

Part II

EDA as Pattern Discovery

Chapter 2

Dimensionality Reduction - Linear Methods

In this chapter we describe several linear methods of dimensionality reduction. We first discuss some classical methods such as principal component analysis (PCA), singular value decomposition (SVD), and factor analysis. Finally, we include a brief discussion of methods for determining the intrinsic dimensionality of a data set.

2.1 Introduction

Dimensionality reduction is the process of finding a suitable lower-dimensional space in which to represent the original data. Our hope is that the alternative representation of the data will help us:

- Explore high-dimensional data with the goal of discovering structure or patterns that lead to the formation of statistical hypotheses.
- Visualize the data using scatterplots when dimensionality is reduced to 2-D or 3-D.
- Analyze the data using statistical methods, such as clustering, smoothing, probability density estimation, or classification.

One possible method for dimensionality reduction would be to just select subsets of the variables for processing and analyze them in groups. However, in some cases, that would mean throwing out a lot of useful information. An alternative would be to create new variables that are functions (e.g., linear combinations) of the original variables. The methods we describe in this book are of the second type, where we seek a mapping from the higher-dimensional space to a lower-dimensional one, while keeping information on all of the available variables. This mapping can be linear or nonlinear.

Since the methods in this chapter transform the data using projections, we take a moment to describe this concept before going on to explain PCA and SVD. A projection will be in the form of a matrix that takes the data from the

original space to a lower-dimensional one. We illustrate this concept in Example 2.1.

Example 2.1

In this example, we show how projection works when our data set consists of the two bivariate points

$$\mathbf{x}_1 = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} -4 \\ 5 \end{bmatrix},$$

and we are projecting onto a line that is θ radians from the horizontal or x -axis. For this example, the projection matrix is given by

$$\mathbf{P} = \begin{bmatrix} (\cos \theta)^2 & \cos \theta \sin \theta \\ \cos \theta \sin \theta & (\sin \theta)^2 \end{bmatrix}.$$

The following MATLAB code is used to enter this information.

```
% Enter the data as rows of our matrix X.
X = [4 3; -4 5];
% Get theta.
theta = pi/3;
% Now obtain the projection matrix.
c2 = cos(theta)^2;
cs = cos(theta)*sin(theta);
s2 = sin(theta)^2;
P = [c2 cs; cs s2];
```

The coordinates of the projected observations are a weighted sum of the original variables, where the columns of $\mathbf{P}$ provide the weights. We project a single observation as follows

$$\mathbf{y}_i = \mathbf{P}^T \mathbf{x}_i \quad i = 1, \dots, n.$$

Expanding this out shows the new y coordinates as a weighted sum of the x variables:

$$\begin{aligned} y_{i1} &= P_{11}x_{i1} + P_{21}x_{i2} \\ y_{i2} &= P_{12}x_{i1} + P_{22}x_{i2} \end{aligned} \quad i = 1, \dots, n.$$

We have to use some linear algebra to project the data matrix $\mathbf{X}$ because the observations are *rows*. Taking the transpose of both sides of our projection equation above, we have

$$\begin{aligned} \mathbf{y}_i^T &= (\mathbf{P}^T \mathbf{x}_i)^T \\ &= \mathbf{x}_i^T \mathbf{P} . \end{aligned}$$

Thus, we can project the data using this MATLAB code:

```
% Now project the data onto the theta-line.
% Since the data are arranged as rows in the
% matrix X, we have to use the following to
% project the data.
Xp = X*P;
plot(Xp(:,1),Xp(:,2),'o') % Plot the data.
```

This projects the data onto a 1-D subspace that is at an angle θ with the original coordinate axes. As an example of a projection onto the horizontal coordinate axis, we can use these commands:

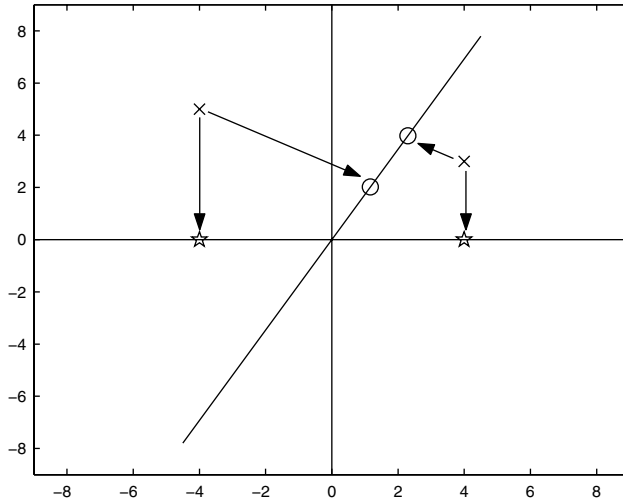
```
% We can project onto the 1-D space given by the
% horizontal axis using the projection:
Px = [1;0];
Xpx = X*Px;
```

These data now have only one coordinate value representing the number of units along the x -axis. The projections are shown in [Figure 2.1](#), where the o's represent the projection of the data onto the θ line and the stars represent the projections onto the x -axis.

□

2.2 Principal Component Analysis - PCA

The main purpose of *principal component analysis* (PCA) is to reduce the dimensionality from p to d , where $d < p$, while at the same time accounting for as much of the variation in the original data set as possible. With PCA, we transform the data to a new set of coordinates or variables that are a linear combination of the original variables. In addition, the observations in the new principal component space are uncorrelated. The hope is that we can gain information and understanding of the data by looking at the observations in the new space.

**FIGURE 2.1**

This figure shows the orthogonal projection of the data points to both the θ line (o's) and the x -axis (stars).

2.2.1 PCA Using the Sample Covariance Matrix

We start with our centered data matrix $\mathbf{X}_c$ that has dimension $n \times p$. Recall that this matrix contains observations that are centered about the mean; i.e., the sample mean has been subtracted from each row. We then form the sample covariance matrix $\mathbf{S}$ as

$$\mathbf{S} = \frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c,$$

where the superscript T denotes the matrix transpose. The jk -th element of $\mathbf{S}$ is given by

$$s_{jk} = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \quad j, k = 1, \dots, p,$$

with

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}.$$

The next step is to calculate the eigenvectors and eigenvalues of the matrix $\mathbf{S}$. The eigenvalues are found by solving the following equation for each l_j , $j = 1, \dots, p$

$$|\mathbf{S} - l_j \mathbf{I}| = 0, \quad (2.1)$$

where $\mathbf{I}$ is a $p \times p$ identity matrix and $|\bullet|$ denotes the determinant. Equation 2.1 produces a polynomial equation of degree p . The eigenvectors are obtained by solving the following set of equations for $\mathbf{a}_j$

$$(\mathbf{S} - l_j \mathbf{I})\mathbf{a}_j = 0 \quad j = 1, \dots, p,$$

subject to the condition that the set of eigenvectors is orthonormal. This means that the magnitude of each eigenvector is one, and they are orthogonal to each other:

$$\begin{aligned} \mathbf{a}_i \mathbf{a}_i^T &= 1 \\ \mathbf{a}_j \mathbf{a}_i^T &= 0, \end{aligned}$$

for $i, j = 1, \dots, p$ and $i \neq j$.

A major result in matrix algebra shows that any square, symmetric, nonsingular matrix can be transformed to a diagonal matrix using

$$\mathbf{L} = \mathbf{A}^T \mathbf{S} \mathbf{A},$$

where the columns of $\mathbf{A}$ contain the eigenvectors of $\mathbf{S}$, and $\mathbf{L}$ is a diagonal matrix with the eigenvalues along the diagonal. By convention, the eigenvalues are ordered in descending order $l_1 \geq l_2 \geq \dots \geq l_p$, with the same order imposed on the corresponding eigenvectors.

We use the eigenvectors of $\mathbf{S}$ to obtain new variables called *principal components* (PCs). The j -th PC is given by

$$z_j = \mathbf{a}_j^T (\mathbf{x} - \bar{\mathbf{x}}) \quad j = 1, \dots, p, \quad (2.2)$$

and the elements of $\mathbf{a}$ provide the weights or coefficients of the old variables in the new PC coordinate space. It can be shown that the PCA procedure defines a principal axis rotation of the original variables about their means [Jackson, 1991; Strang, 1988], and the elements of the eigenvector $\mathbf{a}$ are the direction cosines that relate the two coordinate systems. Equation 2.2 shows that the PCs are linear combinations of the original variables.

Scaling the eigenvectors to have unit length produces PCs that are uncorrelated and whose variances are equal to the corresponding eigenvalue. Other scalings are possible [Jackson, 1991], such as

$$\mathbf{v}_j = \sqrt{l_j} \mathbf{a}_j,$$

and

$$\mathbf{w}_j = \frac{\mathbf{a}_j}{\sqrt{l_j}}.$$

The three eigenvectors $\mathbf{a}_j$, $\mathbf{v}_j$, and $\mathbf{w}_j$ differ only by a scale factor. The $\mathbf{a}_j$ are typically needed for hypothesis testing and other diagnostic methods. Since they are scaled to unity, their components will always be between ± 1 . The vectors $\mathbf{v}_j$ are useful at times, because they and their PCs have the same units as the original variables. Using $\mathbf{w}_j$ in the transformation yields PCs that are uncorrelated with unit variance.

We transform the observations to the PC coordinate system via the following

$$\mathbf{Z} = \mathbf{X}_c \mathbf{A}. \quad (2.3)$$

The matrix $\mathbf{Z}$ contains the *principal component scores*. Note that these PC scores have zero mean (because we are using the data centered about the mean) and are uncorrelated. We could also transform the original observations in $\mathbf{X}$ by a similar transformation, but the PC scores in this case will have mean $\bar{\mathbf{z}}$. We can invert this transformation to get an expression relating the original variables as a function of the PCs, which is given by

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{A}\mathbf{z}.$$

To summarize: the *transformed variables* are the PCs and the individual *transformed data values* are the PC scores.

The dimensionality of the principal component scores in Equation 2.3 is still p , so no dimensionality reduction has taken place. We know that the sum of the variances of the original variables is equal to the sum of the eigenvalues. The idea of dimensionality reduction with PCA is that one could include in the analysis only those PCs that have the highest eigenvalues, thus accounting for the highest amount of variation with fewer dimensions or PC variables. We can reduce the dimensionality to d with the following

$$\mathbf{Z}_d = \mathbf{X}_c \mathbf{A}_d, \quad (2.4)$$

where $\mathbf{A}_d$ contains the first d eigenvectors or columns of $\mathbf{A}$. We see that $\mathbf{Z}_d$ is an $n \times d$ matrix (each observation now has only d elements), and $\mathbf{A}_d$ is a $p \times d$ matrix.

2.2.2 PCA Using the Sample Correlation Matrix

We can scale the data first to have standard units, as described in [Chapter 1](#). This means we have the j -th element of $\mathbf{x}^*$ given by

$$x_j^* = \frac{(x_j - \bar{x}_j)}{\sqrt{s_{jj}}}; \quad j = 1, \dots, p,$$

where s_{jj} is the variance of x_j (i.e., the jj -th element of the sample covariance matrix $\mathbf{S}$). The standardized data $\mathbf{x}^*$ are then treated as observations in the PCA process.

The covariance of this standardized data set is the same as the correlation matrix. The ij -th element of the sample correlation matrix $\mathbf{R}$ is given by

$$r_{ij} = \frac{s_{ij}}{\sqrt{s_{ii}}\sqrt{s_{jj}}},$$

where s_{ij} is the ij -th element of $\mathbf{S}$ and s_{ii} is the i -th diagonal element of $\mathbf{S}$. The rest of the results from PCA hold for the $\mathbf{x}^*$. For example, we can transform the standardized data using Equation 2.3 or reduce the dimensionality using Equation 2.4, where now the matrices $\mathbf{A}$ and $\mathbf{A}_d$ contain the eigenvectors of the correlation matrix.

The correlation matrix should be used for PCA when the variances along the original dimensions are very different; i.e., if some variables have variances that are very much greater than the others. In this case, the first few PCs will be dominated by those same variables and will not be very informative. This is often the case when the variables are of different types or units. Another benefit of using the correlation matrix rather than the covariance matrix arises when one wants to compare the results of PCA among different analyses.

PCA based on covariance matrices does have certain advantages, too. Methods for statistical inference based on the sample PCs from covariance matrices are easier and are available in the literature. The PCs obtained from the correlation and covariance matrices do not provide equivalent information. Additionally, the eigenvectors and eigenvalues from one process do not have simple relationships or correspondence with those from the other one [Jolliffe, 1986]. Since this text is primarily concerned with exploratory data analysis, not inferential methods, we do not discuss this further. In any event, in the spirit of EDA, the analyst should take advantage of both methods to describe and to explore the data.

2.2.3 How Many Dimensions Should We Keep?

One of the key questions at this point is how many PCs to keep. We offer the following possible ways to address this question; more details and options (such as hypothesis tests for equality of eigenvalues, cross-validation, and correlation procedures) can be found in Jackson [1991]. All of the following techniques will be explored in Example 2.2.

Cumulative Percentage of Variance Explained

This is a popular method of determining the number of PCs to use in PCA dimensionality reduction and seems to be implemented in many computer packages. The idea is to select those d PCs that contribute a cumulative percentage of total variation in the data, which is calculated using

$$t_d = 100 \sum_{i=1}^d l_i \div \sum_{j=1}^p l_j.$$

If the correlation matrix is used for PCA, then this is simplified to

$$t_d = \frac{100}{p} \sum_{i=1}^d l_i.$$

Choosing a value for t_d can be problematic, but typical values range between 70% and 95%. We note that Jackson [1991] does not recommend using this method.

Scree Plot

A graphical way of determining the number of PCs to retain is called the *scree plot*. The original name and idea is from Cattell [1966], and it is a plot of l_k (the eigenvalue) versus k (the index of the eigenvalue). In some cases, we might plot the log of the eigenvalues when the first eigenvalues are very large. This type of plot is called a *log-eigenvalue* or LEV plot. To use the scree plot, one looks for the ‘elbow’ in the curve or the place where the curve levels off and becomes almost flat. Another way to look at this is by the slopes of the lines connecting the points. When the slopes start to level off and become less steep, that is the number of PCs one should keep.

The Broken Stick

In this method, we choose the number of PCs based on the size of the eigenvalue or the proportion of the variance explained by the individual PC.

If we take a line segment and randomly divide it into p segments, then the expected length of the k -th longest segment is

$$g_k = \frac{1}{p} \sum_{i=k}^p \frac{1}{i}.$$

If the proportion of the variance explained by the k -th PC is greater than g_k , then that PC is kept. We can say that these PCs account for more variance than would be expected by chance alone.

Size of Variance

One rule that can be used for correlation-based PCA is due to Kaiser [1960], but is more commonly used in factor analysis. Using this rule, we would retain PCs whose variances are greater than 1 ($l_k \geq 1$). Some suggest that this is too high [Jolliffe, 1972], so a better rule would be to keep PCs whose variances are greater than 0.7 ($l_k \geq 0.7$). We can use something similar for covariance-based PCA, where we use the average of the eigenvalues rather than 1. In this case, we would keep PCs if

$$l_k \geq 0.7 \bar{l} \quad \text{or} \quad l_k \geq \bar{l}.$$

Example 2.2

We show how to perform PCA using the **yeast** cell cycle data set. Recall from [Chapter 1](#) that these contain 384 genes corresponding to five phases, measured at 17 time points. We first load the data and center each row.

```
load yeast
[n,p] = size(data);
% Center the data.
datac = data - repmat(sum(data)/n,n,1);
% Find the covariance matrix.
covm = cov(datac);
```

We are going to use the covariance matrix in PCA since the variables have common units. The reader is asked to explore the correlation matrix approach in the exercises. The **eig** function is used to calculate the eigenvalues and eigenvectors. MATLAB returns the eigenvalues in a diagonal matrix, and they are in ascending order, so they must be flipped to get the scree plot.

```
[eigvec,eigval] = eig(covm);
eigval = diag(eigval); % Extract the diagonal elements
% Order in descending order
eigval = flipud(eigval);
eigvec = eigvec(:,p:-1:1);
```

```
% Do a scree plot.
figure, plot(1:length(eigval),eigval,'ko-')
title('Scree Plot')
xlabel('Eigenvalue Index - k')
ylabel('Eigenvalue')
```

We see from the scree plot in Figure 2.2 that keeping four PCs seems reasonable. Next we calculate the cumulative percentage of variance explained.

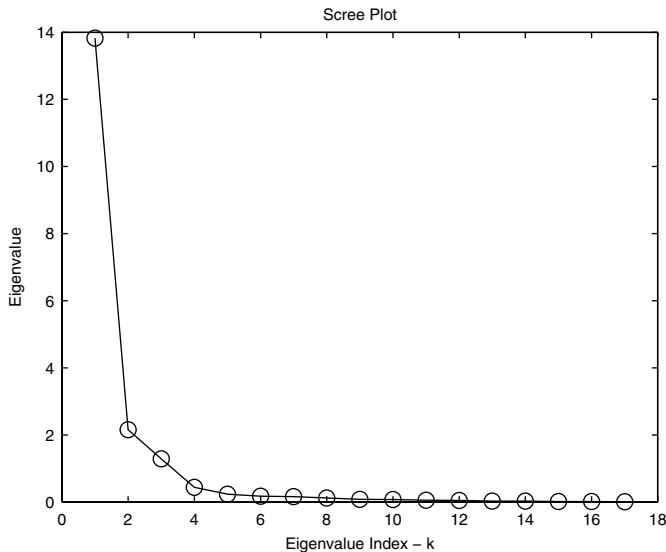


FIGURE 2.2

This is the scree plot for the **yeast** data. The elbow in the curve seems to occur at $k = 4$.

```
% Now for the percentage of variance explained.
pervar = 100*cumsum(eigval)/sum(eigval);
```

The first several values are:

```
73.5923    85.0875    91.9656    94.3217    95.5616
```

Depending on the cutoff value, we would keep four to five PCs (if we are using the higher end of the range of t_d). Now we show how to do the broken stick test.

```
% First get the expected sizes of the eigenvalues.
g = zeros(1,p);
for k = 1:p
    for i = k:p
```

```

    g(k) = g(k) + 1/i;
end
end
g = g/p;

```

The next step is to find the proportion of the variance explained.

```
propvar = eigval/sum(eigval);
```

Looking only at the first several values, we get the following for g_k and the proportion of variance explained by each PC:

```

g(1:4) =          0.2023      0.1435      0.1141      0.0945
propvar(1:4) = 0.7359      0.1150      0.0688      0.0236

```

Thus, we see that only the first PC would be retained using this method. Finally, we look at the size of the variance.

```

% Now for the size of the variance.
avgeig = mean(eigval);
% Find the length of ind:
ind = find(eigval > avgeig);
length(ind)

```

According to this test, the first three PCs would be retained. So, we see that different values of d are obtained using the various procedures. Because of visualization issues, we will use the first three PCs to reduce the dimensionality of the data, as follows.

```

% So, using 3, we will reduce the dimensionality.
P = eigvec(:,1:3);
Xp = datac*P;
figure,plot3(Xp(:,1),Xp(:,2),Xp(:,3),'k*')
xlabel('PC 1'),ylabel('PC 2'),zlabel('PC 3')
grid on, axis tight

```

These results are shown in [Figure 2.3](#).

□

We illustrated the use of the **eig** function that comes in the main MATLAB package. It contains another useful function called **eigs** that can be used to find the PCs and eigenvalues of sparse matrices. For those who have the Statistics Toolbox, there is a function called **princomp**. It returns the PCs, the PC scores and other useful information for making inferences regarding the eigenvalues. It centers the observations at the means, but does not rescale (Statistics Toolbox, Version 5). For PCA using the covariance matrix, the **pcacov** function is provided.

Before moving on to the next topic, we recall that the PCA described in this book is based on the sample covariance or sample correlation matrix. The procedure is similar if the population version of these matrices is used. Many interesting and useful properties are known about principal components and

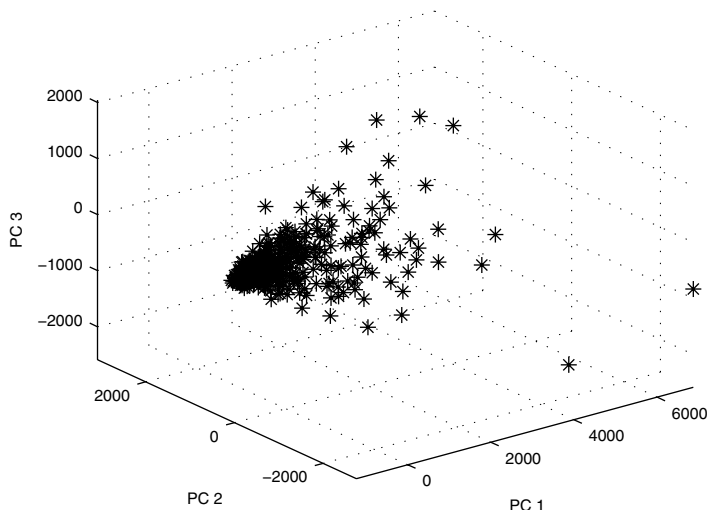


FIGURE 2.3

This shows the results of projecting the **yeast** data onto the first three PCs.

the associated data transformations, but are beyond the scope and purpose of this book. We provide references in the last section.

2.3 Singular Value Decomposition - SVD

Singular value decomposition (SVD) is an important method from matrix algebra and is related to PCA. In fact, it provides a way to find the PCs without explicitly calculating the covariance matrix [Gentle, 2002]. It also enjoys widespread use in the analysis of gene expression data [Alter, Brown and Botstein, 2000; Wall, Dyck and Brettin, 2001; Wall, Rechtsteiner and Rocha, 2003] and in information retrieval applications [Deerwester, et al., 1990; Berry, Dumais and O'Brien, 1995; Berry, Drmac and Jessup, 1999], so it is an important technique its own right.

As before, we start with our data matrix $\mathbf{X}$, where in some cases, we will center the data about their mean to get $\mathbf{X}_c$. We use the noncentered form in the explanation that follows, but the technique is valid for an arbitrary matrix; i.e., the matrix does not have to be square. The SVD of $\mathbf{X}$ is given by

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T, \quad (2.5)$$

where $\mathbf{U}$ is an $n \times n$ matrix, $\mathbf{D}$ is a diagonal matrix with n rows and p columns, and $\mathbf{V}$ has dimensions $p \times p$. The columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal. $\mathbf{D}$ is a matrix containing the *singular values* along its diagonal, which are the square roots of the eigenvalues of $\mathbf{X}^T \mathbf{X}$, and zeros everywhere else.

The columns of $\mathbf{U}$ are called the *left singular vectors* and are calculated as the eigenvectors of $\mathbf{X}\mathbf{X}^T$ (this is an $n \times n$ matrix). Similarly, the columns of $\mathbf{V}$ are called the *right singular vectors*, and these are the eigenvectors of $\mathbf{X}^T \mathbf{X}$ (this is a $p \times p$ matrix). An additional point of interest is that the singular values are the square roots of the eigenvalues of $\mathbf{X}^T \mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$.

Let the rank of the matrix $\mathbf{X}$ be given by r , where

$$r \leq \min(n, p).$$

Then the first r columns of $\mathbf{V}$ form an orthonormal basis for the column space of $\mathbf{X}$, and the first r columns of $\mathbf{U}$ form a basis for the row space of $\mathbf{X}$ [Strang, 1993]. As with PCA, we order the singular values largest to smallest and impose the same order on the columns of $\mathbf{U}$ and $\mathbf{V}$. A lower rank approximation to the original matrix $\mathbf{X}$ is obtained via

$$\mathbf{X}_k = \mathbf{U}_k \mathbf{D}_k \mathbf{V}_k^T, \quad (2.6)$$

where $\mathbf{U}_k$ is an $n \times k$ matrix containing the first k columns of $\mathbf{U}$, $\mathbf{V}_k$ is the $p \times k$ matrix whose columns are the first k columns of $\mathbf{V}$, and $\mathbf{D}_k$ is a $k \times k$ diagonal matrix whose diagonal elements are the k largest singular values of $\mathbf{X}$. It can be shown that the approximation given in Equation 2.6 is the best one in a least squares sense.

To illustrate the SVD approach, we look at an example from information retrieval called *latent semantic indexing* or LSI [Deerwester, et al., 1990]. Many applications of information retrieval (IR) rely on lexical matching, where words are matched between a user's query and those words assigned to documents in a corpus or database. However, the words people use to describe documents can be diverse and imprecise, so the results of the queries are often less than perfect. LSI uses SVD to derive vectors that are more robust at representing the meaning of words and documents.

Example 2.3

This illustrative example of SVD applied to information retrieval (IR) is taken from Berry, Drmac and Jessup [1999]. The documents in the corpus comprise a small set of book titles, and a subset of words have been used in the analysis, where some have been replaced by their root words (e.g., *bake* and *baking* are both *bake*). The documents and terms are shown in Table 2.1. We start with a data matrix, where each row corresponds to a term, and each column corresponds to a document in the corpus. The elements of the term-document matrix $\mathbf{X}$ denote the number of times the word appears in the

TABLE 2.1

Document Information for Example 2.3

Number	Title
Doc 1	How to <i>Bake Bread</i> Without <i>Recipes</i>
Doc 2	The Classic Art of Viennese <i>Pastry</i>
Doc 3	Numerical <i>Recipes</i> : The Art of Scientific Computing
Doc 4	<i>Breads, Pastries, Pies</i> and <i>Cakes</i> : Quantity <i>Baking Recipes</i>
Doc 5	<i>Pastry</i> : A Book of Best French <i>Recipes</i>
Term 1	bak (e, ing)
Term 2	recipes
Term 3	bread
Term 4	cake
Term 5	pastr (y, ies)
Term 6	pie

document. In this application, we are not going to center the observations, but we do pre-process the matrix by normalizing each column such that the magnitude of the column is 1. This is done to ensure that relevance between the document and the query is not measured by the absolute count of the terms.¹ The following MATLAB code starts the process:

```
load lsidx
% Loads up variables: X, termdoc, docs and words.
% Convert the matrix to one that has columns
% with a magnitude of 1.
[n,p] = size(termdoc);
for i = 1:p
    termdoc(:,i) = X(:,i)/norm(X(:,i));
end
```

Say we want to find books about *baking bread*, then the vector that represents this query is given by a column vector with a 1 in the first and fourth positions:

```
q1 = [1 0 1 0 0 0]';
```

If we are seeking books that pertain only to *baking*, then the query vector is:

```
q2 = [1 0 0 0 0 0]';
```

We can find the most relevant documents using the original term-document matrix by finding the cosine of the angle between the query vectors and the columns (i.e., the vectors representing documents or books); the higher cosine values indicate greater similarity between the query and the document. This would be a straightforward application of lexical matching. Recall from matrix algebra that the cosine of the angle between two vectors $\mathbf{x}$ and $\mathbf{y}$ is given by

¹Other term weights can be found in the literature [Berry and Browne, 1999].

$$\cos\theta_{x,y} = \frac{\mathbf{x}^T \mathbf{y}}{\sqrt{\mathbf{x}^T \mathbf{x}} \sqrt{\mathbf{y}^T \mathbf{y}}}.$$

The MATLAB code to find the cosines for the query vectors in our example is:

```
% Find the cosine of the angle between
% columns of termdoc and query.
% Note that the magnitude of q1 is not 1.
m1 = norm(q1);
cosq1a = q1'*termdoc/m1;
% Note that the magnitude of q2 is 1.
cosq2a = q2'*termdoc;
```

The resulting cosine values are:

```
cosq1a = 0.8165, 0, 0, 0.5774, 0
cosq2a = 0.5774, 0, 0, 0.4082, 0
```

If we use a cutoff value of 0.5, then the relevant books for our first query are the first and the fourth ones, which are those that describe *baking bread*. On the other hand, the second query matches with the first book, but misses the fourth one, which would be relevant. Researchers in the area of IR have applied several techniques to alleviate this problem, one of which is LSI. One of the powerful uses of LSI is in matching a user's query with existing documents in the corpus whose representations have been reduced to lower rank approximations via the SVD. The idea being that some of the dimensions represented by the full term-document matrix are noise and that documents will have closer semantic structure after dimensionality reduction using SVD. So, we now find the singular value decomposition using the function `svd`.

```
% Find the singular value decomposition.
[u,d,v] = svd(termdoc);
```

We then find the representation of the query vector in the reduced space given by the first k columns of $\mathbf{U}$ in the following manner

$$\mathbf{q}_k = \mathbf{U}_k^T \mathbf{q},$$

which is a vector with k elements. We note that in some applications of LSI, the following is used as the reduced query

$$\mathbf{q}_k = \mathbf{D}_k^{-1} \mathbf{U}_k^T \mathbf{q}.$$

This is simply a scaling by the singular values, since $\mathbf{D}$ is diagonal. The following code projects the query into the reduced space and also finds the cosine of the angle between the query vector and the columns. Berry, Drmac

and Jessup show that we do not have to form the full reduced matrix $\mathbf{X}_k$. Instead, we can use the columns of $\mathbf{V}_k$, saving storage space.

```
% Project the query vectors.
q1t = u(:,1:3) '* q1;
q2t = u(:,1:3) '* q2;
% Now find the cosine of the angle between the query
% vector and the columns of the reduced rank matrix,
% scaled by D.
for i = 1:5
    sj = d(1:3,1:3) * v(i,1:3)';
    m3 = norm(sj);
    cosq1b(i) = sj' * q1t / (m3 * m1);
    cosq2b(i) = sj' * q2t / (m3);
end
```

From this we have

```
cosq1b = 0.7327, -0.0469, 0.0330, 0.7161, -0.0097
cosq2b = 0.5181, -0.0332, 0.0233, 0.5064, -0.0069
```

Using a cutoff value of 0.5, we now correctly have documents 1 and 4 as being relevant to our queries on *baking bread* and *baking*.

□

Note that in the above loop, we are using the magnitudes of the original query vectors as in Berry, Drmac and Jessup [Equation 6.1, 1999]. This saves on computations and also improves precision (disregarding irrelevant information). We could divide by the magnitudes of the reduced query vectors (**q1t** and **q2t**) in the loop to improve recall (retrieving relevant information) at the expense of precision.

Before going on to the next topic, we point out that the literature describing SVD applied to LSI and gene expression data defines the matrices of Equation 2.5 in a different way. The decomposition is the same, but the difference is in the size of the matrices **U** and **D**. Some definitions have the dimensions of **U** as $n \times p$ and **D** with dimensions $p \times p$ [Golub & Van Loan, 1996]. We follow the definition in Strang [1988, 1993], which is also used in MATLAB.

2.4 Factor Analysis

There is much confusion in the literature as to the exact definition of the technique called *factor analysis* [Jackson, 1981], but we follow the commonly used definition given in Jolliffe [1986]. In the past, this method has also been confused with PCA, mostly because PCA is sometimes provided in software

packages as a special case of factor analysis. Both of these techniques attempt to reduce the dimensionality of a data set, but they are different from one another in many ways. We describe these differences at the end of the discussion of factor analysis.

The idea underlying factor analysis is that the p observed random variables can be written as linear functions of $d < p$ unobserved *latent variables* or *common factors* f_j , as follows:

$$\begin{aligned} x_1 &= \lambda_{11}f_1 + \dots + \lambda_{1d}f_d + \varepsilon_1 \\ &\dots \\ x_p &= \lambda_{p1}f_1 + \dots + \lambda_{pd}f_d + \varepsilon_p. \end{aligned} \tag{2.7}$$

The λ_{ij} ($i = 1, \dots, p$ and $j = 1, \dots, d$) in the above model are called the *factor loadings*, and the error terms ε_i are called the *specific factors*. Note that the error terms ε_i are specific to *each* of the original variables, while the f_j are common to *all* of the variables. The sum of the squared factor loadings for the i -th variable

$$\lambda_{i1}^2 + \dots + \lambda_{id}^2$$

is called the *communality* of x_i .

We see from the model in Equation 2.7 that the original variables are written as a linear function of a smaller number of variables or factors, thus reducing the dimensionality. It is hoped that the factors provide a summary or clustering of the original variables (not the observations), such that insights are provided about the underlying structure of the data. While this model is fairly standard, it can be extended to include more error terms, such as measurement error. It can also be made nonlinear [Jolliffe, 1986].

The matrix form of the factor analysis model is

$$\mathbf{x} = \mathbf{\Lambda} \mathbf{f} + \mathbf{e}. \tag{2.8}$$

Some assumptions are made regarding this model, which are

$$E[\mathbf{e}] = \mathbf{0} \quad E[\mathbf{f}] = \mathbf{0} \quad E[\mathbf{x}] = \mathbf{0},$$

where $E[\bullet]$ denotes the expected value. If the last of these assumptions is violated, the model can be adjusted to accommodate this, yielding

$$\mathbf{x} = \mathbf{\Lambda} \mathbf{f} + \mathbf{e} + \boldsymbol{\mu}, \tag{2.9}$$

where $E[\mathbf{x}] = \boldsymbol{\mu}$. We also assume that the error terms ε_i are uncorrelated with each other, and that the common factors are uncorrelated with the specific

factors f_j . Given these assumptions, the sample covariance (or correlation) matrix is of the form

$$\mathbf{S} = \mathbf{\Lambda}^T \mathbf{\Lambda} + \mathbf{\Psi},$$

where $\mathbf{\Psi}$ is a diagonal matrix representing $E[\mathbf{e}\mathbf{e}^T]$. The variance of ε_i is called the *specificity* of x_i , so the matrix $\mathbf{\Psi}$ is also called the *specificity matrix*. Factor analysis obtains a reduction in dimensionality by postulating a model that relates the original variables x_i to the d hypothetical variables or factors.

The matrix form of the factor analysis model is reminiscent of a regression problem, but here both $\mathbf{\Lambda}$ and $\mathbf{f}$ are unknown and must be estimated. This leads to one problem with factor analysis: the estimates are not unique. Estimation of the parameters in the factor analysis model is usually accomplished via the matrices $\mathbf{\Lambda}$ and $\mathbf{\Psi}$. The estimation proceeds in stages, where an initial estimate is found by placing conditions on $\mathbf{\Lambda}$. Once this initial estimate is obtained, other solutions can be found by rotating $\mathbf{\Lambda}$. The goal of some rotations is to make the structure of $\mathbf{\Lambda}$ more interpretable, by making the λ_{ij} close to one or zero. Several methods that find rotations such that a desired criterion is optimized are described in the literature. Some of the commonly used methods are varimax, equimax, orthomax, quartimax, promax, and procrustes.

These factor rotation methods can either be *orthogonal* or *oblique*. In the case of orthogonal rotations, the axes are kept at 90 degrees. If this constraint is relaxed, then we have oblique rotations. The orthogonal rotation methods include quartimax, varimax, orthomax, and equimax. The promax and procrustes rotations are oblique.

The goal of the quartimax rotation is to simplify the rows of the factor matrix by getting a variable with a high loading on one factor and small loadings on all other factors. The varimax rotation focuses on simplifying the columns of the factor matrix. From the varimax approach, perfect simplification is obtained if there are only ones and zeros in a single column. The output from this method tends to have high loadings close to ± 1 and some near zero in each column. The equimax rotation is a compromise between these two methods, where both the rows and the columns of the factor matrix are simplified as much as possible.

Just as we did in PCA, we might want to transform the observations using the estimated factor analysis model either for plotting purposes or for further analysis methods, such as clustering or classification. We could think of these observations as being transformed to the ‘factor space.’ These are called *factor scores*, similarly to PCA. However, unlike PCA, there is no single method for finding the factor scores, and the analyst must keep in mind that the factor scores are really *estimates* and depend on the method that is used.

An in-depth discussion of the many methods for estimating the factor loadings, the variances $\mathbf{\Psi}$, and the factor scores, as well as the rotations, is beyond the scope of this book. For more information on the details of the

methods for estimation and rotation in factor analysis, see Jackson [1991], Lawley and Maxwell [1971], or Cattell [1978]. Before we go on to an example of factor analysis, we note that the MATLAB Statistics Toolbox uses the maximum likelihood method to obtain the factor loadings, and it also implements some of the various rotation methods mentioned earlier.

Example 2.4

In this example, we examine some data provided with the Statistics Toolbox, called **stockreturns**. An alternative analysis of these data is provided in the Statistics Toolbox User's Guide. The data set consists of 100 observations, representing the percent change in stock prices for 10 companies. Thus, the data set has $n = 100$ observations and $p = 10$ variables. It turns out that the first four companies can be classified as technology, the next three as financial, and the last three as retail. We can use factor analysis to see if there is any structure in the data that supports this grouping. We first load up the data set and perform factor analysis using the function **factoran**.

```
load stockreturns
% Loads up a variable called stocks.
% Perform factor analysis: 3 factors, default rotation.
[LamVrot, PsiVrot] = factoran(stocks, 3);
```

This is the basic syntax for **factoran**, where the user must specify the number of factors (3 in this case), and the default is to use the **varimax** rotation, which optimizes a criterion based on the variance of the loadings. See the MATLAB **help** on **factoran** for more details on the rotations. Next, we specify no rotation, and we plot the matrix **Lam** (the factor loadings) in [Figure 2.4](#).

```
[Lam, Psi] = factoran(stocks, 3, 'rotate', 'none');
```

These plots show the pairwise factor loadings, and we can see that the factor loadings are not close to one of the factor axes, making it more difficult to interpret the factors. We can try rotating the matrix next using one of the oblique (nonorthogonal) rotations called **promax**, and we plot these results in [Figure 2.5](#).

```
% Now try the promax rotation.
[LProt, PProt] = factoran(stocks, 3, 'rotate', 'promax');
```

Note that we now have a more interpretable structure with the factor loadings, and we are able to group the stocks. We might also be interested in estimating the factor scores. The user is asked to explore this aspect of it in the exercises.

□

It can be very confusing trying to decide whether to use PCA or factor analysis. Since the objective of this book is to describe exploratory data analysis techniques, we suggest that both methods be used to explore the

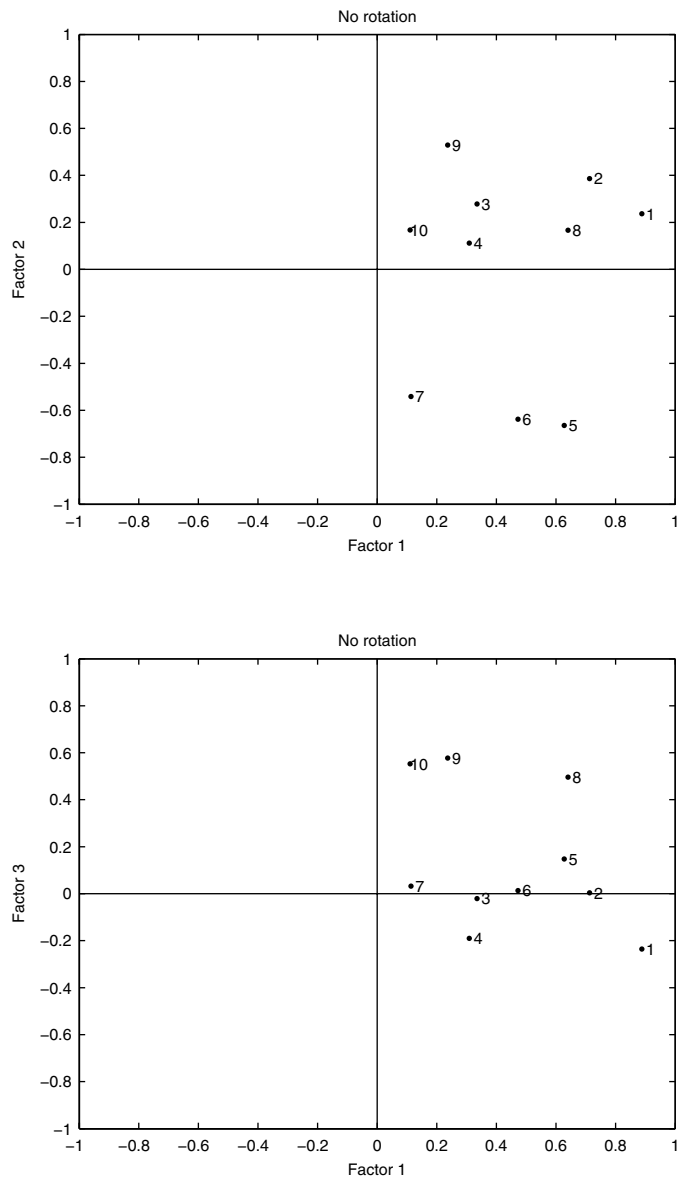


FIGURE 2.4
These show the factor loadings in their unrotated form. We see that the loadings are not grouped around the factor axes, although it is interesting to note that we have the three financial companies (points 5, 6, & 7) grouped together in the upper plot (factors 1 and 2), while the three retail companies (points 8, 9, & 10) are grouped together in the lower plot (factors 1 and 3).

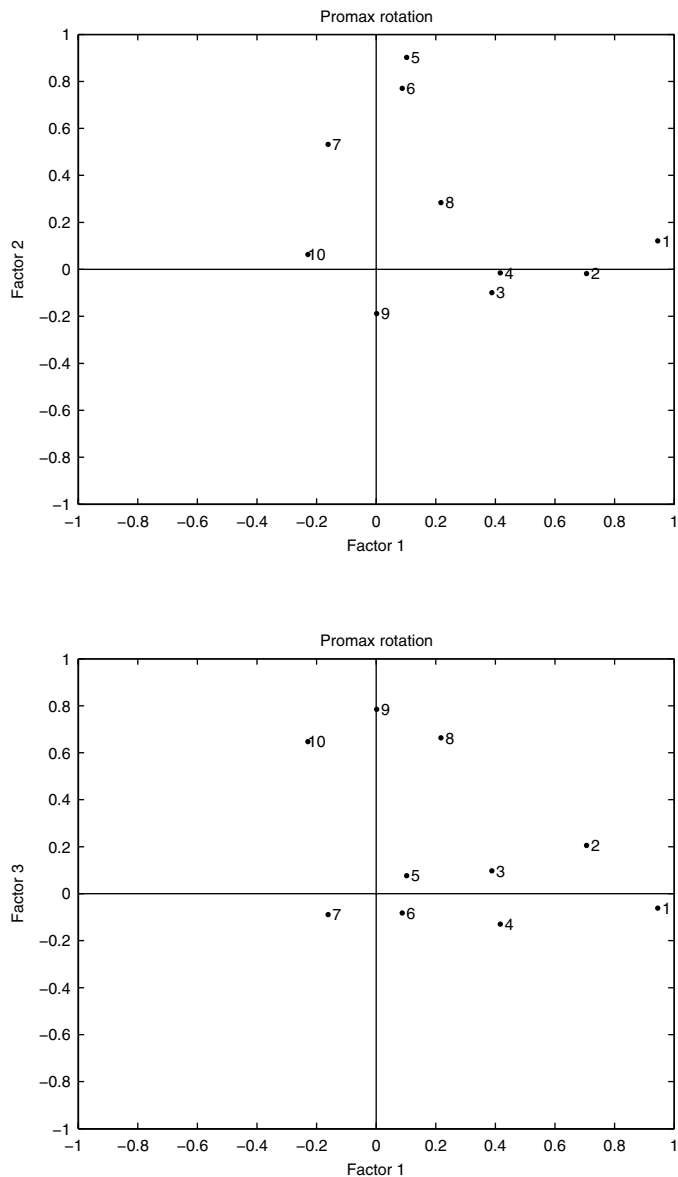


FIGURE 2.5
These plots show the factor loadings after the promax rotation. We see that the stocks can be grouped as technology companies {1, 2, 3, 4}, financial {5, 6, 7}, and retail {8, 9, 10}. The rotation makes the factors somewhat easier to interpret.

data, because they take different approaches to the problem and can uncover different facets of the data. We now outline the differences between PCA and factor analysis; a discussion of which method to use can be found in Velicer and Jackson [1990].

- Both factor analysis and PCA try to represent the structure of the data set based on the covariance or correlation matrix. Factor analysis tries to explain the off-diagonal elements, while PCA explains the variance or diagonal elements of the matrix.
- Factor analysis is typically performed using the correlation matrix, and PCA can be used with either the correlation or the covariance matrix.
- Factor analysis has a model, as given in Equation 2.8 and 2.9, but PCA does not have an explicit model associated with it (unless one is interested in inferential methods associated with the eigenvalues and PCs, in which case, distributional assumptions are made).
- If one changes the number of PCs to keep, then the existing PCs do not change. If we change the number of factors, then the entire solution changes; i.e., existing factors must be re-estimated.
- PCA has a unique solution, but factor analysis does not.
- The PC scores are found in an exact manner, but the factor scores are estimates.

2.5 Intrinsic Dimensionality

Knowing the *intrinsic dimensionality* (sometimes called *effective dimensionality*) of a data set is useful information when exploring a data set. This is defined as the smallest number of dimensions or variables needed to model the data without loss [Kirby, 2001]. Fukunaga [1990] provides a similar definition of intrinsic dimensionality as “the minimum number of parameters needed to account for the observed properties of the data.”

Several approaches to estimating the intrinsic dimensionality of a data set have been proposed in the literature. Trunk [1968, 1976] describes a statistical approach using hypothesis testing regarding the most likely local dimensionality. Fukunaga and Olsen [1971] present an algorithm where the data are divided into small subregions, and the eigenvalues of the local covariance matrix are computed for each region. The intrinsic dimensionality is then defined based on the size of the eigenvalues. Pettis, et al. [1979] develop an algorithm for estimating intrinsic dimensionality that is based on nearest neighbor information and a density estimator. We choose to

implement the original Pettis algorithm as outlined below; details of the derivation can be found in the original paper.

We first set some notation, using 1-D representation for the observations as in the original paper. However, this method works with the *distance* between observations, so it easily extends to the multi-dimensional case. Let $r_{k,x}$ represent the distance from x to the k -th nearest neighbor of x . The average k -th nearest neighbor distance is given by

$$\bar{r}_k = \frac{1}{n} \sum_{i=1}^n r_{k,x_i} . \quad (2.10)$$

Pettis, et al. [1979] show that the expected value of the average distance in Equation 2.10 is

$$E(\bar{r}_k) = \frac{1}{G_{k,d}} k^{1/d} C_n , \quad (2.11)$$

where

$$G_{k,d} = \frac{k^{1/d} \Gamma(k)}{\Gamma\left(k + \frac{1}{d}\right)} ,$$

and C_n is independent of k . If we take logarithms of Equation 2.11 and do some rearranging, then we obtain the following

$$\log(G_{k,d}) + \log E(\bar{r}_k) = (1/d) \log(k) + \log(C_n) . \quad (2.12)$$

We can get an estimate for $E(\bar{r}_k)$ by taking the observed value of $\bar{r}_k$ based on the sample, yielding the following estimator $\hat{d}$ for the intrinsic dimensionality d

$$\log(G_{k,\hat{d}}) + \log(\bar{r}_k) = (1/\hat{d}) \log(k) + \log(C_n) . \quad (2.13)$$

This is similar to a regression problem, where the slope is given by $(1/\hat{d})$. The term $\log(C_n)$ affects the intercept, not the slope, so we can disregard this in the estimation process.

The estimation procedure must be iterative since $\hat{d}$ also appears on the response side of Equation 2.13. To get started, we set the term $\log(G_{k,\hat{d}})$ equal to zero and find the slope using least squares, where the predictor values are given by $\log(k)$, and the responses are $\log(\bar{r}_k)$, for $k = 1, \dots, K$. Once we have this initial value for $\hat{d}$, we then find $\log(G_{k,\hat{d}})$ using Equation 2.13. Using this

value, we find the slope where the responses are now $\log(G_{k, \hat{d}}) + \log(\bar{r}_k)$. The algorithm continues in this manner until the estimates of intrinsic dimensionality converge.

We outline the algorithm shortly, but we need to discuss the effect of outliers first. Pettis, et al. [1979] found that their algorithm works better if potential outliers are removed before estimating the intrinsic dimensionality. To define outliers in this context, we first define a measure of the maximum average distance given by

$$m_{\max} = \frac{1}{n} \sum_{i=1}^n r_{K, x_i},$$

where r_{K, x_i} represents the i -th K nearest neighbor distance (i.e., the maximum distance). A measure of the spread is found by

$$s_{\max}^2 = \frac{1}{n-1} \sum_{i=1}^n (r_{K, x_i} - m_{\max})^2.$$

The data points x_i for which

$$r_{K, x_i} \leq m_{\max} + s_{\max} \quad (2.14)$$

are used in the nearest neighbor estimate of intrinsic dimensionality. We are now ready to describe the algorithm.

Procedure - Intrinsic Dimensionality

1. Set a value for the maximum number of nearest neighbors K .
2. Determine all of the distances r_{K, x_i} .
3. Remove outliers; i.e., keep only those points that satisfy Equation 2.14.
4. Calculate $\log(\bar{r}_k)$.
5. Get the initial estimate $\hat{d}_0$ by fitting a line to

$$\log(\bar{r}_k) = (1/\hat{d})\log(k),$$

and taking the inverse of the slope.

6. Calculate $\log(G_{k, \hat{d}_j})$ using Equation 2.13.
7. Update the estimate of intrinsic dimensionality by fitting a line to

$$\log(G_{k, \hat{d}_j}) + \log(\bar{r}_k) = (1/\hat{d})\log(k).$$

8. Repeat steps 6 and 7 until the estimates converge.

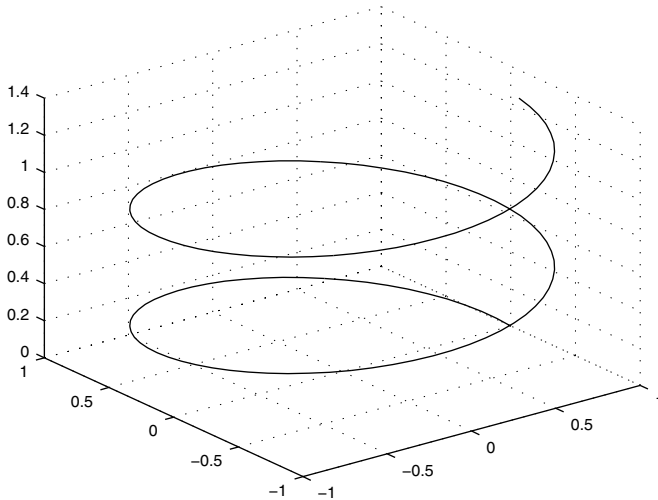


FIGURE 2.6

This shows the helix used in Example 2.5. This is a 1-D structure embedded in 3-D.

Example 2.5

The following MATLAB code implements the Pettis algorithm for estimating intrinsic dimensionality, which is also contained in the EDA Toolbox function called **idpettis**. We first generate some data to illustrate the functionality of the algorithm. The helix is described by the following equations, and points are randomly chosen along this path:

$$\begin{aligned}x &= \cos\theta \\y &= \sin\theta \\z &= 0.1\theta\end{aligned}$$

for $0 \leq \theta \leq 4\pi$. For this data set, the dimensionality is 3, but the intrinsic dimensionality is 1. We show a picture of the helix in Figure 2.6. We obtain the data by generating uniform random numbers in the interval $0 \leq \theta \leq 4\pi$.

```
% Generate the random numbers
% unifrnd is from the Statistics Toolbox.
n = 500;
```

```

theta = unifrnd(0,4*pi,1,n);
% Use in the equations for a helix.
x = cos(theta);y=sin(theta);z = 0.1*(theta);
% Put into a data matrix.
X = [x(:),y(:),z(:)];

```

We note that the function `unifrnd` is from the Statistics Toolbox, but users can also employ the `rand` function and scale to the correct interval [Martinez and Martinez, 2002]. In the interest of space and clarity, we show only the code for the algorithm after the outliers have been removed. Thus, you cannot run the code given below, as it stands. We refer curious readers to the MATLAB function `idpettis` for the implementation of the rest of the procedure.

```

% Get initial value for d. Values n, k, & logrk
% are defined in the idpettis function. This is
% an extract from that function.
logk = log(k);
[p,s] = polyfit(logk,logrk,1);
dhat = 1/p(1);
dhatold = realmax;
maxiter = 100;
epstol = 0.01;
i = 0;
while abs(dhatold - dhat) >= epstol & i < maxiter
    % Adjust the y values by adding logGRk
    logGRk = (1/dhat)*log(k)+...
             gammaln(k)-gammaln(k+1/dhat);
    [p,s] = polyfit(logk,logrk + logGRk,1);
    dhatold = dhat;
    dhat = 1/p(1);
    i = i+1;
end
idhat = dhat;

```

As an example of using the `idpettis` function, we offer the following:

```

% Get the distances using the pdist function.
% This returns the interpoint distances.
ydist = pdist(X);
idhat = idpettis(ydist,n);

```

where the input variable `X` is the helix data. The resulting estimate of the intrinsic dimensionality is 1.14. We see from this result, that in most cases the estimate will need to be rounded to the nearest integer. Thus, the estimate in this case is the correct one: 1-D.

□

2.6 Summary and Further Reading

In this chapter, we introduced the concept of dimensionality reduction by presenting several methods that find a linear mapping from a high-dimensional space to a lower one. These methods include principal component analysis, singular value decomposition, and factor analysis. We also addressed the issue of determining the intrinsic dimensionality of a data set.

For a general treatment of linear and matrix algebra, we recommend Strang [1985, 1993] or Golub and van Loan [1996]. Discussions of PCA and factor analysis can be found in most multivariate analysis books; some examples are Manly [1994] or Seber [1984]. Readers who would like a more detailed treatment of these subjects are referred to Jackson [1991] and Jolliffe [1986]. These texts provide extensive discussions (including many extensions and applications) of PCA, SVD, and factor analysis. There are many recent applications of PCA and SVD to microarray analysis, some examples include gene shaving [Hastie, et al., 2001], predicting the clinical status of human breast cancer [West, et al., 2001], obtaining a global picture of the dynamics of gene expression [Alter, Brown and Botstein, 2000], and clustering [Yeung and Ruzzo, 2001].

The Fukunaga-Olsen method of estimating intrinsic dimensionality is further explored in Verveer and Duin [1995] and Bruske and Sommer [1998]. The Pettis algorithm is also described in Fukunaga [1990] and is expanded in Verveer and Duin [1995]. Verveer and Duin [1995] revise both the Pettis and the Fukunaga-Olsen algorithms and provide an evaluation of their performance. Levina and Bickel [2004] describe a new method for estimating the intrinsic dimensionality of a data set by applying maximum likelihood to the distances between close neighbors. They derive the estimator by a Poisson process approximation. Costa and Hero [2004] propose a geometric approach based on entropic graph methods. Their method is called the geodesic minimal spanning tree (GMST), and it yields estimates of the manifold dimension and the α -entropy of the sample density on the manifold.

Exercises

- 2.1 Generate $n = 50$, $p = 3$ normally distributed random variables that have high variance in one dimension. For example, you might use the following MATLAB code to do this:

```
x1 = randn(50,1)*100;
```

```
x2 = randn(50,2);
X = [x1,x2];
```

Try PCA using both correlation and covariance matrices. Is the one with the covariance matrix very informative? Which one would be better to use in this case?

- 2.2 Do a scree plot of the data set used in Example 2.2. Generate multivariate, uncorrelated normal random variables for the same n and p . (Hint: use **randn(n,p)**.) Perform PCA for this data set, and construct its scree plot. It should look approximately like a straight line. Plot both scree plots together; where they intersect is an estimate of the number of PCs to keep. How does this compare with previous results? [Jackson, 1991, p. 46].
- 2.3 Generate a set of $n = 30$ trivariate normal random variables using **randn(30,3)**.
 - a. Subtract the mean from the observations and find the covariance matrix, using **cov**. Get the eigenvectors and eigenvalues based on the covariance matrix of the centered observations. Is the total variance of the original data equal to the sum of the eigenvalues? Verify that the eigenvectors are orthonormal. Find the PC scores and their mean.
 - b. Impose a mean of $[2, 2, 2]^T$ on the original variables. Perform PCA using the covariance of these noncentered data. Find the PC scores and their mean.
 - c. Center and scale the original variables, so that each one has zero mean and a standard deviation of one. Find the covariance of these transformed data. Find the correlation matrix of the original, non-transformed data. Are these two matrices the same?
 - d. Verify that the PC scores produce data that are uncorrelated by finding the correlation matrix of the PC scores.
- 2.4 Repeat Example 2.2 using the correlation matrix. Compare with the previous results.
- 2.5 Generate multivariate normal random variables, centered at the origin (i.e., centered about 0). Form the matrix $\mathbf{X}^T\mathbf{X}$, find the eigenvectors. Form the matrix $\mathbf{X}\mathbf{X}^T$, find the eigenvectors. Now use the SVD on $\mathbf{X}$. Compare the columns of $\mathbf{U}$ and $\mathbf{V}$ with the eigenvectors. Are they the same? Note that the columns of $\mathbf{U}$ and $\mathbf{V}$ are unique up to a sign change. What can you say about the eigenvalues and singular values?
- 2.6 Generate a set of multivariate normal random variables, centered at the origin. Obtain the eigenvalue decomposition of the covariance matrix. Multiply them to get the original matrix back, and also perform the multiplication to get the diagonal matrix $\mathbf{L}$.
- 2.7 Construct a plot based on the slopes of the lines connecting the points in the scree plot. Apply to the **yeast** data. Does this help in the analysis? Hint: use the **diff** function.

- 2.8 Apply PCA to the following data sets. What value of d would you get using the various methods for choosing the number of dimensions?
- Other gene expression data sets.
 - oronsay** data.
 - sparrow** data.
- 2.9 Repeat Example 2.3 for the SVD - LSI using $k = 2$. Comment on the results and compare to the document retrieval using $k = 3$.
- 2.10 Using the term-document matrix in Example 2.3, find the cosine of the angle between the reduced query vectors and the reduced document vectors. However, this time use the magnitude of the reduced query vectors in the loop, not the magnitudes of the original queries. Discuss how this affects your results in terms of precision and recall.
- 2.11 Generate a bivariate data set using either **rand** or **randn**. Verify that the singular values are the square roots of the eigenvalues of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$.
- 2.12 Repeat Example 2.4, using other rotations and methods implemented in MATLAB for estimating the factor loadings. Do they change the results?
- 2.13 Plot factors 2 and 3 in Example 2.4 for no rotation and promax rotation. Discuss any groupings that might be evident.
- 2.14 Try Example 2.4 with different values for the number of factors - say two and four. How do the results change?
- 2.15 The MATLAB function **factoran** includes the option of obtaining estimates of the factor scores. Using the data in Example 2.4, try the following code

```
[lam,psi,T,stats,F]=...  
    factoran(stocks,3,'rotate','promax');
```

The factor scores are contained in the variable **F**. These can be viewed using **plotmatrix**.

- 2.16 Try factor analysis on some of the gene expression data sets to cluster the patients or experiments.
- 2.17 Generate 2-D standard bivariate random variables using **randn** for increasing values of n . Use the **idpettis** function to estimate the intrinsic dimensionality, repeating for several Monte Carlo trials for each value of n . The true intrinsic dimensionality of these data is 2. How does the algorithm perform on these data sets?
- 2.18 This data set is taken from Fukunaga [1990], where he describes a Gaussian pulse characterized by three parameters **a**, **m**, and σ . The waveform is given by

$$\mathbf{x}(t) = \mathbf{a} \exp\left(-\frac{(t-\mathbf{m})^2}{2\sigma^2}\right).$$

These parameters are generated randomly in the following ranges:

$$0.7 \leq \mathbf{a} \leq 1.3$$

$$0.3 \leq \mathbf{m} \leq 0.7$$

$$0.2 \leq \sigma \leq 0.4.$$

The signals are generated at 8 time steps in the range $0 \leq t \leq 1.05$, so each of the signals is an 8-D random vector. The intrinsic dimensionality of these data is 3. Generate random vectors for various values of n , and apply **idpettis** to estimate the intrinsic dimensionality and assess the results.

- 2.19 Estimate the intrinsic dimensionality of the **yeast** data. Does this agree with the results in Example 2.2?
- 2.20 Estimate the intrinsic dimensionality of the following data sets. Where possible, compare with the results from PCA.
 - a. All BPM data sets.
 - b. **oronsay** data.
 - c. **sparrow** data
 - d. Other gene expression data.
- 2.21 The Statistics Toolbox, Version 5, has a new function called **rotatefactors** that can be used with either factor analysis or principal component analysis loadings. Do a **help** on this function. Apply it to the factor loadings from the **stockreturns** data. Apply it to the PCA results of the **yeast** data in Example 2.2.