w_{jk}\beta_j, \ k = 1, 2, \cdots, q, \tag{4}$$

with $\{w_{jk}; \ j = 1, \cdots, p, \ k = 1, \cdots, q\}$ appropriately defined constants. We then get that $C = BW$, where $W = \{w_{jk}\}$. Notice that $W = B'C$ and therefore $W'W = C'BB'C = C'C = I_q$, since B is orthogonal by construction and C orthonormal ex hypothesis. Some algebra shows that

$$\Sigma_z = C'\Sigma C = W'B'\Sigma BW = W'B'(B\Lambda B')BW = W'\Lambda W. \tag{5}$$

Hence, $\mathbf{trace}(\Sigma_z) = \sum_{j=1}^{p}\sum_{k=1}^{q} \lambda_j w_{jk}^2$. The fact $W'W = I_q$ implies that $\sum_{j=1}^{p}\sum_{k=1}^{q} w_{jk}^2 = q$ and the columns of W are orthonormal. Hence, $\sum_{k=1}^{q} w_{jk}^2 \leq 1$.

But $\sum_{k=1}^{q} w_{jk}^2$ corresponds to the coefficient of λ_j in $\mathbf{trace}(\Sigma_z)$, their sum is equal to q and they are smaller or equal than 1. The ordering of the eigenvalues $\lambda_1 > \lambda_2 > \cdots > \lambda_p$ implies that $\sum_{j=1}^{p}(\sum_{k=1}^{q} w_{jk}^2)\lambda_j$ would be maximized if we can identify a set of w_{jk} for which $\sum_{k=1}^{q} w_{jk}^2 = 1, \ j = 1, \cdots, q$ and 0 otherwise. Setting $C' = B_q'$ satisfies the above and therefore maximizes $\mathbf{trace}(\Sigma_z)$.

An analogous derivation shows that $\mathbf{trace}(\Sigma_z)$ is minimized by setting $C = \tilde{B}_q$, where $\tilde{B}_q$ contains the last $q-$columns of B.

(4) Correlations of the original random variables $\mathbf{x}$ with the principal components $\mathbf{y}$.
We have that $\mathbf{Cov}(\mathbf{x}, \mathbf{y}) = \mathbf{Cov}(\mathbf{x}, B'\mathbf{x}) = \Sigma B = B\Lambda$. This shows that for a fixed y_j we have $\mathbf{Cov}(y_j, \mathbf{x}) = \lambda_j \beta_j$. Hence, $\mathbf{Corr}(x_i, y_j) = \lambda_j \beta_{ji} / \sqrt{\lambda_j}$. The quantities $\sqrt{\lambda_j}\beta_j$ are called *factor loadings*.

(5) Proportion of variance explained by the principal components.
Notice that $\mathbf{trace}(\Sigma) = \mathbf{trace}(B\Lambda B') = \mathbf{trace}(\Lambda B' B) = \mathbf{trace}(\Lambda)$. Thus, each principal component "explains" $\lambda_j / (\sum_{j=1}^{p} \lambda_j)$ proportion of the total variance.

(6) Consider the set of p-dimensional ellipsoids of the form

$$\mathbf{x}'\Sigma^{-1}\mathbf{x} = c,$$

Then, the principal components define the principal axes of these ellipsoids.

The set of principal components is given by $\mathbf{y} = B'\mathbf{x}$. Because of the fact that B is orthogonal, we get $\mathbf{x} = B\mathbf{y}$. Plugging this in we obtain

$$\mathbf{x}'\Sigma^{-1}\mathbf{x} = \mathbf{y}'B'\Sigma^{-1}B\mathbf{y} = c.$$

But the eigenvalue decomposition of Σ^{-1} gives

$$\Sigma^{-1} = B\Lambda^{-1}B',$$

i.e. the inverse of the covariance matrix has the same set of eigenvectors as the covariance matrix and the reciprocals of its eigenvalues (provided that Σ is strictly positive definite). Some algebra shows that $B'\Sigma^{-1}B = \Lambda^{-1}$ and therefore $\mathbf{y}'\Lambda^{-1}\mathbf{y} = c$. The last expression corresponds to the equation of an ellipsoid referred to its principal axes. It also implies that the half lengths of the principal axes are proportional to $\lambda_j^{1/2}$.

Remark: Notice that if the random vector $\mathbf{x} \sim \mathbf{N}(0, \Sigma)$, then the above defined ellipsoids define contours of constant probability for the distribution of $\mathbf{x}$. The first principal axis of such ellipsoids defines the direction in which statistical variation is greatest. The second one, defines the direction of maximum statistical variability, subject to an orthogonality constraint and so on. This property is relevant even without the normality assumption.

2.2. **Principal Components based on the Correlation Matrix.** .

Let $\tilde{\mathbf{x}} = (x_j/\sigma_j)$, where σ_j^2 is the variance of x_j. We can define the principal components of $\tilde{\mathbf{x}}$ as before; namely,

$$\tilde{\mathbf{y}} = \tilde{B}'\tilde{\mathbf{x}}, \tag{6}$$

where $\tilde{B}$ contains the eigenvectors of the correlation matrix of the $\tilde{x}_j$'s.

Remark: It has been proposed in the literature to consider principal components of other rescaled versions of the x_j's of the form x_j/w_j. The properties of the principal components discussed in the previous section carry over with some minor adjustments.

Remark: It is worth noting that the principal components derived from the covariance and the correlation matrix are not the same, except in very special circumstances. This is because the principal components are *invariant* under *orthogonal* transformations of $\mathbf{x}$ but not under other transformations. However, the transformation from $\mathbf{x}$ to $\tilde{\mathbf{x}}$ is not orthogonal.

Remark: The main advantage of using the correlation matrix in PCA is that the results are not sensitive to the units of measurement for $\mathbf{x}$. Consider the following example involving two variables x_1 and x_2: the covariance matrix is given by

$$\Sigma_1 = \begin{pmatrix} 0.75 & 0.60 \\ 0.60 & 0.73 \end{pmatrix}$$

The eigenvalues are $\lambda_1 = 1.34$ and $\lambda_2 = 0.14$ and the eigenvectors are

$$B = \begin{pmatrix} -0.71 & 0.70 \\ -0.70 & -0.71 \end{pmatrix}$$

It can be seen that the two variables participate with equal weights in the first principal component. Suppose that x_1 was measured in different units (say seconds instead of minutes), so that the covariance matrix becomes

$$\Sigma_2 = \begin{pmatrix} 2715 & 3.60 \\ 3.60 & 0.07 \end{pmatrix}$$

The eigenvalues are $\lambda_1 = 1.34$ and $\lambda_2 = 0.14$ as before, but the eigenvectors are now

$$B = \begin{pmatrix} -0.99 & 0.01 \\ -0.01 & -0.99 \end{pmatrix}$$

In this case, the first component is dominated by the first variable. If on the other hand, the analysis was based on the correlation matrix, we would have obtained identical results.

PCA analysis based on the correlation matrix satisfies the following property. Let $\mathbf{z} = C'\mathbf{x}$ be an orthonormal transformation with C a $p \times q$ matrix. Let R_j^2 denote the squared multiple correlation coefficient between the variable x_j and $\mathbf{z}$. The criterion $\sum_{j=1}^{p} R_j^2$ is maximized when $C = \tilde{B}_q$; i.e. the $\mathbf{y}$ corresponds to the principal components associated with the largest q eigenvalues of the correlation matrix.

Since the principal components are uncorrelated (by construction), the criterion reduces to $\sum_{j=1}^{p} \sum_{k=1}^{q} r_{jk}^2$, where r_{jk}^2 is the squared correlation between principal component k and variable j. The criterion would be maximized by any matrix C that produces $\mathbf{z}$ that spans the same $q-$dimensional space as the first q principal components. The interesting thing about PCs based on the correlation matrix is that they maximize the criterion successively for $q = 1, \cdots, p$.

2.3. **Some Special Cases.** We examine next the following two special cases:

(1) Equality of eigenvalues ($\lambda_i = \cdots = \lambda_j,\ 1 \leq i \leq j \leq p$). In this case, the corresponding eigenvectors span a certain unique $(j-i)$-dimensional subspace, but within that subspace they are arbitrary. Geometrically the principal axes of a hypersphere can not be uniquely defined.

It is worth noting that equality of the eigenvalues in the covariance matrix need not imply equality of the eigenvalues in the correlation matrix.

(2) The presence of zero eigenvalues ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_q = 0,\ 1 \leq q \leq p$). In this case, the covariance matrix Σ is singular and with rank equal to $(p-q)$. Therefore, the principal components associated with the zero eigenvalues define an exact linear relationship between the $\mathbf{x}'$s. In such a case, we only have to consider the remaining $(p-q)$ variables. This property also holds for correlation matrices.

3. Sample Principal Components

We turn our attention to PCA applied to data. Let X be an $n \times p$ data matrix. Further, assume that the observations are *independent* replicates from the p-dimensional random vector $\mathbf{x}$. We will denote the columns (variables) of the data matrix by $X_1, X_2. \cdots, X_n$. Without loss of generality assume that Mean(X_i)=0. Then, the *sample covariance* matrix is given by $S = X'X/(n-1)$.

Let $S = B\Lambda B'$ be the eigenvalue decomposition of the sample covariance matrix. Then, the sample principal components are defined by

$$Y = XB, \tag{7}$$

where Y is a $n \times p$ matrix containing PC scores, and B a $p \times p$ orthonormal matrix containing the weights. In direct analogy to the population version of PCA the elements of the $p \times p$ diagonal matrix Λ are the variances of the p principal components; i.e. $\mathbf{Var}(Y_j) = \lambda_j,\ j = 1, \cdots, p$.

Remark: The properties derived in section 2.1 carry over, with the sample covariance matrix S replacing the population covariance matrix Σ. One small difference arises in property (6). The ellipsoids $\mathbf{x}'S^{-1}\mathbf{x} = c$ do not represent contours of constant probability; rather, they correspond to contours of equal Mahalanobis distance from the sample mean, which it is assumed to be 0 (the data have been centered). In the literature, the following interpretation of PCA has appeared; namely, PCA finds successive orthogonal directions for which the Mahalanobis distance from the data set to a hypersphere enclosing all the data is minimized.

The next geometric property of sample principal components gives another interpretation of PCA. In fact, it is a generalization of an idea used by Karl Pearson back in 1901 to define PCA for 2-dimensional data.

Consider n observations denoted by $X(i,\cdot)$ ($X(i,\cdot)$ being a $p-$dimensional column vector). Let C be an orthogonal $p \times q$ matrix and let $Y(i,\cdot) = C'X(i,\cdot)$ be the projection of the data onto a $q-$dimensional subspace. A goodness-of-fit measure of this $q-$dimensional subspace to $X(1,\cdot), \cdots, X(n,\cdot)$ is given by the sum of *squared perpendicular distances* of the $X(i,\cdot)$'s from the subspace. This measure is *minimized* when $C = B_q$.

The vector $Y(i,\cdot)$ is an orthogonal projection of $X(i,\cdot)$ onto a $q-$dimensional subspace defined by the matrix C. Let $\mathbf{v}_i$ denote its position on the subspace and define $\mathbf{r}_i = X(i,\cdot) - \mathbf{v}_i$ (think of $\mathbf{v}_i$ as a perpendicular to the subspace residual). Notice that $\mathbf{r}_i$ is orthogonal to the subspace, so that $< \mathbf{r}_i, \mathbf{v}_i >= 0$. Moreover, $\mathbf{r}_i'\mathbf{r}_i$ gives the squared perpendicular distance of $X(i,\cdot)$ from the subspace. Hence, the goodness-of-fit measure under consideration is given by

$$\sum_{i=1}^{n} \mathbf{r}_i'\mathbf{r}_i. \tag{8}$$

Some algebra (in analogy to the Pythagorean theorem in 2 dimensions) gives that

$$X(i,\cdot)'X(i,\cdot) = \mathbf{r}_i'\mathbf{r}_i + \mathbf{v}_i'\mathbf{v}_i. \tag{9}$$

Therefore,

$$\sum_{i=1}^{n} \mathbf{r}_i'\mathbf{r}_i = \sum_{i=1}^{n} X(i,\cdot)'X(i,\cdot) - \sum_{i=1}^{n} \mathbf{v}_i'\mathbf{v}_i. \tag{10}$$

Since the first term in the right hand side is fixed, minimization of the goodness-of-fit measure is equivalent to maximization of $\sum_{i=1}^{n} \mathbf{v}_i'\mathbf{v}_i$. But distances under orthogonal transformations are preserved, which implies that it is equivalent to maximizing

$$\sum_{i=1}^{n} Y(i,\cdot)'Y(i,\cdot) = \sum_{i=1}^{n} X(i,\cdot)'CC'X(i,\cdot) = \mathbf{trace} \sum_{i=1}^{n} \big(X(i,\cdot)'CC'X(i,\cdot)\big)$$

$$= \sum_{i=1}^{n} \mathbf{trace}\big(X(i,\cdot)'CC'X(i,\cdot)\big) = \mathbf{trace}\big(C'\big(\sum_{i=1}^{n} X(i,\cdot)X(i,\cdot)'\big)C\big) =$$

$$\mathbf{trace}\big(C'\big[\sum_{i=1}^{n} X(i,\cdot)X(i,\cdot)\big]'C\big) = \mathbf{trace}(C'X'XC) = (n-1)\mathbf{trace}(C'SC).$$

But the third property in section 2.1 establishes the result.

The plot in Figure 1 schematically illustrates this property.

Remark: The problems discussed for the population version of PCA with equal and/or zero variances of the principal components carry over to the sample version. Of course, exactly equal or zero eigenvalues are extremely rare, but when you spot nearly equal or zero eigenvalues in your analysis you need to deal with this issue.

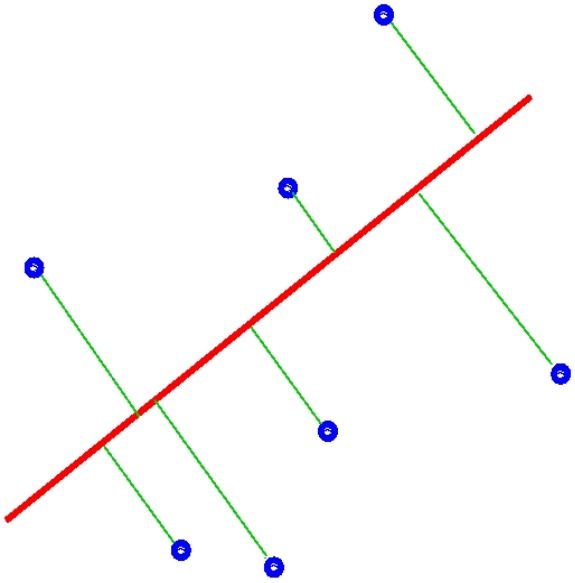


FIGURE 1. Geometry of PC lines; PC line depicted in red, data points in blue and their orthogonal projections to the PC line in green

3.1. **An Application: City crime data.** In this example, we analyze using PCA the crime data set. The data give crime rates per 100,000 people for the 72 largest US cities in 1994.

The variables are:

```
1) Murder
2) Rape
3) Robbery
4) Assault
5) Burglary
6) Larceny
7) Motor Vehicle Thefts
```

A scatterplot matrix of the data is given in Figure 2.

The first 3 PCs account for 54%, 17% and 11% of total variance respectively, and in total for 82%. So, it suffices to look at a 2-dim or a 3-dim representation of the results (the *screeplot* of the eigenvalues/variances of the PCs is given in Figure 3).

```
                        Comp. 1  Comp. 2   Comp. 3   Comp. 4
Standard deviation      1.948    1.095     0.877     0.714
Proportion of Variance  0.54     0.17      0.11      0.073
Cumulative Proportion   0.54     0.71      0.82      0.89
```

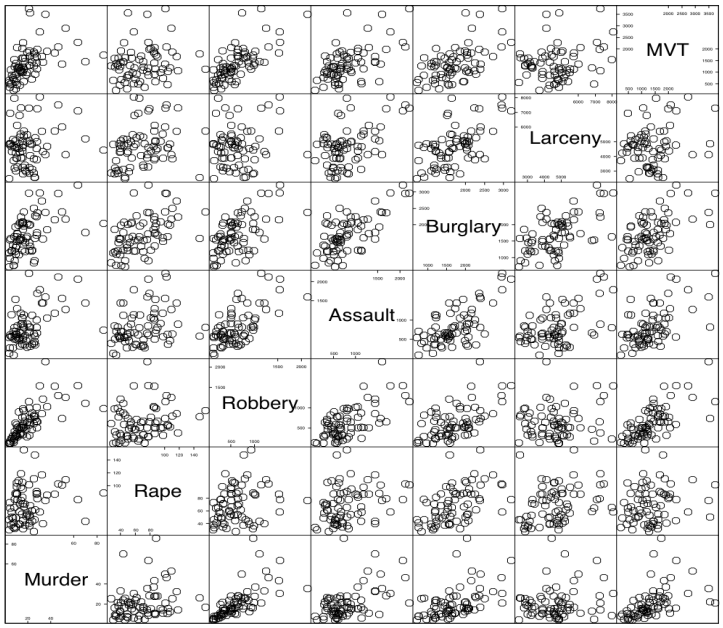


FIGURE 2. Scatterplot matrix of the city crime data

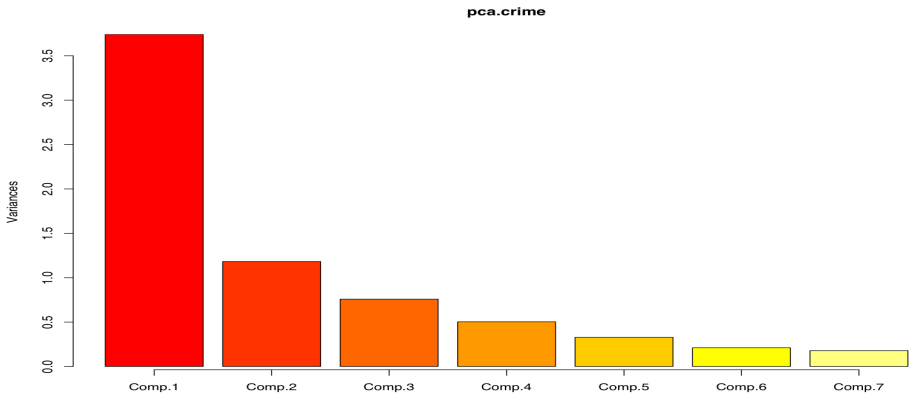


FIGURE 3. Screeplot of eigenvalues of crime data

The optimal weights (the matrix B) of the crime variables on the first 3 components are given next.

	PC1	PC2	PC3
Murder	0.370	-.339	0.202
Rape	0.249	0.466	0.782
Robbery	0.426	-.387	0.079
Assault	0.434	0.042	-.282
Burglary	0.449	0.238	0.015
Larceny	0.276	0.605	-.492
MVT	0.390	-.302	-.134

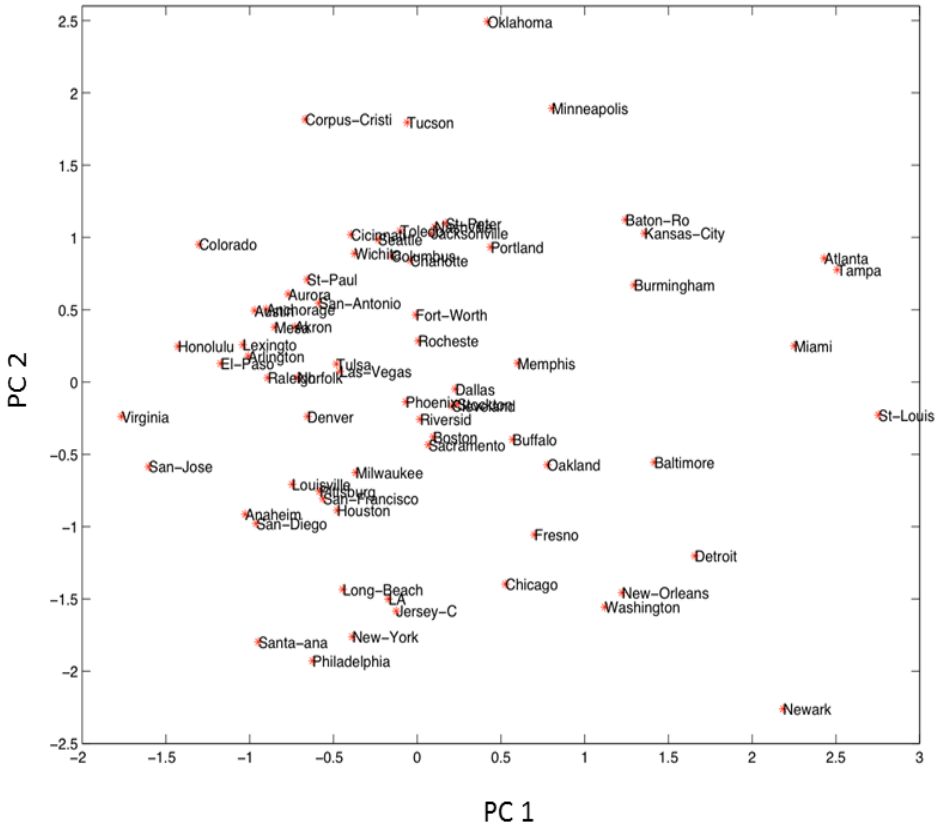


FIGURE 4. Projection of the cities to the subspace spanned by the first two principal components

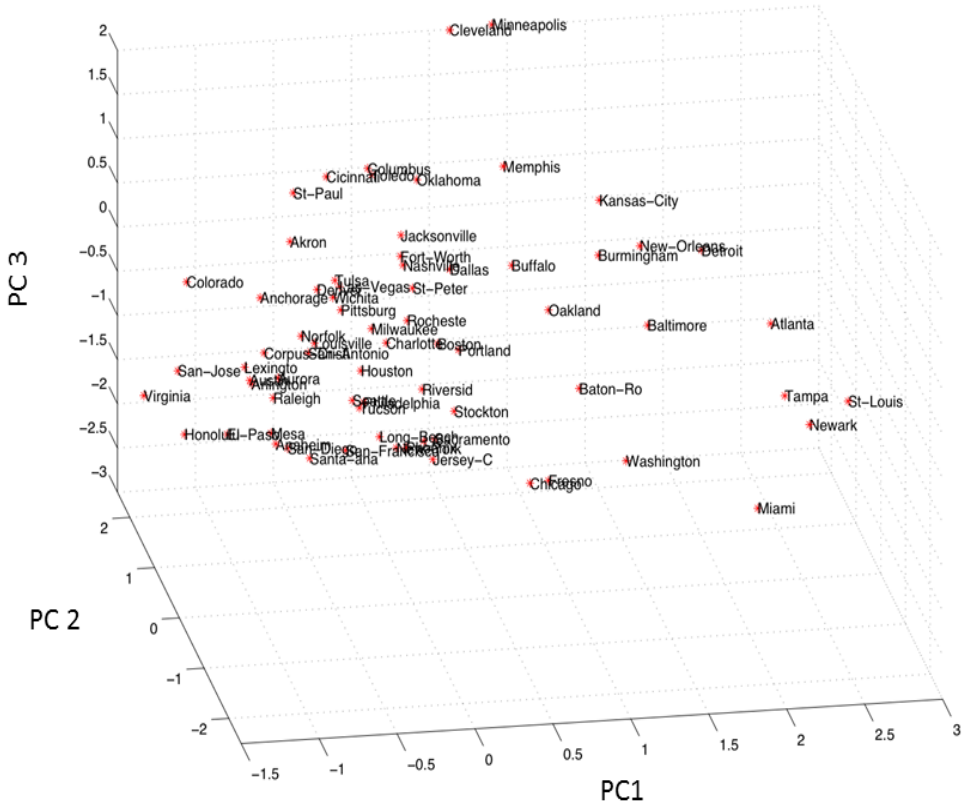


FIGURE 5. Projection of the cities to the subspace spanned by the first three principal components

Looking at these results, it can be seen that the first component can be interpreted as an overall measure of crime activity and examining the corresponding plot of the first 2 PCs we see that cities such as St. Louis, Atlanta, Tampa Bay, Newark, Detroit, Miami, etc. can be characterized as ”dangerous,” while cities such as Virginia Beach, San Jose, Colorado, Honolulu, etc. as ”safe” (recall that these results correspond to 1994 data). The second PC distinguishes between cities with high rape and larceny incidents (and to some degree burglaries) and cities with high murder, robbery and MVT incidents. So, on the bottom of the picture we find cities such as Newark, Jersey City, Philadelphia, Santa Ana, Detroit, Washington DC, NYC, Chicago and Long Beach, characterized by relatively more murder, robbery and MVT crimes, while in Oklahoma, Corpus Cristi, Tucson and Minneapolis rapes and larcenies are more frequent. However, you should be careful on how far you

should go with such an interpretation. It is safe to make such statements for Newark and Detroit, which score high on the first component as well. But the story is not that clear between Washington and Santa Ana, since Santa Ana according to its score on the first component is a relatively "safe" city. On the other hand, the second component allows you to distinguish between Corpus Cristi and Santa Ana, that score similarly on the first component. So, the plot tells you that there are many more rapes in Corpus Cristi compared to Santa Ana or NYC, while more murders in the latter two cities. Finally, the third PC basically contrasts cities with lots of rapes vs cities with lots of larcenies, but since it accounts for only 10% of the total variance, you should be cautious when interpretating the findings based on this component.

3.2. **PCA and the SVD.** Given an $n \times p$ centered data matrix X of full column rank we can apply the Singular Value Decomposition and obtain

$$X = ULB', \tag{11}$$

where U and B are $n \times p$ and $p \times p$ orthonormal matrices and L a $p \times p$ diagonal matrix.

Some straightforward algebra shows that

$$(n-1)S = X'X = BLU'ULB' = BL^2B'. \tag{12}$$

Hence, the matrix B contains the eigenvectors of the sample covariance matrix, while a rescaled version of L its eigenvalues. Specifically $\mathbf{Var}(y_i) = \ell_i^2/(n-1)$. Further, notice that the PC scores $Z = XB$ are given by $Z = XB = ULB'B = UL$.

It is also of interest the fact that $XX' = ULB'BLU' = UL^2U$; i.e. the matrix U contains the eigenvectors of XX'. This result comes in handy in some applications where the role of 'variables' and 'observations' is reversed.

The main interest in the SVD is that in addition to providing a computationally efficient way to do PCA, it also gives additional insight into the technique. For example, let $\tilde{X}_q = U_q L_q B_q'$ be the $n \times q$ matrix formed by retaining the q largest in terms of variance principal components. Then, the Householder-Young theorem implies that $\tilde{X}_q$ gives the best $q-$rank approximation of the data in the Frobenius norm; i.e. $\mathbf{argmin}_Q ||X - Q||_F = \tilde{X}_q$.

3.3. **Biplots:** The SVD also provides an efficient way of visualizing together objects (observations) and variables. The emphasis of the PCA of the city crime data was on cities (objects). However, a graphical representation of the data that contains some information about the variables is particularly useful. One such representation is given by the *biplot*, introduced by Gabriel in 1971.

Let $X = ULB'$ denote the SVD of a centered $n \times p$ full column rank data matrix. Now define a $p \times p$ diagonal matrix L^γ with elements $\ell^\gamma,\ 0 \leq \gamma \leq 1$. We can then write X as

$$X = ULB' = UL^\gamma L^{1-\gamma} B' = GH', \tag{13}$$

with $G = UL^{\gamma}$ and $H' = L^{1-\gamma}B'$. Hence, $X(i,j) = g_i'h_j$, with g_i' being the $i-$th row of G and h_j' the $j-$the row of H, respectively.

When a 2-rank approximation $\tilde{X}_2$ works well, we can use this decomposition to project both variables and objects onto two dimensional space. The two most interesting values for γ are 0 and 1.

(1) When $\gamma = 0$, then $G = U$ and $H' = LB'$. In this case we get after some algebra $X'X = HH'$. The product $h_k'h_j$ is $(n-1)S(k,j)$, while the length $h_j'h_j$ is proportional to the variance of the variables. Hence, the cosines (angles on a plot) between the h_j's provide information about the correlation of the variables.

In addition, let $d_{ij}^2 = (X(i,\cdot) - X(j,\cdot))'S^{-1}(X(i,\cdot) - X(j,\cdot))$ denote the Mahalanobis distance between observations i and j. Write $X(i,\cdot)' = g_i'H'$ and compute

$$d_{ij}^2 = (g_i - g_j)'H'S^{-1}H(g_i - g_j) = (n-1)(g_i - g_j)'LB'(X'X)^{-1}BL(g_i - g_j) = (n-1)(g_i - g_j)'L(B'B)L^{-2}(B'B)L(g_i - g_j) = (n-1)(g_i - g_j)'(g_i - g_j).$$

Therefore, the Mahalanobis distance between observations is a multiple of the Euclidean distance between the g_i's.

(2) When $\gamma = 1$, we have $G = UL$ and $H' = B'$. Analogous calculations show that the Euclidean distance between the g_i's is proportional to the Euclidean distance between the observations. Further, the angle between the variables still provides qualitative information about their correlations.

(3) Finally, for other values of $\gamma \in (0,1)$, a similar qualitative interpretation of the resulting biplot holds, but the exact properties discussed above are no longer valid.

We discuss next the biplot (Figure 6) of the city crime data, in order to look at a joint representation of data points and variables. The arrows on the biplot indicate the directions along which one can locate cities with high values on a particular variable. Note that on this plot, Virginia Beach that was deemed a "safe" city in our discussion of the results (refer to Figure 4) of this data set, is located on the right end of the plot, while St. Louis (a "dangerous" city) on the left. This is due to the *rotational invariance* of the PCA solution; namely, the signs of the loadings of the first PC have been reversed, but the interpretation of the results does not change. Keeping this in mind, we see that V3 (rape) and V7 (larceny) point towards Minneapolis and Oklahoma, which is consistent with the discussion above.

3.4. **Some Practical Considerations.** .

(1) *Number of principal components to retain.* The mathematics of PCA does not provide a direct solution to this issue. However, several suggestions have appeared over the year in the literature. The most popular heuristic criteria for deciding how many principal components to keep are:

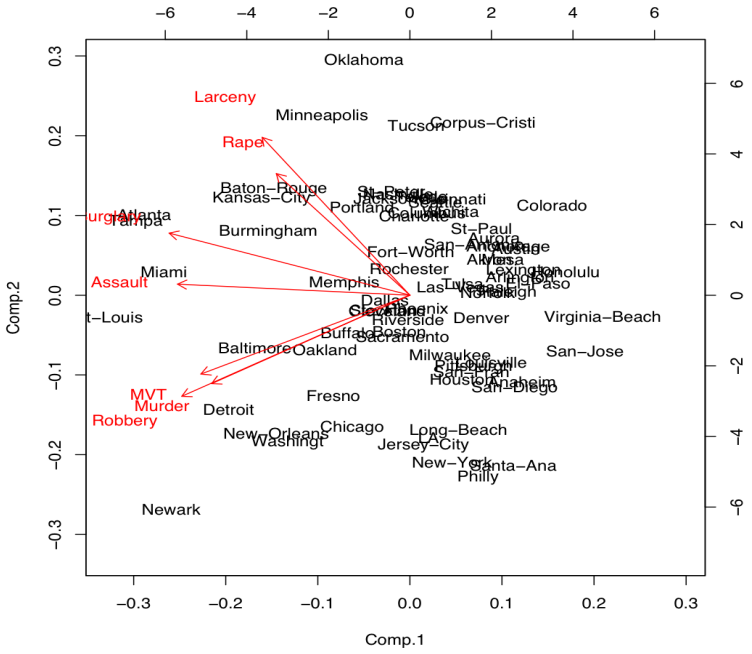


FIGURE 6. Biplot of city crime data

(i) Scree plot [Cattell (1966)]
(ii) Use the cutoff point 1 (i.e. $\lambda_j < 1$) suggested by [Kaiser (1968)]
(iii) Use the cutoff point .7 (i.e. $\lambda_j < .7$) suggested by Jolliffe (1971)]
(iv) Pick enough PCs to explain 70-90% of the variation in the data
None of these criteria has a rigorous theoretical justification, but some of them have been supported by simulation studies.

A different approach based on cross-validation ideas borrowed from regression analysis is advocated by Wold (1978). We have seen that the best $q-$rank approximation of X is given by $\tilde{X}_q = U_q L_q B_q'$. Let X^m denote the data matrix based on a subset of the observations and let $\tilde{X}_q^m$ be its best $q-$rank approximation. Specifically,

$$\tilde{X}_q^m(i,j) = \sum_{k=1}^{q} \hat{U}(i,k)\hat{L}(k,k)\hat{B}(j,k),$$

where $\hat{U}(i,k), \hat{L}(k,k), \hat{B}(j,k)$ are calculated from suitable subsets of the data. Then, the sum of squared differences between predicted and observed values is

given by

$$P(q) = \sum_{i=1}^{n}\sum_{j=1}^{p}(\tilde{X}_q^m(i,j) - X(i,j))^2.$$

Wold proposes to break the data set into g blocks and estimate the above quantities with the data of the h-th block deleted and its observations subsequently predicted.

He further suggests to examine the ratio

$$R = \frac{P(q)}{\sum_{j=1}^{p}(\tilde{X}_{q-1}^m(i,j) - X(i,j))^2}.$$

If $R < 1$ then it is implied that a better prediction is achieved by including the $q-$th principal component.

Another test statistic was proposed by Eastment and Krzanowski (1982);

$$W = \frac{(P(q-1) - P(q))/d_1}{P(q)/d_2},$$

where d_1, d_2 denote the degrees of freedom of the numerator and denominator, respectively. If $W > 1$, then the $q-$the principal component needs to be included in the analysis.

(2) *Selecting a subset of variables*. PCA does not quite reduce the dimensionality of the data, since there are as many principal components as number of variables. The technique, however, can help us *discard* some variables by keeping only those with the highest factor loadings.

(3) *Outliers and influential observations in PCA*. Since the extraction of PCs is based on variances, the presence of outliers can have a significant impact on the results. However, detecting outliers in high-dimensional data is a notoriously difficult task. An alternative is to try to use a more *robust* version of the correlation/covariance matrix, but this is computationally expensive (defining multivariate versions of univariate robust to outliers statistics, such as the median, is far from trivial). Another alternative approach can be based in finding a projection that maximizes a robust measure of variance in p dimensional space.

4. Inference for Sample Principal Components

Assume that the data $\mathbf{x} \sim \mathbf{N}(0, \Sigma)$, with Σ *known*. Then,

$$(n-1)S \sim \mathbf{W}_p(\Sigma, n-1), \tag{14}$$

where $\mathbf{W}$ denotes the Wishart distribution with parameters Σ and $(n-1)$. Therefore, by examining sampling properties of eigenvectors and eigenvalues of Wishart distributed random variables is equivalent to looking at sampling properties of the coefficients and variances of the sample principal components.

The Wishart distribution has been thoroughly investigated over the years due to its connection with various multivariate analysis problems. The density function of a $p \times p$ real symmetric positive definite matrix $A \sim \mathbf{W}_p(\Sigma, n-1)$ is given by

$$c|A|^{(n-p-2)/2} \exp\left(-1/2\mathbf{trace}(\Sigma^{-1}A)\right),$$

with

$$c^{-1} = 2^{p(n-1)/2}|A|^{(n-1)/2}\pi^{p(p-1)/4}\prod_{j=1}^{p}\Gamma((n-j)/2).$$

In order to get a feeling about the Wishart distribution, the following two facts (proofs given in Srivastava and Khatri, 1973) prove useful.

Fact 1: If $A \sim \mathbf{W}_p(\Sigma, n-1)$ and C is a $q \times p$ real matrix of rank q, then $CAC' \sim \mathbf{W}_q(C\Sigma C', n-1)$.

Fact 2: If $\mathbf{z} \neq 0$ is a $p-$dimensional vector and $A \sim \mathbf{W}_p(\Sigma, n-1)$, then $\mathbf{z}'A\mathbf{z} \sim \sigma_z^2\chi^2_{n-1}$, where $\sigma_z^2 = z'\Sigma z$. With a special $\mathbf{z}' = (0, \cdots, 0, 1, 0, \cdots, 0)$ we get $A(j,j) \sim \Sigma(j,j)\chi^2_{n-1}$.

We need to make the following additional technical assumption: all eigenvalues of Σ are distinct and strictly positive (i.e. $\lambda_1 > \lambda_2 > ... > \lambda_p$). Let $\hat{\lambda}$ denote the vector of estimated eigenvalues from our large sample and λ their corresponding population values. Similarly, let $\hat{\beta} = (\hat{\beta}_1, \cdots, \hat{\beta}_p)$ denote the estimated eigenvectors from the data and $\beta = (\beta_1, \cdots, \beta_p)$ their population values. Further, $\hat{\beta}_j(m), j = 1, \cdots, p,\ m = 1, \cdots, p$ denotes the $m-$th component of the j-th estimated eigenvector and analogously for the corresponding population value. We then have the following *asymptotic* results:

(1) $\mathbb{E}(\hat{\lambda}) = \lambda,\ \mathbb{E}(\hat{\beta}_j) = \beta_j$.
(2) $\mathbf{Cov}(\hat{\lambda}_i, \hat{\lambda}_j) = 0,\ \mathbf{Var}(\hat{\lambda}_j) = 2\lambda_j^2/(n-1)$.
(3) $\mathbf{Cov}(\hat{\beta}_j(k), \hat{\beta}_i(m)) = -(\lambda_j\lambda_i\beta_j(k)\beta_i(m))/[(n-1)(\lambda_j - \lambda_i)^2]$ and
$\mathbf{Cov}(\hat{\beta}_j(k), \hat{\beta}_j(m)) = (\lambda_j/(n-1))\sum_{i=1}^{p}(\lambda_i\beta_i(k)\beta_i(m))/(\lambda_i - \lambda_j)^2)$
(4) all the $\hat{\lambda}$'s are *independent* of all the $\hat{\beta}$'s.

Remark: Note that in (3) we give results regarding the covariance of the estimated lodings for *two different variables* on *two different principal components* and for *two different variables* on the *same principal component*.

Remark: It should be emphasized that the above results are asymptotic in nature and therefore only approximate for small samples. Exact results are available only for special cases, such as $\Sigma = I$.

Remark: In general, the larger eigenvalues tend to be overestimated and the smaller ones underestimated; hence, more accurate approximations involve a bias correction.

The asymptotic results previously discussed can be used to construct confidence intervals for the parameters of interest. Specifically,

(1) $\hat{\lambda}_j \sim \mathbf{N}(\lambda_j, 2\lambda_j^2/(n-1))$. An alternative is based on the distribution of $\log(\hat{\lambda}_j) \sim \mathbf{N}(\log(\lambda_j), 2/(n-1))$, whose variance does not depend on the unknown population parameter λ_j.

(2) $\sqrt{n}(\hat{\beta}_j - \beta_j) \sim \mathbf{N}(0, V_j)$, where

$$V_j = \frac{\lambda_j}{n-1} \sum_{k=1, k\neq j}^{m} \frac{\lambda_k}{(\lambda_k - \lambda_j)^2} \beta_k \beta_k' \tag{15}$$

The matrix V_j has rank $(p-1)$, since it has a zero eigenvalue associated with the β_j eigenvector, which introduces further complications. However, it can be shown (Mardia et al., 1979) that approximately

$$(n-1)\beta_j'(\hat{\lambda}_j S^{-1} + \hat{\lambda}_j^{-1} S - 2I_p)\beta_j \sim \chi^2_{(p-1)}.$$

4.1. **Computational Inference for PCA.** We discuss next some inference techniques based on *resampling* and *permutation* ideas.

The question we are trying to address is "How accurate is the percentage of variance explained by the first two components?" (i.e. how accurate is $\theta = (\lambda_1 + \lambda_2)/\sum_{i=1}^m \lambda_i$, in case we decided to use the first two principal components. This is the question that the *bootstrap* was designed to answer. Notice that the mathematical complexity going into the computation of θ is irrelevant, as long as we can compute $\theta^\star$ for any *bootstrap data set*.

Let X be the $n \times p$ data matrix. In this case a bootstrap data set is an $n \times p$ data matrix $X^\star$, with the rows of $X^\star$ being a random sample of size n from the rows of the actual data matrix X. Since the rows of $X^\star$ are generated by sampling *with replacement* the rows of X, some of the original observations appear zero times in $X^\star$, some once, some twice, etc., for a total of n rows.

Having generated $X^\star$, we can calculate its covariance (or correlation matrix) $\S^\star(R^\star)$, compute its eigenvalue matrix $\Lambda^\star$ and finally obtain

$$\theta^\star = (\lambda_1^\star + \lambda_2^\star)/\sum_{i=1}^m \lambda_i^\star$$

the bootstrap replication of θ. In Figure 7, the histogram of $B = 10,000$ bootstrap replications $\theta^\star$ is shown, for the city crime data presented in Section 7. The mean value of the 10,000 replications is .745, larger than the observed one in the data of .71. The histogram looks fairly normal and the observed value in the data is contained in the central 95% part of the distribution.

In order to calculate $\theta^\star$, we had to obtain the bootstrap values of Λ. The bootstrap distributions of λ_1 and λ_2 (namely, the percentage of variance explained by the 1st and 2nd PCs,

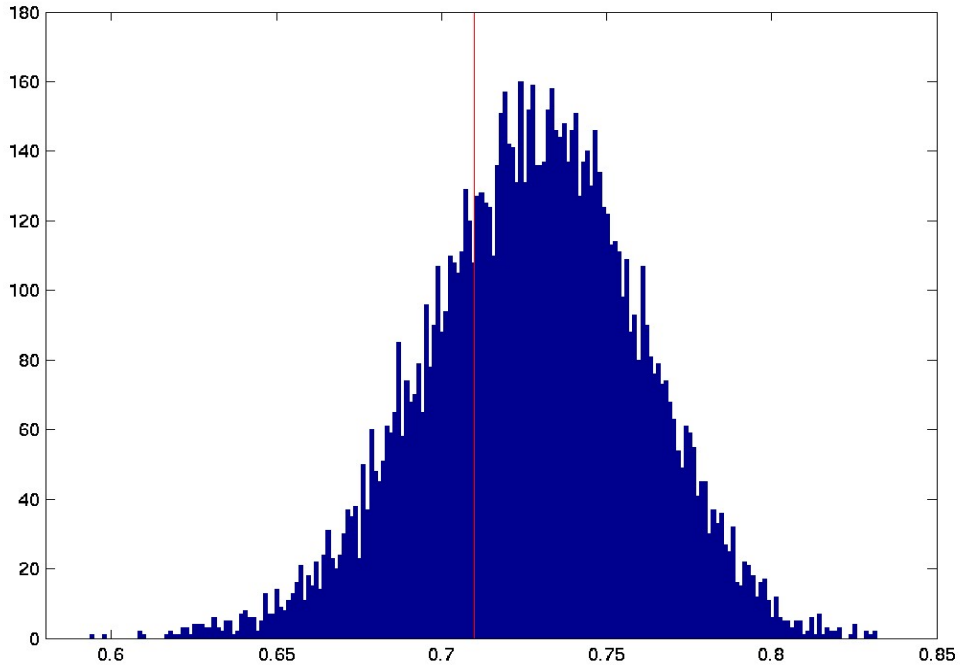


FIGURE 7. Histogram of bootstrap distribution of the statistic θ. The red line indicates the observed value in the actual data set $\hat{\theta}$.

respectively) are shown in Figure 8. Analogously one can calculate bootstrap distributions for the PCs (the eigenvectors of the covariance or correlation matrix).

Another method whose goal is to assess the overall significance of the results obtained from the PCA borrows ideas from permutation tests. Although our exposition of PCA emphasized the exploratory nature of the technique, nevertheless we would like to determine whether the structure observed in the data is "too pronounced to be easily explained away as some kind of fluke", to paraphrase Freedman and Lane (1983). Permutation tests can help to study the concept of "no structure at all." The idea behind using such tests is that they represent a nice way of formalizing the notion of no structure. The random variation is introduced *conditionally on the observed data*, which implies that we do not have to assume a particular model that generated the data, thus making them useful in non-stochastic settings as well. Each new data set $\tilde{X}$ is generated by permuting the values the objects are assigned for each variable, resulting in destroying the original profiles of the objects, and hence their original correlation structure. Then, PCA is applied to $\tilde{X}$ and the eigenvalues $\tilde{\Lambda}$ computed. For small data sets in terms of both objects and variables it is possible to derive the permutation distribution of the eigenvalues by complete enumeration of the possible

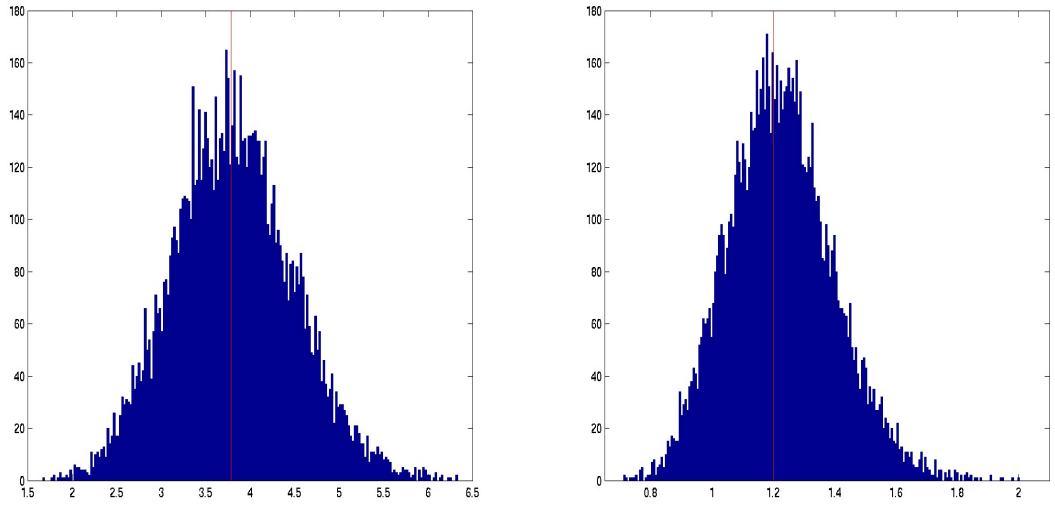


FIGURE 8. Histogram of **bootstrap** distributions of the first (left panel) and second (right panel) largest eigenvalues. The red line indicates the observed value in the actual data set.

cases. However, for all other cases one has to resort to Monte Carlo methods (i.e. do this exercise a large number of times). The histogram of the permutations distributions of the largest two eigenvalues of the city crime data, based on 10000 Monte Carlo replications, are shown in Figure 9. It can easily be seen that the first principal component captures structure in the data, while the second one is more questionable.

5. CONCLUDING REMARKS

PCA is a dimension reduction technique that constructs linear combinations of the original variables. These linear combinations are constructed in such a way, so as to capture the maximum amount of variability in the data.

The criterion of maximum variability leads to utilizing information contained in the covariance (correlation) matrix of the data. This implies that if nonlinear structure exists, PCA is not going to explore it. Further, although the assumption of multivariate normality is not strictly necessary for employing PCA, the technique fully utilizes the available information in such a setting. For other multivariate distributions, higher than second moments contain useful information that is ignored by PCA.

In terms of interpreting the results, in many instances linear combinations of variables that do not measure comparable things may not be meaningful. Hence, although PCA can be

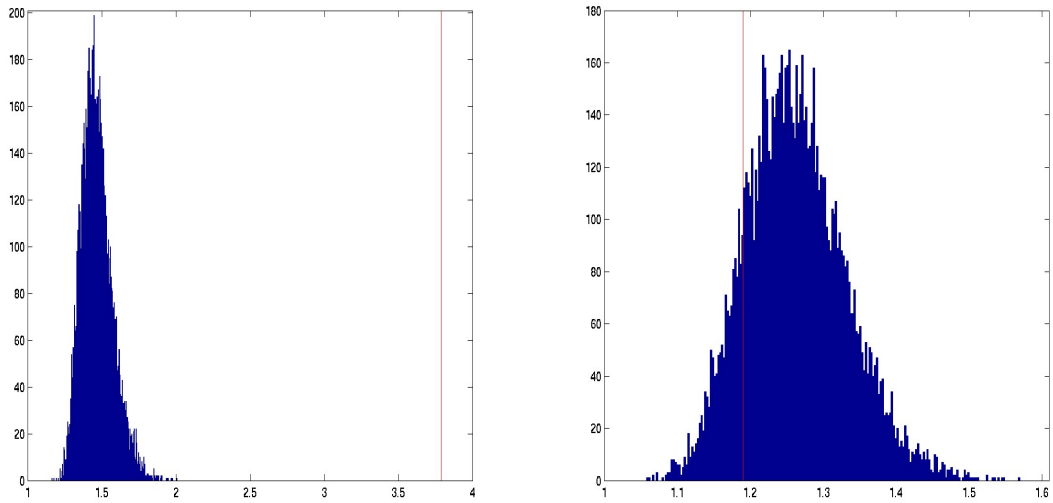


FIGURE 9. Histogram of **permutation distributions** of the first (left panel) and second (right panel) largest eigenvalues. The red line indicates the observed value in the actual data set.

applied to any set of variables, it is most useful when applied to correlated variables representing one or more common domains; for example, indicators of socioeconomic status, job satisfaction, political attitudes, etc.

Finally, extreme outliers may distort the results. However, as pointed out above, determining such outliers is a hard task.

6. EXAMINING ANOTHER DATA SET: ATHLETIC PERFORMANCE

This data set shows the men's athletic records for 55 countries and come from Belsham and Hymans (1984). Records for 100, 200 and 400 meters are in seconds, and for 800, 1500, 5000, 10,000 metros and the marathon are in minutes.

We will see how to produce the necessary output and plots in R.

The next set of commands reads in the data (and stores it in the variable data and also extracts the numerical values stored in numerical.data), extracts the row labels (country abbreviation stored in Abrrevn) to be used subsequently when plotting the results and also creates the variable names (Event).

```
Event <- c("X100","X200","X400","X800","X1500","X5000","X10000","Marathon")
data <- read.table("athletic.txt", col.names=c("Abbrevn","Country",Event))
Abbrevn <- data[1]
```

```
Country <- data[2]
numerical.data <- data[,3:ncol(dat)]
```

Next, we take a look at the scatterplot of the data, shown in Figure 10.

```
pairs(numerical.data,labels=Event)
```

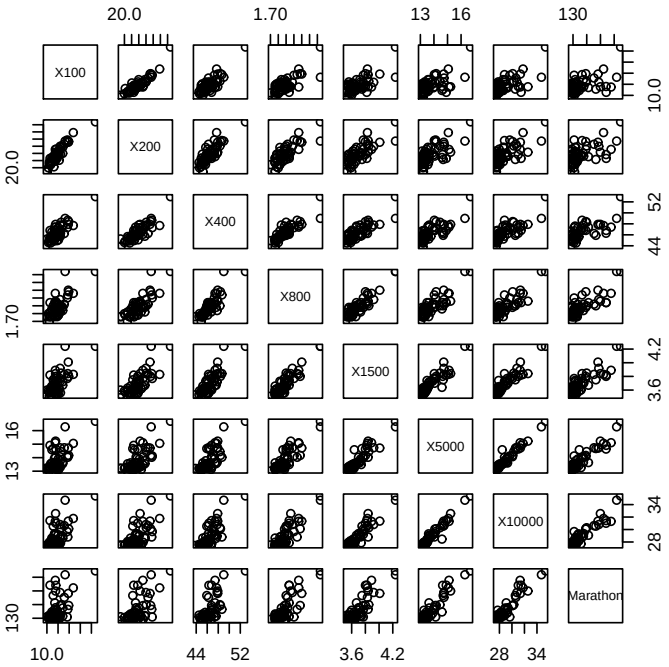


FIGURE 10. Scatter plot of the athletic data set.

Next, we run PCA on the covariance matrix and display the results.

```
athletic.PCAcov <- princomp(numerical.data,cor=F)

summary(athletic.PCAcov)

# shows the contribution of each component

Importance of components:
                          Comp.1     Comp.2      Comp.3      Comp.4
Standard deviation     9.3956859 1.17769448 0.505093087 0.327849412
Proportion of Variance 0.9801097 0.01539866 0.002832439 0.001193346
Cumulative Proportion  0.9801097 0.99550834 0.998340782 0.999534128
```

```
                            Comp.5       Comp.6       Comp.7       Comp.8
Standard deviation     0.1637285643 0.1118690306 4.692618e-02 2.091677e-02
Proportion of Variance 0.0002976231 0.0001389432 2.444827e-05 4.857433e-06
Cumulative Proportion  0.9998317510 0.9999706943 9.999951e-01 1.000000e+00

loadings(athletic.PCAcov)

# Note that R defines loadings without multiplying them by the
square root of eigenvalue
```

```
Loadings:
         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
X100             -0.211        -0.358  0.191  0.887
X200             -0.359        -0.834        -0.410
X400     -0.111 -0.828 -0.378  0.396
X800                                                -0.261  0.965
X1500                                               -0.959 -0.262
X5000            -0.130  0.336        -0.909  0.185
X10000   -0.181 -0.299  0.849  0.135  0.364
Marathon -0.973  0.181 -0.142
```

We repeat the analysis using the correlation matrix instead

```
athletic.PCAcor <- princomp(numerical.data,cor=T)

summary(athletic.PCAcor)

Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
Standard deviation     2.5733517 0.9367424 0.39920904 0.35222270 0.28270507
Proportion of Variance 0.8277673 0.1096858 0.01992098 0.01550760 0.00999027
Cumulative Proportion  0.8277673 0.9374531 0.95737413 0.97288173 0.98287200
                           Comp.6     Comp.7      Comp.8
Standard deviation     0.260809261 0.21542870 0.150309667
Proportion of Variance 0.008502684 0.00580119 0.002824125
Cumulative Proportion  0.991374685 0.99717588 1.000000000

loadings(athletic.PCAcor)
```

```
Loadings:
         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
X100     -0.318 -0.567  0.332 -0.128  0.267  0.592  0.137 -0.106
X200     -0.337 -0.462  0.361  0.259 -0.159 -0.654 -0.113
X400     -0.356 -0.248 -0.561 -0.652 -0.221 -0.156
X800     -0.369        -0.533  0.480  0.540        -0.238
X1500    -0.373  0.140 -0.153  0.405 -0.486  0.161  0.610 -0.140
X5000    -0.364  0.312  0.190        -0.253  0.144 -0.591 -0.547
X10000   -0.367  0.307  0.182        -0.132  0.221 -0.176  0.797
Marathon -0.342  0.439  0.263 -0.300  0.495 -0.320  0.398 -0.158
```

The main point conveyed by the analysis of this particular data set is that the *units in which the variables are measured may influence the results and the conclusions*. It can be seen that if we decide to use the variance-covariance matrix the marathon clearly dominates the first component; in addition, the 400 meters has a much larger influence than most of the remaining track events and dominates the second component. Therefore, the choice of units produces a rather uninformative PC analysis, since the first component captures to a large extent countries' performances (records) in the marathon and the second component in the 400 meters. However, it is not necessary to go through a PC analysis in order to 'rank' countries' records according to these two variables, since the information available in the original data can be used for this purpose.

On the other hand, when we *scale* the variables to have unit variances (thus use the correlation matrix) we get that the first component can be interpreted as a measure of overall performance, while the second component contrasts performance in short distances (100, 200, 400, 800 meters) and in long distances (1500, 5000, 10000 meters and the marathon).

Figure 11 shows the projection of the 55 countries on the first 2-dimensional PC space. To produce this figure (for both the analysis based on the covariance and the correlation matrix of the data) we do

```
athletic.PCcov <- predict(athletic.PCAcov)

athletic.PCcor <- predict(athletic.PCAcor)

plot(athletic.PCcov[,1],athletic.PCcov[,2],type="n")
text(athletic.PCcov[,1],athletic.PCcov[,2],t(Abbrevn))
title("First two PCs from covariance matrix")


plot(athletic.PCcor[,1],athletic.PCcor[,2],type="n")
text(athletic.PCcor[,1],athletic.PCcor[,2],t(Abbrevn))
title("First two PCs from correlation matrix")
```

It can be seen that the relative positions of the countries in the space are affected by the choice of the input (covariance vs correlation) especially when it comes to the second component. On the other hand, the US clearly ranks first in overall performance (1st PC) in both analyses, while the Cook Islands (CI) rank last. However, the rankings of the remaining countries exhibit slight different for the two analyses (see Great Britain (GB), Italy (IT), USSR (UR), etc).

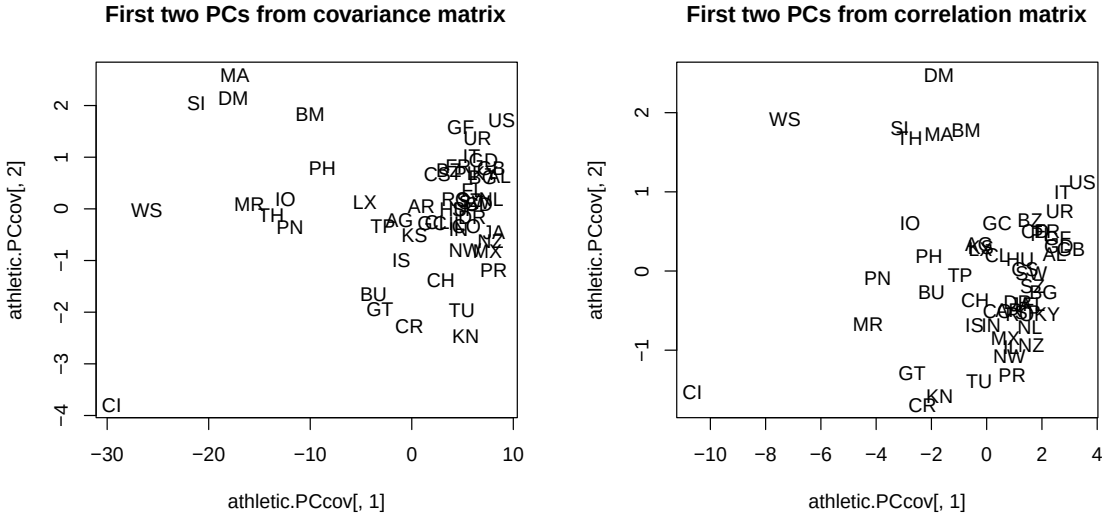


FIGURE 11. First two principal components of the athletic data (left panel: from the covariance matrix; right panel: from the correlation matrix)

Figure 12 gives a 'scree' diagram of the eigenvalues (the variances of the PCs) from the analysis of the correlation matrix. It can be seen that the first component captures over 80% of the total variance, and the first two over 95% of the total variance. Therefore, in this case there is no need to examine the results for the remaining components.

To produce this plot do:

```
barplot(athletic.PCAcor$std)
title("Scree diagram based on the correlation matrix")
```

Finally, in Figure 13 we show the biplot for the analysis based on the correlation matrix that is produced by the command

```
biplot(athletic.PCAcor)
```

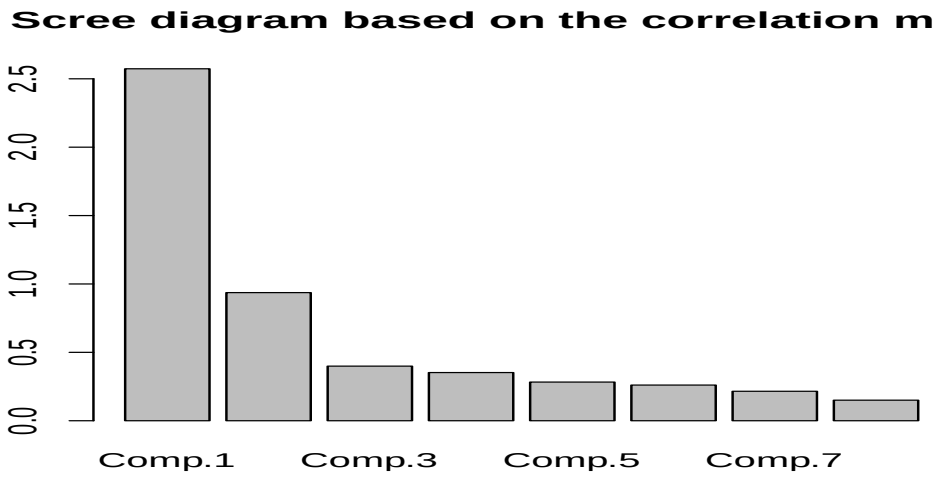


FIGURE 12. Scree diagram of the eigenvalues from the analysis of the correlation matrix of the athletic performance data

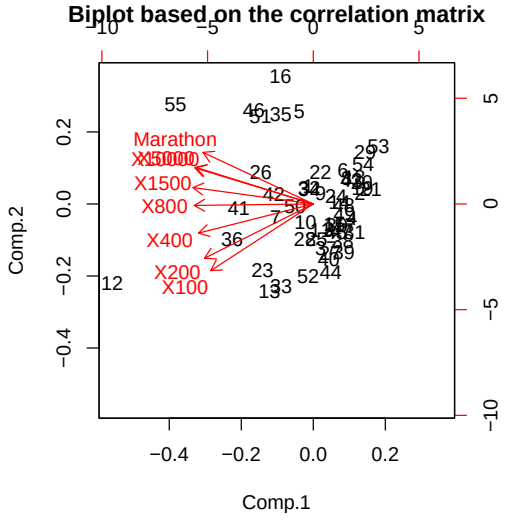


FIGURE 13. Biplot based on the correlation matrix

This data set brings up the following interesting point: should one analyze the covariance matrix or the correlation matrix? Opinions differ on this matter. For example, Krzanowski

(1988, Section 2.2) says, "As a general guideline, however, it would seem sensible to standardize first whenever the measured variables show marked differences in variances, or wherever they concern very different measured entities or units. Unstandardized data are preferable whenever the measured variables are comparable both with respect to units and variances, or when a simple and straightforward data plot is the sole purpose of the analysis." On the other hand, Seber (1984, Section 5.2) says, "Some practitioners maintain that R should always be used instead of Σ, as R does not depend on the scales used for the original variables. This approach would seem reasonable in psychological and educational studies where the scales may be arbitrary and the data little more than ranks. However, the distribution theory associated with R is much more complex and there are some problems in interpreting the coefficients given by the eigenvectors ... About all that can be said is that similar variables should be measured in the same units where possible." Another opinion is voiced by Gnanadesikan (1977, page 12 (quoted with approval by Seber, *ibid.*) who says that there, "does not seem to be any *general* elementary rationale to motivate the choice of scaling of the variables as a preliminary to principal components analysis on the resulting covariance matrix."

7. PCA ANALYSIS OF TEMPORAL DATA: CANADIAN TEMPERATURES

The data represent average monthly temperatures (in degrees Celsius) for 35 weather stations in Canada (this data set has also been analyzed in Ramsey and Silverman's book *Functional Data Analysis*, 1997, New York: Springer). A time series plot of the data is given in Figure 14. It can be seen that the data follow a regular pattern, with colder on average temperatures in the winter and spring months and warmer in the summer and fall months.

We will use PCA to uncover the basic features in this data set.

```
Importance of components:
                          Comp.1    Comp.2     Comp.3
Standard deviation     3.1447615 1.2037888 0.46514235
Proportion of Variance 0.8483661 0.1243107 0.01856007
Cumulative Proportion  0.8483661 0.9726768 0.99123686
```

It can be seen that the first 3 components capture over 99% of the variation in the data set. Moreover, the type of variation captured by the first PC strongly dominates all other types of variation. In order to interpret the components we examine their loadings next.

```
       Comp.1      Comp.2     Comp.3
Jan 0.2724928  0.38864197 -0.1669129
Feb 0.2840193  0.32068851  0.2823290
Mar 0.3024111  0.17776321  0.2837999
Apr 0.3043605 -0.05148195  0.4575479
```

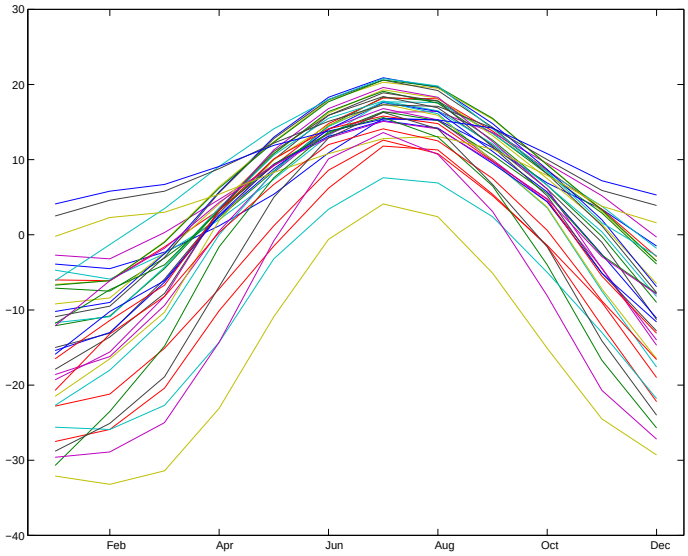


FIGURE 14. Average monthly temperatures in degrees C

```
May 0.2906974 -0.24656985  0.4464331
Jun 0.2663673 -0.41940374  0.1242702
Jul 0.2600561 -0.44041802 -0.2575670
Aug 0.2843447 -0.31342729 -0.3382340
Sep 0.3086486 -0.08632847 -0.2394660
Oct 0.3083248  0.04508856 -0.1589914
Nov 0.2960035  0.22103108 -0.3196341
Dec 0.2812635  0.35302259 -0.1492018
```

The first PC captures the average trend in the data. Hence, weather stations with high scores will have much warmer than average winters combined with warm summers. From figure 2 we see that Vancouver and Victoria receive the highest scores, while Resolute in the high Arctic receives the lowest one. The second PC captures the positive contribution of the winter/spring months and the negative contribution of the summer/fall months, thys corresponding to a measure of uniformity of temperature throughout the year. Low scores go to prairie stations such as Winnipeg that have hot summers and cold winters, while weather stations on the Pacific coast (e.g. Prince Rupert) exhibit very uniform temperatures throughout the year. Finally, the third PC corresponds to a time shift combined with an overall increase in temperature between summer and winter.

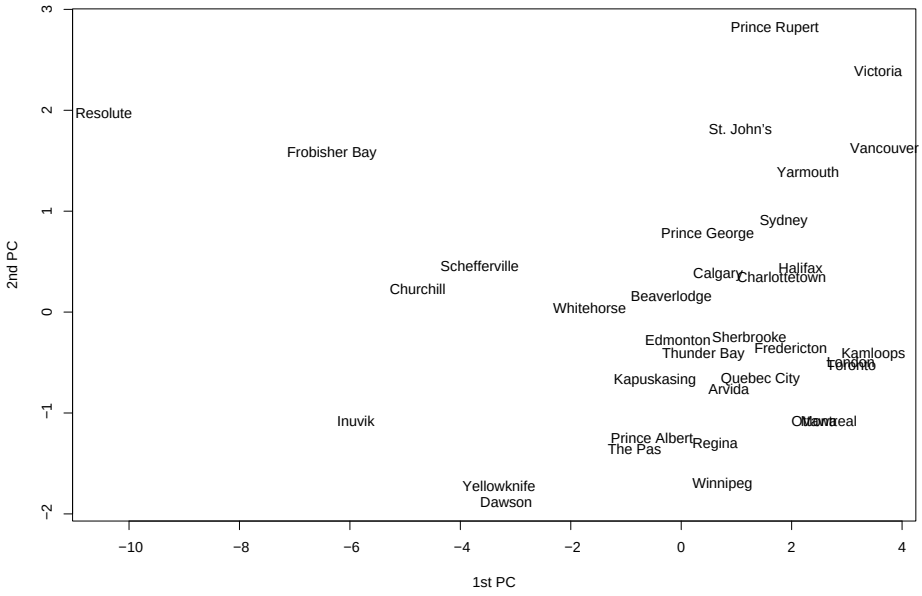


FIGURE 15. Plot of first 2 PCs for Canadian Temperature Data

A projection of the weather stations on the space spanned by the first two PCs is shown in Figure 15, while the corresponding biplot is shown in Figure 16 (note again the rotational invariance issue of the PCA solution). It can be seen that locations such as Resolute, Inuvik, Iqualit, Churchill and Schefferville that are close or above the Arctic circle are cold throughout the year, while some British Columbia locations (Vancouver, Prince Ruppert and Victoria) are warm year-round, but warmerst compared to the remaining locations in the winter months (December to February).

Remark: One may wonder whether it is appropriate to apply PCA on time series type of data. For the data set at hand we have to deal with only 12 values (the average monthly temperatures). However, the technique is more generally applicable for capturing the main features of stochastic processes through the Karhunen-Loeve transform.

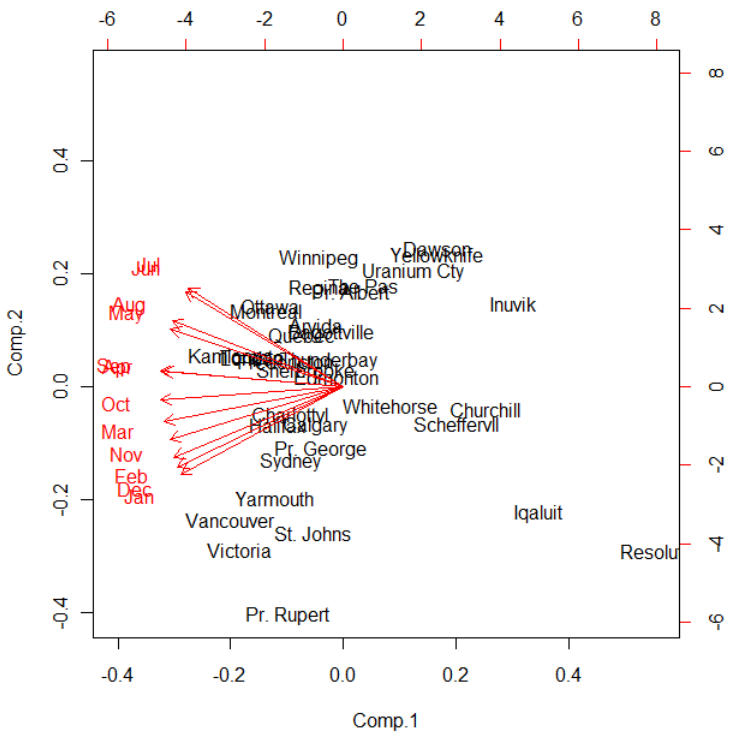


FIGURE 16. Biplot of first 2 PCs for Canadian Temperature Data

Chapter 2: Factor Analysis

1. Introduction

As Johnson and Wichern observe in their book, *factor analysis* (FA) has provoked a rather turbulent controversy throughout its history. The origins of the technique date back to 1905 and the work of Charles Spearman on defining and measuring human intelligence.

The main objective of FA is to describe the relationships between many variables (as captured by the covariance matrix) by a few underlying, but unobservable (latent) variables, called *factors*.

Remark: Both FA and PCA can be viewed as attempts in approximating the covariance matrix Σ. However, the primary concern in FA is whether the observed data are consistent with a prescribed structure/model. But when used for exploratory purposes, it can be viewed as a model based version of PCA, and therefore suitable for data reduction purposes.

2. The FA Model

We start by examining the population version of the model. Let $\mathbf{x}$ be a p-dimensional random vector with $\mathbf{x} = (\mathbf{x_1}, \cdots, \mathbf{x_p})'$ with $\mathbb{E}(\mathbf{x}_i) = 0$ $Var(\mathbf{x}_i) = \Sigma(i,i) < \infty$ and $Cov(x_i, x_j) = \Sigma(i,j) < \infty$ for all $i, j = 1, \cdots, p$. According to the FA model each $\mathbf{x}_i$ can be modeled as a *linear combination* of $k < p$ *common factors* $\mathbf{F} = (F_1,\ F_2, ..., F_k)'$ plus some element (variable) specific random error u_i. Hence, we posit the following model

$$
\begin{aligned}
x_1 &= \sum_{j=1}^{k} \lambda_{1j} F_j + u_1 \\
x_2 &= \sum_{j=1}^{k} \lambda_{2j} F_j + u_2 \\
&..... \\
x_p &= \sum_{j=1}^{k} \lambda_{pj} F_j + u_p
\end{aligned} \tag{1}
$$

or in matrix notation

$$
\mathbf{x} = \Lambda \mathbf{F} + \mathbf{u}, \tag{2}
$$

where Λ is a $p \times k$ matrix of *factor loadings*, $\mathbf{F}$ is a $p \times 1$ random vector containing the *common factors*, and $\mathbf{u}$ is a $p \times 1$ vector of error terms (also known as *variable specific factors*).

It is interesting to observe that in equation (2), on the left-hand side we have an observed random vector $\mathbf{x}$, while on the right-hand side both random vectors $\mathbf{F}$ and $\mathbf{u}$, as well as the coefficient matrix Λ are latent (unobservable). To overcome potential identifiability issues (namely that the elements of Λ can not be uniquely estimated from data) the following assumptions are made regarding random vectors $\mathbf{F}$ and $\mathbf{u}$:

- $\mathbf{F}$ and $\mathbf{u}$ are *independent*.
- $\mathbb{E}(\mathbf{F}) = \mathbf{0}$ and $\text{Cov}(\mathbf{F}) = I_p$.
- $\mathbb{E}(\mathbf{u}) = 0$ and $\text{Cov}(\mathbf{u}) = \Psi$, where Ψ is a $p \times p$ *diagonal* matrix.

It can be seen that the above model implies that the elements of $\mathbf{x}$ are independent given the common factors $\mathbf{F}$.

We can then express the covariance matrix of $\mathbf{x}$ as

$$\text{(3)} \quad \text{Cov}(\mathbf{x}) \equiv \Sigma = \mathbb{E}(\mathbf{x}\mathbf{x}') = \mathbb{E}\big[(\Lambda\mathbf{F} + \mathbf{u})(\Lambda\mathbf{F} + \mathbf{u})'\big] = \\ \Lambda\mathbb{E}(\mathbf{F}\mathbf{F}')\Lambda' + 2\Lambda\mathbb{E}(\mathbf{F}\mathbf{u})' + \mathbb{E}(\mathbf{u}') = \Lambda\Lambda' + \Psi.$$

Specifically, note that the variance of the i-th element of $\mathbf{x}$ can be written as

$$\text{(4)} \qquad \text{Var}(x_i) = \sum_{j=1}^{k} \Lambda^2(i,j) + \Psi(i,i), \ i = 1, ..., p.$$

It can be seen that $\text{Var}(x_i)$ is comprised of two components: the *communality* of variable x_i given by $\sum_{j=1}^{k} \Lambda^2(i,j)$ (i.e. the variance shared with other variables via the common factors), and the *uniqueness* $\Psi(i,i)$ (i.e. the variance not shared with other variables).

Therefore, the objective of the FA model is to estimate the factor loadings matrix Λ and the diagonal matrix Ψ containing the uniquenesses, from the covariance matrix Σ.

Remark: (Rotational indeterminacy of the solution). The solution to the problem given in (3) is not unique. Take for example an orthonormal $p \times p$ matrix T, i.e. $T'T = TT' = I_p$. We have $\mathbf{x} = (\mathbf{\Lambda T})(\mathbf{T'F}) + \mathbf{u}$ and hence, $S = (\Lambda T)(\Lambda T)' + \Psi = \Lambda\Lambda' + \Psi$. This "ambiguity" provides the rational for "factor rotations." Thus, factor loadings are determined up to an orthogonal rotation matrix T.

3. Estimation of the FA Model

In practice, one has gathered data on p variables from n independent and identically distributed observations, in a $n \times p$ matrix X. The sample covariance matrix S will be used to estimate the parameters of the FA model. Towards that objective there are two popular

algorithms, the principal factor analysis one and one based on the maximum likelihood principle.

3.1. **Principal Factor Analysis.** It is not usually possible in practice to find matrices $\hat{\Lambda}$ and $\hat{\Psi}$ so that $S = \hat{\Lambda}\hat{\Lambda}' + \hat{\Psi}$ holds exactly. Hence, the goal becomes to find such matrices, so that trace$(S - \hat{S})'(S - \hat{S})$ is minimized, where $\hat{S} = \hat{\Lambda}\hat{\Lambda}' + \hat{\Psi}$. The following iterative algorithm works in this case:

- Guess $\hat{\Psi}$.
- Set $\hat{\Lambda} =$ the largest k eigenvectors of the eigendecomposition of $S - \hat{\Psi}$.
- Set $\hat{\Psi} = \text{diag}(S - \hat{\Lambda}\hat{\Lambda}')$.

Repeat steps 2 and 3 until convergence.

Remark: Notice that in case $\hat{\Psi} = 0$ this corresponds to principal components analysis. In this case the factor loadings correspond to rescaled versions of the PCs.

3.2. **Maximum Likelihood Factor Analysis.** In this case, one makes the following distributional assumptions regarding the common factors and the random error term:

$$\mathbf{F} \sim N(0, I_k) \tag{5}$$

$$\mathbf{u} \sim N(0, \Psi) \tag{6}$$

The log-likelihood function is then given by

$$\mathcal{L}(\Lambda, \Psi) = \text{constant} - \frac{N}{2} log|\Sigma| - \frac{N}{2}\text{trace}(\Sigma^{-1}S), \tag{7}$$

and it is maximized with respect to Λ and Ψ by an Expectation-Maximization algorithm (details given in the paper by Rubin and Thayer (1982), EM algorithm for ML factor analysis, Psychometrika, 47, 69-76). However, we must be careful regarding *Heywood cases*, that is solutions with $\hat{\Psi}(i,i) < 0$. In those circumstances we must force $\hat{\Psi}(i,i) = 0$.

4. Implementation Issues

4.1. **Number of Factors.** We begin by examining how many common factors we can estimate in the first place. The number of free observations in S is $p(p+1)/2$, since the covariance matrix is symmetric. The number of free parameters in the representation $\Lambda\Lambda' + \Psi$ is p from the matrix of uniquenesses Ψ, pk from Λ minus $k(k-1)/2$ from the rotational freedom in Λ, for a total of $p(k+1) - (k-1)k/2$.

For example, suppose that we have $p = 5$ variables. The number of free observations is 15, while the number of free parameters for a $k = 2$ common factor model is 14, which works fine. However, we can not estimate a $k = 3$ common factor model. The quantity $p(p+1)/2 - [p(k+1) - (k-1)k/2]$ in the factor model is known as *degrees of freedom*.

However, for this simple example there still remains the question whether a one factor model is better than a two factor model. An informal method suggests the use of the scree plot, as in PCA. A more formal approach is based on the following hypothesis testing problem.

$H_0 : \Sigma = \Lambda\Lambda' + \Psi$ with k common factors, vs
$H_A : \Sigma$ is unconstrained.

By assuming normality and using the ML estimates, we can construct a likelihood ratio test to decide between H_0 and H_A.

4.2. **Factor Rotations.** As previously discussed, the solution $\hat{\Lambda}$ is not unique, since its multiplication by an orthonormal matrix T does not alter the decomposition of S. In many cases the given solution may be difficult to interpret; for example, many common factors may have similar large loadings, or both positive and negative loadings. A rotation by T may succeed in revealing a "simpler structure." For factor models with two components (2-factor models) a visual inspection suffices, but for more than two factor solutions some other method is necessary. A popular choice is the *varimax* rotation (Kaiser, 1958).

The rotation matrix T corresponds to the solution of the following criterion:

$$\hat{T} = \text{argmax}_T \Big(\sum_{j=1}^{k} \sum_{i=1}^{p} Z^4(i,j) - \frac{1}{p} \sum_{k=1}^{j} \big(\sum_{i=1}^{p} Z^2(i,j) \big)^2 \Big), \tag{8}$$

where $Z = \Lambda T$.

Remark: In some cases even orthogonal rotations do not provide an easy interpretation of the solution. It is possible to allow non-orthogonal rotations called *oblique* rotations. This allows for possible simplicity at the expense of losing the orthogonality of the factors.

4.3. **Factor Scores.** How can we calculate an object's location in the common factor space (i.e. the space spanned by the common factors we identified)? Remember that in PCA we just had to project the original data matrix to the space spanned by the PCs we selected. In factor analysis things are a little bit more complicated, since we have two unknown quantities, namely F and U. Several heuristic approaches have been suggested in the literature. Bartlett suggested calculating factor scores by computing $(\hat{\Lambda}\hat{\Psi}^{-1}\hat{\Lambda})^{-1}(\hat{\Lambda}\hat{\Psi}^{-1}X)$ (a generalized least squares estimate), while Thomson suggested using $(I + \hat{\Lambda}\hat{\Psi}^{-1}\hat{\Lambda})^{-1}(\hat{\Lambda}\hat{\Psi}X)$. The latter choice is the default in R.

5. Concluding Remarks

Unlike PCA where we attempt to "summarize" multivariate data, FA is a more complex technique. The data analyst must make several decisions. The most important one is the choice of k, the number of common factors. The availability of a large sample size test

does not always solve the problem, because it assumes that the data are approximately normally distributed and the sample size is large. Most often, the choice of k is based on a combination of (i) proportion of variance explained, (ii) subject-matter knowledge and (iii) how reasonable and interpretable the results are. Another concern is that the principal factor solution and the ML one may differ.

6. Factor Analysis in R

In the following synthetic data set with 50 observations, we generate three mean 0, variance 1 variables X_1, X_3 and X_5. We then generate another three variables as follows: $X_2 = X_1 + \epsilon_1$, $X_4 = X_3 + \epsilon_2$, $X_6 = X_5 + \epsilon_3$ and $X_7 = 0.5X_1 + 0.5X_3 + \epsilon_4$, where $\epsilon_i \sim N(0, .4),\ i = 1, 2, 3$ and $\epsilon_4 \sim N(0, .8)$ Therefore, we expect that a 3-factor model should fit the data well.

```
set.seed(14156)
x1=rnorm(50,0,1)
x2=.8*x1+rnorm(50,0,.5)
x3=rnorm(50,0,1)
x4=.8*x3+rnorm(50,0,.5)
x5=rnorm(50,0,1)
x6=.8*x5+rnorm(50,0,.5)
x7=.5*x1+.5*x2+rnorm(50,0,.8)
x=cbind(x1,x2,x3,x4,x5,x6,x7)
```

The correlation matrix of this synthetic data set is given next (in order to reproduce the results, you need to set the seed of the random number generator to the same value)

```
> options(digits=3)
> cor(x)
        x1      x2      x3     x4      x5      x6      x7
x1  1.0000  0.8935 -0.1129 -0.196 -0.1491 -0.0706  0.8729
x2  0.8935  1.0000 -0.0688 -0.144 -0.0725 -0.0111  0.9058
x3 -0.1129 -0.0688  1.0000  0.893  0.2680  0.2671 -0.0826
x4 -0.1963 -0.1436  0.8933  1.000  0.2602  0.2612 -0.1522
x5 -0.1491 -0.0725  0.2680  0.260  1.0000  0.8396 -0.1532
x6 -0.0706 -0.0111  0.2671  0.261  0.8396  1.0000 -0.0893
x7  0.8729  0.9058 -0.0826 -0.152 -0.1532 -0.0893  1.0000
```

Next we try to fit a 4-factor model.

```
> library(MASS)
```

```
> factanal(x,factors=4)

Error in factanal(artificial, factors = 4) :

4 factors is too many for 7 variables
```

The software complains that 4 factors are too many for 7 variables, so we go with a 3-factor model.

```
dataFA=factanal(x,factors=3)

Call:
factanal(x = x, factors = 3)

Uniquenesses:
   x1    x2    x3    x4    x5    x6    x7
0.137 0.066 0.192 0.005 0.031 0.267 0.115

Loadings:
   Factor1 Factor2 Factor3
x1  0.923
x2  0.965
x3          0.884   0.158
x4 -0.108   0.983   0.131
x5          0.126   0.973
x6          0.152   0.842
x7  0.937

                Factor1 Factor2 Factor3
SS loadings       2.681   1.799   1.707
Proportion Var    0.383   0.257   0.244
Cumulative Var    0.383   0.640   0.884

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 0.31 on 3 degrees of freedom.
The p-value is 0.959
```

The fit is pretty good, since the first three factors capture 88.4% of the total variance in the data. Also, the uniquenesses (variances of the error term $\mathbf{u}$ in the model) are also fairly small.

Finally, as expected (due to the way the data were generated) the first factor captures the associations between X_1, X_2 and X_7, the second factor between X_3 and X_4 and the third factor between X_5, X_6.

The test also does not reject the 3-factor model.

To get the degrees of freedom we do

```
dataFA$dof
[1] 3
```

If we want to visualize the observations we ask for Bartlett scores; another option is to ask for regression scores (scores="regression").

```
dataFA=factanal(x,factors=3,scores="Bartlett")

% we can plot the scores
plot(dataFA$scores[,1:2],type="n")
text(dataFA$scores[,1:2]
```

In R the default rotation is varimax, but also an oblique rotation like promax is available and the results are given next.

```
Call:
factanal(x = x, factors = 3, scores = "Bartlett", rotation = "promax")

Uniquenesses:
   x1    x2    x3    x4    x5    x6    x7
0.137 0.066 0.192 0.005 0.031 0.267 0.115

Loadings:
   Factor1 Factor2 Factor3
x1  0.921
x2  0.971
x3          0.897
x4          1.000
x5                  0.989
x6                  0.851
x7  0.938

               Factor1 Factor2 Factor3
SS loadings      2.675   1.808   1.709
Proportion Var   0.382   0.258   0.244
```

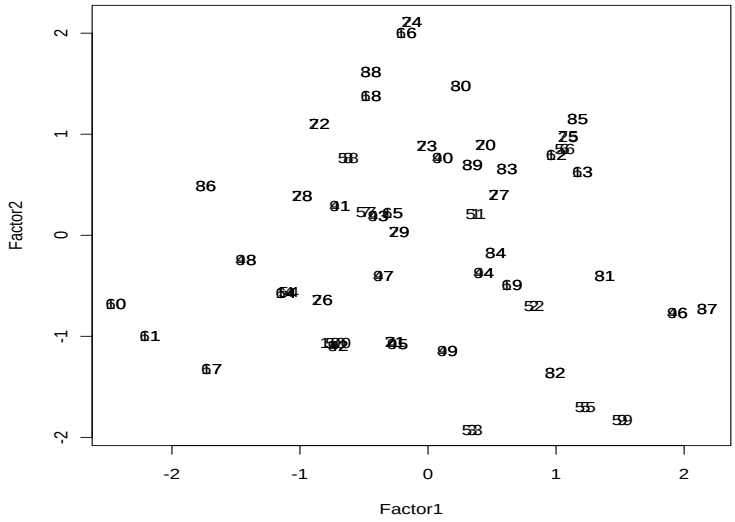


FIGURE 1. Bartlett scores for the first two factors of a 3-factor model

```
Cumulative Var   0.382   0.640   0.884

Factor Correlations:
        Factor1 Factor2 Factor3
Factor1   1.000  -0.309   0.137
Factor2  -0.309   1.000  -0.103
Factor3   0.137  -0.103   1.000

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 0.31 on 3 degrees of freedom.
The p-value is 0.959
```

It can be seen that to a large extent the structure of the model is very similar to the one obtained with the default varimax rotation, although this rotation made the structure crisper.

7. FACTOR ANALYSIS OF TWO REAL DATA SETS

7.1. **Ability data set.** Next an analysis on a data set from Bartholomew's book on *Latent Variable Analysis and Factor Analysis* (1987) is presented. The data deal with "ability" scores on several tasks administered to 112 individuals.

In this case, the data available come in the form of a covariance matrix given next.

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

We start with a 1-factor model

```
> ability.FA <- factanal(factors=1, covmat=ability.cov)

> ability.FA

Call:
factanal(factors = 1, covmat = ability.cov)

Uniquenesses:
general picture  blocks    maze reading   vocab
  0.535   0.853   0.748   0.910   0.232   0.280

Loadings:
        Factor1
general 0.682
picture 0.384
blocks  0.502
maze    0.300
reading 0.877
vocab   0.849

               Factor1
```

```
SS loadings      2.443
Proportion Var   0.407

Test of the hypothesis that 1 factor is sufficient.
The chi square statistic is 75.18 on 9 degrees of freedom.
The p-value is 1.46e-12
```

So, we reject the null and go for a 2-factor model.

```
> ability.FA <- factanal(factors=2, covmat=ability.cov)
> ability.FA

Call:
factanal(factors = 2, covmat = ability.cov)

Uniquenesses:
general picture  blocks    maze reading   vocab
  0.455   0.589   0.218   0.769   0.052   0.334

Loadings:
        Factor1 Factor2
general 0.503   0.540
picture 0.160   0.620
blocks  0.212   0.859
maze    0.112   0.467
reading 0.957   0.176
vocab   0.786   0.220

               Factor1 Factor2
SS loadings      1.871   1.711
Proportion Var   0.312   0.285
Cumulative Var   0.312   0.597

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 6.11 on 4 degrees of freedom.
The p-value is 0.191
```

Since the interpretation of the results is not particularly easy, we use the 2-factor model with a promax rotation.

```
Call:
factanal(factors = 2, covmat = ability.cov, rotation = "promax")

Uniquenesses:
general picture  blocks    maze reading   vocab
  0.455   0.589   0.218   0.769   0.052   0.334

Loadings:
        Factor1 Factor2
general  0.364   0.470
picture           0.671
blocks            0.932
maze              0.508
reading  1.023
vocab    0.811

               Factor1 Factor2
SS loadings      1.853   1.807
Proportion Var   0.309   0.301
Cumulative Var   0.309   0.610

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 6.11 on 4 degrees of freedom.
The p-value is 0.191
```

It can be seen that the first factor is associated with general, reading and vocabulary tasks, while the second factor captures associations between ”visual” tasks, plus general tasks.

7.2. **Detergent data set.** Let us examine another data set where 512 consumers were asked to evaluate various detergents. The variables used in the study were:

(1) V1: gentle to natural fabrics
(2) V2: won’t harm colors
(3) V3: won’t harm synthetics
(4) V4: safe for lingerie
(5) V5: strong, powerful
(6) V6: gets dirt out
(7) V7: makes colors bright
(8) V8: removes grease stains
(9) V9: good for greasy oil

(10) V10: pleasant fragrance
(11) V11: removes stubborn stains

Once again the data are given in the form of a correlation matrix, shown next.

```
V1 1
V2 .419 1
V3 .518 .576 1
V4 .566 .499 .643 1
V5 .181 .186 .290 .383 1
V6 .174 .246 .344 .396 .579 1
V7 .230 .229 .411 .377 .594 .577 1
V8 .306 .225 .340 .404 .676 .701 .677 1
V9 .240 .219 .328 .423 .692 .623 .684 .698 1
V10 .212 .258 .388 .365 .437 .621 .541 .685 .586 1
V11 .206 .258 .367 .367 .652 .578 .638 .719 .691 .635 1
```

Notice that we have enough *free observations* and we can fit up to a 6 common factor model. Suppose that we decided to fit 6 factors, using the MLE method (with the default in R varimax rotation). The cumulative variance explained is 75%. The uniquenesses are given next.

```
   V1    V2   V3   V4   V5   V6   V7    V8   V9  V10  V11
4e-10 .546 .248 .378 .180 .331 4e-10 .197 .312 .289 .164
```

It can be seen that V1=gentle to natural fabrics and V7=makes colors bright are fitted almost exactly, while the fit of some of the variables can be characterized as poor (e.g. V2 and V4). This is an example of Heywood cases; i.e. the uniqueness are exactly 0.

The factor loadings are given next:

```
Loadings:
    Factor1 Factor2 Factor3 Factor4 Factor5 Factor6
================================================
V1            0.468                   0.873
V2            0.650
V3            0.820
V4            0.669
V5   0.453           0.744
V6   0.705
V7   0.494                   0.805
V8   0.770
V9   0.600           0.443
V10  0.787
```

```
V11  0.646                                          0.455
```

But this is a very ”undesirable” solution, since some factors have high loadings on a single variable only and they also share high loadings with factors 1 and 2.

Since, we are performing *exploratory* factor analysis we decide to look for a model with fewer factors. If on the other hand you has some theory that advocated 6 factors (*confirmatory* FA) you would conclude that this particular dataset does not render strong support to the theory. In the absence of some theory that would explain why we may need 6 factors we search for a simpler, more interpretable model.

Next, we fit next a 2 factor model. The total variance explained is 60% and the uniquenesses reveal that some of the variables are not particularly well fitted (e.g. V1, V2).

```
   V1   V2   V3   V4   V5   V6   V7   V8   V9  V10  V11
 .566 .551 .317 .368 .407 .415 .394 .228 .305 .460 .325
```

However, the factor loadings (shown next) suggest that factor 1 captures the ”softness/gentleness” of the detergent, while factor 2 its ”strength” to remove stains. Notice that this model is much easier to explain; namely that the results of this customer survey (as captured by the correlation matrix of the items) can be explained by a softness and a strength factor.

```
Loadings:
    Factor1 Factor2
==================
V1          0.646
V2          0.659
V3          0.790
V4          0.732
V5  0.755
V6  0.736
V7  0.741
V8  0.856
V9  0.810
V10 0.693
V11 0.797
```

As a final remark, in general it is hard to find examples in the literature for which a factor analysis model fits well and has an elegant interpretation of the factors.

Chapter 3: Multidimensional Scaling

1. Introduction

The term 'Multidimensional Scaling' (MDS) is used in two essentially different ways in statistics. MDS in the more general sense refers to any technique that produces a multidimensional geometric representation of data, where quantitative or qualitative relationships in the data correspond with geometric relationships in the representation. MDS in the narrow sense starts with information about some *dissimilarities* between a set of objects and constructs their geometric representation from this information (de Leeuw and Heiser, 1980, de Leeuw, 2000).

Multidimensional scaling (MDS) is a method that represents measurements of *similarity* or *dissimilarity* among pairs of objects as distances between points of a low-dimensional space (Borg & Groenen, 1997, p. 3).

Let us look at a simple example to start clarifying ideas. In the following table distances in km between 10 European cities are given. (see Borg & Groenen, p. 16).

	London	Stockholm	Lisbon	Madrid	Paris	Amsterdam	Berlin	Prague	Rome	Dubli
London	0	569	667	530	141	140	357	396	570	19
Stockholm	569	0	1212	1043	617	446	325	423	787	64
Lisbon	667	1212	0	201	596	768	923	882	714	71
Madrid	530	1043	201	0	431	608	740	690	516	62
Paris	141	617	596	431	0	177	340	337	436	32
Amsterdam	140	446	768	608	177	0	218	272	519	30
Berlin	357	325	923	740	340	218	0	114	472	51
Prague	396	423	882	690	337	272	114	0	364	57
Rome	569	787	714	516	436	519	472	364	0	75
Dublin	190	648	714	622	320	302	514	573	755	

Let us look at a simple example to clarify the notions of similarity and dissimilarity. The following table shows the correlation matrix of the crime dataset (data taken from the U.S. Statistical Abstract of 1994).

	Murder	Rape	Robbery	Assault	Burglary	Larceny	MVT
Murder	1.0000000	0.2260129	0.6991807	0.5022187	0.5129268	0.1642008	0.4784107
Rape	0.2260129	1.0000000	0.2390043	0.2937968	0.5179029	0.2972498	0.1771553
Robbery	0.6991807	0.2390043	1.0000000	0.6605061	0.5961077	0.1519019	0.7056040
Assault	0.5022187	0.2937968	0.6605061	1.0000000	0.6818327	0.5113177	0.5753717
Burglary	0.5129268	0.5179029	0.5961077	0.6818327	1.0000000	0.5891757	0.5771577
Larceny	0.1642008	0.2972498	0.1519019	0.5113177	0.5891757	1.0000000	0.2166964
MVT	0.4784107	0.1771553	0.7056040	0.5753717	0.5771577	0.2166964	1.0000000

We regard the correlation coefficient as a measure of *proximity* (similarity) between two variables. The higher the correlation coefficient the closer the two variables are in some unspecified space. We would like to approximate these similarities by points in a low dimensional Euclidean space, in order to visualize the overall structure of the correlation matrix. For technical reasons that we are going to examine later on, it is easier to work with dissimilarities instead, so we transform the table by taking the reciprocals of its entries. So, the data I am going to approximate are

	Murder	Rape	Robbery	Assault	Burglary	Larceny	MVT
Murder	1.000000	4.424527	1.430246	1.991164	1.949596	6.0901055	2.090254
Rape	4.424527	1.000000	4.184025	3.403713	1.930864	3.3641742	5.644764
Robbery	1.430246	4.184025	1.000000	1.513991	1.677549	6.5831954	1.417225
Assault	1.991164	3.403713	1.513991	1.000000	1.466635	1.9557311	1.738007
Burglary	1.949596	1.930864	1.677549	1.466635	1.000000	1.6972866	1.732629
Larceny	6.090106	3.364174	6.583195	1.955731	1.697287	1.0000000	4.614750
MVT	2.090254	5.644764	1.417225	1.738007	1.732629	4.6147505	1.000000

The picture below tells us that robbery is highly correlated with murder and motor vehicle thefts and lowly correlated with rape and larceny, since in this 2-dimensional representation it is closer to the first two variables and further away from the last two. A look at the correlation matrix confirms these findings.

Remark: There is a large literature of how to convert similarity data to dissimilarity data. Some suggestions include:

1 $D_{ij} = \text{constant} - S_{ij}$, where D_{ij} (S_{ij}) denotes the dissimilarity (similarity) between objects i and j.
2 $D_{ij} = 1/S_{ij}$.
3 $D_{ij}^2 = S_{ii} + S_{jj} - 2S_{ij}$. This derivation takes its cue from the following identity involving norms and inner products: $||x_i - x_j||^2 = ||x_i||^2 + ||x_j||^2 - 2 < x_i, x_j >$.

Historically, MDS was developed as a method for analyzing proximity data that arise in the social sciences (similarity ratings for pairs of stimuli such as tastes, colors, etc), classification problems, archaeology (similarity in the features of artifacts). Another early use of MDS has been *dimension reduction* (Kruskal-Shepard MDS). The idea is to compute the pairwise distances between the objects in a high dimensional space, and attempt to find an arrangement of the objects in a low dimensional space so that their distances approximate as close as possible the original distances (see example in Section 4).

2. METRIC SCALING

The setting is as follows: given N objects with proximity matrix $\Delta = \{\delta_{ij}\}$ find the best possible arrangement of the objects in $d-$dimensions (p small) with proximity matrix $D = \{d_{ij}\}$, so that $\Delta \approx D$ (in some appropriate norm). The coordinates of the objects in

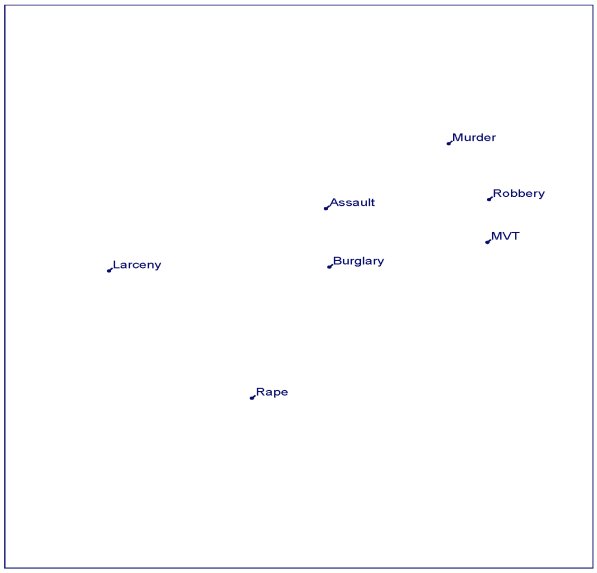


FIGURE 1. MDS representation of the crime correlation matrix

the low dimensional space ($p = 2$ or 3 in practice) are collected in a $N \times p$ matrix X. The proximities $d_{ij}(X)$ correspond to the Euclidean distances between points x_i and x_j, in the majority of cases, although some times we compute ℓ_1 or general ℓ_p distances.

Remark: If the original proximities δ_{ij} also correspond to distances (i.e. satisfy the usual properties of a distance function), they we talk about *classical* (or Torgerson-Gower) MDS.

We measure how good the approximation $\Delta \approx D(X)$ is, by employing the following *loss* function, known in the literature as the *Stress* function (Kruskal, 1964):

$$\text{Stress}(X) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} (d_{ij}(X) - \delta_{ij})^2, \tag{1}$$

which is invariant under rotations and translations, but not invariant to stretching and shrinking. This implies that the value of *Stress* is scale dependent and a better criterion is given by

$$\sqrt{\frac{\text{Stress}(X)}{\sum_{i<j} d_{ij}^2(X)}}. \tag{2}$$

A more general form of the Stress function incorporates weights w_{ij} that reflect variability, measurement error or missing data:

$$\text{WStress}(X) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} w_{ij}(d_{ij}(X) - \delta_{ij})^2. \tag{3}$$

If the weights are chosen so that $w_{ij} = \delta_{ij}^{-1}$, we then recover the *Sammon mapping*.

Finding the configuration X is a non-trivial mathematical problem. However, the SMACOF algorithm of de Leeuw (1977) guarantees convergence to a stationary point. Unfortunately the rate of convergence is linear.

The underlying model in the Stress function is

$$d_{ij}(X) = \delta_{ij} + \text{error term} \tag{4}$$

and our goal is to minimize the error sum of squares. We could generalize this model to allow

$$d_{ij}(X) = \beta\delta_{ij} + \text{error term}, \tag{5}$$

which gives rise to *ratio scaling* or

$$d_{ij}(X) = \alpha + \beta\delta_{ij} + \text{error term}, \tag{6}$$

which corresponds to *interval scaling*. Obviously in these cases the optimization problem becomes considerably more difficult. An even more general specification is

$$d_{ij}(X) = g(\delta_{ij}) + \text{error term}, \tag{7}$$

where g is some smooth function. The logarithmic and the exponential functions have been found particularly useful in various contexts in psychology.

3. NON-METRIC SCALING

If we are unwilling to trust the actual values of the proximities but only their *ordering* then we need to fit *non-metric* models. Ordinal models typically require that

$$\text{if } \delta_{ij} < \delta_{ik} \text{ then } d_{ij}(X) \leq d_{ik}(X). \tag{8}$$

Such models represent only the *ordinal* properties of the data. Finding the configuration X becomes a more challenging mathematical problem (for details see the recent book by Borg & Groenen, *Modern Multidimensional Scaling*, Springer, 1997).

Stress	Goodness of fit
20%	Poor
10%	Fair
5%	Good
2.5%	Excellent
0%	Perfect

4. Choosing the Dimensionality of the Configuration X

The usual choice of p is 2 or 3, since we can then plot X.

A rough rule of thumb proposed by Kruskal (1964) is:

This table suggests to pick a large enough p so that the fit is at least fair.

5. A Dimension Reduction Example

Let us examine the crime dataset. We begin by calculating Euclidean distances between the cities in the 7-dimensional space defined by the 7 variables. Notice that in this case the proximities δ_{ij} correspond to true distances. We fit next a 2-dimensional metric MDS model. The goodness of fit is 3% and according to Kruskal's criterion we don't need to look for higher dimensions. The arrangement of the cities in this 2-dimensional space is shown next. This map places (by construction) cities with *similar* crime profiles close together and cities with *dissimilar* crime profiles far apart. It is interesting to note that the 2-dim MDS representation of the cities is very similar to the 2-dim PCA representation. However, all the variable information has been absorbed in the matrix Δ and therefore it is not possible to draw a *biplot*.

6. MDS Implementation in R

The function to use for metric scaling in R is cmdscale. We illustrate the use of the function with the Iris data set.

```
> data(iris)
> library(cluster)

I would like to use the daisy distance function from the cluster
package

> iris.dist <- daisy(iris[,1:4])

Distances are calculated based on the 4 numerical variables only.
```

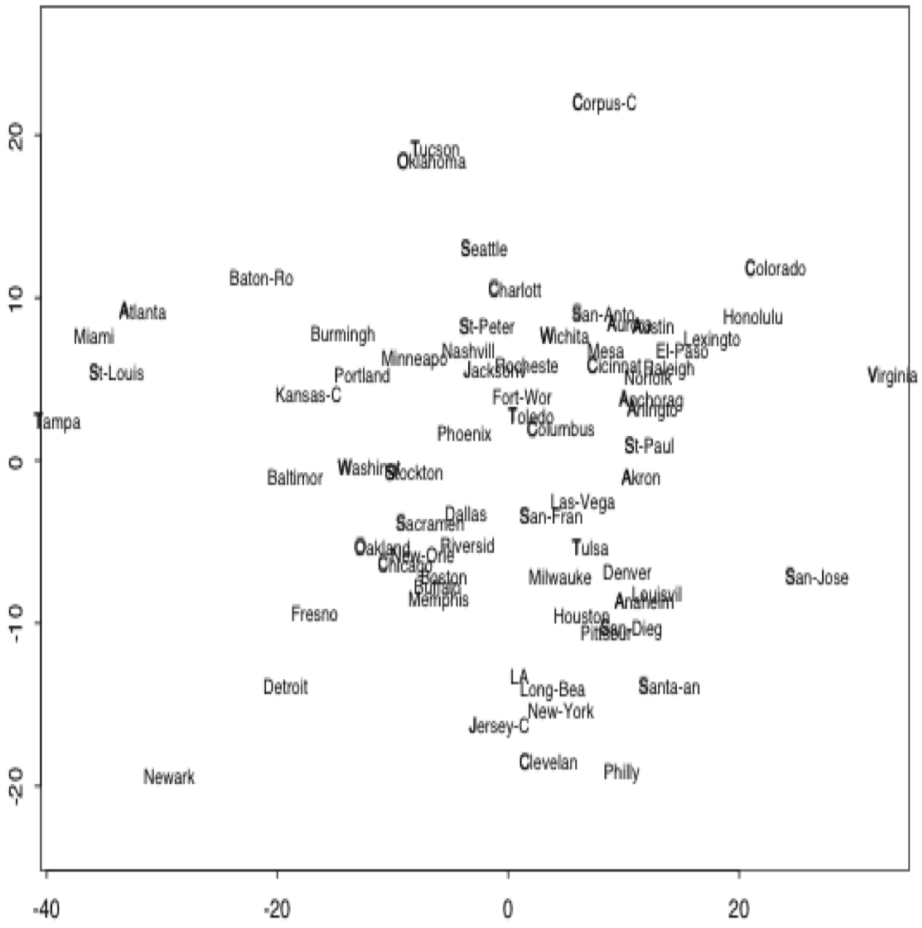


FIGURE 2. MDS representation of the crime data

```
> library(stats)

Loading the necessary package for MDS.

> iris.mds <- cmdcale(iris.dist,k=2)

Request a 2-dimensional solution.

> fitted.dist <- daisy(iris.mds)
> sum((iris.dist-fitted.dist)^2)/sum(fitted.dist)

[1] 0.006458607
```

```
Calculate the quality of the fit, which according to
Kruskal's scale is excellent. Therefore, no higher
dimensional solution is necessary.

> plot(iris.mds[,1],iris.mds[,2])
```

The MDS solution is shown in Figure 3

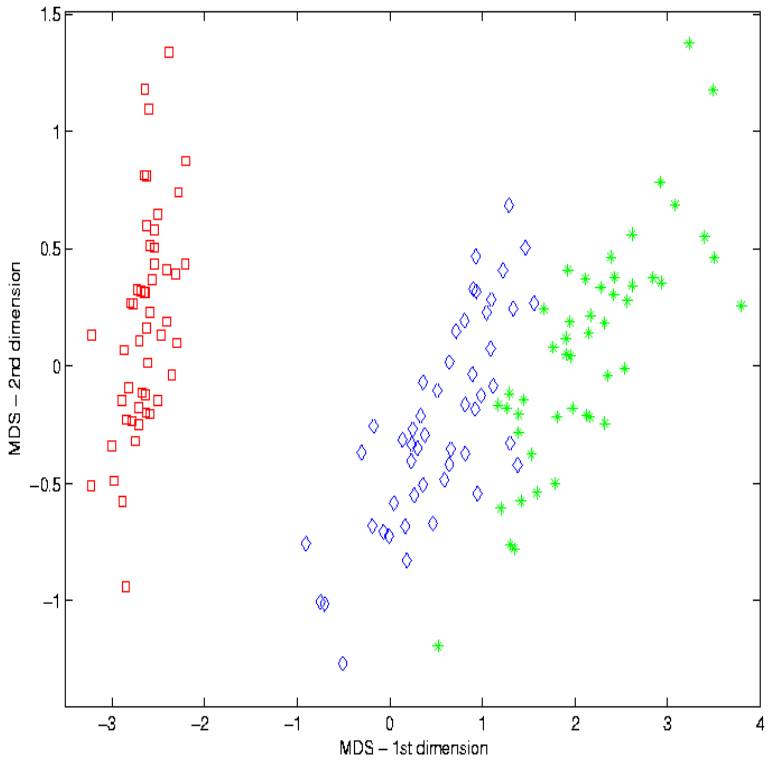


FIGURE 3. 2-dim MDS solution of the Iris data set. The red points correspond to class Setosa, the green points to the class Virginica and the green points to the class Versicolor.

6.1. **A categorical data set.** We analyze next the animal data set from the cluster library. Notice that the function daisy comes in handy.

```
> library(cluster)

> library(stats)
```

```
> data(animals)

> animals.diss <- daisy(animals,list=type(symm=1:6))

> animals.mds <- cmdscale(animals.diss)

> fitted.dist <-dist(animals.diss)
> sum((animals.diss-fitted.dist)^2)/sum(fitted.dist)

[1] 0.4498873

> plot(animals.mds[,1],animals.mds[,2],'type="n")
> text(animals.mds,row.names(animals))
```

Notice that the fit is inadequate; however, this is common when dealing with categorical data. However, the plot is fairly informative.

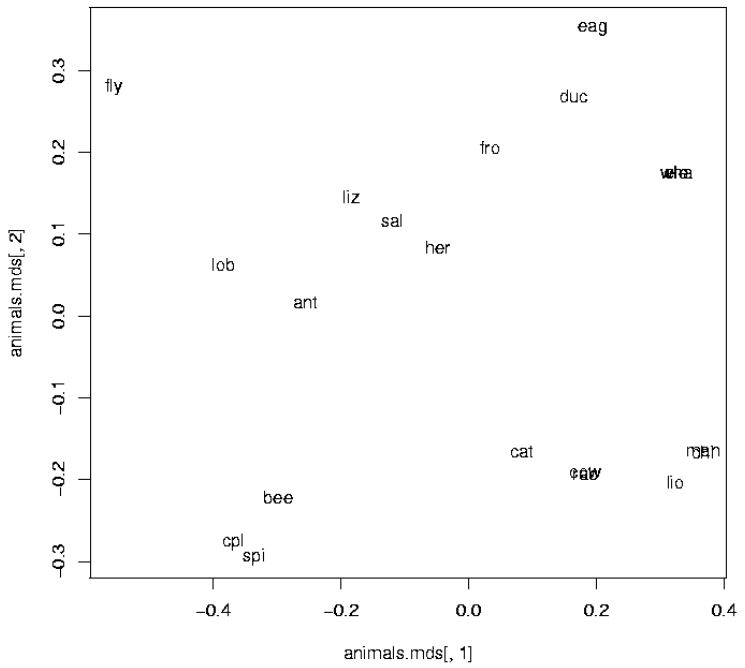


FIGURE 4. 2-dimensional MDS solution of the animals data set.

Chapter 4: Classification

1. Introduction and Problem Formulation

The area of classification, also known as pattern recognition and supervised learning has grown substantially over the last twenty years, primarily due to the availability of increased computing power, necessary for executing sophisticated algorithms. The need for classification arises in most scientific fields, ranging from disease diagnosis, to classifying galaxies by shape, to text and image classification, to applications in the financial industry, such as deciding which customers are good credit risks or constructing efficient portfolios, just to name a few. In bioinformatics, examples of classification tasks include classification of samples to different diseases based on gene and or protein expression data, prediction of protein secondary structure and identification and assignment of spectra to peptides and proteins obtained from mass spectrometry. It should be noted that the emergence of genomic and proteomic technologies, such as cDNA microarrays and high density oligonucleotide chips and antibody arrays gave a big impetus to the field, but also introduced a number of technical challenges, since the availability of more features (variables) than samples represented a shift in the classical paradigm.

The field of classification originated with the work of Fisher (1936) and experienced fast growth in the subsequent 40 years with the introduction of flexible classification techniques, such as nearest neighbor classifiers, the perceptron and neural networks. More recently, more flexible classifiers were proposed in the literature, such as classification trees, support vector machines, as well as regularized versions of more classical classifiers. Finally, over the last ten years ensemble methods emerged such as bagging and boosting, based in PAC learning theory (Valliant, 1984) which established that classifiers whose performance is slightly better than random guessing when appropriately combined can exhibit a superior performance.

On the theoretical front, some of the major developments include the development of a general statistical decision framework for classification (see below) and the connection of the problem to learning theory. The objective of learning theory is to study mathematical

properties of learning machines. Such properties are usually expressed as those of the function class that the learning machine can implement (see below).

The classification problem in layman's terms is as follows: given a number of measurements on a particular object, *assign* the object to one of a *prespecified* fixed number of *classes* (groups). The following examples provide some motivation about the task at hand.

Example 1: The goal is to assign an iris plant to one of the following three classes *setosa, virginica, versicolor* using information on the length and width of its petals and sepals. In order to achieve this goal, data on 150 plants were collected. Figure 1 shows the profiles (in a parallel coordinates plot) of 150 plants, 50 from each class (Fisher, 1936). It can be immediately seen that petal length (PL) and width (PW) separate the three classes well enough. The scatterplot of these variables given in Figure 2 shows a fairly clean separation of the three classes.

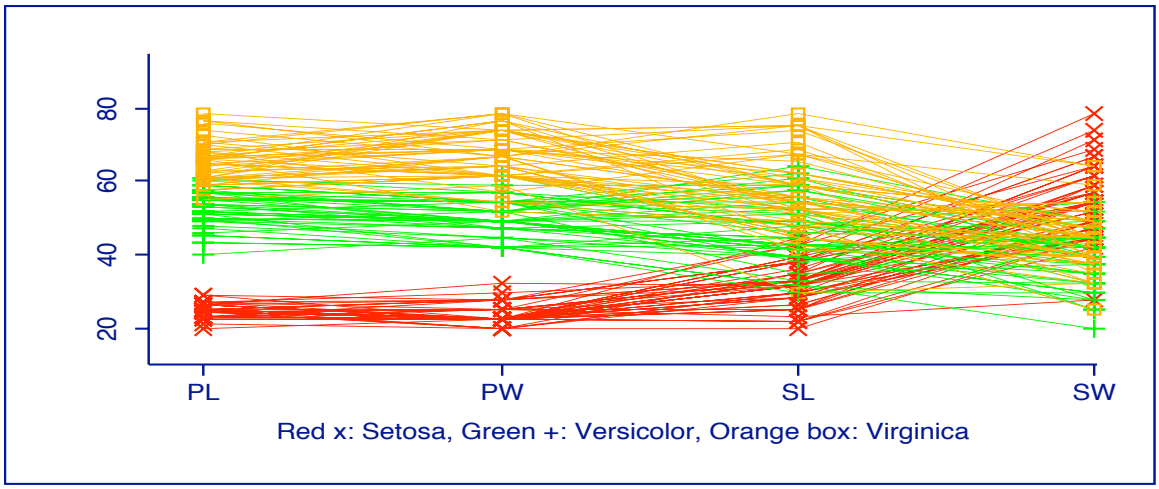


FIGURE 1. Parallel coordinates plot for the iris data

Example 2: This example comes from forensic testing of glass used by Evett and Spiehler from the Central Research Establishment, Home Office Forensic Science Service. The variables are the refractive index and weight percent of the following oxides: sodium, magnesium, aluminum, silicon, potassium, calcium, barium and iron. The six possible classes are: A=Building windows (float processed) (Green), B=Building windows (non-float processed) (Red), C=Vehicle windows (Yellow), D=Containers (Blue), E=Tableware

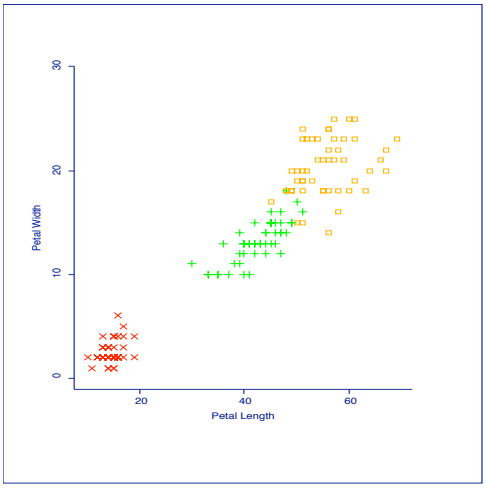


FIGURE 2. Scatterplot of petal length and petal width for the iris data

(Pink), and F=Headlamps (Orange). Figure 3 shows the profiles (in a parallel coordinates plot) of 214 cases; 70 from A, 76 from B, 17 from C, 13 from D, 9 from E and 29 from F. Some discrimination between glass types is apparent even from single attributes. For example class F is high on barium and aluminum and low on magnesium, while class A is high on magnesium. Nevertheless, the goal of assigning objects to their appropriate classes seems more difficult in this case than in the iris dataset.

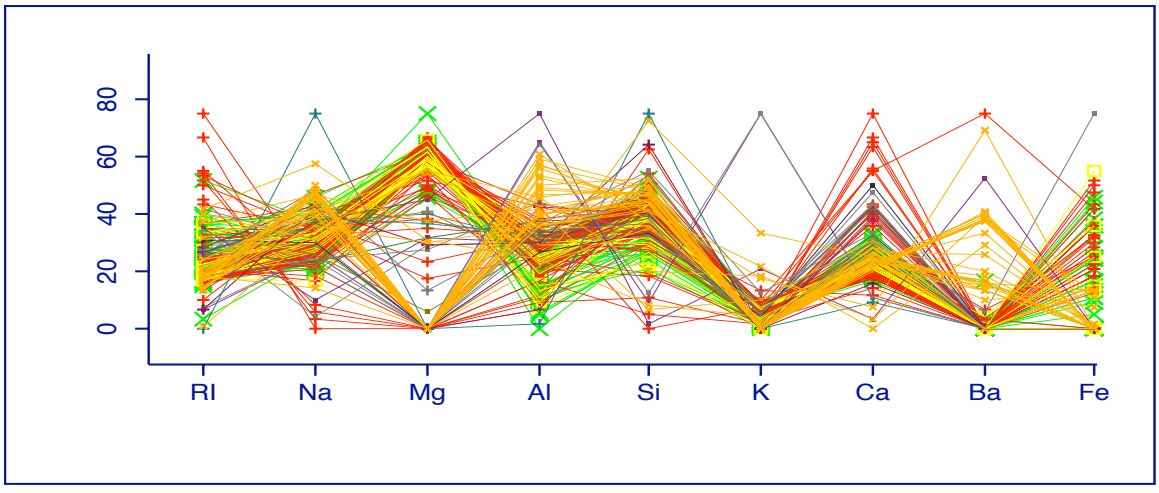


FIGURE 3. Parallel coordinates plot for the glass data

2. THE FRAMEWORK

Suppose we have collected data on N objects. Each object is characterized by an attribute vector x comprised of p elements, belonging to a suitable space (e.g. a subset of $\mathbb{R}^p$ or $\mathbb{Z}^p$). Each object needs to be classified into (assigned to) one of K prespecified classes (groups, types). In order to achieve this goal we need to *design* a **decision rule** ($\hat{c}$) that would assign object i with attribute vector x_i to class $c_k,\ k = 1, ..., K$

$$\hat{c}(x_i) \to c_k. \tag{1}$$

A good classification rule would be one that minimizes "inaccuracy," to be formally discussed in Section 2.1.

2.1. **A decision theoretic framework:** Let C denote the class label of a random feature vector $\mathbf{x} \in \mathbb{R}^p$. Assume that the prior probability in the population is given by $\pi_k = P(C = k),\ k = 1, \cdots, K$. Hence, the class labels are considered to be random. Let $\hat{c}(\mathbf{x}) : \mathcal{X} \to \{1, \cdots, K, D\}$ be a decision or classification rule, with $\mathcal{X}$ denoting the sample space and D the *in doubt* option.

In order to determine whether a classification rule is 'good' or not, we need a *loss function* that measures its quality. In classification, the most commonly used loss function is the 0/1 loss; i.e. $L(\ell, k) = 1$, if $\hat{c}(\mathbf{x}) = \ell$ but the true class is k, and 0 otherwise. Further, we assume that $L(\ell, D) = d, 0 < d < 1$ for all classes k. Finally, assume that the observations from class k have a *class conditional distribution function* denoted by $p_k(\mathbf{x})$ (i.e. density or probability mass function).

The risk function for classifier $\hat{c}$ is the expected loss as a function of the unknown class k:

$$R(\hat{c}, C = k) = E[L(\hat{c}, k)|C = k] = \sum_{\ell=1}^{K} L(\ell, k)P(\hat{c} = \ell|C = k) + L(D, k)P(\hat{c} = D|C = k)$$

The *total risk* is the total expected loss, viewing both the class C and the vector $\mathbf{x}$ as random:

$$R(\hat{c}) = ER(\hat{c}, C) = \sum_{k=1}^{K} \pi_k P(\hat{c}(\mathbf{x}) \neq k|C = k) + d \sum_{k=1}^{K} \pi_k P(\hat{c}(\mathbf{x}) = D|C = k).$$

This gives the overall misclassification probability plus d times the overall doubt probability.

Let $p(k|\mathbf{x}) = P(C = k|\mathbf{x} = x) = \frac{\pi_k p_k(\mathbf{x})}{\sum_{\ell=1}^{K} \pi_\ell p_\ell(\mathbf{x})}$ be the *posterior probability* of class k given a feature vector $\mathbf{x} = x$.

Proposition: The classification rule which minimizes the total risk under the 0/1 loss function is given by

$$c(x) = \begin{cases} k & \text{if } p(k|\mathbf{x}) = \max_{1 \le \ell \le K} p(\ell|\mathbf{x}) \text{ and exceeds } 1 - d \\ D & \text{if each } p(k|\mathbf{x}) \le 1 - d \end{cases}$$

Proof: We have that

$$R(\hat{c}) = E[E[L(\hat{c}(\mathbf{x}), C)|\mathbf{x} = x]] = \int E[L(\hat{c}(\mathbf{x}, C))|\mathbf{x} = x]p(\mathbf{x})d\mathbf{x}$$

where $p(\mathbf{x}) = \sum_{\ell=1}^{K} \pi_\ell p_\ell(\mathbf{x})$ is the marginal density for $\mathbf{x}$. It suffices to minimize the conditional expectation, which can be written as

$$E(\hat{c}) = \sum_{\ell=1}^{K} L(c, \ell)p(\ell|\mathbf{x})$$

with respect to c for each $\mathbf{x}$. For $c = D$ we have that

$$\sum_{\ell=1}^{K} L(\ell, D)p(\ell|\mathbf{x}) = d$$

Under our loss function, it is easy to see that the minimum is given by

$$1 - p(1|\mathbf{x}), 1 - p(2|\mathbf{x}), ..., 1 - p(K|\mathbf{x}), d$$

for $\hat{c} = 1, 2, ..., K, D$, respectively. So, the solution is given by the $\hat{c} = \max_{1 \le \ell \le K}\{p(K|\mathbf{x}), 1 - d\}$.

Under the 0/1 loss function, another way to write the optimal classification rule is to choose the class with the highest $\pi_k p_k(\mathbf{x})$, provided that it exceeds $(1 - d)p(\mathbf{x})$.

The optimal classification rule is referred to in the literature as the *Bayes rule*. When two or more classes attain the maximal posterior probability, the tie can in principle be broken in an arbitrary fashion. The value $R(c)$ of the total risk for the Bayes rule is called the

Bayes risk. This value is the best one can achieve if both the prior probabilities π_k and the class conditional distributions $p_k(\mathbf{x})$ are a priori known; hence, it provides a theoretical benchmark for all other procedures.

Some algebra shows that without the in doubt option, for a two class problem the Bayes risk is given by $E\min\{p(1|\mathbf{x}), p(2|\mathbf{x})\}$, while for K classes by $E[1 - \max_k p(k|\mathbf{x})]$.

Example: In order to illustrate the decision theoretic framework, we look at an example involving two normal populations with *common* covariance matrix. Assume that $p_k(\mathbf{x}) \sim N(\mu_k, \Sigma),\ k = 1, \cdots, K$ and disregard the doubt option. Then, an observation with feature vector $\mathbf{x} = x$ is allocated to the class k with smallest value of $\delta(x, \mu_k) - 2\log\pi_k$ (by calculating the posterior probability), where

$$\delta(x, \mu_k) = [(x - \mu_k)'\Sigma^{-1}(x - \mu_k)]^{1/2}.$$

Notice that the quadratic term $x'\Sigma^{-1}x$ is common to all classes and therefore the optimal rule can be written as

$$\text{minimize} - 2\mu_k'\Sigma^{-1}x + \mu_k'\Sigma^{-1}\mu_k - 2\log\pi_k, \text{ over } k = 1, \cdots, K.$$

If all classes are equally likely, then an observation with feature vector x is classified to the *nearest* population in the sense of having the smallest Mahalanobis distance to its mean (recall that this distance for a vector $x \in \mathbb{R}^p$ with mean μ and covariance Σ is defined as $d_M(x) = (x - \mu)'\Sigma^{-1}(x - \mu)$).

We revisit this example in more detail in Section 3.1.1.

2.2. **Learning a classification rule** $c(x)$**.** Notice that the theoretical framework previously defined assume perfect knowledge of the prior distribution over classes and of the class conditional distributions (or densities).

In order to make this decision theoretical framework *applicable* to real data, a *training data set* $T = \{c_i, x_i\}_{i=1}^N$ will be used to "learn" a classification rule $\hat{c}(x|T)$, by estimating $\hat{p}_k(x|T)$. Notice that under very mild conditions the frequency of N_k/N in T (i.e. the number of observations of class k in the training data over the total size of the training data) is a consistent estimate of π_k. There are in general two approaches for learning

$c(x)$: (i) parametric and (ii) non-parametric methods. In parametric methods, the class conditional densities are explicitly specified; e.g. multivariate normal or t distribution. In nonparametric methods, the class conditional densities are estimated from the training data, *without assuming a particular form*.

2.3. **Assessing the performance of classification rules.** The performance of the classification rule $c(x)$ is usually assessed by calculating the misclassification error rate; i.e. the probability of making an incorrect classification for a future randomly sampled observation. The misclassification error rate is the proportion of mistakes made, when classifying either a training or an *independent test* data set. If the training set is used, the error rate will be usually biased downwards relative to using an independent test data set, since the training data were used both for constructing the classifier and also assessing its performance. Hence, an independent test data set is preferable for performance assessment purposes. The error rate is calculated by classifying each observation in the test data, counting the errors and dividing by the size of the test data set. This measure is clearly unbiased, but can be highly variable. Further, the use of an independent test set wastes data that could have been used for training purposes, especially in cases where labeled examples are expensive to acquire, which is often the case in biological applications. In order to overcome this difficulty, the idea of *cross-validation* proves useful. Suppose that the training data T are partitioned into M pieces. Then, one part can be used as the test set and the remaining $M-1$ pieces for training purposes and this exercise can be repeated M times, with each part serving as the test set once. Each estimate of the error rate is unbiased and by averaging over the M estimates the variability is reduced. The extreme version of this strategy is to take $M = N$, the so called *leave-one-out* cross validation (Mosteller and Wallace, 1963). However, this is computationally the most demanding strategy. Further, excluding one observation leads to assessment of the classifier through $\mathcal{O}(1/N)$ perturbations from T. But the variability in the parameter estimates is usually of the order $\mathcal{O}(1/\sqrt{N})$, which implies that for large sample sizes they are calculated from smaller perturbations. Thus, cross validated estimates in this case can be rather variable; on the other hand, a smaller M may lead to larger bias, but smaller variance and consequently mean squared error.

2.4. **Connection to Statistical Learning Theory.** In Section 2.1, a decision theoretic framework was introduced for deriving optimal classification rules. It was shown that an optimal rule minimizes the Bayes risk. In practice, since the true distribution of the data is unknown and has to be estimated, one is interested in minimizing the empirical counterpart of the Bayes risk, defined as:

$$R_n(c) = \frac{1}{n} \sum_{i \in T} I(c(x_i) \neq y_i) \tag{2}$$

Since the risk $R(c)$ can not be computed directly, but only approximated by $R_n(c)$, it is not reasonable to look at classifiers (functions) that minimize $R_n(c)$ among all possible functions. The reason is that one can always construct such a function that performs perfectly on the training data and always misclassifies new observations. Hence, avoiding *over-fitting* the training data becomes an important consideration. This can be achieved in two ways: (i) restrict the class of functions (classification rules) over which the minimization takes place and (ii) modify the optimization criterion; for example, by adding a term that penalizes 'complicated' functions, or in other words *regularize* the problem under consideration.

A classification rule trained on n observations ($c_n(x)$) is designed to map inputs (variables x) to outputs (class labels). Its risk is a random variable $R(c_n)$ (since it depends on the data) and can not be computed directly. Statistical learning theory is interested in producing bounds for the following quantities:

- Error bound for the estimation of the risk from an empirical quantity.
- Error bound for the rule given the functional class it is assumed to belong to.
- Error bound relative to the Bayes risk.

These questions have attracted a lot of attention over the last decade; a good overview is given in Vapnik's book (1996), in Scholkopf et al. (2002) book and in the review paper by Bousquet et al. (2004.)

3. CLASSIFICATION METHODS

In this section, we discuss a number of classification methods widely used in practice.

3.1. **Classification Rules via Probability Models.**

3.1.1. *Linear and Quadratic Discriminant Analysis.* The objects in class k are assumed to be sampled from a normal distribution with mean vector μ_k and covariance matrix Σ_k. Then, the Bayes rule decision rule minimizes

$$(3) \quad Q_k = -2\log(p(x|k) - 2\log(\pi_k) = (x-\mu_k)'\Sigma_k^{-1}(x-\mu_k) + \log|\Sigma_k| - 2\log(\pi_k).$$

The first term of (3) corresponds to the Mahalanobis distance of object i from the center of class k. This rule is known in the literature as *quadratic discriminant analysis*.

The expression in (3) simplifies if one is willing to assume equal covariance matrices amongst the K groups; i.e. $\Sigma_k = \Sigma$ for $k = 1, \cdots, K$. In that case the rule minimizes,

$$(4) \qquad Q_k = -2\log(p(x|k) - 2\log(\pi_k) = \mu_k\Sigma^{-1}x + \mu_k'\Sigma^{-1}\mu_k - 2\log(\pi_k).$$

Therefore, the rule becomes *linear* in the data x. For a 2-class problem, it corresponds to Fisher's linear discriminant that was derived based on a different criterion. The above rule is known in the literature as *linear discriminant analysis*.

In practice, the following quantities need to be estimated from the training data set T by their sample counterparts: π_k, μ_k and Σ_k. The maximum likelihood estimates are given by $\hat{\mu}_k = \bar{X}_k$ (the multivariate mean for each class k and $\hat{\Sigma}_k = \frac{1}{n_k}\sum_{j=1}^{n_k}(x_j - \hat{\mu}_k)(x_j - \hat{\mu}_k)'$. For linear discriminant analysis, the pooled covariance estimate $\hat{\Sigma} = \sum_{k=1}^{K}(n_k/N)\hat{\Sigma}_k$ is used. Often the bias-corrected estimator of Σ with divisor $N - K$ is used, but makes very little difference to the linear rule (and none if the class prior probabilities π_k are the same).

It should be noted that in linear discriminant analysis $K(p+1) + p(p+1)/2$ parameters need to be estimated from the training data: K means μ_k of dimensionality p plus K values of the vector of prior probabilities π and $p(p+1)/2$ parameters for the common covariance matrix Σ. In contrast, $K(p+1) + Kp(p+1)/2$ parameters need to be estimated in quadratic discriminant analysis; again, $K(p+1)$ parameters for the class means and the prior probability vector plus K class specific covariance matrices Σ_k. This suggests that unless there are sufficient training data for each class, parameter estimates can be rather volatile, which in turn implies that the quadratic rule can be well outperformed by the

linear one for moderate sample sizes. For this reason, in practice a diagonal version of quadratic discriminant analysis is often employed, where $\hat{\Sigma}_k = \text{diag}(\hat{\sigma}_j^k),\ j = 1, \cdots, p$.

Another multivariate distribution used for classification purposes is the multivariate t with a moderate number of degrees of freedom, that exhibits heavier tails than the multivariate normal. It can be thought of as a flexible model for distributions with elliptical densities with heavy tails. The density with $\nu > 2$ degrees of freedom is given by

$$p(x|k) = \frac{\Gamma(\frac{1}{2}(\nu+p))}{(\nu\pi)^{p/2}\Gamma(\nu/2)}|\Sigma_k|^{-1/2}\big(1+\frac{1}{\nu}(x-\mu_k)'\Sigma_k^{-1}(x-\mu_k)\big)^{-1/2(\nu+p)}. \tag{5}$$

The optimal classifier minimizes

$$Q_k = \frac{\nu+p}{2}\log\big(1+\frac{1}{\nu}(x-\mu_k)'\Sigma_k^{-1}(x-\mu_k)\big) + \frac{1}{2}\log|\Sigma_k| - \log(\pi_k). \tag{6}$$

If the covariance matrix is common to all classes, a linear rule is once again obtained. The effect of the heavier tails of the t distribution is to down-weigh observations which are far from the class mean.

An illustration of the quadratic type of decision boundaries produced by linear and quadratic discriminant analysis is given in Figure 4. It can be seen that the linear discriminant analysis misclassifies four observations, while the more flexible boundary of quadratic discriminant analysis results in three misclassifications.

Parametric models can be used in more complex situations, where for example the data exhibit clear multi-modality. In that case, one can employ a finite mixture of multivariate normal distributions as the conditional class density. However, the problem of estimating all the parameters involved (means, covariances and mixing coefficients) is not a trivial one and care should be taken regarding identifiability issues (see Venables and Ripley, 1994).

3.1.2. *Logistic Discrimination.* This model arises when one wants to model the posterior probabilities of the K classes through linear functions of x. As a motivating example, consider the normal model for the K classes with common covariance matrix. By comparing the ratio of the class k posterior probability to that of class 1, we obtain

$$\log(p(k|x))/\log(p(1|x)) = (x-\mu_k)'\Sigma^{-1}(x-\mu_k) - (x-\mu_1)'\Sigma^{-1}(x-\mu_1) + \log(\pi_k/\pi_1) =$$

$$(\mu_k-\mu_1)'\Sigma^{-1}x - (\mu_k+\mu_1)'\Sigma^{-1}(\mu_k-\mu_1) + \log(\pi_k/\pi_1) = \alpha_k + \beta_k'x,$$

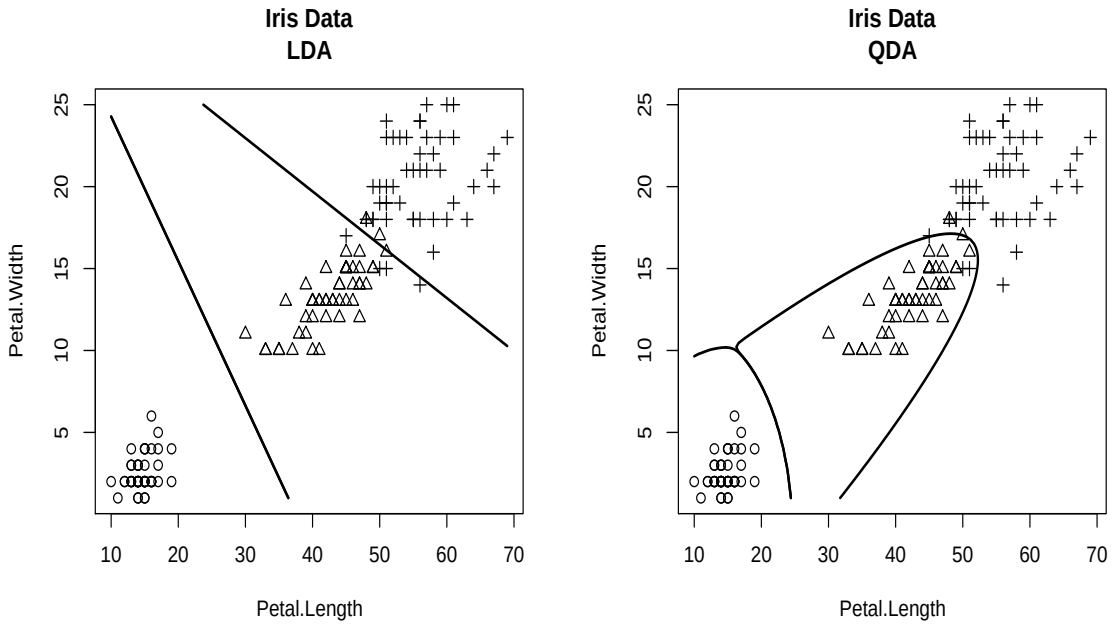


FIGURE 4. Decision boundaries for linear (left panel) and quadratic discriminant analysis (right panel) for the iris data set.

a linear model in x.

In general, we can model $\log(p(k|x)) - \log(p(1|x))$ by

$$p(c = k|x) = \frac{\exp(\beta_k' x)}{\sum_{k=1}^{K} \exp(\beta_k' x)}, \tag{7}$$

thus modeling the log-odds by a linear function. This model implies that odds increase multiplicatively by $exp(\beta_k)$ for every unit increase in the value of the attribute x_h. This procedure can be generalized to allow modeling of the log-odds by more general functions.

The logistic regression model given in (7) is estimated by maximum likelihood methods. Since the resulting score equations are nonlinear in the parameter β_k, an iterative algorithm like Newton-Raphson needs to be employed (for more details, see Hastie et al. 2001).

3.2. **Nonparametric Methods.** In case one is not willing to assume that the underlying populations come from a known parametric distribution (e.g. multivariate normal, or multivariate t), we need to resort to nonparametric methods for estimating $p(x|k)$ and its sample

counterpart $p(x|k,T)$. The naive estimate of the underlying density is the sample histogram of the training dataset T. The main problem with this estimate is its lack of smoothness. A possible solution is the use of *kernel* methods to smooth it. The estimate then takes the following form

$$p(x|k,T) = \frac{1}{n_k} \sum_{i=1}^{n_k} \frac{1}{h} K(\frac{x - x_i}{h}), \tag{8}$$

where h is the *bandwidth* and K the *kernel* that satisfies $\int_{\text{sample space}} K(x)dx = 1$.

Remarks:

- The choice of h (a scalar parameter) controls the smoothness of the resulting estimate. For small values of h we get rough estimates, while for large values of h fairly smooth ones.
- Possible choices of the kernel function K include the Gaussian, the rectangular, the triangular, the Epanechnikov, etc.
- There are various automatic approaches for choosing the bandwidth.
- In high dimensions most of the underlying sample space is empty of objects. The latter fact implies that we need to choose either a very large value of h, or use a product estimate of $p(x|k,T)$.

3.2.1. *Nearest Neighbor Methods.* One simple adaptive kernel method is to choose the kernel K to be constant over the nearest r objects and zero elsewhere. As mentioned in Ripley (1996), this does not in fact define a density as the estimate of $p(x|k)$, since its integral is infinite but we can nevertheless estimate the posterior distribution as the proportions of the classes amongst the nearest r objects. The resulting classification rule is known as the *nearest neighbor rule* and dates back to Fix and Hodges (1951).

The basic steps to classify a new object with attribute vector x_0 are:

Step 1: Determine the r nearest neighbors (usually with respect to the Euclidean distance) of the new object in T.

Step 2: Classify the new object to the class that contains the majority of the r nearest neighbors.

The version with $r = 1$ corresponds to dividing the data space into the cells of the Dirichlet tessellation of the data points, and every new observation is classified according to the label of the cell it falls in. The behavior of the 1-nearest neighbor classification can be characterized asymptotically. Specifically, let E^* denote the error rate of the Bayes rule in a K-class problem. Them the error rate of the 1-nearest neighbor rule *averaged* over training sets converges to a value E_1 which is bounded above by $E^*(2 - K/(K-1)E^*)$ (see Cover and Hart, 1967). In another development, Stone (1977) showed that if the number of neighbors used $r \to \infty$, while $r/N \to 0$, the risk for the r-nearest neighbor rule converges in probability to the Bayes risk (not averaged over training data sets). However, these results are asymptotic in nature and do not necessarily apply to finite samples. In particular, these results are independent of the metric used for defining the nearest neighbor (e.g. Euclidean). Experience shows that the choice of metric can be important.

The flexible boundaries produced by nearest neighbor classifiers are illustrated on the iris data set in Figure 5, where the value $r = 3$ was chosen. It can be seen that one observation from the Viriginica class is classified as Versicolor and vice versa, thus exhibiting a slightly better apparent misclassification error rate than linear discriminant analysis.

3.2.2. *Classification Trees.* The goal of classification trees is to partition the *sample space* (feature space) into *hypercubes* and assign a class to every hypercube.

We illustrate their use by an example. In the left panel of Figure 6 the classification boundaries for the three classes in the iris data are shown, corresponding to the tree shown in the right panel. It can be seen that if the objects' petal length is less than 24.5 then they are assigned to class Setosa, while if it is larger than 24.5 to some other class. Then, if petal width is larger than 17.5 and petal length less than 49.5 the objects are classified to class Virginica, and so on. In the plot, the misclassifications are also included; for example, all the observations in the Setosa class are correctly classified, while 5 observations in the Viriginica class are classified as Versicolor, etc. This is an example of a binary tree, where each node *splits* into two branches. The number of possible binary splits is $m(N-1)$. The main advantage of binary splits is their inherent simplicity: numerical variables are split according to whether their values are above or below some threshold value τ, and the same holds for ordinal variables; nominal variables with L levels can be split in $2^{L-1} - 1$ possible

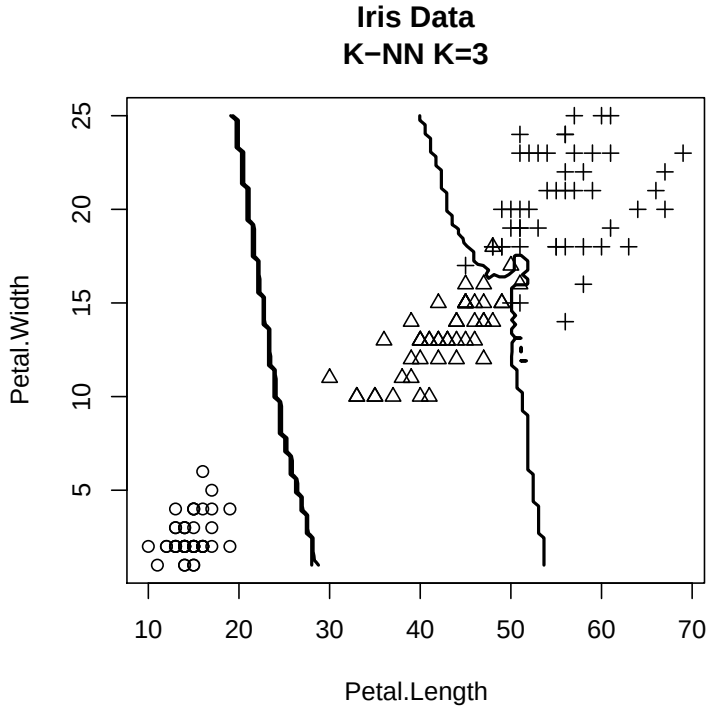


FIGURE 5. Decision boundaries produced by $r = 3$ nearest neighbor classifier for the iris data set.

ways. Classification trees are *rooted*, with the root corresponding to the top node. Observations are passed down the tree along the branches, with decisions being made at each node until a terminal node, also called *leaf* node, is reached. Each non-terminal node contains a question on which a split is based, while each leaf contains the label of a classified observation.

It can be seen that the goal of this procedure is to split the sample space in such a way that most members belonging to a particular class fall in the same hypercube. However, nothing prevents the tree to grow in such a way so that there is a single object only in each hypercube. The main issues then become on how the tree should be constructed and also pruned, in order to avoid the one object per hypercube phenomenon.

Almost all current tree-construction methods use a one-step lookahead approach. That is, they choose the next split (which variable to use in the next split) in an optimal way,

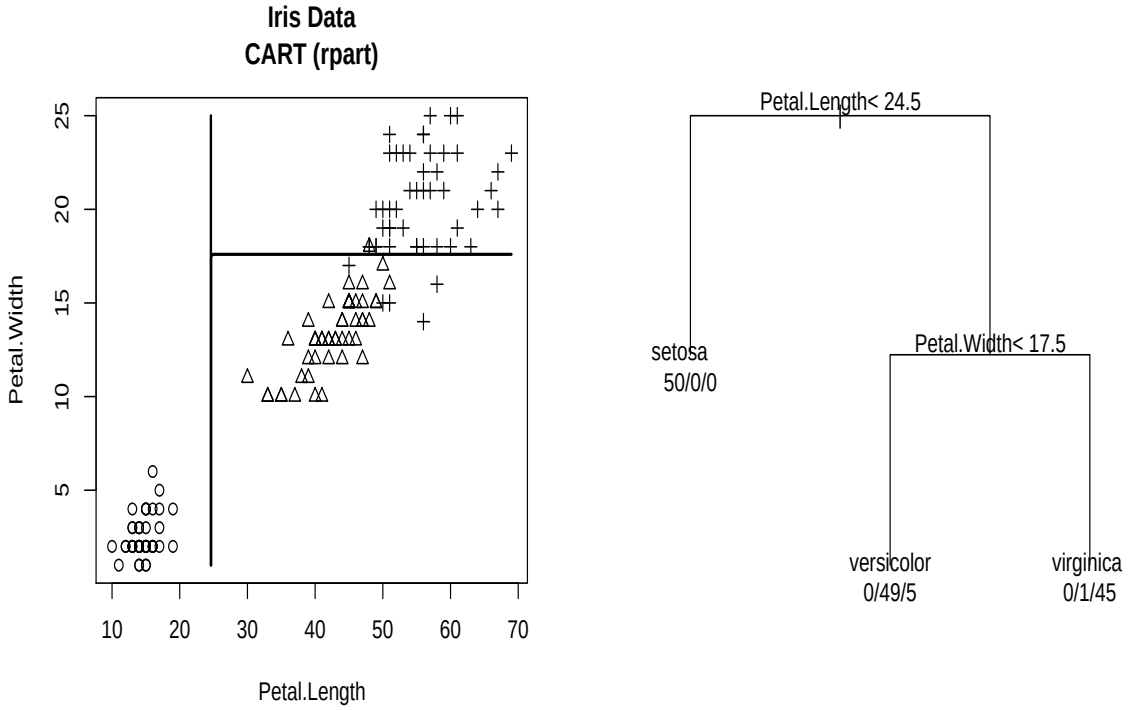


FIGURE 6. Left panel: classification boundaries for the Iris data. Right panel: Corresponding classification tree with number of misclassifications. For the figure shown in the right panel, the left branch corresponds to when the posited condition is satisfied (i.e. petal length < 24.5), while the numbers under the leaf nodes indicate the composition of the partition (i.e. for the left most partition, all objects are classified as Setosa, 0 as Versicolor and 0 as Virginica, while for the middle partition 0 objects are classified as Setosa, 49 as Versicolor and 5 as Virginica.

without attempting to optimize the performance of the whole tree. A common measure for choosing splits is the *deviance*. At each node ν of a tree we have a probability distribution $p_{\nu k}$ over the classes. By conditioning on the observed attributes x_i of the objects in the training dataset T we learn the numbers n_ν assigned to every node of the tree. This allows

us to define a deviance for node ν (through the conditional likelihood) as

$$D_\nu = -2\sum_{k=1}^{K} n_{\nu k}\log(p_{\nu k}) \tag{9}$$

and for the tree as a whole $D = \sum_\nu D_\nu$. Now consider splitting a particular node ν into nodes ν_1 and ν_2. The split that maximizes the reduction in the deviance $D_\nu - D_{\nu_1} - D_{\nu_2}$ is the one that is chosen. Other approaches for choosing future splits involve entropy measures ($\sum_k p_{\nu k}\log(p_{\nu k})$ and the Gini index ($1 - \sum_k p_{\nu k}^2$) (see Breiman et al. (1984)).

With noisy data there is always the danger of over-fitting the training dataset T. The established methodology to avoid this problem is to "prune" the tree.

For any tree TR, define $R(\text{TR})$ to be the error rate (i.e. the number of misclassifications) in T. Define an error-complexity measure

$$C_\alpha(\text{TR}) = R(\text{TR}) + \alpha\text{Size}(\text{TR}), \tag{10}$$

which we are interested in minimizing over all trees. Notice that for $\alpha = 0$ we are only interested in minimizing $R(\text{TR})$ while for $\alpha \to \infty$ we are only interested in constructing the simplest possible tree. Note that by choosing a particular value of α appropriately we can strike a good compromise between the two objectives; namely minimizing the number of misclassifications vs the size of the tree. The value of $\alpha = 2(K-1)$ corresponds to the tree with minimum AIC value.

3.2.3. *Support Vector Machines.* Support vector machines and kernel methods have proven to be powerful tools in numerous applications and have thus gained widespread popularity e.g. in machine learning and bioinformatics. In order to motivate the idea behind the technique we start our exposition with the simplest case, that of two *separable classes by a linear boundary*. The data in the training set T are the pairs $(y_i, x_i),\ i = 1, \cdots, N$ with the response taking values in the set $\{-1, 1\}$ (positive and negative examples) and $x_i \in \mathbb{R}^p$. Suppose there exists a *hyperplane* that *separates* the positive from the negative examples. The points x that lie on the hyperplane satisfy $< w, x > + b = 0$, where w is a vector normal (orthogonal) to the hyperplane and $b/||w||$ is the perpendicular distance from the hyperplane to the origin, with $||w||$ denoting the Euclidean norm of w and $< \cdot, \cdot >$ the inner product operator. Further, define the *margin* of a separating hyperplane to be m_+

and m_- for the positive and negative examples, respectively. The margin corresponds to the distance between the hyperplane and the closest postive or negative example. For the linearly separable case, the support vector algorithm searches for the separating hyperplane with the largest margin. To formulate such a program notice that in the current setting all the training data satisfy: $< w, x_i > +b \geq +1$ for $y_1 = +1$ and $< w, x_i > +b \leq -1$ for $y_i = -1$. Consider the the positive and negative examples for which these relations hold with equality. The positive examples lie on a hyperplane $h_+ :< w, x_{i>+b=1}$ with a normal vector w and perpendicular distance from the origin $|1-b|/||w||$, while the negative examples on another hyperplane $h_- :< w, x_i > +b = -1$, with the same normal vector w and perpendicular distance to the origin $|-1-b|/||w||$. Therefore, $m_+ = m_- = 1/||w||$ and the margin is given by $2/||w||$. As shown in Figure 7, h_+ and h_- are parallel hyperplanes, since they have the same normal vector and no training points fall between them. Hence, we can find the pair of hyperplanes that gives the maximum margin by minimizing

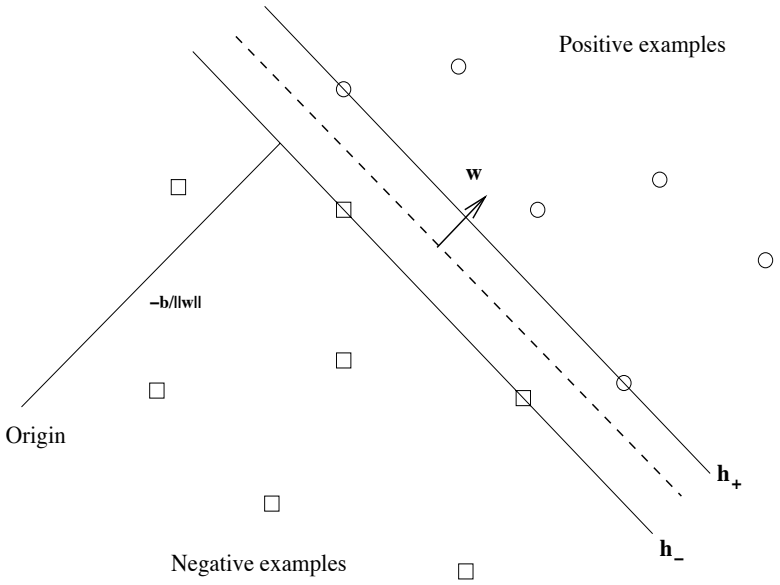


FIGURE 7. Illustration of linear separating hyperplanes for the separable case

Hence, we can find the pair of hyperplanes that gives the maximum margin by minimizing

$$\min ||w||^2 \text{ subject to } y_i(< w, x_i > +b) - 1 \geq 0, \text{ all } i \in T \tag{11}$$

This is a convex optimization problem with a quadratic criterion and linear constraints, whose solution can be found by first switching to a Lagrangian unconstrained formulation and then solving its corresponding dual problem.

The above formulation and the resulting quadratic programming algorithm, when applied to non-separable data, will find no feasible solution. A relaxation of the constraints on h_+ and h_- is required, which can be achieved by the introduction of *slack* variables $\xi_i \geq 0$ for all i. The defining hyperplanes are then given by $h_+ :< w, x_i > +b \geq 1 - \xi_i$ for $y_i = +1$ and $h_- :< w, x_i > +b \geq -1 + \xi_i$ for $y_i = -1$. Hence, when mistakes occur, the corresponding ξ_i must be larger than one and so $\sum_i \xi$ serves as an upper bound on the number of training errors. A natural way to assign costs for errors is to change the objective function to be minimized from $||w||^2$ to $||w||^2 + a(\sum_i \xi)$, with a being a tuning parameter that controls the assigned penalty in the presence of mistakes. This leads once again to a quadratic programming problem (for more details on the Lagrangian formulation of the dual, see Hastie et al. (2001)).

The resulting classifier is given by:

$$c(x) = \text{sign} < w, x > +b, \tag{12}$$

where $w = \sum_{i \in \mathcal{S}} \alpha_i y_i x_i$, α_i are obtained from the solution of the dual problem and $\mathcal{S}$ denotes the set of examples that define the support vectors h_+ and h_-.

In most real life examples, the two classes are not going to be approximately linearly separable and that is where the so-called 'kernel trick' comes into play. The main idea is that separation of the classes by a hyperplane may be easier in higher dimensions. The construction depends on inner products that have to be evaluated in the variables space; the latter task may become computationally intractable, if the dimensionality of that space is too large. Kernel (generalized covariance) functions that are defined in lower dimensional space, but behave like inner products in higher (possibly infinite) dimensional space will accomplish this goal, as originally observed by Boser et. al (1992). Suppose that the data are mapped to some other Euclidean space Q through:

$$\Phi : \mathbb{R}^p \longrightarrow Q, \tag{13}$$

where Q may be infinite dimensional. Notice that the training algorithm would depend on the data through inner products of the form $< \Phi(x_{ell}), \Phi(x_j) >$ in Q space. The challenge is to find a bivariate function K such that $K(x_\ell), x_j) =< \Phi(x_{ell}), \Phi(x_j) >$; this would require only knowledge of K in the computations. Examples of K include γ-degree polynomials of the form $K(x_\ell, x_j) = (1+ < x_\ell, x_j >)^\gamma$, radial basis functions $K(x_\ell, x_j) = \exp(-||x_\ell - x_j||^2/u)$ and neural network functions $K(x_\ell, x_j) = \tanh(\eta_1 < x_\ell, x_j > +\eta_2)$. The classifier for a new point x is given by

$$c(x) = \sum_{i \in \mathcal{S}} \alpha_i y_i K(x, x_i). \tag{14}$$

Remark 1: An extensive treatment of the theory of support vector machines, together with learning with kernels is given in the book by Scholkopf and Smola (2002). Another fairly extensive presentation and connections to a regularization framework is given in Hastie et al. (2001).

Remark 2: Support vector machines have proved particularly successful in a variety of applications. Lin (2002) investigated their empirical success and argued that they implement the Bayes classifier in an efficient manner. Specifically, as discussed above, the Bayes rule minimizes the misclassification error rate, given by $\mathbb{P}(y = +1|x) - 1/2$ for the positive examples class. Lin (2002) showed that the support vector machines solution targets directly this rate without estimating the posterior probability of the positive (or negative) class.

The classical support vector machine paradigm presented above, primarily deals with binary (2-class) classification problems and has a nice geometric interpretation. The most common strategy adopted to solve multi-category classification problems is to treat them as a series of of binary problems, with the positive examples corresponding to class k and the negative examples to all the remaining classes. Dietterich and Bakiri (1995) discuss a general scheme for carrying out this task and Allwein et al. (2000) proposed a unifying framework for it. More direct formulation of the multi-category problem for support vector machines are given in Cramer and Singer (2001) and Lee et al. (2004).

The flexible decision boundaries produced by support vector machines are shown in Figure 8. It can be seen that five observations from the training data set are misclassified. It should

be noted that in the presence of only two variables, support vector machines can not take full advantage of their flexibility.

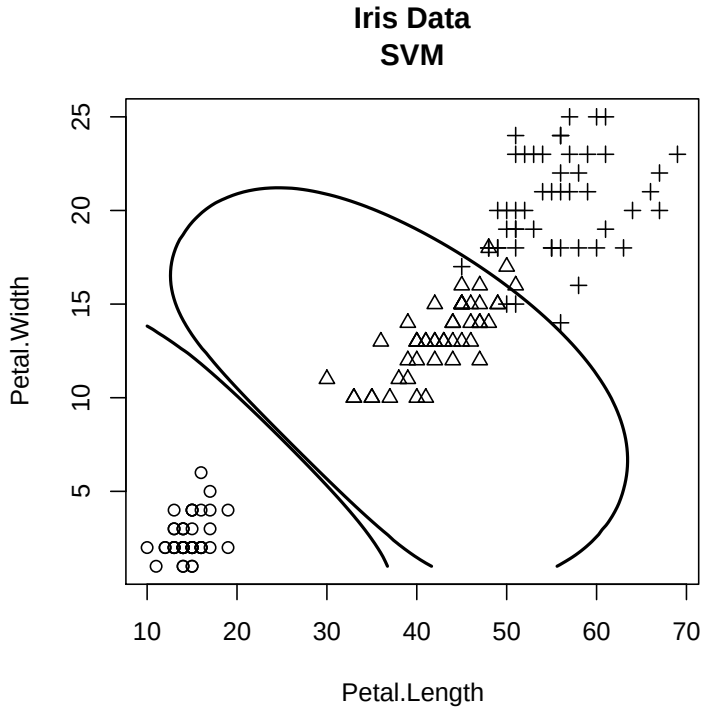


FIGURE 8. Classification boundaries for the Iris data using SVM and a radial basis function kernel.

3.2.4. *Neural Networks.* The term *neural network* encompasses a wide variety of models. We focus on the simplest possible model. the single layer perceptron, to convey the main ideas. It should be noted that the literature on the subject is very extensive and good references are the books by Bishop (1995) and Ripley (1996); some recent advances are presented in the edited volume by Bishop (1998). Neural networks can be thought of as non-linear statistical models as discussed below.

The single layer perceptron is a two-stage classification model that can be represented schematically by an acyclic graph (see Figure 9). At the top there are K nodes, representing the probability of class k, coded as $y_k \in \{0, 1\}$. At the bottom layer the nodes

represent the p variables. The intermediate layer nodes represent linear combinations of the variables that are subsequently used for predicting the outcome variables. Specifically, let $Z_m = h(X\alpha),\ m = 1, \cdots, M$, with $h(\cdot)$ usually chosen to be the sigmoid function $h(u) = 1/(1 + \exp(u))$. Other choices for h include radial basis functions Further, let $g_k(y_k) = Z\beta,\ k = 1, \cdots, K$, which allows an additional non-linear transformation for the output vector. A popular function for $g(\cdot)$ is the softmax function given by $g(u_k) = \exp(u_k)/(\sum_{\ell=1}^{K} \exp(u_k))$. The introduction of the functions h and g significantly increases the space of available models. More complicated models can be derived by considering a larger number of intermediate layers (for more see Ripley (1996)).

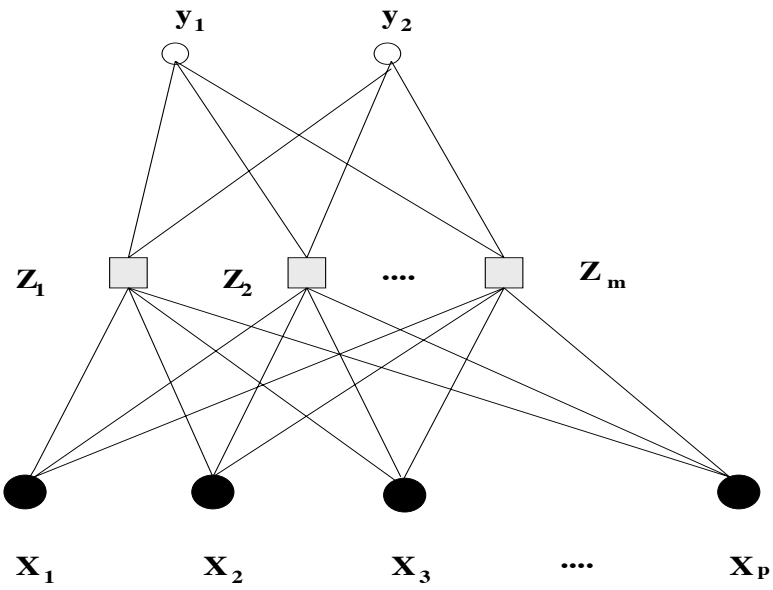


FIGURE 9. Illustration of single layer perceptron with two classes

3.2.5. *Ensemble Methods: Bagging and Boosting.* In this section, we discuss some techniques that do not produce a single classification rule, but an *ensemble* of classifiers. We focus on bagging and its extension random forests and on boosting.

Bagging:

The main idea is to generate *perturbed* versions of the training data of the same size as the original set T, train a classifier on each perturbed version and aggregate the results

by majority voting. The perturbed versions are obtained by bootstrapping observations from T. The bootstrap aggregation or *bagging* algorithm introduced in Breiman (1996) is described next:

(1) Create B bootstrapped training data sets $T_1, \cdots, T_B$
(2) Train a classifier $c_b(x)$ on each bootstrapped training data set T_b
(3) Output: for a new observation x, classify it to the majority class predicted by the B classifiers

The final prediction is based on averaging a number of individual predictions produced by each of the B classifiers. It can be shown that bagging reduces the variance of the classification procedure, but does not affect its bias. Further, empirical evidence suggests that high variance classifiers (those with the tendency to overfit the data, such as classification trees) primarily benefit from bagging. Unlike techniques that produce a single classification rule, bagging produces an ensemble of rules that although may exhibit superior performance in terms of misclassification error rate, they are nevertheless hard to interpret as a 'model'. A variation of bagging based on creating convex pseudo-sets of training data was proposed in Breiman (1998). The idea is to generate a new training set where the observations are linear combinations of two observations selected at random from the original training set T. This procedure is repeated B times, so that the desired classifier can be trained on B perturbed training data sets and its results averaged through majority voting as in bagging.

Empirical evidence showed that bagging improved performance in practice but not by a wide margin. A variation of bagging specifically designed for classification trees as the base classifier was introduced in Breiman (2001). The main steps of the algorithm, called Random Forests, are described next:

(1) Create B bootstrapped training data sets $T_1, \cdots, T_B$. The observations not included in the bootstrapped sample T_b form an *out-of-bag* sample
(2) Train a classification tree on each bootstrapped training data set T_b as follows:
at every node, consider *only a random subset of variables* for the next split
Note that trees are recommended to be grown to maximum size and *not pruned*
(3) Output: for a new observation x, classify it to the majority class predicted by the B classifiers

Empirical evidence suggest that over-fitting due to growing the tree to maximum size is not an issue. Further, selection of a small number of variables at every node works well and the performance of random forests seems to be insensitive to it. Finally, the algorithm provides a mechanism for estimating the importance of each variable in the ensemble. Specifically, to assess the importance of variable j its values in the out-of-bag data set are randomly permuted and each out-of-bag sample is run down the B classification trees in the ensemble and a prediction obtained. Repeating this procedure for all the out-of-bag samples of variable j a misclassification error rate is computed. Following the same steps smisclassification error rates are computer for the remaining variables. These p misclassification error rates are then compared to those obtained from the out-of-bag samples if all the variables were kept intact. The percentage increase claulctaed between the perturbed and original misclassification error rates gives a measure of variable importance.

Boosting:

Boosting has proved to be an effective method to improve the performance of base classifiers, both theoretically and empirically. The underlying idea is to combine simple classification rules (the base classifiers) to form an ensemble, whose performance is significantly improved. The origins of boosting lie in PAC learning theory (Valiant, 1984), which established that learners that exhibit a performance slightly better than random guessing when appropriately combined can perform very well. A provably polynomial complexity boosting algorithm was derived in Schapire (1990), whereas the Adaptive Boosting (AdaBoost) algorithm and its numerous variants (Freund and Schapire, 1996, Freund and Schapire, 1997) proved to be a practical implementation of the boosting ensemble method. The basic steps of the AdaBoost algorithm are described next for a binary classifier. Let $c(x)$ be the classifier under consideration and $(y_i, x_i),\ i = 1, \cdots, N$ the data in the training set T. Then,

(1) Initialize weights $w_i = 1/N$
(2) for $m = 1$ to M do:
(3) fit $y = c_m(x)$ as the base weighted classifier using weights w_i
(4) let $W_{c_m} = \sum_{i=1}^{N} w_i I(y_i c_m(x_i) = -1)$ and $\alpha_m = \log[(1 - W_{-}(h))]/W_{-}$
(5) $w_i = w_i \exp\{\alpha_m I(y_i \neq c_m(x_i))\}$ rescaled to sum to one

(6) end for

(7) Output: $\hat{c}(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m c_m(x))$

The popular intuition behind the success of the algorithm is that hard to classify observations receive higher weights α_i at later iterations; thus, the algorithm concentrates more on these cases and hence manages to drive the misclassification error rate on the training data to zero.

Remark 1: Friedman et al. (2000) established connections between the AdaBoost algorithm and statistical concepts such as loss functions, additive models and logistic regression. Specifically, it was shown that this algorithm fits a stage-wise additive logistic regression model that minimizes the expectation of the exponential loss function $\exp(-yc(x))$. A good discussion is also included in Hastie et al. (2001). For some interesting observations on boosting see also Mease and Wyner (2006).

Remark 2: It can be seen that this algorithm produces as output the new observation's predicted class. Proposals for the algorithm to produce as output the class probability estimates include the RealBoost and the GentleBoost algorithms (Friedman et al. (1998)).

Remark 3: The boosting algorithm performs a gradient descent for minimizing the underlying exponential loss function and consequently determine the weights and the classifier at each iteration. However, this is a greedy strategy, since the solution pays attention to reducing the misclassification error rate on the training data. A variant of the algorithm that focuses more on the misclassification error rate for future observations along with a regularized version (Stochastic Gradient Boosting) are discussed in Friedman (2002).

3.3. **Variable Selection.** Variable selection is an important topic in classification, especially in applications where the number of variables exceeds the number of available observations in the training data set T ($p >> n$). Two domains where $p >> n$ are text classification and gene/protein microarray data. The goal of variable selection is to improve the predictive performance of the classifier, to reduce its computational cost and finally to provide a more interpretable rule. From the description of the most commonly used techniques to train classifiers, it can be seen that some algorithms have a *built-in* variable selection mechanism; examples include classification trees, random forests. On the

other hand, logistic discrimination, support vector machines and especially nearest neighbor methods do not offer such mechanisms.

Various approaches have been proposed in the literature to deal with this topic. We briefly outline two popular approaches: (i) regularization and (ii) dimension reduction. The idea behind regularization is to modify the objective function that is being optimized by adding a penalty term that penalizes complex models - in this setting, models that include a large number of variables. This applies directly to logistic, support vector machines and linear and quadratic discriminant analysis. The ideal penalty would be an ℓ_0 norm that would eliminate 'unnecessary' variables (the ℓ_0 norm for a vector $x \in \mathbb{R}^p$ is defined as follows: $||x||_0 = \{i : x_i \neq 0\}$, i.e. the number of non-zero elements of the vector). However, it generally leads to a non-convex optimization problem which in turn is hard to solve. Hence, the ℓ_1 norm has become a very popular alternative, since it shrinks the coefficients of variables to zero, thus effectively performing variable selection, while at the same time keeping the underlying optimization problem convex. Another popular penalty is the ℓ_2 norm that shrinks the coefficients of the variables, but not all the way to zero. This penalty is best suited for reducing the mean squared error of the estimated model, but does not eliminate variables. Another approach is based on the idea of reducing the dimensionality of the variable space. Possible techniques include principal components analysis and clustering. Principal components analysis creates new variables that are linear combinations of the original ones, which can then be used as inputs to the various classification techniques. Clustering of variables replaces a group of 'similar' variables by a representative one, that can be their weighted average. A more ad hoc method is variable ranking, which is attractive because of its simplicity and scalability. A classification rule is used with a single variable and the final output is a ranking of all variables according to their predictive ability. This procedure is not necessarily designed to produce a good final model, but to discard the most 'uninformative' variables.

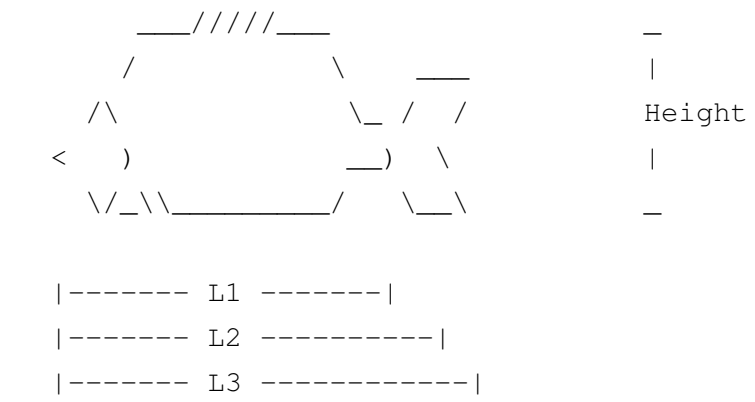
4. Classification techniques in R

We are going to examine next implementation of various classification techniques introduced in the previous sections in R. The data set to be used for illustration is the fish data

available from www.amstat.org. It consists of 157 fishes of 7 species that were caught and measured in the lake Laengelmavesi near Tamper in Finland.

A description of the variables is given next along with a picture that explains some of the them.

```
Weight      Weight of the fish (in grams)
Length1     Length from the nose to the beginning of the tail (in cm)
Length2     Length from the nose to the notch of the tail (in cm)
Length3     Length from the nose to the end of the tail (in cm)
Height%     Maximal height as % of Length3
Width%      Maximal width as % of Length3
```



After reading the data, we examine the distribution of the 7 classes.

```
fish=read.table('fish-data.txt',h=T)

Class=fish[,1]
table(Class)

Class
```

```
    Bream    Parkki     Perch      Pike     Roach     Smelt Whitewish
       34        11        56        17        19        14         6
```

The next table gives the correlation matrix for the 6 variables in the data set.

```
  print(cor(fish[,-1]),digits=3)
       Weight     L1     L2     L3 Height  Width
Weight  1.000 0.9165 0.9194 0.9245 0.1944 0.1355
L1      0.916 1.0000 0.9995 0.9921 0.0351 0.0318
L2      0.919 0.9995 1.0000 0.9942 0.0548 0.0443
L3      0.925 0.9921 0.9942 1.0000 0.1326 0.0377
Height  0.194 0.0351 0.0548 0.1326 1.0000 0.4561
Width   0.136 0.0318 0.0443 0.0377 0.4561 1.0000
```

It can be seen that the three length variables are very highly correlated. This finding implies that the variables L2 and L3 contribute very little additional information. Whether this would have any consequences for classification remains to be seen. The parallel coordinate plot along with some 2-dim and 3-dim views of the data are given in Figures 4 and 4.

In some of the plots we see a fairly nice separation of the classes. But given the very high correlations between the length variables we decided to compute the following new variables:

```
L21=fish[,4]-fish[,3]
L31=fish[,5]-fish[,3]
L32=fish[,5]-fish[,4]

fish=data.frame(fish,L21,L31,L32)
```

An inspection of pictures of some of the fish contained in this data set suggest that these variables are informative.

The correlation matrix becomes

```
       Weight     L1     L2     L3 Height  Width   L21    L31
Weight  1.000 0.9165 0.9194 0.9245 0.1944 0.1355 0.876 0.7460
L1      0.916 1.0000 0.9995 0.9921 0.0351 0.0318 0.906 0.7218
```

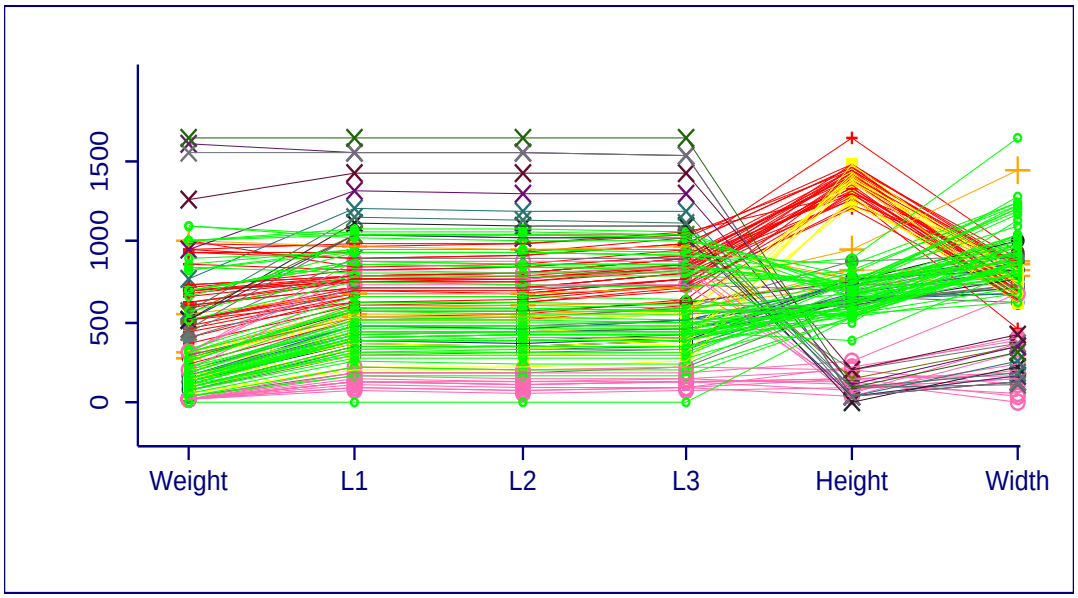


FIGURE 10. Parallel coordinates plot for the fish data. Bream: red +, White: orange big+, Roach: blue solid circle, Parkki: yellow solid square, Smelt: pink open circle, Pike: cyan x, Perch: green small open circle

```
L2      0.919 0.9995 1.0000 0.9942 0.0548 0.0443 0.919 0.7355
L3      0.925 0.9921 0.9942 1.0000 0.1326 0.0377 0.934 0.8028
Height  0.194 0.0351 0.0548 0.1326 1.0000 0.4561 0.299 0.5655
Width   0.136 0.0318 0.0443 0.0377 0.4561 1.0000 0.199 0.0572
L21     0.876 0.9061 0.9188 0.9336 0.2988 0.1987 1.000 0.8456
L31     0.746 0.7218 0.7355 0.8028 0.5655 0.0572 0.846 1.0000
```

It can be seen that the correlations between the 3 length variables and the new variables vary substantially. The parallel coordinates plot for all 9 variables is given next and we can see that especially variable L32 seems promising when it comes to separating the 7 classes.

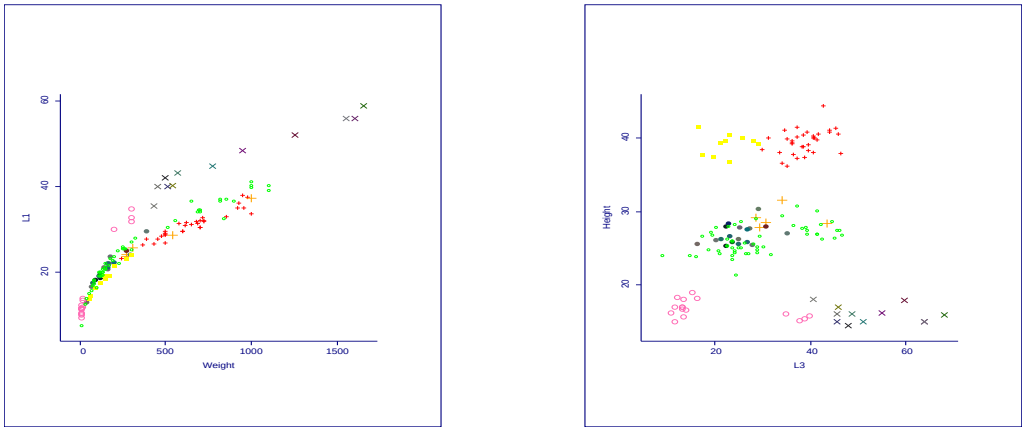


FIGURE 11. Selected 2-dimensional scatterplots of the Fish Data

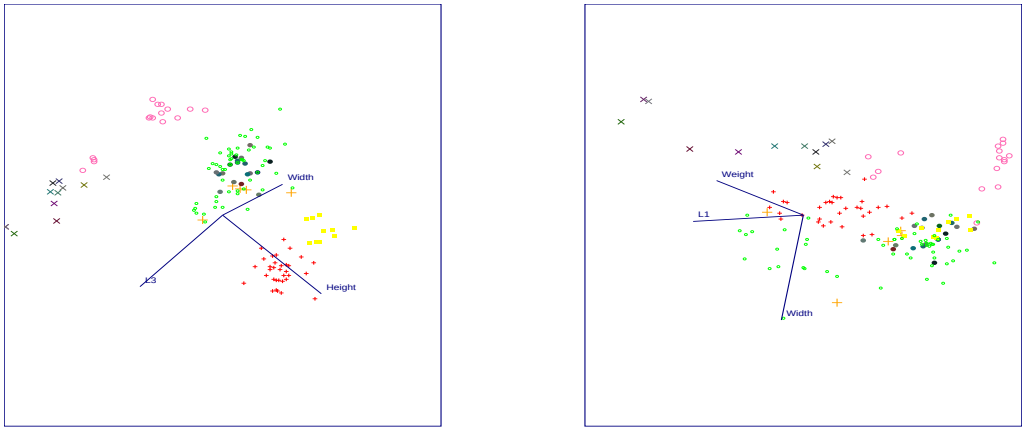


FIGURE 12. Selected 3-dimensional scatterplots of the Fish Data

4.1. **Linear and Quadratic Discriminant Analysis.** We start by employing Linear Discriminant Analysis (LDA).

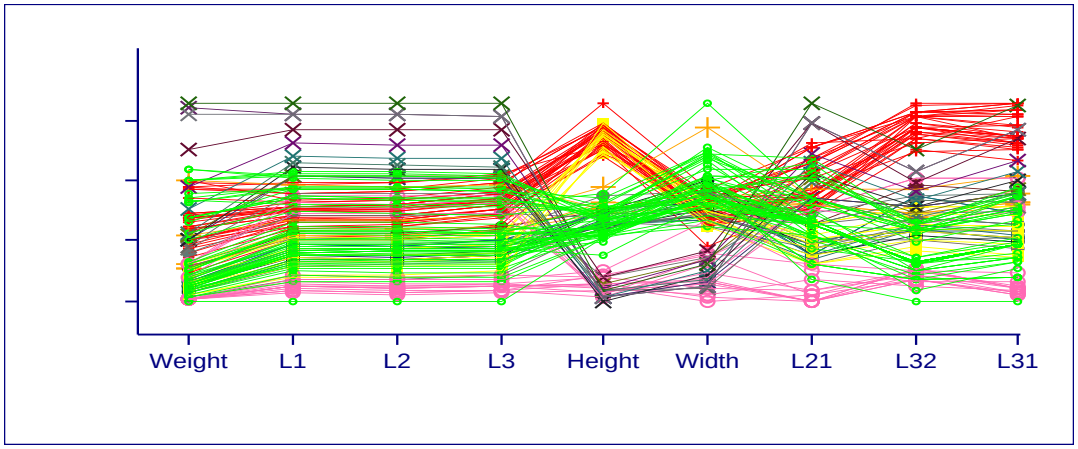


FIGURE 13. Parallel coordinates plot for the fish data (9 variables). Bream: red +, White: orange big+, Roach: blue solid circle, Parki: yellow solid square, Smelt: pink open circle, Pike: cyan x, Perch: green small open circle

```
library(MASS)
fish.lda = lda(Species ~ Weight+L1+L2+L3+Height+Width+L21+L32+L31,
     data=fish)

Warning messages:
  variables are collinear in: lda.default(x, grouping)
```

The program complains that the variables are collinear, which renders the covariance matrix singular. So, from our previous exploratory analysis we fit the following model

```
fish.lda = lda(Species ~ Weight+L1+Height+Width+L21+L32,
   data=fish)

# The output is given next

print(fish.lda,digits=2)
Call:
```

```
lda(Species ~ Weight + L1 + Height + Width + L21 + L32, data = fish)

Prior probabilities of groups:
    Bream    Parkki     Perch      Pike     Roach     Smelt Whitewish
    0.217     0.070     0.357     0.108     0.121     0.089     0.038

Group means:
           Weight L1 Height Width  L21 L32
Bream         626 30     40    14 2.81 5.2
Parkki        155 19     39    14 1.62 2.4
Perch         382 26     26    16 2.16 1.7
Pike          719 42     16    10 3.01 3.2
Roach         160 21     27    15 1.64 2.7
Smelt          11 11     17    10 0.66 1.1
Whitewish     531 29     29    16 2.52 3.0

Coefficients of linear discriminants:
             LD1    LD2     LD3     LD4     LD5     LD6
Weight  0.00095 -0.003  0.0078  0.0016  0.0062  0.0034
L1      0.13770  0.041 -0.2736 -0.2337 -0.3333  0.1303
Height -0.62108 -0.328 -0.0514 -0.3303 -0.0275  0.0266
Width   0.43249 -0.357 -0.3519  0.8531 -0.2292  0.1410
L21    -0.15448  0.730 -2.2916  0.3026  2.7712 -2.8401
L32    -2.29634  2.224  0.5801  1.8116 -0.4880  0.0905

Proportion of trace:
   LD1    LD2    LD3    LD4    LD5    LD6
0.7885 0.1349 0.0540 0.0188 0.0039 0.0000
```

It can be seen that the first two linear discriminant (LD) functions capture $\sim 93\%$ of the between groups variance (remember that the linear discriminant functions for the case of two classes correspond to the columns of $\Sigma^{-1}(\mu_1 - \mu_0)$. The coefficients of the variables on the first 2 LDs suggest that we may decide to keep only Height, Width, L21 and L32. We next calculate the apparent misclassification rate.

```
fish.ldapredict = predict(fish.lda,fish[,-1])
table(fish$Species,fish.ldapredict$class)
```

	Bream	Parkki	Perch	Pike	Roach	Smelt	Whitewish
Bream	34	0	0	0	0	0	0
Parkki	0	11	0	0	0	0	0
Perch	0	0	56	0	0	0	0
Pike	0	0	0	17	0	0	0
Roach	0	0	0	0	19	0	0
Smelt	0	0	0	0	0	14	0
Whitewish	0	0	0	0	1	0	5

The resulting misclassification rate is .6%. The above table with the true classification as the rows and the predicted classes as columns is known as the *confusion* matrix. A plot of the various classes projected onto the space spanned by the first 2 LDs can be obtained by

```
eqscplot(fish.ldapredict$x,type="n",xlab="1st LD",ylab="2nd LD")
fish.species = c(rep("B",34),rep("W",6),rep("R",19),rep("Pa",11),rep("S",14),
     rep("Pi",17),rep("Pe",56))
text(fish.ldapredict$x[,1:2],fish.species)
```

In case we wanted to fit the model on a subset of the data and then predict the class labels of the remaining points we can do it as follows

```
select = c(1:30,35:38,41:55,60:67,71:80,85:96,102:145)

# select a subset of the training data

fish.ldasel = lda(Species ~ Weight+L1+Height+Width+L21+L32,
   data=fish[select,])

# build the LDA model

fish.ldapredsel = predict(fish.ldasel,fish[-select,-1])
```

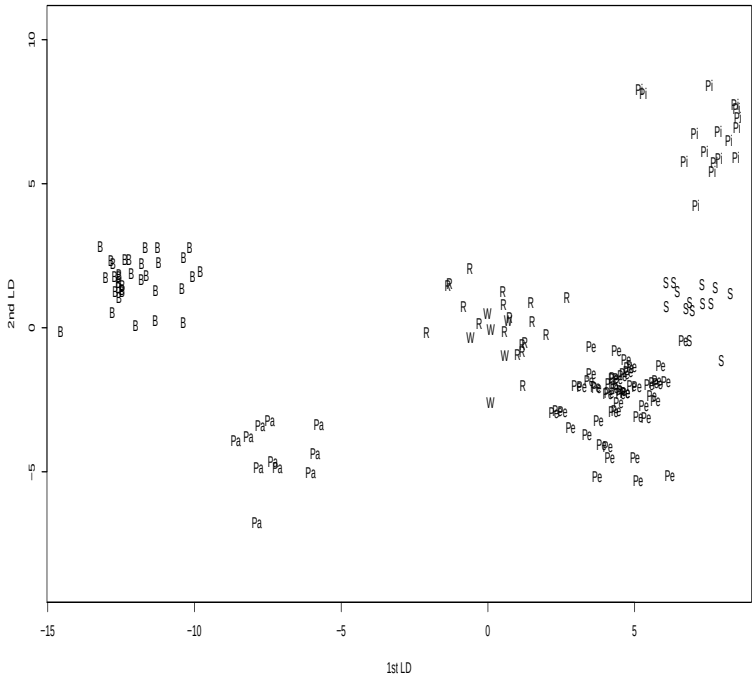


FIGURE 14. Plot of fish data projected on the first 2 LDs

```
# predict the remaining observations

table(fish$Species[-select],fish.ldapredsel$class)
```

	Bream	Parkki	Perch	Pike	Roach	Smelt	Whitewish
Bream	4	0	0	0	0	0	0
Parkki	0	3	0	0	0	0	0
Perch	0	0	12	0	0	0	0
Pike	0	0	0	5	0	0	0
Roach	0	0	0	0	4	0	0
Smelt	0	0	0	0	0	4	0
Whitewish	0	0	0	0	1	0	1

We try next quadratic discriminant analysis (QDA).

```
fish.qda = qda(Species ~  Weight+L1+Height+Width+L21+L32,
     data=fish)
```

```
Error in qda.default(x, grouping): some group is too small for qda
Dumped
```

The program complains that one of the groups is too small; remember that there are several classes with small sample sizes (e.g. Whitewash with only 6 observations) and hence we are unable to calculate the separate covariance matrices for each class, as required by the technique.

So, we try a model with only two variables, Weight and L32 trained on a subset of the observations.

```
fish.qda = qda(Species ~  Weight+L32,data=fish[select,])
```

```
fish.qdapredict = predict(fish.qda,fish[-select,-1])
table(fish$Species[-select],fish.qdapredict$class)
```

	Bream	Parkki	Perch	Pike	Roach	Smelt	Whitewish
Bream	4	0	0	0	0	0	0
Parkki	0	1	0	2	0	0	0
Perch	0	0	12	0	0	0	0
Pike	0	0	4	1	0	0	0
Roach	0	1	0	3	0	0	0
Smelt	0	0	4	0	0	0	0
Whitewish	0	0	0	2	0	0	0

It can be seen that this model has a hard time predicting objects in the following classes: Roach, Smelt and Whitewash.

Versions of LDA and QDA that employ diagonal covariance matrices have been implemented in the R package dDA.

4.2. **Classification Trees.** We illustrate next the use of classification trees in R.

```
library(rpart)
fish.control = rpart.control(minsplit=10,minbucket=3,xval=0)
```

Notice that with the parameters minsplit=10 and minbucket=3 you control the growth of your tree. In this example I require that at least 10 objects must exist in a node for a split to be attempted (minsplit=10), and that in any terminal node at least 3 objects must be present. The default values are 20 and 7, which for this data set result in a very small tree that does not contain the class White.

```
fish.tree = rpart(Species ~ .,data=fish,control=fish.control)

printcp(fish.tree)

Classification tree:
rpart(formula = Species ~ ., data = fish, control = fish.control)

Variables actually used in tree construction:
[1] Height L21    L31    L32    Weight

Root node error: 101/157 = 0.64331

n= 157

        CP nsplit rel error
1 0.336634      0  1.000000
2 0.158416      1  0.663366
3 0.138614      2  0.504950
4 0.108911      3  0.366337
5 0.099010      4  0.257426
6 0.059406      5  0.158416
7 0.044554      6  0.099010
8 0.010000      8  0.009901
```

This model used the variables Height, L21, L31, L32 and Weight. To calculate the apparent misclassification error rate we multiply the root node error rate (after the first split) by the the relative error given in the above title of a tree with k splits; for example, for a tree with 4 splits the error rate is $.64 \times .25 = .16$, while for one with 8 splits is .06%.

To visualize the tree we do

```
plot(fish.tree)
text(fish.tree)
```

to get the following plot

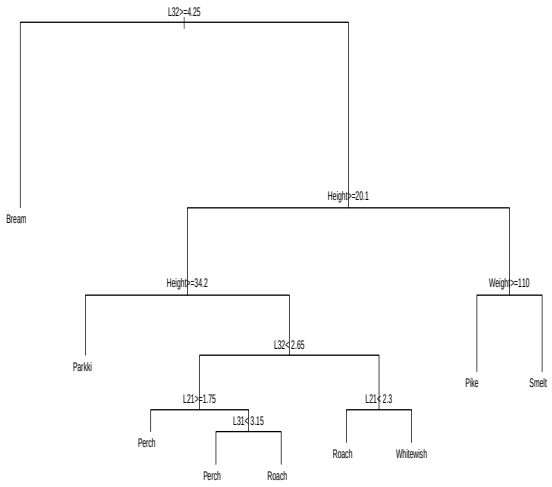


FIGURE 15. Classification tree for the fish data set using all the variables.

It can be seen that we need to construct a fairly deep tree to get class Whitewash to show up. A pruned tree would miss it as shown next (the selection of the cp parameter is such that it would lead to 5 splits as indicated by the command printcp shown above):

```
fish.treepruned=prune(fish.tree,cp=.06)
plot(fish.treepruned)
text(fish.treepruned)
```

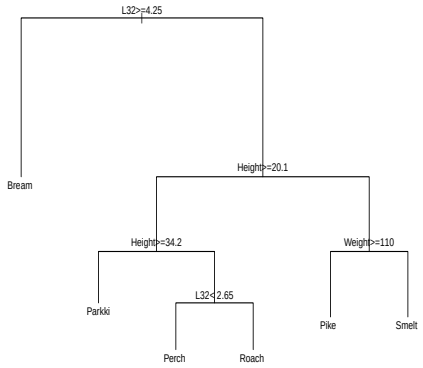


FIGURE 16. Pruned classification tree for the fish data set using all the variables; note that class Whitewash is missing

To get an estimate of the true misclassification error rate we can use cross-validation by modifying the rpart.control and in addition using the entropy as the splitting criterion:

```
fish.control = rpart.control(minbucket=3,minsplit=10,xval=100)

fish.tree=rpart(Species ~ ., data=fish,parms=list(split='information'),
       control=fish.control)

printcp(fish.tree)

Classification tree:
rpart(formula = Species ~ ., data = fish, parms = list(split = "information"),
    control = fish.control)

Variables actually used in tree construction:
[1] Height L21    L3     L31    L32    Weight

Root node error: 101/157 = 0.64331

n= 157
```

```
        CP nsplit rel error   xerror     xstd
1 0.336634      0  1.000000 1.000000 0.059427
2 0.168317      1  0.663366 0.663366 0.061360
3 0.138614      2  0.495050 0.534653 0.058931
4 0.108911      3  0.356436 0.396040 0.054057
5 0.099010      4  0.247525 0.366337 0.052653
6 0.059406      5  0.148515 0.168317 0.038549
7 0.044554      6  0.089109 0.178218 0.039525
8 0.010000      8  0.000000 0.029703 0.016984
```

It can be seen that the cross-validated error rate is $.643 * .017 = .01$. Finally, we show how to predict the labels of new observations.

```
fish.tree=rpart(Species ~ ., data=fish[select,],control=fish.control)

fish.predict=predict(fish.tree,fish[-select,],type="class")

table(fish$Species[-select],fish.predict)
           fish.predict
            Bream Parkki Perch Pike Roach Smelt Whitewish
  Bream         4      0     0    0     0     0          0
  Parkki        0      3     0    0     0     0          0
  Perch         0      0    12    0     0     0          0
  Pike          5      0     0    0     0     0          0
  Roach         0      0     0    0     1     0          3
  Smelt         0      0     0    0     0     4          0
  Whitewish     0      0     2    0     0     0          0
```

It can be seen that the classification tree systematically misclassifies objects from class Pike, Whitewash and has a hard time with those from class Roach.

4.3. **Random Forests.** As discussed above, random forests is an ensemble of classification trees. The R library used to create random forests is called randomForest.

```
library(randomForest)

set.seed(80)
# This command fixes the random number generator used to
# bootstrap the training data

fish.rf=randomForest(Species ~ .,data=fish,ntree=1000,mtry=3,importance=T)

# The ntree option requires 1000 bootstrap samples and hence corresponding
# number of trees in the ensemble. The mtry option determines the number
# of variables to be considered for splitting purposes at every node

# The importance option is going to calculate a measure of importance
# for the variables using the out-of-bag (oob) samples

print(fish.rf)

Call:
 randomForest(formula = Species ~ ., data = fish, ntree = 1000,
       mtry = 3, importance = T)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 3

        OOB estimate of  error rate: 1.91%
Confusion matrix:
          Bream Parkki Perch Pike Roach Smelt Whitewish class.error
Bream        34      0     0    0     0     0         0   0.0000000
Parkki        0     11     0    0     0     0         0   0.0000000
Perch         0      0    56    0     0     0         0   0.0000000
Pike          0      0     0   17     0     0         0   0.0000000
Roach         0      0     2    0    16     0         1   0.1578947
```

```
Smelt          0       0      0     0      0     14           0    0.0000000
Whitewish      0       0      0     0      0      0           6    0.0000000
```

The print command summarizes the results and gives an estimate of the true misclassification error rate using the out-of-bag samples. For our example, it can be seen that all the misclassifications come from class Roach.

The following set of commands provides useful information about the structure of the random forest. The varUsed command gives how many times each variable was used in the ensemble. So, variable Weight was used 1060 times (on average once in every tree), variable L1 923 times, variable L32 2323 times, etc. The treesize command provides information about the size of the trees constructed for the ensemble.

```
varUsed(fish.rf)
# Weight   L1   L2   L3 Height Width  L21  L31   L32
[1] 1060  923 1050 1027  2459  1490  1628  1664 2323

hist(treesize(fish.rf))
```

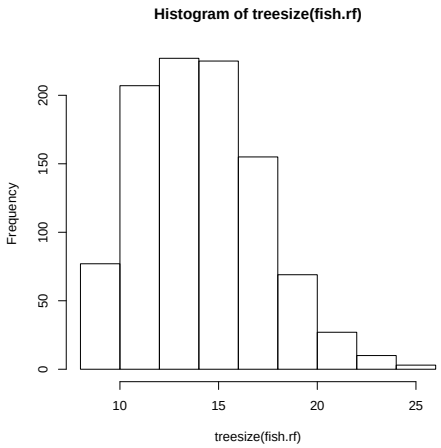


FIGURE 17. Histogram of the tree sizes in the random forest.

It can be seen that the average tree size in the random forest is about 15 and there are no trees of size larger than 25.

The margin command calculates the margin of each observation, which is defined as the proportion of votes for the correct class minus the maximum proportion of votes for the other classes. Therefore, under a simple majority rule, a positive margin implies correct classification and a negative one a false classification. Obviously, the larger the value of the margin the less disagreement among the votes.

```
margin(fish.rf,fish$Species)

# we print the margins for the first few observation in the data set
    Bream        Bream        Bream        Bream        Bream        Bream
 0.334246575  0.938829787  0.872928177  0.912928760  0.953168044  1.000000000
       Bream        Bream        Bream        Bream        Bream        Bream
 0.984886650  0.965517241  1.000000000  0.964769648  1.000000000  0.993993994
...........
```

It can be seen that the first observation is more challenging to classify, since there is quite some disagreement between the various trees. On the other hand, *all* trees classify the sixth observation to class Bream. To better understand this point, we show next the percentage of votes each class got by the forest for the first six observations.

```
print(fish.rf$votes[1:6,],digits=2)

  Bream Parkki  Perch   Pike  Roach Smelt Whitewish
1  0.60 0.2685 0.0000 0.0055 0.0767     0    0.0466
2  0.96 0.0239 0.0027 0.0053 0.0027     0    0.0027
3  0.91 0.0414 0.0028 0.0055 0.0055     0    0.0304
4  0.94 0.0132 0.0000 0.0106 0.0053     0    0.0290
5  0.97 0.0055 0.0000 0.0000 0.0028     0    0.0193
6  1.00 0.0000 0.0000 0.0000 0.0000     0    0.0000
```

It can be seen that the margin .33 of the first observation is the difference in the percentage of votes between the majority class Bream and the second most popular class Parkki, while for the sixth observation it is necessarily 1, since no other class received any votes.

The next command provides a measure of importance of the variables for the various classes (larger values, more important). It produces a matrix with rows corresponding to the variables and columns corresponding to the number classes plus two. The first K columns correspond to the class-specific measures that indicate the mean decrease in accuracy for that specific class, while the last two columns are the mean decrease in accuracy over all classes and of the Gini index, respectively.

```
print(round(fish.rf$importance, 2))
```

```
       Bream Parkki Perch Pike Roach Smelt Whitewish MeanDecreaseAccuracy
Weight  0.01   0.01  0.03 0.03  0.05  0.20      0.01                 0.04
L1      0.01   0.03  0.03 0.06  0.03  0.06     -0.01                 0.03
L2      0.02   0.05  0.04 0.08  0.05  0.06     -0.01                 0.04
L3      0.03   0.06  0.03 0.07  0.04  0.09      0.00                 0.04
Height  0.24   0.52  0.21 0.36  0.22  0.34      0.13                 0.26
Width   0.03   0.06  0.11 0.36  0.05  0.20      0.04                 0.11
L21     0.05   0.13  0.07 0.08  0.22  0.15      0.19                 0.10
L31     0.24   0.08  0.16 0.07  0.10  0.15      0.10                 0.15
L32     0.38   0.14  0.32 0.12  0.20  0.18      0.27                 0.27
       MeanDecreaseGini
Weight             6.59
L1                 4.81
L2                 5.97
L3                 6.08
Height            26.81
Width             13.71
L21               11.85
L31               17.14
L32               29.63
```

It can be seen that both measures show agreement regarding the importance of the L32. On the other hand, the second most important variable in terms of accuracy is L21, while for the Gini index Height. Nevertheless, the top four variables are the same according to both

measures. To visualize these two measures, which comes in handy in the presence of many variables, we can use the following command

```
varImpPlot(fish.rf)
```

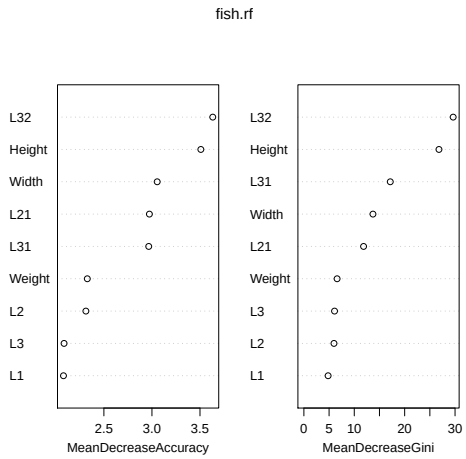


FIGURE 18. Variable importance plot obtained from a random forest for the fish data set.

Finally, in the presence of a large number of variables, it is not clear how one should set the value for the mtry parameter; namely, how many variables the algorithm should consider for each split. The command tuneRF resolves this issue

```
tuneRF(fish[,-1],fish[,1],StepFactor=1)
mtry = 3  OOB error = 2.55%
Searching left ...
mtry = 2        OOB error = 5.1%
-1 0.05
Searching right ...
mtry = 6        OOB error = 2.55%
0 0.05
      mtry   OOBError
2.OOB    2 0.05095541
3.OOB    3 0.02547771
6.OOB    6 0.02547771
```

For this example, our initial choice of 3 was the optimal one; allowing a larger number of variables to be considered for splitting would not have improved the performance of the random forest.

Classifying the Glass Data Set: We have argued that random forests represents a flexible and powerful classification technique. We examine next the glass data discussed in the introduction.

Using the tuneRF command we get that the optimal number of variables to be used at each split is three, to get

```
glass.rf=randomForest(Class ~.,data=glass,mtry=3,ntree=10000,importance=T)

glass.rf

Call:
 randomForest(formula = Class ~ ., data = glass, mtry = 3, ntree = 10000,
    importance = T)
               Type of random forest: classification
                     Number of trees: 10000
No. of variables tried at each split: 3

        OOB estimate of  error rate: 19.63%
Confusion matrix:
      Con Head Tabl Veh WinF WinNF class.error
Con     9    1    0   0    0     3   0.3076923
Head    0   25    0   0    1     3   0.1379310
Tabl    0    0    7   0    0     2   0.2222222
Veh     0    0    0   7    6     4   0.5882353
WinF    0    0    0   1   63     6   0.1000000
WinNF   2    1    2   1    9    61   0.1973684
```

The results confirm that this is a challenging classification problem; especially, the class Veh(icle) windows performs worse than random guessing. The variable importance plot shows that the following variables are important for both accuracy and the Gini index: Al,

Mg, Ca and RI. One may be tempted to exclude non-important variables, but the following

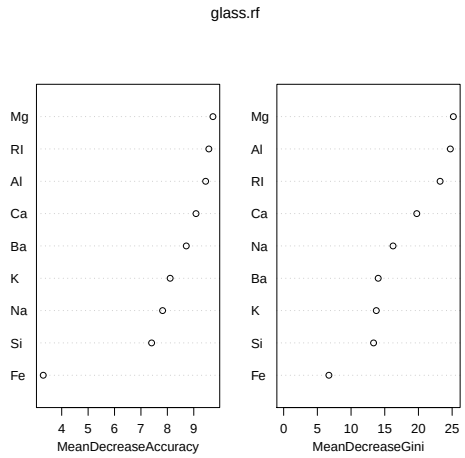


FIGURE 19. Variable importance plot obtained from a random forest for the glass data set

results suggest that this is usually not a good idea.

```
glass.rfres=randomForest(Class ~ Al+Mg+RI+Ca,data=glass,mtry=3,
   ntree=10000,importance=T)

glass.rfres

Call:
 randomForest(formula = Class ~ Al + Mg + RI + Ca, data = glass,
   mtry = 3, ntree = 10000, importance = T)
               Type of random forest: classification
                     Number of trees: 10000
No. of variables tried at each split: 3

        OOB estimate of  error rate: 22.9%
Confusion matrix:
      Con Head Tabl Veh WinF WinNF class.error
Con     9    3    0   0    0     1   0.3076923
Head    1   23    1   0    1     3   0.2068966
```

```
Tabl    2    1    4    0    0     2   0.5555556
Veh     0    0    0    8    7     2   0.5294118
WinF    0    0    0    2   62     6   0.1142857
WinNF   2    1    2    4    8    59   0.2236842
```

It can be seen that the objects in class Tabl(eware) are consistently misclassified. Plotting the margin from the original random forests model shows that about 45 observations have a negative margin.

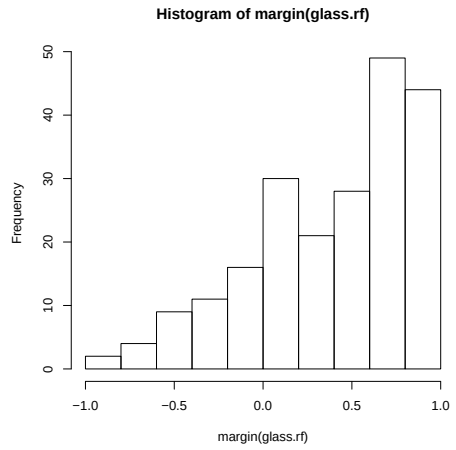


FIGURE 20. Histogram of margin for the glass data set

5. BOOSTING CLASSIFICATION TREES

The main learner in boosting is classification trees. There are a number of packages in R that can boost classification trees, although the majority of them focuses on the two class problem (e.g. adaboost, ada, gbm). Hence, for illustration purposes we will restrict attention to the largest classes in the fish data set, Bream and Perch.

```
library(ada)

fish.select=read.table('fish-data=2class.txt',h=T)

# We select a subset for training purposes
```

```
select.train=c(1:25,35:75)

# together with a test set

select.test=setdiff(1:90,select.train)

# We train the model and predict the remaining observations

fish.boost=ada(Species ~ .,data=fish.select[select.train,],type="discrete")

fish.predict=predict(fish.boost,fish.select[select.test,-1])
table(fish.select$Species[select.test],fish.predict)
       fish.predict
        Bream Perch
  Bream     9     0
  Perch     0    15
```

To assess variable importance we use the command

```
varplot(fish.boost)
```

In this 2-class problem, Height remains an important variable, complemented with L3.

6. NEAREST NEIGHBORS

We start by classifying a subset of the observation using all the variables and setting the nearest neighbor parameter $r = 3$.

```
library(class)
fish.knn=knn(fish[select,-1],fish[-select,-1],fish$Species[select],
  k=3,prob=T)

fish.knn
```

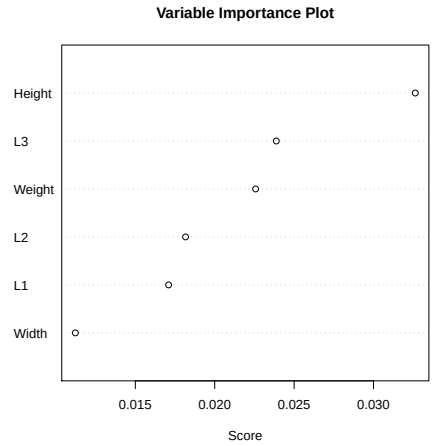


FIGURE 21. Variable importance plot for the fish data set obtained from boosting

```
[1] Bream     Bream     Bream     Bream     Pike      ....
attr(,"prob")
 [1] 1.0000000 1.0000000 1.0000000 1.0000000 0.3333333 ....

# It can be seen that for the first observation all its neighbors belong to
# the class Bream, while for the fifth observation each neighbor belong to a
# different class and the corresponding tie is broken at random

table(fish$Species[-select],fish.knn)
           fish.knn
            Bream Parkki Perch Pike Roach Smelt Whitewish
  Bream         4      0     0    0     0     0         0
  Parkki        1      0     1    0     0     0         1
  Perch        10      0     1    1     0     0         0
  Pike          5      0     0    0     0     0         0
  Roach         1      0     2    0     0     0         1
  Smelt         0      0     0    0     0     4         0
  Whitewish     1      0     0    1     0     0         0
```

It can be seen that using all the variables results in consistently misclassifying the observations from the class Perch. We try next using the important variables identified from the random forest (namely Height and L32), although there are no guarantees that such a strategy will work.

```
fish.knn=knn(fish[select,c(6,10)],fish[-select,c(6,10)],
     fish$Species[select],k=3,prob=T)
table(fish$Species[-select],fish.knn)
           fish.knn
            Bream Parkki Perch Pike Roach Smelt Whitewish
  Bream         4      0     0    0     0     0          0
  Parkki        0      3     0    0     0     0          0
  Perch         0      0     5    0     5     0          2
  Pike          0      0     0    5     0     0          0
  Roach         0      0     0    0     3     0          1
  Smelt         0      0     0    0     0     4          0
  Whitewish     0      0     0    0     1     0          1
```

7. SUPPORT VECTOR MACHINES

There are many implementations of support vector machines (SVM) in R. We illustrate their use on the fish data set with the library e1071. It should be noted that for a multiclass problem, a "one-against-one"'-approach is used,in which $K(K-1)/2$ binary classifiers are trained and the appropriate class is found by a voting scheme.

```
library(e1071)
fish.svm = svm(Species ~ .,data=fish[select,],kernel="linear")

# The kernel used was a linear one, other choices include polynomial,
# radial and sigmoid.
# The default option is the radial kernel which is more flexible
# than the linear one used here.

summary(fish.svm)
```

```
Call:
svm(formula = Species ~ ., data = fish[select, ], kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.1111111

Number of Support Vectors:  41

 ( 6 4 9 4 2 4 12 )


Number of Classes:  7

Levels:
 Bream Parkki Perch Pike Roach Smelt Whitewish


# Next, we predict the remaining observations and tally the result


fish.pred=predict(fish.svm,fish[-select,-1],decision.values=T)
table(fish$Species[-select],fish.pred)
           fish.pred
            Bream Parkki Perch Pike Roach Smelt Whitewish
  Bream         4      0     0    0     0     0         0
  Parkki        0      2     0    0     0     0         1
  Perch         0      0    12    0     0     0         0
  Pike          0      0     0    5     0     0         0
  Roach         0      0     0    0     1     0         3
  Smelt         0      0     2    0     0     2         0
```

```
  Whitewish      0       0      0     0      0      0          2
```

It can be seen that the model performs very well; note that other choices of kernels do not affect the results.

REFERENCES

[1] Allwein, E.L., Schapire, R.E. and Singer, Y. (2000), Reducing Multi-class to Binary: a Unifying Approach for Margin Classifiers, Journal of Machine Learning Research, 113-141

[2] Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK

[3] Bishop, C.M. (editor) (1998), *Neural Networks and Machine Learning*, Springer, NY

[4] Boser, B.E., Guyon, I.M. and Vapnik, V. (1992), A Training Algorithm for Optimal Margin Classifiers, in Proceedings of ACM Workshop on Computational Learning Theory, 144-152, Pittsburgh, PA

[5] Bousquet, O., Boucheron, S. and Lugosi, G. (2004), Introduction to Statistical Learning Theory, Advanced Lectures on Machine Learning Lecture Notes in Artificial Intelligence 3176, 169-207. (Eds.) Bousquet, O., von Luxburg, U. and Ratsch, R. Springer, Heidelberg, Germany

[6] Breiman, L., Friedman, J.H., Olshen, R. and Stone, C. (1984), *Classification and Regression Trees*, CRC Press, Boca Raton, FL

[7] Breiman, L. (1996), Bagging Predictors, Machine Learning, 24, 123-140

[8] Breiman, L. (1998), Arcing Classifiers, Annals of Statistics, 26, 801-849

[9] Breiman, L. (2001), Random Forests, Machine Learning, 45, 5-32

[10] Cover, T. and Hart, P. (1967), Nearest neighbor pattern classification, IEEE Transactions on Information Theory, 13, 21-27

[11] Crammer, K. and Singer, Y. (2001), On the Algorithmic Implementation of Multi-class Kernel-based Vector Machines, Journal of Machine Learning Research, 2, 265-292

[12] Dudoit, S., Fridlyand, J. and Speed, T.P. (2002), Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data, Journal of the American Statistical Association, 97, 77-87

[13] Dietterich, T.G. and Bakiri, G. (1995), Solving Multi-class Learning Problems via Error Correcting Output Codes, Journal of Artificial Intelligence Research, 2, 263-286

[14] Fisher, R.A. (1936), The use of multiple measurements in taxonomic problems, Annals of Eugenics, 7, 179-188

[15] Fix, E. and Hodges, J.L. (1951), Discriminatory Analysis. Nonparametric Discrimination; Consistency Properties, Report Number 4, USAF School of Aviation Medicine, Randolph Field, TX

[16] Friedman, J.H., Hastie, T. and Tibshirani, R. (1998), Additive Logistic Regression: a Statistical view of Boosting, Annals of Statistics, 28, 337-407 (with discussion)

[17] Friedman, J.H. (2002), Stochastic Gradient Boosting, Computational Statistics and Data Analysis, 38, 367-378

[18] Freund, Y. and Schapire, R.E. (1997), Experiments with a New Boosting Algorithm, In Proceedings of the International Conference on Machine Learning, 148156.

[19] Freund, Y. and Schapire, R.E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, Journal of Computer and System Sciences, 55, 119139

[20] Hastie, T., Tibshirani, R. and Friedman, J. (2001), *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer, NY

[21] Lee, Y., Lin, Y. and Wahba, G. (2004). Multi-category Support Vector Machines, Theory and Application to the Classification of Microarray Data and Satellite Radiance Data , Journal of American Statistical Association, 99, 67-81.

[22] Mease, D. (2006), Evidence Contrary to the Statistical View of Boosting, Technical Report

[23] Mosteller, F. and Wallace, D.L. (1963), Inference in an authorship Problem, Journal of the American Statistical Association, 58, 275-309

[24] Ripley, B.D. (1996), *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge, UK

[25] Scholkopf, B., Smola, A.J. (2002), *Learning with Kernels*, MIT Press, Cambridge, MA

[26] Tibshirani, R., Hastie, T., Narasimhan, B. and Chu, G. (2002), Diagnosis of multiple cancer types by shrunken centroids of gene expression, Proceedings of the National Academies of Science USA, 99, 6567-6572

[27] Valiant, L.G. (1984), A Theory of the Learnable, Communications of the ACM, 1134-1142

[28] Vapnik, V. (1998), *Statistical Learning Theory*, Wiley, NY

Chapter 5: Cluster Analysis Techniques

1. Introduction

The idea is that given amultivariate data set comprising of N objects and p variables, we would like to group the objects (and in some cases the variables) into *homogeneous* groups. The motivation for this set of technique stems from the fact that large multivariate data sets can prove difficult to comprehend. It is usually particularly useful to be able to summarize them and extract the most important patterns from them.

1.1. Problem Formulation and Algorithms.

The main idea behind cluster analysis is to investigate relationships between the objects in order to establish whether or not the data can validly be summarized by a small number of groups (clusters) of similar objects. A good outcome manages to assign the N objects in the dataset into a small number of groups in such a way that members in the same group are as 'similar' as possible, while members of different groups are as 'dissimilar' as possible.

Formally we have: let $\mathcal{O} = \{o_1, o_2, \ldots, o_N\}$ be a set of N objects and let $\mathcal{C} = \{C_1, ..., C_K\}$ be a *partition* of $\mathcal{O}$ into subsets; i.e. $C_i \cap C_j = \emptyset,\ i \neq j$ and $\cup_k C_k = \mathcal{O}$. Each subset is called a *cluster*, and $\mathcal{C}$ is a clustering solution.

The input data for a clustering problem is typically given in one of two forms:

- Profile data matrix $X_{N\times p}$, where each object o_i is associated with a real-valued vector, called its profile, which contains measurements on p variables, e.g. gene expression levels at different experimental conditions.
- Dissimilarity matrix $\Delta_{N\times N} = \{\delta_{ij}\}_{i,j=1}^{N}$, that contains the pairwise dissimilarities that are usually computed from the profile data.

A dissimilarity d_{ij} between observations i and j is a distance-like measure that satisfies the following properties:

(1) Non-negativity property: $d_{ij} \geq 0$ for all i and j, with $d_{ii} = 0$.
(2) Symmetry property: $d_{ij} = d_{ji}$ for all i, j.

If in addition, d_{ij} satisfies the triangle inequality ($d_{ij} \leq d_{ik} + d_{kj}$ for any k) then it is a proper *distance*. Euclidean, Mahalanobis and Manhattan distances are commonly used in cluster analysis techniques for defining dissimilarities.

When dealing with a cluster analysis problem the following issues need to be studied (Hansen and Jaumard, 1997):

- what is the criterion used (e.g. homogeneity vs separation).
- what type of clustering should be considered.
- how difficult it is to perform the clustering (issue of complexity).
- how should the clustering be done (issue of algorithmic design).
- is the resulting clustering meaningful (i.e. how should the results be interpreted).

Remark: Many clustering techniques make few assumptions, usually posed in set-theoretic terms. Some other techniques consider each cluster to correspond to a component from a multivariate mixture distrbution, whose number of components and parameters need to be estimated from the available data (Mirkin, 1996).

We examine next some of the most popular algorithmic approaches for cluster analysis. Primarily they can be categorized into two groups: (i) *partition methods* and (ii) *tree-type methods*.

Partition methods create a family of clusters where each object belongs to just a single member of the partition. To generate such partitions the requirement is that distances between pairs of objects belonging to the same cluster are smaller than distances between pairs of objects in different clusters. Formally, suppose that objects i and j belongs to cluster A, while object k belongs to cluster B. We then require that $d_{ij} < d_{ik}$ and $d_{ij} < d_{jk}$.

Tree methods build a tree of clusters that includes all the objects and for which any two clusters are either disjoint or one cluster is a superset of the other. In tree methods it is usually required that the distance be an *ultrametric*; i.e. $D_{ij} \leq \max\{d_{ik}, d_{jk}\}$. Equivalently we have that for every triple of objects (i, j, k) we have that the 2 largest values in the set $\{d_{ij}, d_{jk}, d_{ik}\}$ are equal.

2. AGGLOMERATIVE HIERARCHICAL ALGORITHMS

In general, hierarchical methods methods create an iterated partition. The clustering solution is represented by a *dendrogram*, which is a rooted weighted tree $\mathcal{T}$, with leaves corresponding to the objects. A tree satisfies the following conditions:

(1) (i) $\mathcal{O} \in \mathcal{T},$ (ii) $\emptyset \notin \mathcal{T}$. (iii) $o_i \in \mathcal{T}$all $i \in \mathcal{O}$

(2) (iv) if $A,\ B \in \mathcal{T},$ then $A \cap B \in \{\emptyset, A, B\}$.

Each edge defines the cluster of objects contained in the subtree below that particular edge. The edge's length (or weight) reflects the dissimilarity between that cluster and the remaining clusters.

For each pair of objects $(i, j),\ \ell_{ij} = \ell_{ji}$ is defined to be the length of the smallest subset containing both the ith and the jth objects. The value of ℓ_{ij} measures the difference between the two objects, with small values indicating a high degree of similarity. The set of lengths $\{\ell_{ij},\ i, j \in \mathcal{O}\}$ satisfies the *ultrametric* inequality

(3) $$\ell_{ij} \leq \max\{\ell_{ik}, \ell_{jk}\},\ \text{for all triples } i, j, k \in \mathcal{O}.$$

An equivalent condition to (3) is that every triple of objects (i, j, k) possesses the property that the two largest values in the set $\{\ell_{ij}, \ell_{ik}, \ell_{jk}\}$ are equal.

Remark: Although hierarchical clustering of objects determines a set of ultrametric distances, practitioners rarely make use of the values themselves. Attention is most often paid only in the ordering of the lengths.

More specifically, such methods start with one object per cluster and merges them to form progressively larger groups. Since groups are merged to form larger groups we need a definition of distance between groups. Some popular choices possibilities are:

Single linkage: The dissimilarity between two clusters is defined by

(4) $$d(C_i, C_j) = \min_{s,t}\{d(s, t) | o_s \in C_i,\ o_t \in C_j\}.$$

Complete linkage: In this case, the dissimilarity is defined by

(5) $$d(C_i, C_j) = \max_{s,t}\{d(s, t) | o_s \in C_i,\ o_t \in C_j\}.$$

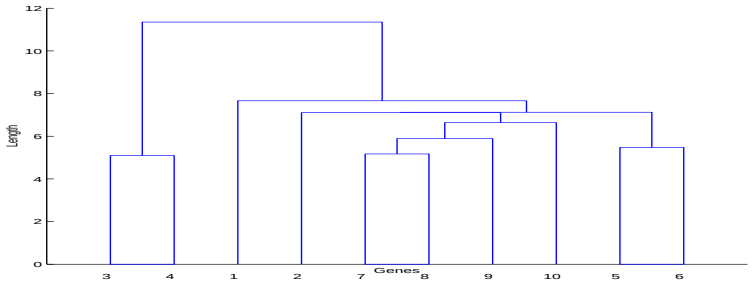


FIGURE 1. The dendrogram of a small cluster analysis example

Average linkage: In this case, the dissimilarity is defined by

$$d(C_i, C_j) = \frac{\sum_{s \in C_i} \sum_{t \in C_j} d(i,j)}{n_i n_j}. \tag{6}$$

Remark: A nice feature about these three methods is that they are invariant to monotone transformations.

Other possibilities to perform the join operation include:

(1) *Ward's method* that merges clusters that minimize $\sum_{\text{all clusters}}$(SS about the mean in cluster).
(2) *The weighted average linkage.*

An example using simple linkage. Suppose that the dissimilarity matrix is given by

$$\Delta = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & & & & \\ 9 & 0 & & & \\ 3 & 7 & 0 & & \\ 6 & 5 & 9 & 0 & \\ 11 & 10 & 2 & 8 & 0 \end{bmatrix} \end{array}$$

Treating each object as a cluster, the first join corresponds to merging clusters 3 & 5. To implement the next level of clustering we need to calculate the distances between the cluster (35) and the remaining objects 1, 2 and 4. We have

$$d((35),1) = \min\{d(3,1), d(5,1)\} = \min\{3, 11\} = 3$$
$$d((35),2) = \min\{d(3,2), d(5,2)\} = \min\{7, 10\} = 7$$
$$d((35),4) = \min\{d(3,4), d(5,4)\} = \min\{9, 8\} = 8$$

The new dissimilarity matrix is then given by

$$D = \begin{array}{c} \\ (35) \\ 1 \\ 2 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} (35) & 1 & 2 & 4 \end{array} \\ \begin{bmatrix} 0 & & & \\ 3 & 0 & & \\ 7 & 9 & 0 & \\ 8 & 6 & 5 & 0 \end{bmatrix} \end{array}$$

The smallest distance between pairs of clusters is now $d((35),1) = 3$; hence, the next cluster obtained is (135). Calculating $d((135),2) = 7$ and $d((135),4) = 6$ we get the new dissimilarity matrix

$$D = \begin{array}{c} \\ (135) \\ 2 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} (135) & 2 & 4 \end{array} \\ \begin{bmatrix} 0 & & \\ 7 & 0 & \\ 6 & 5 & 0 \end{bmatrix} \end{array}$$

The smallest distance between pairs of clusters is now $d(2,4)$ and therefore the resulting dendrogram is given in Figure 2.

Remark: Since single linkage joins clusters by the shortest link between them, the technique can not discern poorly separated clusters.

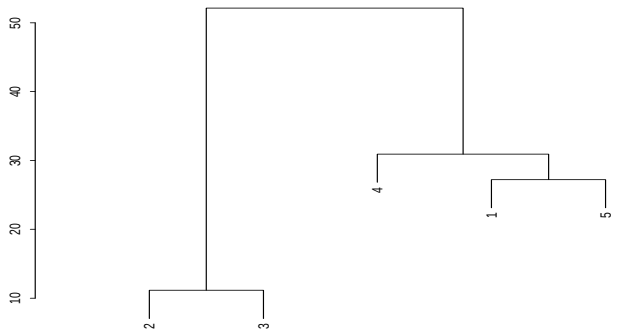


FIGURE 2. Single linkage algorithm applied to a toy data set.

An example using complete linkage: Using the same dissimilarity matrix as above, we again get that at the first stage clusters 1 & 3 are merged as the most similar. At the second stage, we compute

$$d((35),1) = \max\{d(3,1), d(5,1)\} = \max\{3, 11\} = 11$$
$$d((35),2) = \max\{d(3,2), d(5,2)\} = \max\{7, 10\} = 10$$
$$d((35),4) = \max\{d(3,4), d(5,4)\} = \max\{9, 8\} = 9$$

and the modified dissimilarity matrix becomes

$$D = \begin{array}{c} (35) \\ 1 \\ 2 \\ 4 \end{array} \begin{bmatrix} 0 & & & \\ 11 & 0 & & \\ 10 & 9 & 0 & \\ 9 & 6 & 5 & 0 \end{bmatrix}$$

The next join occurs between objects 2 and 4, to give cluster (24). At the next stage, we compute

$$d((24),(35)) = \max\{d(2,(35)), d(4,(35))\} = \max\{10, 9\} = 10$$
$$d((24),1) = \max\{d(2,1), d(4,1)\} = \max\{9, 6\} = 9$$

(7)

to get

$$D = \begin{matrix} (35) \\ (24) \\ 1 \end{matrix} \begin{bmatrix} 0 & & \\ 10 & 0 & \\ 11 & 9 & 0 \end{bmatrix}$$

The next merger produces the cluster (124) and so on. The resulting dendrogram is shown in Figure 3.

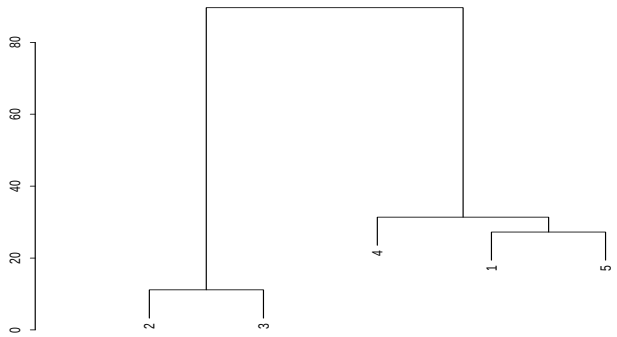


FIGURE 3. Complete linkage algorithm applied to a toy data set.

An example using average linkage: Average linkage treats the dissimilarity between two clusters as the average dissimilarity between all pairs of items where one member of the pair belongs to each cluster. The resulting dendrogram for our toy example is shown in Figure 4.

Remarks:

(1) Sources of error and variation are not formally considered in hierarchical procedures. This implies that such methods are sensitive to outliers.
(2) The *stability* of a hierarchical solution is sometimes checked by applying the clustering algorithm before and after *small* perturbations have been added to the data. If the groups are fairly well distinguished, the clusterings before and after perturbation should agree.

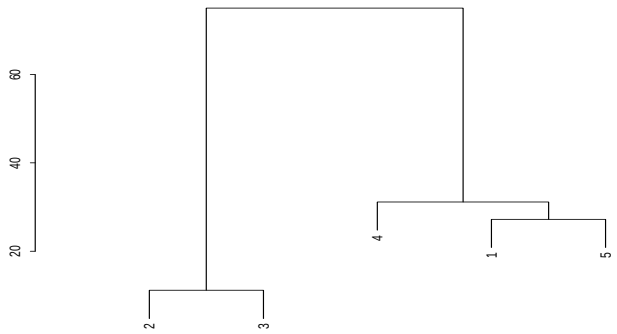


FIGURE 4. Average linkage algorithm applied to a toy data set.

(3) Ties in the dissimilarity matrix can produce multiple solutions to a hierarchical clustering problem. Thus, the dendrograms corresponding to different treatments of the tied dissimilarities can be different, particularly at the lower levels.

Some practical considerations

Agglomerative algorithms rely on the distance between points, so variables need to be scaled appropriately before calculating dissimilarities. For example, if we decide to standardize all variables, it implies that we deem all variables to be approximately equally important. On the other hand if we scale only some of the variables, it implies that we consider some of them more important than others. A data analyst should be aware that scaling will affect the results of many clustering algorithms, since they are not robust to scale changes.

3. PARTITION METHODS

Points are assigned to clusters with the objective of optimizing some criterion. We can decompose the variation as

$$T = W + B, \tag{8}$$

where

$$T = \sum_{\text{all objects}i} (x_i - \bar{x})(x_i - \bar{x})' \tag{9}$$

and

$$W = \sum_{\text{all clusters}k} n_k (x_i - \bar{x}_k)(x_i - \bar{x}_k)' \tag{10}$$

where n_k is the number of objects in cluster k, $\bar{x}$ is the overall mean of the data and $\bar{x}_k$ is the mean of cluster k. It is worth noting that the above definitions are identical to the ones in MANOVA. The only difference is that we *don't know a priori* which group an object belongs to.

Given that T is fixed a good clustering algorithm seeks to minimize some measure of W (the within groups variation) and hence maximize some measure of B (the between groups variation). Criteria that have been suggested in the literature include:

(1) Minimize trace(W) (not invariant to changes in scale).
(2) Minimize det(W).
(3) Maximize the sum of the eigenvalues of $W^{-1}B$.

Remark: This optimization problem is very challenging from a computational point of view, since there are exponentially many ways of allocating N points even in two groups. Hence, an exhaustive search does not represent a viable option beyond toy problems.

The way to proceed in practice is through greedy heuristic algorithms. One of the most popular ones is the K-means algorithm.

The K-means formulation assumes that the clusters are defined by the distance of the points to their class centers only. In other words, the goal of clustering is to find those K multivariate mean vectors $z_1, \cdots, z_K$ and provide the cluster assignment $y_i \in \{1, \cdots, K\}$ of each object with feature vector $x_i, i = 1, \cdots, N$ in the data set. The K-means algorithm is based on an interleaving approach where the cluster assignments y_i are established given the centers and the centers are computed given the assignments. The optimization criterion

is as follows:

$$\min_{y_1,\cdots,y_N,c_1,\cdots,c_K} \sum_{j=1}^{K} \sum_{y_i=j} ||x_i - z_j||^2, \tag{11}$$

where $||\cdot||$ denotes the Euclidean norm.

Notice that assuming that the K cluster centers $z_1, \cdots, z_K$ are given from the previous iteration, then

$$y_i =_j ||x_i - z_j||^2,$$

while for given assignments of the observations to cluster y_i, then for any set $S \subseteq \{1, \cdots, N\}$ we have that

$$\frac{1}{|S|} \sum_{i \in S} x_i =_z \sum_{i \in S} ||x_i - z||^2. \tag{13}$$

The implementation of K-means is as follows: *The K-means algorithm*. We start by fixing the number of clusters (say 2, 3, 5, 15, etc) and making an initial partition of the data. We then consider moving a point to another cluster, and if it reduces the criterion we move it. Otherwise we look for another point. We repeat this procedure until no further improvements are possible. This algorithm converges in a finite number of steps, since each step is guaranteed to reduce the optimization criterion, but usually converges to a local minimum.

The main drawback of the K-means algorithm is that the quality of the local optimum strongly depends on the initial guess (either the centers or the assignments). If we start with a wild guess for the centers it would be fairly unlikely that the process would converge to a good local minimum (i.e. one that is close to the global optimum). In many cases, the solution of a hieararchical clustering algorithm can be used as a starting points for the assignments.

An illustration of the convergence process of K-means is shown in Figure 5 and its failure to converge to a good solution due to poor initial assignements in Figure 6.

Remark: Variable selection. Both agglomerative and partition algorithms offer no guidance on which variables may be important for obtaining high quality clustering solutions.

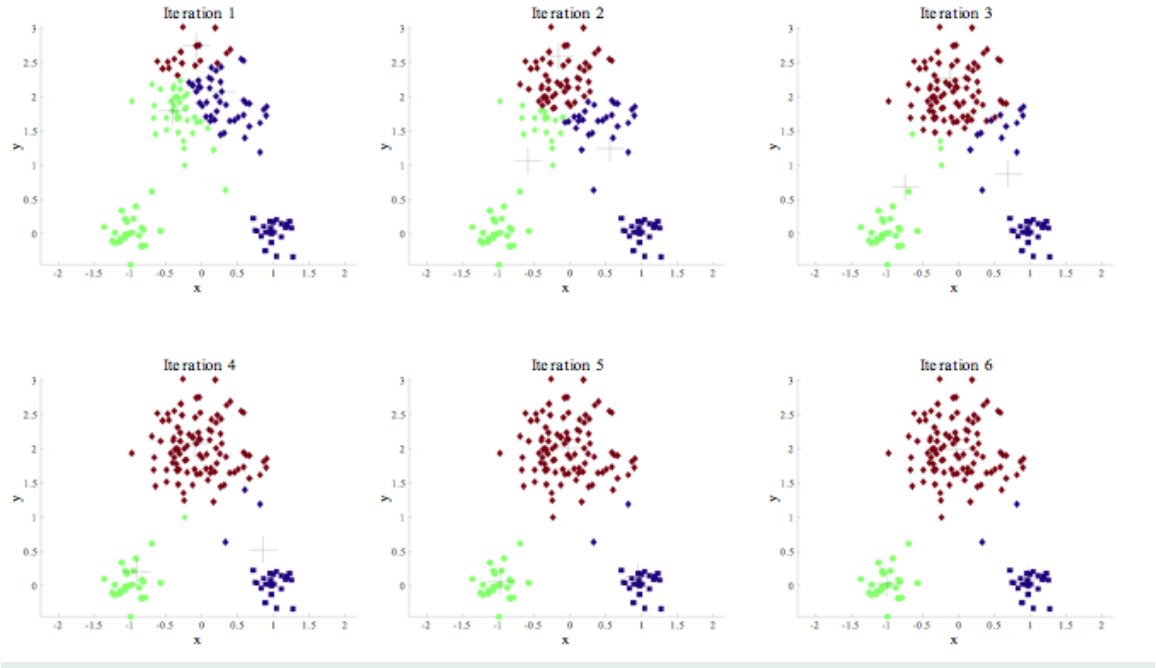


FIGURE 5. The K-means algorithm in action.

4. MODEL BASED CLUSTERING

In this approach, the goal is to provide a principled statistical approach to clustering (Banfield and Raftery, 1993). The idea is that the objects are independent samples from K different groups (populations), but the group labels have been lost. If we knew that the vector α gave the group labels, and each group had class-conditional probability density function $f_k(x_i;\theta_k)$ for all i's in the group, where α_k denotes the proportion of points belonging to the k-th group. Then, the data likelihood would be given by

$$\Pi_{i=1}^{N}\Pi_k \alpha_k f_k(x_i;\theta). \tag{14}$$

Since the labels are unknown, these are regarded as parameters, and hence we need to maximize the likelihood function over $(\theta_k,\alpha_k)_{k=1}^{K}$.

A popular choice for the conditional probability density function of each component, due to its computational tractability, is multivariate normal with mean μ_k and covariance Σ_k; i.e. $f_k \equiv N(\mu_k,\Sigma_k)$. Banfield and Raftery (1993) proposed a general framework for

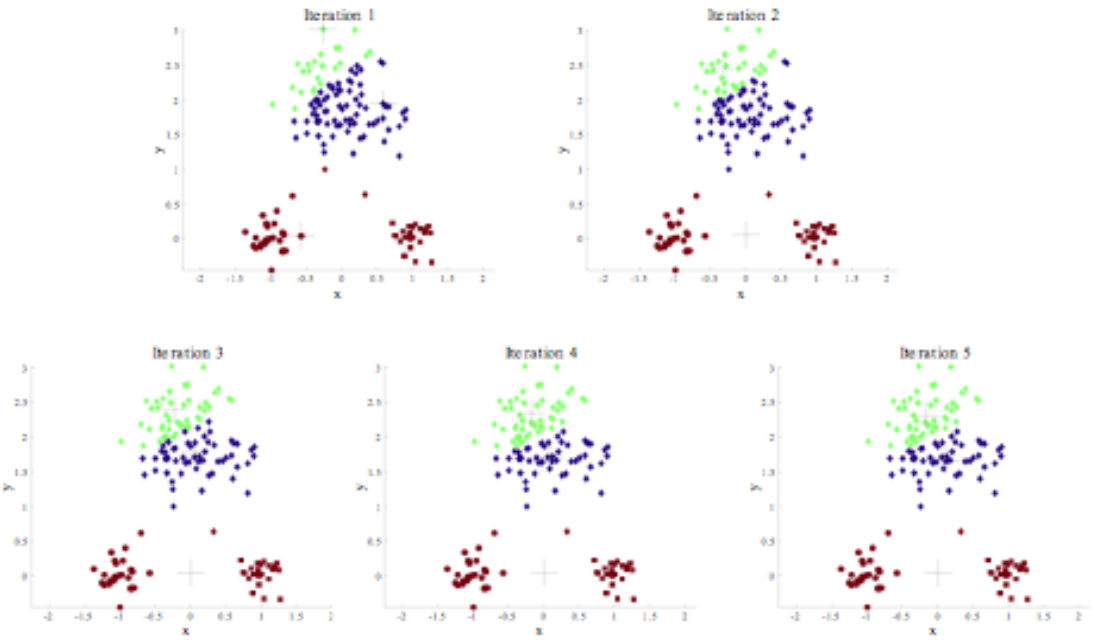


FIGURE 6. Failure of K-means to converge to a good solution due to poor initial assignments.

imposing geometric constraints in mixtures of multivariate normal distributions through the following parameterization of the k-th covariance matrix obtained from an eigenvalue decomposition.

$$\Sigma_k = \lambda_k D_k A_k D_k', \ k = 1, \ldots, K, \tag{15}$$

where D_k is the orthonormal matrix of eigenvectors, A_k is a diagonal matrix whose elements are proportional to the eigenvalues and λ_k is an associated constant of proportionality. D_k governs the orientation of the k-th cluster, A_k its shape and λ_k its volume, which is proportional to $\lambda_k \det(A_k)$. For example, if the largest eigenvalue of Σ_k is much larger in magnitude than the remaining ones, then the k$-th$ cluster will be concentrated close to a line, which will correspond to the first principal component of the k$-th$ group. If the first two eigenvalues are much larger than the rest, then the k$-th$ cluster will be concentrated close to a plane. If finally all the eigenvalues of the k$-th$ group are about the same, then the k$-th$ cluster will be roughly spherical.

The above decomposition allows one to define a rich taxonomy of possible cluster shapes for the data at hand. We discuss next a number of choices, all implemeted in the R package mclust (discussed in the next section).

(1) $\Sigma_k = \lambda_k I$ for all $k = 1, \cdots, K$. In this case, all clusters are assumed to exhibit spherical shapes, but have different volumes. The number of parameters to be estimated is K. Note that this model can be thought as a probabilistic analogue of K-means. A special case is to require $\lambda_k = \lambda$ for all k.

(2) $\Sigma_k = \lambda_k A_k$ for all $k = 1, \cdots, K$. Notice that in this case the eigenvectors $D_k = I$ for all k, which implies that the orientation of the principal axes of all the clusters is parallel to the coordinate axes. However, each cluster is allowed to have its own shape (namely the variances in each direction can be different between the clusters) and volume. This model requires estimation of Kp parameters. Another way to write this model is to require that the Σ_k be diagonal and different. Constrained versions of this model include $\Sigma_k = \lambda_k A$, where all the clusters have the same shape but their volumes vary and $\Sigma_k = \lambda A_k$, where all the clusters have the same volume but different shapes (they are similarly elongated but along different axes).

(3) $\Sigma_k = \lambda_k D_k A_k D_k'$ for all $k = 1, \cdots, K$. This is the most general model that allows clusters to have different orientations, shapes and volumes. The number of parameters to be estimated is $Kp(p+1)/2$. Constrainted versions of this model include $\Sigma_k = \lambda D A D'$, where the orientation of the cluster can be anything in p-dimensional space, but all the clusters are identical in terms of orientation, shape and volume. Other versions include $\Sigma_k = \lambda D_k A D_k'$, where the shape and the volume for all clusters is fixed, but the orientation is free to vary and $\Sigma_k = \lambda_k D_k A D_k'$, where only the shape is fixed.

These mixture models can be estimated by maximum likelihood method using the EM algorithm (Celeux and Govaert, 1995) and the best model both in terms of number of clusters and parameterization by a model selection criterion, such as the Bayesian Information one implemented in the R package mclust.

5. CLUSTERING TECHNIQUES IN R

We are going to use the wine data set for illustration purposes available from the UCI Machine Learning repository
`http://archive.ics.uci.edu/ml/machine-learning-database`. It has information about the chemical content of 178 wines that come from three different classes. In our subsequent analysis we will ignore the class information. A description of the variables is given next:

(1) Alcohol
(2) Malic acid
(3) Ash
(4) Alcalinity of ash
(5) Magnesium
(6) Total phenols
(7) Flavanoids
(8) Nonflavanoid phenols
(9) Proanthocyanins
(10) Color intensity
(11) Hue
(12) OD280/OD315 of diluted wines
(13) Proline

5.1. **Hierarchical Algorithms.** We are going to use the R package cluster that has a number of nice features, plus fast algorithms for large size data sets.

One of the features are the daisy function to calculate dissimilarities between objects using variables of mixed type, such as nominal, ordinal and numerical. Specifically, a dissimalrity betwen objects i and j is defined as

$$\delta_{ij} = \frac{\sum_{q=1}^{p} w_{ij}^{q} \delta_{ij}^{q}}{\sum_{q=1}^{p} w_{ij}^{q}} \in [0,1], \tag{16}$$

where δ_{ij}^{k} denotes the dissimilarity measure for the q-th variable, w_{ij}^{q} the corresponding weight. The weight option allows to handle missing data by setting the corresponding

weight $w_{ij}^q = 0$. The δ_{ij}^q dissimilarity measure is given by (i) $\delta_{ij}^q = |x_{iq} - x_{jq}|/(\max_\ell x_{\ell q} - \min_\ell x_{\ell q})$ for numerical variables, where $x_{\ell q}$ denotes the value of observation ℓ for the q-th variable; i.e. it uses a normalized by the range of the variable Manhattan distance, (ii) $\delta_{ij}^q = 1$ if $x_{iq} \neq x_{jq}$ for nominal variables; i.e. it uses the Hamming distance and (iii) for an ordinal variable, one first computes the ranks r_{iq} of the observations x_{iq}, then calculates $h_{iq} = (r_{iq} - 1)/(\max_\ell r_{\ell q} - 1)$ and treates the resulting h's as numerical variables.

The other interesting feature in the package is the *silhouette* measure that assesses the quality of the obtained cluster. Specifically, let $a(i)$ denote the average dissimilarity of object i that has been allocated to cluster C; it is defined as follows:

$$a(i) = \frac{1}{|C| - 1} \sum_{j \in C}, j \neq i \delta_{i,j}. \tag{17}$$

It is a measure of the homogeneity of the cluster, since if all $a(i)$'s are small, then all objects in the same cluster are fairly similar. For any cluster $\tilde{C} \neq C$ we can calculate the average dissimilarity of object $i \in C$ to all objects in $\tilde{C}$. It is given by

$$d(i, \tilde{C}) = \frac{1}{|\tilde{C}| - 1} \sum_{j \in \tilde{C}} \delta_{i,j}. \tag{18}$$

The smallerst dissimilarity over all other clusters $\tilde{C}$ is given by

$$b(i) = \min_{\tilde{C} \neq C} d(i, \tilde{C}). \tag{19}$$

The cluster that attains this minimum is the second best alternative for that object. Finally, the silhouette value $s(i)$ is defined by

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}. \tag{20}$$

It can be seen that $s(i) \in [-1, 1]$, with values close to 1 indicating a very good assignment for object i and values close to -1, a poor assignment.

We cluster next the wines in our data set, using the daisy function for calculating their corresponding dissimilarities.

```
library(cluster)

wine=read.table('wine-data.txt',h=T)
```

```
wine.dist=daisy(wine)

wine.ave=agnes(wine.dist, method="average")

# obtain average linkage clustering solution
# other options include single, complete, ward

plot(wine.ave)

# visualize the resulting dendrogram

plot(silhouette(cutree(wine.ave,k=5),daisy(wine)))

# obtain a plot of the silhouette values by using 5 clusters
```

A plot of the dendrogram and of the corresponding silhouette values is given in Figure 7. It can be seen that a 5-cluster fits the data fairly well, although there is a small cluster comprising of 6 objects only. Other options such as complete linkage and Ward's method give comparable results, but single linkage gives a very different and rather poor quality solution without any strong clustering pattern, shown in Figure 8.

5.2. **K-means variant.** The cluster package implements in the command pam a version of K-means that instead of taking the mean center as the cluster representative, considers the most central object itself. This variant can be more robust to outliers and hence give better results.

```
wine.pam=pam(wine,k=5,diss=F)

# given our prior experience we look for 5 clusters

plot(wine.pam)
```

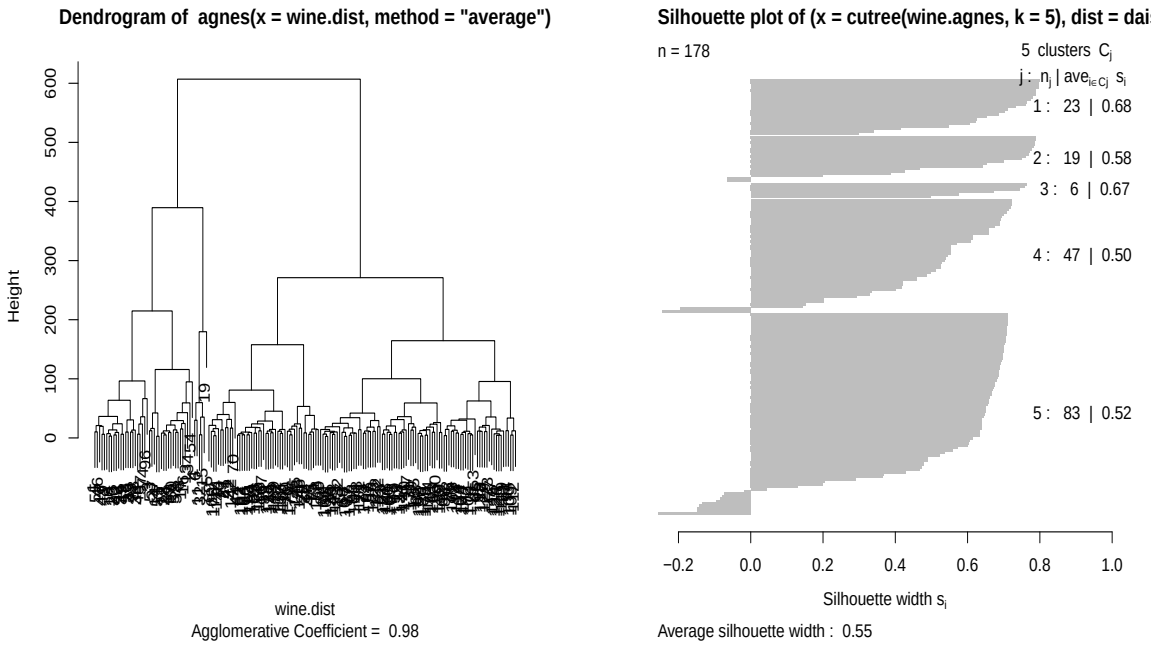


FIGURE 7. Dendrogram and silhouette plot for the wine data using average linkage.

It can be seen from the silhouette plot shown in Figure 9 that a 5-cluster solution seems to fit the data well. The package also offers the possibility to visualize the clustering structure by projecting the objects to the space spanned by the first two principal components. However, in our experience this visual display is usually not informative.

5.3. **Model based clustering.** We start by listing the different models discussed in Section 2, as called in the mclust package. So, the models forcing spherical clusters are EII (most constrained0 and VII (less constrained), those that force clusters to be aligned in orientation with the coordinate axes, EEI ($\Sigma_k = \lambda A$), VEI ($\Sigma_k = \lambda_k A$), EVI ($\Sigma_k = \lambda A_k$) and VVI ($\Sigma_k = \lambda_k A_k$) and finally the general orientation clusters by EEE ($\Sigma_k = \lambda DAD'$), EEV ($\Sigma_k = \lambda D_k A D_k'$), VEV ($\Sigma_k = \lambda_k D_k A D_k'$) and VVV the totally unconstrained one.

```
library(Mclust)

wine.mclust=Mclust(wine)
```

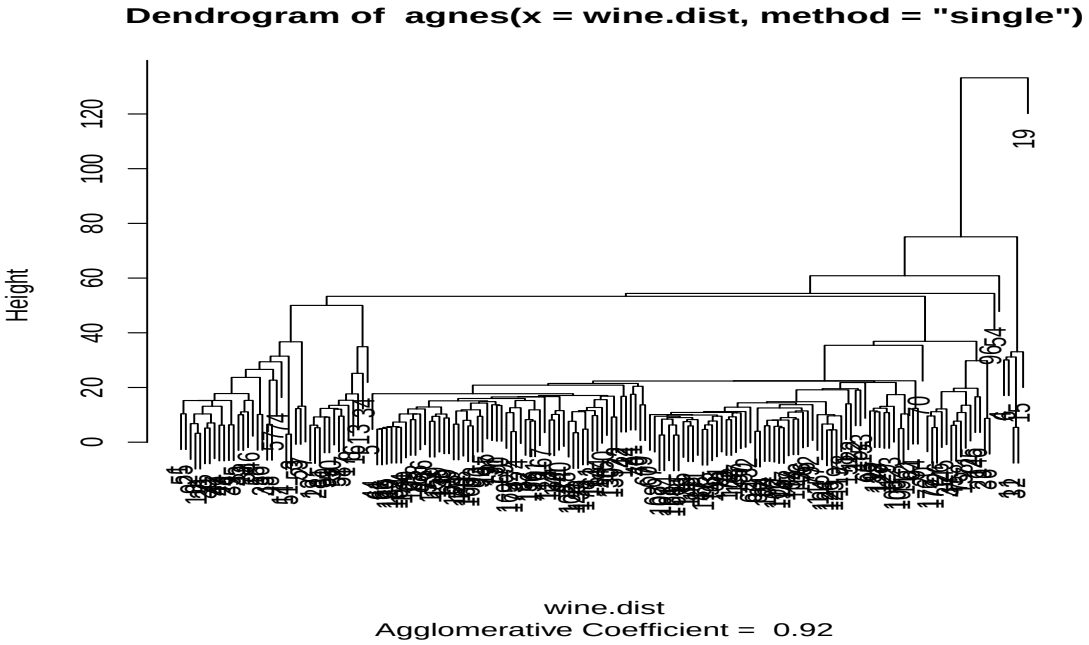


FIGURE 8. Dendrogram for the wine data using single linkage.

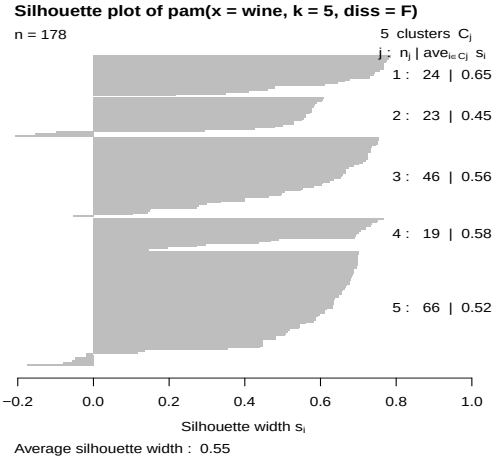


FIGURE 9. Silhouette plot for a 5-cluster solution for the wine data, obtained from a variant of K-means.

```
plot(wine.mclust)
```

```
# obtain a plot of the BIC values for all the models and different
# number of clusters



coordProj(wine,dimens=c(1,6),classification=wine.mclust$classification,
   parameters = wine.mclust$parameters)

# plot the obtained clustering solution on the space spanned by
# variables 1 and 6
```

The model selection results based on BIC values, together with the solution projected on variables 1 (Alcohol content) and 6 (Total phenols) are shown in Figure 10.

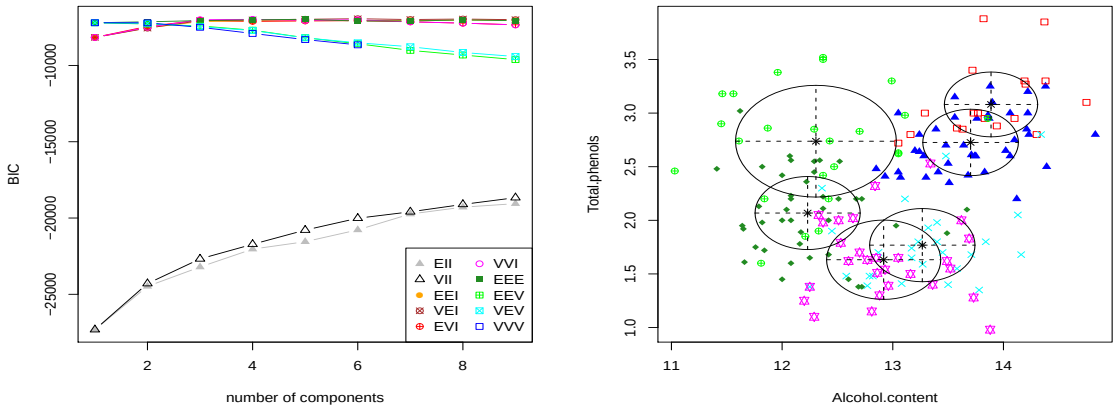


FIGURE 10. BIC values for different models and numbers of clusters and the solution projected on variables 1 and 6.

REFERENCES

[1] Banfield, J.D. and Raftery, A.E. (1993), Model based Gaussian and non-Gaussian clustering, *Biometrics*, 49, 803-821

[2] Celeux, G. and Govaert, G. (1995), Gaussian parsimonious clustering models, *Pattern Recognition*, 28, 781-793

[3] Hansen, P. and Jaumard, B. (1997), Cluster analysis and mathematical programming, *Mathematical Programming*, 79, 191-215

[4] Mirkin, B. (1996), *Mathematical Classification and Clustering*, Kluwer Academic Publishers, Dordrecht, The Netherlands

Appendix: A Review of Matrix Algebra

1. BASIC FACTS

The term *matrix* denotes a *rectangular* array of numbers. A data matrix consists of measurement scores for n objects (e.g. individuals) on m items (attributes, variables). Usually, a data matrix is written so that the objects form the rows and the items the columns. A simple example of a 3×2 matrix A is

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 4 & 7 \end{pmatrix}$$

Matrices are usually denoted by capital letters, e.g. A, B, V, etc. I will denote the typical i, j element of a matrix A by $A(i, j)$, the i^{th} row by $A(i, :)$ and the j^{th} column by $A(:, j)$.

Remark: The number of rows, n, and the number of columns, m, of a matrix define its *order*. The matrix A in our example above has order '3 by 2'. Occasionally, an $n \times m$ matrix A is denoted by $A_{n\times m}$ to show its order explicitly. If $n = m$ we have a *square* or *quadratic* matrix.

Remark: Matrices where $m = 1$ or $n = 1$ are called *vectors*. They are usually denoted by small letters, e.g. x, y, z, etc. A $n \times 1$ vector is called a *column vector* and a $1 \times n$ vector a *row vector*. The matrix A in our example consists of 3 row vectors and 2 column vectors.

Transposition: Suppose x is a column vector. One obtains the row vector x' from the column vector x simply by writing it as a row vector, an operation called *transposition*. More generally, one can also form the *transpose* of a matrix A by writing its rows as columns. The transpose is denoted by A'. For the matrix in our example, we get

$$A' = \begin{pmatrix} 1 & 3 & 4 \\ 2 & 5 & 7 \end{pmatrix}$$

Definition: A matrix A is symmetric if $A(i, j) = A(j, i)$ for all i, j, or equivalently if $A = A'$. Obviously from the definition it follows that symmetric matrices have to be square matrices. In data analysis, symmetric matrices are commonplace (e.g. correlation matrices, covariance matrices).

Matrix Addition: Addition (subtraction) is possible only if A and B have the *same order*. We formally have that $C(i, j) = A(i, j) + B(i, j)$, for all $i,\ j$. We then write $C = A + B$.

Matrix Multiplication by a Scalar: We can multiply a matrix by a number k (i.e. a scalar), which is done by multiplying each and every element of the matrix by k. Formally, $B = kA$ has elements given by $B(i, j) = kA(i, j)$ for all $i,\ j$.

Matrix Multiplication: The product of two matrices $A_{n\times k}$ and $B_{k\times m}$ is a matrix C of order $n \times m$ with elements $C(i,j) = \sum_{a=1}^{k} A(i,a)B(a,j)$, $i = 1,...,n$, $j = 1,...,m$. Matrix multiplication requires that A has as many columns as B has rows. Hence, if both A and B are square matrices, then both products AB and BA exist and are of the same order. However, it is important to realize that $AB \neq BA$ in general (matrix multiplication **is not commutative**). We therefore use special terminology and speak of *multiplication from the left* and *multiplication from the right*.

Comment: There is another way of multiplying two matrices of order $n \times m$. We get that $C = AoB$ with $C(i,j) = A(i,j)B(i,j)$. The new matrix C is called the ***Hadamard*** product of A and B. However, this type of product plays a very minor role in most applications of matrix algebra.

Diagonal and Identity Matrices: A square matrix A with $A(i,i) \neq 0$ and $A(i,j) = 0$ for all $i \neq j$ is called a diagonal matrix. For example

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3.15 & 0 \\ 0 & 0 & -1/125 \end{pmatrix}$$

is a diagonal matrix. A diagonal matrix with $A(i,i) = 1$ for all i is called the *identity* matrix and corresponds to the *neutral element* with respect to matrix multiplication. The identity matrix of order n is denoted by I_n.

Inverse of a Matrix: The inverse of a square matrix $A_{n\times n}$ is another square matrix of order $n \times n$ and denoted by A^{-1} that satisfies

$$A^{-1}A = I_n = AA^{-1} \tag{1}$$

Rank of a Matrix: The *rank* of a $n \times m$ matrix is the number of *linearly independent* rows or columns of this matrix. Obviously, $r \leq \min(n,m)$. A row (column) is linearly independent if it can NOT be expressed as a *weighted* sum of the remaining rows (columns). A matrix with $r = \min(n,m)$ is called a matrix of *full* rank. If a matrix is not of full rank, then it is called *singular*.

Existence of the Inverse of a Matrix: The inverse of a square matrix $A_{n\times n}$ exists if A is a matrix of *full rank*, i.e. $r = n$.

Definition: A $n \times n$ matrix A satisfying $A'A = I_n$ is called *orthonormal*.

But if $A'A = I_n$ this automatically implies that $A' = A^{-1}$ and because A is square we also have that $AA^{-1} = AA' = I_n$. Hence, a square matrix with orthonormal columns has necessarily orthonormal rows.

Some Basic Useful Properties of Matrices:

- $(AB)C = A(BC)$
- $(A + B)(C + D) = A(C + D) + B(C + D)$, associative property

- $(A')' = A$
- $(AB)' = B'A'$
- $(ABC)' = C'B'A'$
- $(A + B)' = A' + B'$
- $(A^{-1})^{-1} = A$
- $(A')^{-1} = (A^{-1})'$
- $(AB)^{-1} = B^{-1}A^{-1}$, provided both A and B are square

Definition: A square matrix $A_{n\times n}$ is called *positive semidefinite* iff for every vector $x \neq 0$ of order n, we have $x'Ax \geq 0$. It is called positive definite if a strict inequality holds.

2. Scalar Functions of Vectors and Matrices

One can take a matrix or a vector and assign to it, by some rule, a number. There are many such rules, and they are useful for different purposes. Here we discuss some cases that are important in the multivariate analysis context.

Scalar Product of Vectors: Given two real valued column vectors x and y of the same order n, their scalar product is given by

$$< x, y >= x'y = \sum_{i}^{n} x(i)y(i) \tag{2}$$

Of particular importance is the case where $x'y = 0$. Vectors whose scalar product is zero are called *orthogonal*. As an exercise draw on the plane the vectors $x = (-2,\ \sqrt{2})$ and $y = (\sqrt{2},\ 2)$.

Length of a Vector: The *length* of a vector is given by

$$||x|| = \sqrt{x'x} = (\sum_{i=1}^{n} x(i)^2)^{1/2}. \tag{3}$$

The length of a vector is an example of a norm. Norms are used to *normalize* a given vector to unit length. If x is a any real valued vector, then $u = (1/||x||)x$ is a *unit* vector, i.e. $||u|| = 1$.

Remark: All norms satisfy the following properties:

- $||x|| \geq 0$ for $x \neq 0$
- $||x|| = 0$ iff $x = 0$
- $||kx|| = k||x||$, for any scalar k
- $||x + y|| \leq ||x|| + ||y||$ (triangle inequality)

For example in regression problems we minimize $||\epsilon(\beta)||$ with respect to β, where $\epsilon(\beta) = y - X\beta$.

Definition: The *trace* of a *square* matrix A is given by

$$\mathbf{trace}(A) = \sum_{i=1}^{n} A(i,i). \tag{4}$$

Properties of the Trace:

- $\mathbf{trace}(A) = \mathbf{trace}(A')$
- $\mathbf{trace}(A_1A_2A_3...A_n) = \mathbf{trace}(A_2A_2...A_nA_1) = \mathbf{trace}(A_3...A_nA_1A_2) = ...$ (invariance under cyclic permutation)
- $\mathbf{trace}(A+B) = \mathbf{trace}(A) + \mathbf{trace}(B)$ (summation rule)

Frobenius Norm of a Matrix: It is given by

$$||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} A(i,j)^2} \tag{5}$$

It is also called the sum of squares norm and represents the natural generalization of the length of a vector to matrices.

A Fact: $||A||_F^2 = \mathbf{trace}(A'A) = \mathbf{trace}(AA')$.

Let us look at an example.

$$A = \begin{pmatrix} 3 & 2 \\ 1 & 0 \\ 5 & 4 \end{pmatrix}$$

Do the necessary calculations to find that $||A||_F^2 = 55$ and verify the fact.

3. EIGENDECOMPOSITIONS

In this section we deal exclusively with **square** matrices.

Definition: For any matrix A (*real or complex*), a number λ (*real or complex*) is an *eigenvalue* if it corresponds to the root of the polynomial equation given by

$$\mathbf{det}(A - \lambda I) = 0. \tag{6}$$

Definition: A nonzero vector x (*real or complex*) is an *eigenvector* of A corresponding to the eigenvalue λ if it satisfies the following equation

$$Ax = \lambda x \tag{7}$$

Remark: If x is an eigenvector of A, so is cx, for any $c \neq 0$. This property follows by plugging in cx in (7).

Remark: (Existence of eigenvalues) A matrix $A \in \mathbb{C}^{n\times n}$ (i.e. a matrix with complex entries) has exactly n complex eigenvalues (counting multiplicities). With each eigenvalue corresponds at least one eigenvector.

To see why this result is true, by writing explicitly $\mathbf{det}(A - \lambda I_n)$ we get that we deal with a complex polynomial of degree n in λ, which, by the fundamental theorem of algebra, together with the unique factorization theorem for polynomials, has precisely n complex roots. Moreover, if $\mathbf{det}(A - \lambda I_n) = 0$, then $A - \lambda I_n$ is singular, which means that zero is in the column (or row) space of A. Thus, at least one eigenvector exists.

Let us look at a few examples.

Example 1: Let

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

We want to calculate its eigenvalues, by solving (6). We have

$$\mathbf{det}(A - \lambda I) = (1 - \lambda)^2 = 0 \Longrightarrow \lambda = 1. \tag{8}$$

We call $\lambda = 1$ an eigenvalue of A with *multiplicity* 2. Hence, $Ax = 1x$, so any $x \neq 0$ is an eigenvector.

Example 2: Let

$$A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

Let us solve (6) again. We have

$$\mathbf{det}(A - \lambda I) = (1 - \lambda)^2 + 1 = 0 \Longrightarrow (1 - \lambda)^2 = -1. \tag{9}$$

Hence, we have that $\lambda_1 = 1+i$ and $\lambda_2 = 1-i$, with corresponding eigenvectors $x_1 = (1,\ i)$ and $x_2 = (1,\ -i)$.

This example shows that real matrices may have *complex* eigenvalues and eigenvectors.

Example 3: Let

$$A = \begin{pmatrix} 2 & \sqrt{2} \\ \sqrt{2} & 3 \end{pmatrix}$$

Let us solve (6) again. We have

$$\mathbf{det}(A - \lambda I) = (2 - \lambda)(3 - \lambda) - 2 = 0 \Longrightarrow \lambda^2 - 5\lambda + 4 = 0. \tag{10}$$

Hence, we have that $\lambda_1 = 4$ and $\lambda_2 = 1$, with corresponding eigenvectors $x_1 = (1,\ \sqrt{2})$ and $x_2 = (\sqrt{2},\ -1)$.

Observe that in all these examples x_1 and x_2 are orthogonal, i.e. $x_1' x_2 = 0$.

If we collect all the eigenvalues of A into a $n \times n$ *diagonal* matrix Λ and all corresponding eigenvectors into a $n \times n$ matrix X (the eigenvectors correspond to the columns of X), (6)

can be rewritten as

$$AX = X\Lambda. \tag{11}$$

In addition we require $X'X = I_n$ (standardize them by making them orthonormal).

A few important results for us are given next.

Theorem: Eigenvalues of real symmetric matrices are real and so are their corresponding eigenvectors.

Comment: Due to this Theorem, the fact that we obtained real eigenvalues and eigenvectors in Example 3 should not come as a surprise.

Theorem: A Gramian (i.e symmetric, positive definite) matrix has real *positive* eigenvalues and eigenvectors.

Proof: The fact that they are real follows trivially from the previous Theorem. For being positive, consider any eigenvalue λ and corresponding eigenvector x of A. They obviously satisfy $Ax = \lambda x$; hence, $0 < x'Ax = \lambda x'x$. Since $x'x$ as a sum of squares is positive, so is λ.

Rayleigh-Ritz Theorem: (Characterization of Eigenvalues). Suppose we want to maximize/minimize the quadratic form $x'Ax$ with respect to x subject to the constraint $x'x = 1$. Then, the maximum eigenvalue achieves the maximum and the minimum eigenvalue the minimum and the solution x is the corresponding eigenvector.

Remark: Suppose we want to calculate $\mathbf{trace}(A)$. Using the eigendecomposition of A we get $\mathbf{trace}(A) = \mathbf{trace}(X\Lambda X') = \mathbf{trace}(\Lambda X'X) = \mathbf{trace}(\Lambda I_n) = \mathbf{trace}(\Lambda) = \sum_{i=1}^{n} \Lambda(i,i)$. Thus, the trace of a matrix A is equal to the *sum* of its eigenvalues.

Remark: Eigenvalues come in handy for computing *inverse* matrices. Let $AX = X\Lambda$ be the eigendecomposition of A, which can be rewritten as $A = X\Lambda X'$. We are looking for a matrix B such that $BA = I_n$ (this fact follows from the definition of the inverse matrix). Notice that A needs to be positive definite in order to have an inverse. So, we want $BX\Lambda X' = I_n$ or $BX\Lambda = X$ or $BX\Lambda\Lambda^{-1} = X\Lambda^{-1}$ or $B = X\Lambda^{-1}X'$. Thus, $A^{-1} = X\Lambda^{-1}X'$.

Geometry: We restrict our attention to real matrices. The real matrix A is among other things a linear transformation of $\mathbb{R}^n$ into $\mathbb{R}^n$. Thus, it associates a vector $y \in \mathbb{R}^n$ with each $x \in \mathbb{R}^n$ by the rule $y = Ax$. If a real vector $x \in \mathbb{R}^n$ is an eigenvector of A, then $Ax = \lambda x$ implies that λ is real. Moreover, the eigen-equation says that A transforms x into a multiple of itself, i.e. x is an *invariant direction* of the transformation.

The Generalized Eigenvalue Problem: Suppose that we have a $n \times n$ *symmetric* matrix A and another $n \times n$ *symmetric and positive definite* matrix B.

Then, any number λ and any nonzero vector x that satisfy

$$Ax = \lambda Bx \tag{12}$$

define a *generalized eigenvalue* problem.

Reduction to a Classical Eigenvalue Problem: Define $y = B^{1/2}x$ ($B^{1/2} = V\Lambda^{1/2}V'$ since B symmetric and positive definite). Then, $x = B^{-1/2}y$ ($B^{-1/2} = V\Lambda^{-1/2}V'$. So, (12) can be rewritten as

(13)
$$Ax = \lambda Bx \Longleftrightarrow AB^{-1/2}y = \lambda BB^{-1/2}y \Longleftrightarrow B^{-1/2}AB^{-1/2}y = \lambda y \Longleftrightarrow \tilde{A}y = \lambda y,$$

and thus we have to deal with just a regular eigenvalue problem.

λ corresponds to stationary values for the ratio $\frac{x'Ax}{x'Bx}$ which is called the *Rayleigh quotient*.

4. Singular Value Decomposition

Let A be a $n \times m$ real valued matrix.

Definition: Suppose that there exist matrices X of order $n \times m$, Σ diagonal and of order $n \times n$ and Y of order $n \times n$, with X and Y orthonormal, such that they satisfy the following two equations:

(14)
$$AY = X\Sigma$$

(15)
$$A'X = Y\Sigma$$

Then,

(16)
$$A = X\Sigma Y',$$

gives the *Singular Value Decomposition* of A and $\Sigma = X'AY$ is called the *canonical form* of A. The diagonal elements of Σ are called the *singular values* of A, matrix X contains the *left singular vectors* and matrix Y contains the *right singular vectors* of A, respectively.

Relation of SVD and Eigendecomposition: From (16) we get that $A'A = Y\Sigma^2Y'$, which implies that the squared singular values of A are the eigenvalues of $A'A$. Similarly, we get that $AA' = X\Sigma^2X'$.

Hence,

$$\text{SVD}(A) \approx \text{EIGEN} \begin{pmatrix} I_n & A' \\ A & I_m \end{pmatrix}$$

Theorem: (Schmidt-Beltrami-Eckart-Young) Let $B = \sum_{i=1}^{k} X(:,i)\Sigma(:,i)Y'(:,i)$, that is form the matrix B by keeping the first k columns of X, Y as well as the first k singular values. Then, B is the *best least squares rank k* approximation of A. In other words, it minimizes $\mathbf{trace}(A'A - B'B)$ over all B.

An Interesting SVD Application: The Orthogonal Procrustes Problem

Main Idea: There are two data sets and we would like by *translating* and *rotating* one of them, to make them as "similar" as possible. For example in the field of morphometrics the data points (called landmarks) describe the essential features of a biological form (e.g. head, face, etc). This is the main idea behind the Orthogonal Procrustes problem.

Notation: Let X and Y be the two $n \times m$ data matrices (see Figures 1 & 2). One of them (say X) is considered to be the *target*, and we are interested in making Y as similar as possible to the target.

The Mathematical Problem: In what follows we assume that we have already translated the Y data set so that X and Y have the same "centers." So, we are only interested in rotating Y. We need first to define how we measure similarity between the two data sets. The appropriate measure is given by the Frobenius norm, which is defined for any $n \times m$ matrix as $||A||_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{M} a_{ij}^2\right)^{1/2}$.

Hence, we are interested in finding an orthonormal matrix B (i.e. $B'B = I_m$) that would minimize $||X - YB||_F$. In words, we would like to rotate the Y data set in such a way that would minimize the sum of squared differences between the target data set and the rotated one.

Solution: It is easier to work with $(||X - YB||_F)^2$. We then have

(17)
$$(||X - YB||_F)^2 = \mathbf{trace}(X - YB)'(X - YB) = \mathbf{trace}(X'X + B'Y'YB - 2B'Y'X)$$
$$= \mathbf{trace}(X'X) + \mathbf{trace}(Y'YBB') - 2\mathbf{trace}(B'Y'X) = \mathbf{trace}(X'X) + \mathbf{trace}(Y'Y) - 2\mathbf{trace}(B'Y'X),$$

where the last equality follows from the orthonormality constraint on the matrix B. Equation (17) shows that in order to minimize $(||X - YB||_F)^2$, it suffices to maximize $\mathbf{trace}(B'Y'X)$.

Define $Z = Y'X$ and let us consider the Singular Value Decomposition (SVD) of the matrix Z. It is given by $Z = U\Sigma V'$ with Σ an $m \times m$ diagonal matrix containing the *singular values* and $U,\ V$ two orthonormal matrices containing the *left* and *right singular vectors* respectively. We then have

(18)
$$\mathbf{trace}(B'Z) = \mathbf{trace}(B'U\Sigma V') = \mathbf{trace}(V'B'U\Sigma) = \mathbf{trace}(C\Sigma) = \sum_{i=1}^{m} C(i,i)\Sigma(i,i),$$

where $C = V'B'U$ and where the last equality follows from the definition of the trace.

But due to the fact that $U,\ V$ and B are orthonormal matrices the following inequality holds:

(19)
$$\sum_{i=1}^{m} C(i,i)\Sigma(i,i) \leq \sum_{i=1}^{m} \Sigma(i,i).$$

Notice that by setting $C(i,i) = 1$ for all $i = 1, ..., m$, we achieve the **maximum** upper bound in (19). Set $B = UV'$ and C becomes

$$C = V'B'U = V'(UV')'U = V'VU'U = I_m, \tag{20}$$

as required, where the last equality follows from the fact that U and V are orthonormal matrices.

Hence the optimal B in the Orthogonal Procrustes problem is given by $B = UV'$, the orthogonal polar factor of $Y'X$. For the data sets shown in Figures 1 and 2, the optimal B is given by

$$B = \begin{bmatrix} 0.9989 & 0.0479 \\ 0.0479 & -0.9989 \end{bmatrix}$$

and the resulting picture is given in Figure 3.

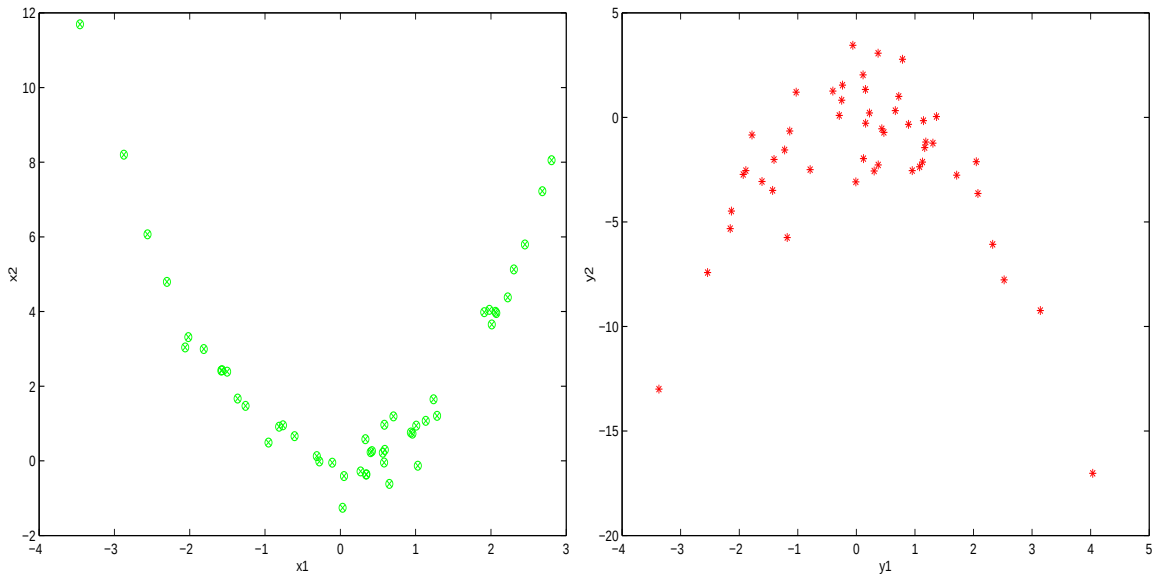


FIGURE 1. Target data set corresponding to data matrix X on the left and the data set corresponding to data matrix Y on the right.

Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an information retrieval technique based on the SVD. Variations of this technique are used by many Web search engines (for more details check the recent book by Berry and Browne, Understanding Search Engines: Mathematical Modeling and Text Retrieval, SIAM Book Series: Software, Environments, and Tools, June 1999).

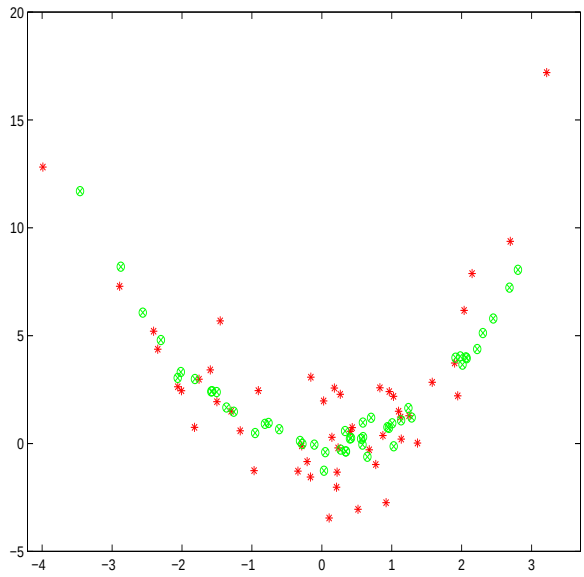


FIGURE 2. Target and rotated data sets shown together

The main problem in information retrieval is to *match* terms in documents with those appearing in the query. For example, suppose that you are interested in finding which books deal with ”multivariate analysis.” Then, your query consists of these two terms. In the old days information was retrieved by literal matching, i.e. books whose titles contained these two terms were only returned. The main shortcoming of literal matching is the presence of synonyms. Suppose that your query consisted of the term ”car.” But documents with the words ”vehicle” or ”automobile” in their titles would also be relevant to you, as well as documents containing the words ”carmakers,” ”engine,” ”chassis,” etc. Another problem with literal matching is that many words have multiple meanings (polysemy), so terms in a user's query will literally match terms in irrelevant documents. LSI attempts to overcome these difficulties by allowing users to retrieve information on the basis of a conceptual topic or meaning of a document.

The technique assumes that there is some underlying (latent) structure in word usage that is partially obscured by variability in word choice. An SVD is used to estimate the structure in word usage across documents. Retrieval is then performed using the singular values and vectors. To clarify things let us look at an example taken from Berry et al. paper (Berry, M.W., Dumais, S.T, and O'Brien, G.W. (1995), Using Linear Algebra for Intelligent Information Retrieval, SIAM Review). The following table contains a list of titles from books reviewed in SIAM review.

Label	Title
B1	A Course on Integral Equations
B2	Attractors for Semigroups and Evolution Equations
B3	Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4	Geometrical Aspects of Partial Differential Equations
B5	Ideals, Varieties and Algorithms - An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6	Intorduction to Hamiltonian Dynamical Systems and the N-body Problem
B7	Knapsack Problems: ALgorithms and Computer Implementations
B8	Methods of Solving Singular Systems of Ordinary Differential Equations
B9	Nonlinear Systems
B10	Ordinary Differential Equations
B11	Oscillation Theory for Neutral Differential Equations with Delay
B12	Oscillation Theory of Delay Differential Equations
B13	Pseudodifferential Operatos and Nonlinear Partial Differentail Equations
B14	Sinc Methods for Quadrature and Differential Equations
B15	Stability of Stochastic Differential Equations with respect to Semimartingales
B16	The Boundary Integral Approach to Static and Dynamic Contact Problems
B17	The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

The terms we are going to examine are: algorithms, application, delay, differential, equations, implementation, integral, introduction, methods, nonlinear, ordinary, oscillation, partial, problem, systems, theory.

So, we can construct a 16x17 terms by documents 0-1 data matrix (terms corresponds to rows, documents to columns) where A(i,j)=1 indicates that the i^{th} term appears in the title of the j^{th} book and 0 otherwise. The data matrix A is given in Table 1.

Remark: Usually in practice the data matrix A has entries that denote the frequency in which term i occurs in document j.

Now we will make use of the Eckart-Young Theorem to find the best k-rank approximation (in the least squares sense) to the matrix A. For illustrative purposes I will use $k = 2$. The idea is that that the first $k = 2$ *factors* will capture most of the important underlying structure in the association of terms and documents. Intuitively, since $k = 2$ is much smaller than 16, the number of unique terms, minor differences in terminology will be ignored.

Let $A = U\Sigma V'$. In the LSI context U provides information about the terms, while V about the documents. The singular values are given next:
4.5353, 2.9542, 2.6228, 1.9856, 1.8004, 1.7346, 1.6326, 1.2359, 1.0576, 1.0000, 0.8215, 0.6161 0.4718 0.2093 0.0000, 0.0000

Terms	Documents																
	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17
algorithms	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
application	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
delay	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
differential	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
equations	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
implementation	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
integral	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1
introduction	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
methods	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
nonlinear	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
ordinary	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
oscillation	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
partial	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
problem	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
systems	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0
theory	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

TABLE 1. Data matrix

It can be seen that the first 3 singular values are much larger than the remaining ones, which implies that the first 3 factors approximate reasonably well the original data. However, I will use only 2 factors to visualize the results. Let $X = U_{16\times 2}\Sigma_{2\times 2}$ denote the matrix obtained by keeping only the first two columns of U and only the first two in magnitude singular values. Similarly, let $Y = V_{17\times 2}\Sigma_{2\times 2}$. Then, I can plot the terms (red points) on the plane (their x-coordinates are contained in the first column of X and their y-coordinates in the second column of X) together with the documents (green points) (their x-coordinates are contained in the first column of Y and their y-coordinates in the second column of Y). The resulting picture is shown in the left panel of Figure 3.

It can be seen that documents B4, B13, B14 and B15 have similar titles dealing with partial differential equations (they are located between partial, methods, nonlinear, differential, equations), while documents B3, B7, B16 and B17 deal with integrals, problems, implementation and algorithms to a large extent. If I have used $k = 3$ I would have obtained a better and more accurate representation.

Suppose that you issue a query containing the terms (application, theory). I can represent it by the vector $a = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$. Then, its position in the above picture can be found by $q = aU_{17\times 2}S^{-1}_{2\times 2}$. It is plotted as the blue square in the right panel of Figure 3 and indicates that the document best matching this query is B3 followed by B17. In LSI there are techniques of determining which documents to be retrieved that go beyond simple visual inspection.

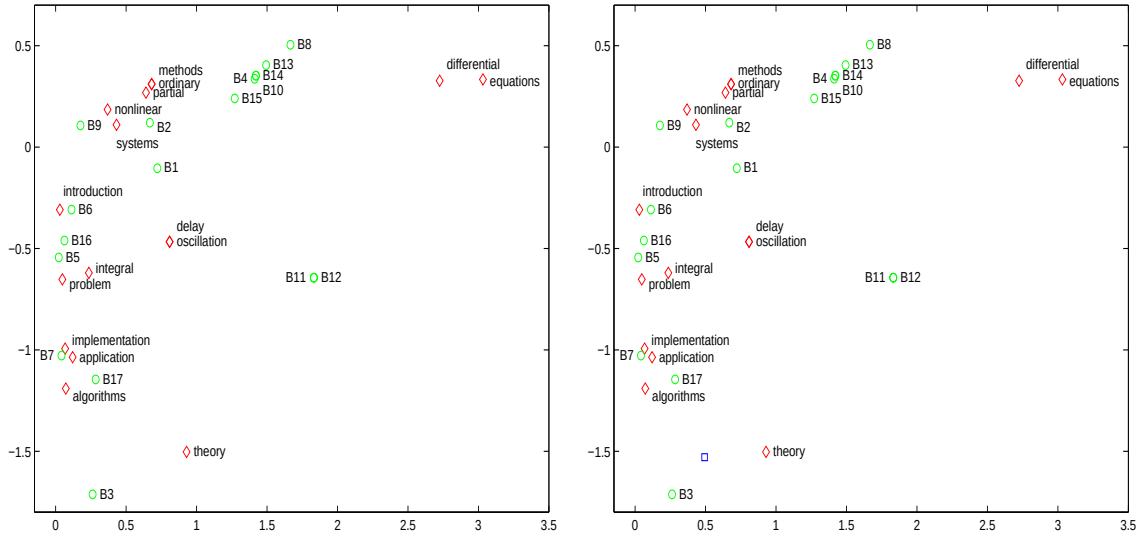


FIGURE 3. Left panel: Terms (red) and Documents (green). Right panel: Query shown as blue square.

A Review of the Multivariate Normal Distribution

A random vector $X = (X_1, \cdots, X_p)$ is said to have a *multivariate normal distribution* if it satisfies the following equivalent conditions:

(1) Every linear combination of its components $Z = a_1X_1 + \cdots + a_pX_p$ is normally distributed; i.e. for any constant vector $a \in \mathbb{R}^p$ the random variable $Z = a'X$ follows a univariate normal distribution.

(2) There exists a random vector Z of dimension ℓ whose components are independent normal random variables, a p-vector $\mu \in \mathbb{R}^p$ and a $p \times \ell$ matrix A such that $X = \mu + AZ$. Here, ℓ is the rank of the covariance matrix $\Sigma_X = \mathbb{E}(X - m)(X - \mu)' = AA'$.

(3) There is a p-vector μ and a symmetric, nonnegative-definite $p \times p$ matrix Σ, such that the characteristic function of X is $\phi_X(u) = \exp(iu'\mu - \frac{1}{2}u'\Sigma u)$.

(4) When the support of X is the entire X^p, there exists a p-vector μ and a symmetric, positive definite $p \times p$ matrix Σ such that the probability density function of X can be written as

$$f_X(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\Big(-\frac{1}{2}(X - \mu)'\Sigma^{-1}(X - \mu)\Big).$$

Remarks:

(1) We denote a p-vector X following a multivariate normal distribution with mean vector μ and covariance matrix Σ by $X \sim N(\mu, \Sigma)$.

(2) Let $Z \sim N(0, I_p)$ and A be a $k \times p$ and B be an $\ell \times p$ real matrices. Then, the multivariate vectors $X = AZ$ and $Y = BZ$ are *independent* if and only if $AB' = (BA')' = 0$.

(3) Let $Z \sim N(0, I_p)$ and $x = a'Z$ and $y = b'Z$ for some $x, y \in \mathbb{R}^p$. Then, x and y are *independent* if and only if $x'y = 0$ (x is orthogonal to y).

(4) Let $Z \sim N(\mu, \Sigma)$ and consider the partition of the p-vector Z into components Z_1 of dimension q and Z_2 of dimension $p - q$. This induces a corresponding partition of μ and Σ as follows:

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Then, the conditional distribution of $Z_1 | Z_2 = z$ is a multivariate normal with mean vector $\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(z - \mu_2)$ and covariance matrix $\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$. This matrix is known as the Schur complement of Σ_{22} in Σ.

(5) If $Z = c + BX$ is an affine transformation of $X \sim N(\mu, \Sigma)$, where c is a constant k-vector and B a $k \times p$ real matrix, then $Z \sim N(c + B\mu, B\Sigma B')$. In particular, any subset of the X_i's has a marginal distribution that is also multivariate normal.

(6) *(Geometric interpretation)* The equidensity contours of a non-singular multivariate normal distribution are ellipsoids (i.e. linear transformations of hyperspheres) centered at the mean. The directions of the principal axes of the ellipsoids are given by the eigenvectors of the corresponding covariance matrix Σ. The squared relative lengths of the principal axes are given by the corresponding eigenvalues.
If Σ is eigendecomponsed as $\Sigma = U'\Lambda U = (U\Lambda^{1/2})(U\Lambda^{1/2})'$, then

$$X \sim N(\mu, \Sigma) \iff X \sim \mu + U\Lambda^{1/2}N(0, I) \iff X \sim \mu + UN(0, \Lambda).$$

Note that if any entry of Λ is zero, this yields a signgular covariance matrix. Geometrically this implies that there is at least a principal axis of the corresponding ellipsoid that has length zero.

References:

Golub, G. and Van Loan, C. (1998), *Matrix Computations*, (3rd ed), Johns Hopkins University Press
Harville, D. (1997), *Matrix Algebra from a Statisticians Perspective*, Springer
Magnus, J. and Neudecker, H. (1988), *Matrix Differential Calculus with Applications in Statistics and Econometrics*, Wiley
Searle, S. (1982), *Matrix Algebra Useful for Statistics*, Wiley