Chapter 3

Dimensionality Reduction - Nonlinear Methods

This chapter covers various methods for nonlinear dimensionality reduction, where the nonlinear aspect refers to the mapping between the high-dimensional space and the low-dimensional space. We start off by discussing a method that has been around for many years called multidimensional scaling. We follow this with several recently developed nonlinear dimensionality reduction techniques called locally linear embedding, isometric feature mapping, and Hessian eigenmaps. We conclude by discussing two related methods from the machine learning community called self-organizing maps and generative topographic maps.

3.1 Multidimensional Scaling - MDS

In general, ***multidimensional scaling*** (MDS) is a set of techniques for the analysis of proximity data measured on a set of objects in order to reveal hidden structure. The purpose of MDS is to find a configuration of the data points in a low-dimensional space such that the proximity between objects in the full-dimensional space is represented with some degree of fidelity by the distances between points in the low-dimensional space. This means that observations that are close together in a high-dimensional space should be close in the low-dimensional space. Many aspects of MDS were originally developed by researchers in the social science community, and the method is now widely available in most statistical packages, including the MATLAB Statistics Toolbox.

We first provide some definitions and notation before we go on to describe the various categories of MDS [Cox and Cox, 2001]. As before, we assume that we have a data set with n observations. In general, MDS starts with measures of proximity that quantify how close objects are to one another or how similar they are. They can be of two types: those that indicate similarity or dissimilarity. A measure of dissimilarity between objects r and s is denoted

by δ_{rs}, and the similarity is denoted by s_{rs}. For most definitions of these proximity measures, we have

$$\delta_{rs} \geq 0 \qquad \delta_{rr} = 0$$

and

$$0 \leq s_{rs} \leq 1 \qquad s_{rr} = 1 .$$

Thus, we see that for a dissimilarity measure δ_{rs}, small values correspond to observations that are close together, while the opposite is true for similarity measures s_{rs}. These two main types of proximity measures are easily converted into the other one when necessary (see Appendix A for more information on this issue). Thus, for the rest of this chapter, we will assume that *proximity* measures *dissimilarity* between objects. We also assume that the dissimilarities are arranged in matrix form, which will be denoted by Δ. In many cases, this will be a symmetric $n \times n$ matrix (sometimes given in either lower or upper triangular form).

In the lower-dimensional space, we denote the *distance* between observation r and s by d_{rs}. It should also be noted that in the MDS literature the matrix of coordinate values in the *lower-dimensional space* is denoted by $\mathbf{X}$. We follow that convention here, knowing that it might be rather confusing with our prior use of $\mathbf{X}$ as representing the original set of n p-dimensional observations.

In MDS, one often starts with or might only have the dissimilarities Δ, not the original observations. In fact, in the initial formulations of MDS, the experiments typically involved qualitative judgements about differences between subjects, and p-dimensional observations do not make sense in that context. So, to summarize, with MDS we start with dissimilarities Δ and end up with d-dimensional[1] transformed observations $\mathbf{X}$. Usually, $d = 2$ or $d = 3$ is chosen to allow the analyst to view and explore the results for interesting structure, but any $d < p$ is also appropriate.

There are many different techniques and flavors of MDS, most of which fall into two main categories: metric MDS and nonmetric MDS. The main characteristic that divides them arises from the different assumptions of how the dissimilarities δ_{rs} are transformed into the configuration of distances d_{rs} [Cox and Cox, 2001]. ***Metric MDS*** assumes that the dissimilarities δ_{rs} calculated from the p-dimensional data and distances d_{rs} in a lower-dimensional space are related as follows

$$d_{rs} \approx f(\delta_{rs}) , \tag{3.1}$$

[1] We realize that the use of the notation d as both the lower dimensionality of the data ($d < p$) and the distance d_{rs} between points in the configuration space might be confusing. However, the meaning should be clear from the context.

where f is a continuous monotonic function. The form of $f(\bullet)$ specifies the MDS model. For example, we might use the formula

$$f(\delta_{rs}) = b\delta_{rs} \ . \tag{3.2}$$

Mappings that follow Equation 3.2 are called ***ratio*** MDS [Borg and Groenen, 1997]. Another common choice is ***interval*** MDS, with $f(\bullet)$ given by

$$f(\delta_{rs}) = a + b\delta_{rs} \ ,$$

where a and b are free parameters. Other possibilities include higher degree polynomials, logarithmic, and exponential functions.

Nonmetric MDS relaxes the metric properties of $f(\bullet)$ and stipulates only that the rank order of the dissimilarities be preserved. The transformation or scaling function must obey the monotonicity constraint:

$$\delta_{rs} < \delta_{ab} \Rightarrow f(\delta_{rs}) \leq f(\delta_{ab}) \ ,$$

for all objects. Because of this, nonmetric MDS is also known as ***ordinal MDS***.

3.1.1 Metric MDS

Most of the methods in metric MDS start with the fact that we seek a transformation that satisfies Equation 3.1. We can tackle this problem by defining an objective function and then using a method that will optimize it. One way to define the objective function is to use the squared discrepancies between d_{rs} and $f(\delta_{rs})$ as follows

$$\sqrt{\frac{\left(\sum_r \sum_s (f(\delta_{rs}) - d_{rs})^2\right)}{\text{scale factor}}} \ . \tag{3.3}$$

In general, Equation 3.3 is called the ***stress***; different forms for the scale factor give rise to different forms of stress and types of MDS. The scale factor used most often is

$$\sum_r \sum_s d_{rs}^2 ,$$

in which case, we have an expression called ***Stress-1*** [Kruskal, 1964a]. The summation is taken over all dissimilarities, skipping those that are missing.

As we stated previously, sometimes the dissimilarities are symmetric, in which case, we only need to sum over $1 = r < s = n$.

Thus, in MDS, we would scale the dissimilarities using $f(\bullet)$ and then find a configuration of points in a d-dimensional space such that when we calculate the distances between them the stress is minimized. This can now be solved using numerical methods and operations research techniques (e.g., gradient or steepest descent). These methods are usually iterative and are not guaranteed to find a global solution to the optimization problem. We will expand on some of the details later, but first we describe a case where a closed form solution is possible.

The phrase *metric multidimensional scaling* is often used in the literature to refer to a specific technique called ***classical*** MDS. However, metric MDS includes more than this one technique, such as least squares scaling and others [Cox and Cox, 2001]. For metric MDS approaches, we first describe classical MDS followed by a method that optimizes a loss function using a majorization technique.

Classical MDS

If the proximity measure in the original space and the distance are taken to be Euclidean, then a closed form solution exists to find the configuration of points in a d-dimensional space. This is the classical MDS approach. The function $f(\bullet)$ relating dissimilarities and distances is the identity function, so we seek a mapping such that

$$d_{rs} = \delta_{rs} .$$

This technique originated with Young and Householder [1938], Torgerson [1952], and Gower [1966]. Gower was the one that showed the importance of classical scaling, and he gave it the name ***principal coordinates analysis***, because it uses ideas similar to those in PCA. Principal coordinate analysis and classical scaling are the same thing, and they have become synonymous with metric scaling.

We now describe the steps of the method only, without going into the derivation. Please see any of the following for the derivation of classical MDS: Cox and Cox [2001], Borg and Groenen [1997], or Seber [1984].

Procedure - Classical MDS

1. Using the matrix of dissimilarities, Δ, find matrix $\mathbf{Q}$, where each element of $\mathbf{Q}$ is given by

$$q_{rs} = -\frac{1}{2}\delta_{rs}^2 .$$

2. Find the centering matrix $\mathbf{H}$ using

$$\mathbf{H} = \mathbf{I} - n^{-1}\mathbf{1}\mathbf{1}^T,$$

where $\mathbf{I}$ is the $n \times n$ identity matrix, and $\mathbf{1}$ is a vector of n ones.

3. Find the matrix $\mathbf{B}$, as follows

$$\mathbf{B} = \mathbf{HQH}.$$

4. Determine the eigenvectors and eigenvalues of $\mathbf{B}$:

$$\mathbf{B} = \mathbf{ALA}^T.$$

5. The coordinates in the lower-dimensional space are given by

$$\mathbf{X} = \mathbf{A}_d\mathbf{L}_d^{1/2},$$

where $\mathbf{A}_d$ contains the eigenvectors corresponding to the d largest eigenvalues, and $\mathbf{L}_d^{1/2}$ contains the square root of the d largest eigenvalues along the diagonal.

We use similar ideas from PCA to determine the dimensionality d to use in Step 5 of the algorithm. Although, $d = 2$ is often used in MDS, since the data can then be represented in a scatterplot.

For some data sets, the matrix $\mathbf{B}$ might not be positive semi-definite, in which case some of the eigenvalues will be negative. One could ignore the negative eigenvalues and proceed to step 5 or add an appropriate constant to the dissimilarities to make $\mathbf{B}$ positive semi-definite. We do not address this second option here, but the reader is referred to Cox and Cox [2001] for more details. If the dissimilarities are in fact Euclidean distances, then this problem does not arise.

Since this uses the decomposition of a square matrix, some of the properties and issues discussed about PCA are applicable here. For example, the lower-dimensional representations are nested. The first two dimensions of the 3-D coordinate representation are the same as the 2-D solution. It is also interesting to note that PCA and classical MDS provide equivalent results when the dissimilarities are Euclidean distances [Cox and Cox, 2001].

Example 3.1

For this example, we use the BPM data described in Chapter 1. Recall that one of the applications for these data is to discover different topics or sub-topics. For ease of discussion and presentation, we will look at only two of the topics in this example: the comet falling into Jupiter (topic 6) and DNA in the O. J.

Simpson trial (topic 9). First, using the match distances, we extract the required observations from the full interpoint distance matrix.

```
% First load the data - use the 'match' interpoint
% distance matrix.
load matchbpm
% Now get the data for topics 9 and 6.
% Find the indices where they are not equal to 6 or 9.
indlab = find(classlab ~= 6 & classlab ~=9);
% Now get rid of these from the distance matrix.
matchbpm(indlab,:) = [];
matchbpm(:,indlab) = [];
classlab(indlab) = [];
```

The following piece of code shows how to implement the steps for classical MDS in MATLAB.

```
% Now implement the steps for classical MDS
% Find the matrix Q:
Q = -0.5*matchbpm.^2;
% Find the centering matrix H:
n = 73;
H = eye(n) - ones(n)/n;
% Find the matrix B:
B = H*Q*H;
% Get the eigenvalues of B.
[A,L] = eig(B);
% Find the ordering largest to smallest.
[vals, inds] = sort(diag(L));
inds = flipud(inds);
vals = flipud(vals);
% Re-sort based on these.
A = A(:,inds);
L = diag(vals);
```

We are going to plot these results using $d = 2$ for ease of visualization, but we can also construct a scree-type plot to look for the 'elbow' in the curve. As in PCA, this can help us determine a good value to use. The code for constructing this plot and finding the coordinates in a 2-D space is given next.

```
% First plot a scree-type plot to look for the elbow.
% The following uses a log scale on the y-axis.
semilogy(vals(1:10),'o')
% Using 2-D for visualization purposes,
% find the coordinates in the lower-dimensional space.
X = A(:,1:2)*diag(sqrt(vals(1:2)));
% Now plot in a 2-D scatterplot
ind6 = find(classlab == 6);
```

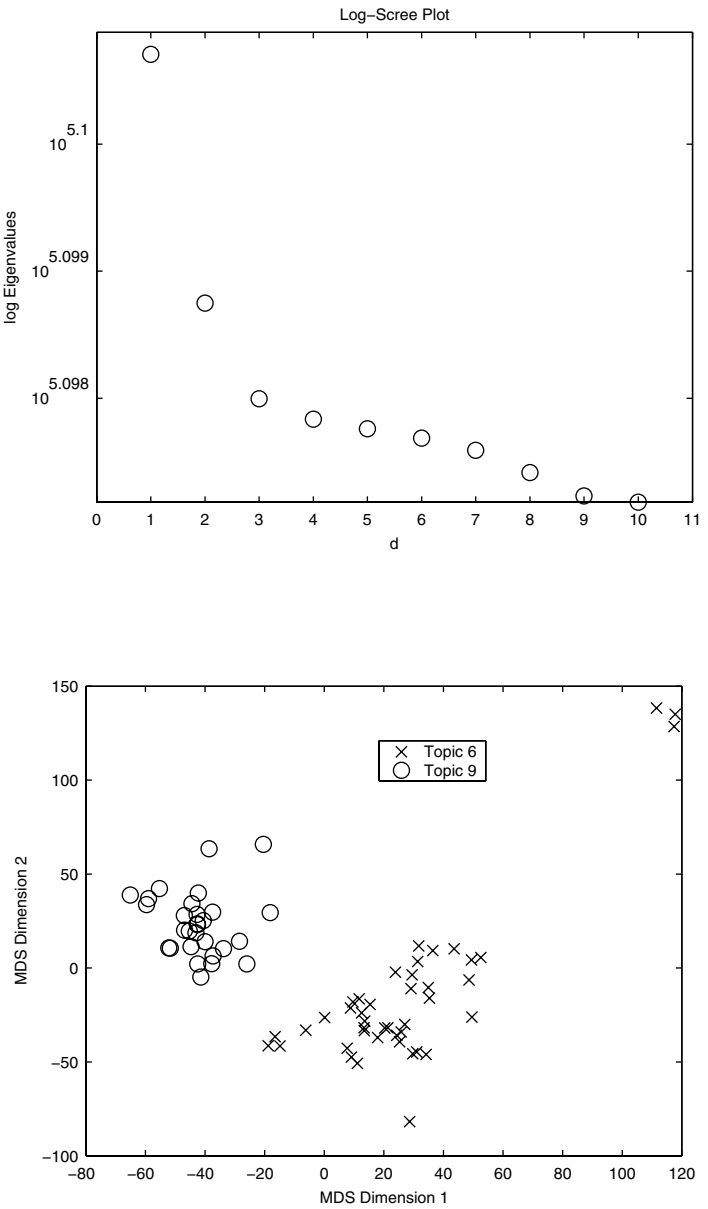

FIGURE 3.1
The top plot shows the logarithm of the eigenvalues. The bottom plot is a scatterplot of the 2-D coordinates after classical MDS. Note the good separation between the two topics, as well as the appearance of sub-topics for topic 6.

```
ind9 = find(classlab == 9);
plot(X(ind6,1),X(ind6,2),'x',X(ind9,1),X(ind9,2),'o')
legend({'Topic 6';'Topic 9'})
```

The scree plot is shown in Figure 3.1 (top), where we see that the elbow looks to be around $d = 3$. The scatterplot of the 2-D coordinates is given in Figure 3.1 (bottom). We see clear separation between topics 6 and 9. However, it is also interesting to note that there seems to be several sub-topics in topic 6. ❑

MATLAB provides a function called **cmdscale** for constructing lower-dimensional coordinates using classical MDS. This is available in the Statistics Toolbox.

Metric MDS - SMACOF

The general idea of the ***majorization method*** for optimizing a function is as follows. Looking only at the one-dimensional case, we want to minimize a complicated function $f(x)$ by using a function $g(x, y)$ that is easily minimized. The function g has to satisfy

$$f(x) \leq g(x, y),$$

for a given y such that

$$f(y) = g(y, y).$$

Looking at graphs of these function one would see that the function g is always above f, and they coincide at the point $x = y$. The method of minimizing f is iterative. We start with an initial value x_0, and we minimize $g(x, x_0)$ to get x_1. We then minimize $g(x, x_1)$ to get the next value, continuing in this manner until convergence. Upon convergence of the algorithm, we will also have minimized $f(x)$.

The SMACOF (Scaling by Majorizing a Complicated Function) method goes back to de Leeuw [1977], and others have refined it since then, including Groenen [1993]. The method is simple, and it can be used for both metric and nonmetric applications. We now follow the notation of Borg and Groenen [1997] to describe the algorithm for the metric case. They show that the SMACOF algorithm satisfies the requirements for minimizing a function using majorization, as described above. We leave out the derivation, but those who are interested in this and in the nonmetric case can refer to the above references as well as Cox and Cox [2001].

The ***raw stress*** is written as

$$\begin{aligned}\sigma(\mathbf{X}) &= \sum_{r<s} w_{rs}(d_{rs}(\mathbf{X}) - \delta_{rs})^2 \\ &= \sum_{r<s} w_{rs}\delta_{rs}^2 + \sum_{r<s} w_{rs}d_{rs}^2(\mathbf{X}) - 2\sum_{r<s} w_{rs}\delta_{rs}d_{rs}(\mathbf{X}),\end{aligned}$$

for all available dissimilarities δ_{rs}. The inequality $r < s$ in the summation means that we only sum over half of the data, since we are assuming that the dissimilarities and distances are symmetric. We might have some missing values, so we define a weight w_{rs} with a value of 1 if the dissimilarity is present and a value of 0 if it is missing. The notation $d_{rs}(\mathbf{X})$ makes it explicit that the distances are a function of $\mathbf{X}$ (d-dimensional transformed observations), and that we are looking for a configuration $\mathbf{X}$ that minimizes stress.

Before we describe the algorithm, we need to present some relationships and notation. Let $\mathbf{Z}$ be a possible configuration of points. The matrix $\mathbf{V}$ has elements given by the following

$$v_{ij} = -w_{ij}, \qquad i \neq j$$

and

$$v_{ii} = \sum_{j=1,\ i\neq j}^{n} w_{ij}.$$

This matrix is not of full rank, and we will need the inverse in one of the update steps. So we turn to the ***Moore-Penrose inverse***, which will be denoted by $\mathbf{V}^+$. [2]

We next define matrix $\mathbf{B}(\mathbf{Z})$ with elements

$$b_{ij} = \begin{cases} -\dfrac{w_{ij}\delta_{ij}}{d_{ij}(\mathbf{Z})} & d_{ij}(\mathbf{Z}) \neq 0 \\ 0 & d_{ij}(\mathbf{Z}) = 0 \end{cases}, \quad i \neq j$$

and

$$b_{ii} = -\sum_{j=1,\ j\neq i}^{n} b_{ij}.$$

[2] The Moore-Penrose inverse is also called the ***pseudoinverse*** and can be computed using the singular value decomposition. MATLAB has a function called **`pinv`** that provides this inverse.

We are now ready to define the ***Guttman transform***. The general form of the transform is given by

$$\mathbf{X}^k = \mathbf{V}^+\mathbf{B}(\mathbf{Z})\mathbf{Z},$$

where the k represents the iteration number in the algorithm. If all of the weights are one (none of the dissimilarities are missing), then the transform is much simpler:

$$\mathbf{X}^k = n^{-1}\mathbf{B}(\mathbf{Z})\mathbf{Z}.$$

The SMACOF algorithm is outlined below.

SMACOF Algorithm

1. Find an initial configuration of points in R^d. This can either be random or nonrandom (i.e., some regular grid). Call this $\mathbf{X}^0$.
2. Set $\mathbf{Z} = \mathbf{X}^0$ and the counter to $k = 0$.
3. Compute the raw stress $\sigma(\mathbf{X}^0)$.
4. Increase the counter by 1: $k = k + 1$.
5. Obtain the Guttman transform $\mathbf{X}^k$.
6. Compute the stress for this iteration, $\sigma(\mathbf{X}^k)$.
7. Find the difference in the stress values between the two iterations. If this is less than some pre-specified tolerance or if the maximum number of iterations has been reached, then stop.
8. Set $\mathbf{Z} = \mathbf{X}^k$, and go to step 4.

We illustrate this algorithm in the next example.

Example 3.2

We turn to a different data set for this example and look at the **leukemia** data. Recall that we can look at either genes or patients as our observations; in this example we will be looking at the genes. To make things easier, we only implement the case where all the weights are 1, but the more general situation is easily implemented using the above description. First we load the data and get the distances.

```
% Use the Leukemia data, using the genes (columns)
% as the observations.
load leukemia
y = leukemia';
% Get the interpoint distance matrix.
% pdist gets the interpoint distances.
```

```
% squareform converts them to a square matrix.
D = squareform(pdist(y,'seuclidean'));
[n,p] = size(D);
% Turn off this warning... :
warning off MATLAB:divideByZero
```

Next we get the required initial configuration and the stress associated with it.

```
% Get the first term of stress.
% This is fixed - does not depend on the configuration.
stress1 = sum(sum(D.^2))/2;
% Now find an initial random configuration.
d = 2;
% Part of Statistics Toolbox
Z = unifrnd(-2,2,n,d);
% Find the stress for this.
DZ = squareform(pdist(Z));
stress2 = sum(sum(DZ.^2))/2;
stress3 = sum(sum(D.*DZ));
oldstress = stress1 + stress2 - stress3;
```

Now we iteratively adjust the configuration until the stress converges.

```
% Iterate until stress converges.
tol = 10^(-6);
dstress = realmax;
numiter = 1;
dstress = oldstress;
while dstress > tol & numiter <= 100000
    numiter = numiter + 1;
    % Now get the update.
    BZ = -D./DZ;
    for i = 1:n
        BZ(i,i) = 0;
        BZ(i,i) = -sum(BZ(:,i));
    end
    X = n^(-1)*BZ*Z;
    Z = X;
    % Now get the distances.
    % Find the stress.
    DZ = squareform(pdist(Z));
    stress2 = sum(sum(DZ.^2))/2;
    stress3 = sum(sum(D.*DZ));
    newstress = stress1 + stress2 - stress3;
    dstress = oldstress - newstress;
    oldstress = newstress;
end
```

A scatterplot of the resulting configuration is shown in Figure 3.2, with two different ways to classify the data. We see that some clustering is visible.
❑

3.1.2 Nonmetric MDS

An algorithm for solving the nonmetric MDS problem was first discussed by Shepard [1962a, 1962b]. However, he did not introduce the idea of using a loss function. That came with Kruskal [1964a, 1964b] who expanded the ideas of Shepard and gave us the concept of minimizing a loss function called *stress*.

Not surprisingly, we first introduce some more notation and terminology for nonmetric MDS. The ***disparity*** is a measure of how well the distance d_{rs} matches the dissimilarity δ_{rs}. We represent the disparity as $\hat{d}_{rs}$. The r-th point in our configuration $\mathbf{X}$ will have coordinates

$$\mathbf{x}_r = (x_{r1}, \ldots, x_{rd})^T .$$

We will use the Minkowski dissimilarity to measure the distance between points in our d-dimensional space. It is defined as

$$d_{rs} = \left[\sum_{i=1}^{d} |x_{ri} - x_{si}|^{\lambda} \right]^{1/\lambda} ,$$

where $\lambda > 0$. See Appendix A for more information on this distance and the parameter λ.

We can view the disparities as a function of the distance, such as

$$\hat{d}_{rs} = f(d_{rs}) ,$$

where

$$\delta_{rs} < \delta_{ab} \Rightarrow \hat{d}_{rs} \leq \hat{d}_{ab} .$$

Thus, the order of the original dissimilarities is preserved by the disparities. Note that this condition allows for possible ties in the disparities.

We define a loss function L, which is really stress, as follows

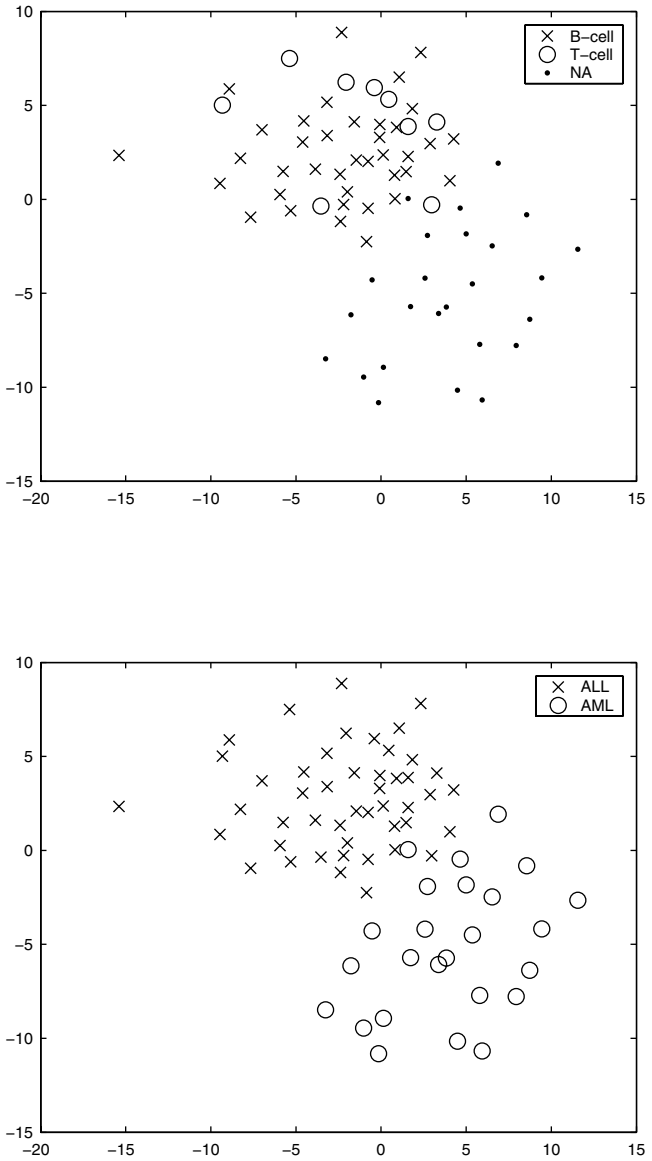

FIGURE 3.2
Here we show the results of using the SMACOF algorithm on the **leukemia** data set. The top panel shows the data transformed to 2-D and labeled by B-cell and T-cell. The lower panel illustrates the same data using different symbols based on the ALL or AML labels. There is some indication of being able to group the genes in a reasonable fashion.

$$L = S = \sqrt{\frac{\sum_{r<s} (d_{rs} - \hat{d}_{rs})^2}{\sum_{r<s} d_{rs}^2}} = \sqrt{\frac{S^*}{T^*}} .$$

It could be the case that we have missing dissimilarities or the values might be meaningless for some reason. If the analyst is faced with this situation, then the summations in the definition of stress are restricted to those pairs (r,s) for which the dissimilarity is available.

As with other forms of MDS, we seek a configuration of points $\mathbf{X}$, such that the stress is minimized. Note that the coordinates of the configuration enter into the loss function through the distances d_{rs}. The original dissimilarities enter into the stress by imposing an ordering on the disparities. Thus, the stress is minimized subject to the constraint on the disparities. This constraint is satisfied by using ***isotonic regression*** (also known as ***monotone regression***)[3] to obtain the disparities.

We now pause to describe the isotonic regression procedure. This was first described in Kruskal [1964b], where he developed an up-and-down-blocks algorithm. A nice explanation of this is given in Borg and Groenen [1997], as well as in the original paper by Kruskal. We choose to describe and implement the method for isotonic regression outlined in Cox and Cox [2001]. Isotonic regression of the d_{rs} on the δ_{rs} partitions the dissimilarities into blocks over which the $\hat{d}_{rs}$ are constant. The estimated disparities $\hat{d}_{rs}$ are given by the mean of the d_{rs} values within the block.

Example 3.3

The easiest way to explain (and hopefully to understand) isotonic regression is through an example. We use a simple one given in Cox and Cox [2001]. There are four objects with the following dissimilarities

$$\delta_{12} = 2.1,\ \delta_{13} = 3.0,\ \delta_{14} = 2.4,\ \delta_{23} = 1.7,\ \delta_{24} = 3.9,\ \delta_{34} = 3.2 .$$

A configuration of points yields the following distances

$$d_{12} = 1.6,\ d_{13} = 4.5,\ d_{14} = 5.7,\ d_{23} = 3.3,\ d_{24} = 4.3,\ d_{34} = 1.3 .$$

We now order the dissimilarities from smallest to largest and use a single subscript to denote the rank. We also impose the same order (and subscript) on the corresponding distances. This yields

[3] This is also known as *monotonic least squares regression*.

$$\delta_1 = 1.7,\ \delta_2 = 2.1,\ \delta_3 = 2.4,\ \delta_4 = 3.0,\ \delta_5 = 3.2,\ \delta_6 = 3.9$$
$$d_1 = 3.3,\ d_2 = 1.6,\ d_3 = 5.7,\ d_4 = 4.5,\ d_5 = 1.3,\ d_6 = 4.3.$$

The constraint on the disparities requires these distances to be ordered such that $d_i < d_{i+1}$. If this is what we have from the start, then we need not adjust things further. Since that is not true here, we must use isotonic regression to get $\hat{d}_{rs}$. To do this, we first get the cumulative sums of the distances d_i defined as

$$D_i = \sum_{j=1}^{i} d_j, \qquad i = 1, \ \ldots, \ N,$$

where N is the total number of dissimilarities available. In essence, the algorithm finds the greatest convex minorant of the graph of D_i, going through the origin. See Figure 3.3 for an illustration of this. We can think of this procedure as taking a string, attaching it to the origin on one end and the last point on the other end. The points on the greatest convex minorant are those where the string touches the points D_i. These points partition the distances d_i into blocks, over which we will have disparities of constant value. These disparities are the average of the distances that fall into the block. Cox and Cox [2001] give a proof that this method yields the required isotonic regression. We now provide some MATLAB code that illustrates these ideas.

```
% Enter the original data.
dissim = [2.1 3 2.4 1.7 3.9 3.2];
dists = [1.6 4.5 5.7 3.3 4.3 1.3];
N = length(dissim);
% Now re-order the dissimilarities.
[dissim,ind] = sort(dissim);
% Now impose the same order on the distances.
dists = dists(ind);
% Now find the cumulative sums of the distances.
D = cumsum(dists);
% Add the origin as the first point.
D = [0 D];
```

It turns out that we can find the greatest convex minorant by finding the slope of each D_i with respect to the origin. We first find the smallest slope, which defines the first partition (i.e., it is on the greatest convex minorant). We then find the next smallest slope, after removing the first partition from further consideration. We continue in this manner until we reach the end of the points. The following code implements this process.

```
% Now find the slope of these.
slope = D(2:end)./(1:N);
```

```
% Find the points on the convex minorant by looking
% for smallest slopes.
i = 1;
k = 1;
while i <= N
    val = min(slope(i:N));
    minpt(k) = find(slope == val);
    i = minpt(k) + 1;
    k = k + 1;
end
```

It turns out that this procedure yields extra points, ones that are *not* on the convex minorant. MATLAB has a function called **convhull** that finds all of the points that are on the convex hull[4] of a set of points. This also yields extra points because we only want those points that are on the 'bottom' of the convex hull. To get the desired points, we take the intersection of the two sets.[5]

```
K = convhull(D, 0:N);
minpt = intersect(minpt+1,K) - 1;
```

Now that we have the points that divide the distances into blocks, we find the disparities, which are given by the average distance in that block.

```
% Now that we have all of the minorant points
% that divide into blocks, the disparities are
% the averages of the distances over those blocks.
j = 1;
for i = 1:length(minpt)
    dispars(j:minpt(i)) = mean(dists(j:minpt(i)));
    j = minpt(i) + 1;
end
```

The disparities are given by

$$\hat{d}_1 = 2.45,\ \hat{d}_2 = 2.45,\ \hat{d}_3 = 3.83,\ \hat{d}_4 = 3.83,\ \hat{d}_5 = 3.83,\ \hat{d}_6 = 4.3\,.$$

The graphs that illustrate these concepts are shown in Figure 3.3, where we see that the disparities fall into three groups, as given above.
❑

Now that we know how to do isotonic regression, we can describe Kruskal's algorithm for nonmetric MDS. This is outlined below.

[4] The convex hull of a data set is the smallest convex region that contains the data set.
[5] Please see the M-file for an alternative approach, courtesy of Tom Lane of The MathWorks.

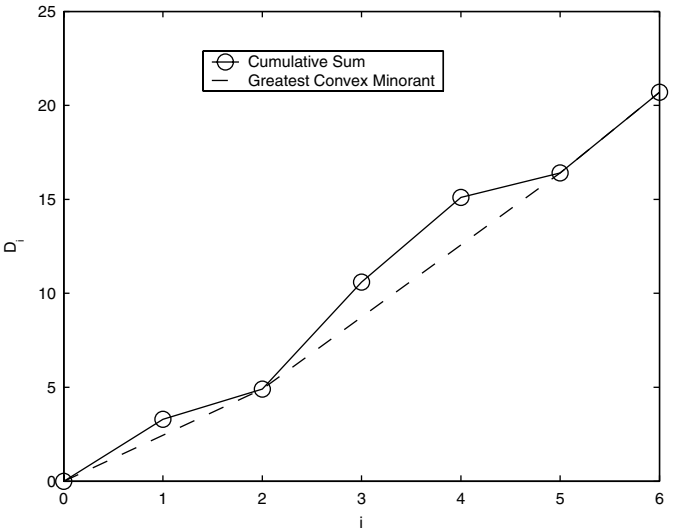

FIGURE 3.3
This shows the idea behind isotonic regression using the greatest convex minorant. The greatest convex minorant is given by the dashed line. The places where it touches the graph of the cumulative sums of the distances partition the distances into blocks. Each of the disparities in the blocks are given by the average of the distances falling in that partition.

Procedure - Kruskal's Algorithm

1. Choose an initial configuration $\mathbf{X}_0$ for a given dimensionality d. Set the iteration counter k equal to 0. Set the step size α to some desired starting value.
2. Normalize the configuration to have mean of zero (i.e., centroid is at the origin) and a mean square distance from the origin equal to 1.
3. Compute the interpoint distances for this configuration.
4. Check for equal dissimilarities. If some are equal, then order those such that the corresponding distances form an increasing sequence within that block.
5. Find the disparities $\hat{d}_{rs}$ using the current configuration of points and isotonic regression.
6. Append all of the coordinate points into one vector such that

$$\mathbf{x} = (x_{11}, \ldots, x_{1d}, \ldots, x_{n1}, \ldots, x_{nd})^T.$$

7. Compute the gradient for the k-th iteration given by

$$\frac{\partial S}{\partial x_{ui}} = g_{ui} = S \sum_{r<s} (\delta^{ur} - \delta^{us}) \left[\frac{d_{rs} - \hat{d}_{rs}}{S^*} - \frac{d_{rs}}{T^*} \right] \frac{|x_{ri} - x_{si}|^{\lambda - 1}}{d_{rs}^{\lambda - 1}} \operatorname{sgn}(x_{ri} - x_{si}),$$

where δ^{ur} represents the Kronecker delta function, *not* dissimilarities. This function is defined as follows

$$\delta^{ur} = 1 \qquad \text{if } u = r$$
$$\delta^{ur} = 0 \qquad \text{if } u \neq r$$

The value of the sgn(•) function is +1, if the argument is positive and –1 if it is negative.

8. Check for convergence. Determine the magnitude of the gradient vector for the current configuration using

$$\text{mag}(\mathbf{g}_k) = \sqrt{\frac{1}{n} \sum_{u,\ i} g_{k_{ui}}^2}.$$

If this magnitude is less than some small value ε or if some maximum allowed number of iterations has been reached, then the process stops.

9. Find the step size α

$$\alpha_k = \alpha_{k-1} \times \text{angle factor} \times \text{relaxation factor} \times \text{good luck factor},$$

where k represents the iteration number and

$$\text{angle factor} = 4^{(\cos\theta)^3} \text{ with } \left(\cos\theta = \frac{\sum_{r<s} g_{k_{rs}}\, g_{k-1_{rs}}}{\sqrt{\sum_{r<s} g_{k_{rs}}^2} \sqrt{\sum_{r<s} g_{k-1_{rs}}^2}} \right)$$

$$\text{relaxation factor} = \frac{1.3}{1 + \beta^5} \text{ with } \left(\beta = \min\left[1,\ \frac{S_k}{S_{k-5}}\right] \right)$$

$$\text{good luck factor} = \min\left[1,\ \frac{S_k}{S_{k-1}}\right].$$

10. The new configuration is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \frac{\mathbf{g}_k}{\text{mag}(\mathbf{g}_k)}.$$

11. Increment the counter: $k = k + 1$. Go to step 2.

A couple of programming issues should be noted. When we are in the beginning of the iterative algorithm, we will not have values of the stress for $k - 5$ or $k - 1$ (needed in Step 9). In these cases, we will simply use the value at the first iteration until we have enough. We can use a similar idea for the gradient. Kruskal [1964b] states that this step size provides large steps during the initial iterations and small steps at the end. There is no claim of optimality with this procedure, and the reader should be aware that the final configuration is not guaranteed to be a global solution. It is likely that the configuration will be a local minimum for stress, which is typically the case for greedy iterative algorithms. It is recommended that several different starting configurations be used, and the one with the minimum stress be accepted as the final answer.

This leads to another programming point. How do we find an initial configuration to start the process? We could start with a grid of points that are evenly laid out over a d-dimensional space. Another possibility is to start from the configuration given by classical MDS. Finally, one could also generate random numbers according to a Poisson process in R^d.

Another issue that must be addressed with nonmetric MDS is how to handle ties in the dissimilarities. There are two approaches to this. The primary approach states that if $\delta_{rs} = \delta_{tu}$, then $\hat{d}_{rs}$ is not required to be equal to $\hat{d}_{tu}$. The secondary and more restrictive approach requires the disparities to be equal when dissimilarities are tied. We used the primary approach in the above procedure and in our MATLAB implementation.

Example 3.4

The **nmmds** function (included in the EDA Toolbox) that implements Kruskal's nonmetric MDS is quite long, and it involves several helper functions. So, we just show how it would be used rather than repeating all of the code here. We return to our BPM data, but this time we will look at two different topics: topic 8 (the death of the North Korean leader) and topic 11 (Hall's helicopter crash in North Korea). Since these are both about North Korea, we would expect some similarity between them. Previous experiments showed that the documents from these two topics were always grouped together, but in several sub-groups [Martinez, 2002]. We apply the nonmetric MDS method to these data using the Ochiai measure of semantic dissimilarity.

```
load ochiaibpm
% First get out the data for topics 8 and 11.
% Find the indices where they are not equal to 8 or 11.
indlab = find(classlab ~= 8 & classlab ~= 11);
% Now get rid of these from the distance matrix.
ochiaibpm(indlab,:) = [];
ochiaibpm(:,indlab)=[];
```

```
classlab(indlab) = [];
% We only need the upper part.
n = length(classlab);
dissim = [];
for i = 1:n
    dissim = [dissim, ochiaibpm(i,(i+1):n)];
end
% Find configuration for R^2.
d = 2;
r = 1;
% The nmds function is in the EDA Toolbox.
[Xd,stress,dhats] = nmmds(dissim,d,r);
ind8 = find(classlab == 8);
ind11 = find(classlab == 11);
% Plot with symbols mapped to class (topic).
plot(Xd(ind8,1),Xd(ind8,2),'.',...
    Xd(ind11,1),Xd(ind11,2),'o')
legend({'Class 8';'Class 11'})
```

The resulting plot is shown in Figure 3.4. We see that there is no clear separation between the two topics, but we do have some interesting structure in this scatterplot indicating the possible presence of sub-topics.
❑

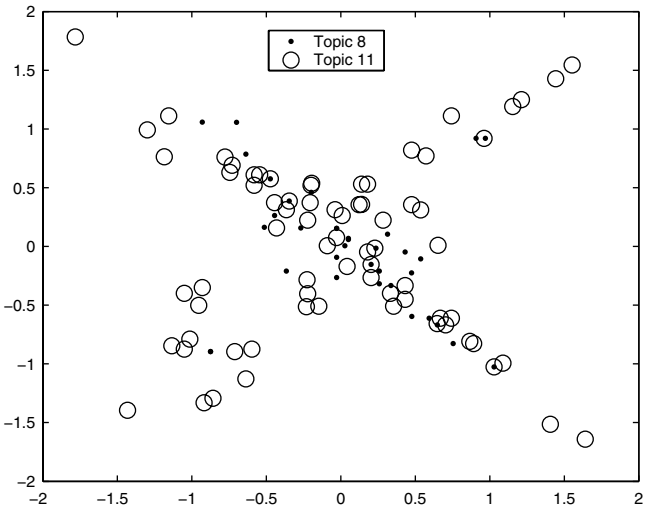

FIGURE 3.4
This shows the results of applying Kruskal's nonmetric MDS to topics 8 and 11, both of which concern North Korea. We see some interesting structure in this configuration and the possible presence of sub-topics.

What should the dimensionality be when we are applying MDS to a set of data? This is a rather difficult question to answer for most MDS methods. In the spirit of EDA, one could apply the procedure for several values of d and record the value for stress. We could then construct a plot similar to the scree plot in PCA, where we have the dimension d on the horizontal axis and the stress on the vertical axis. Stress always decreases with the addition of more dimensions, so we would be looking for the point at which the payoff from adding more dimensions decreases (i.e., the value of d where we see an elbow in the curve).

The Statistics Toolbox, Version 5 includes a new function for multidimensional scaling called **`mdscale`**. It does both metric and nonmetric MDS, and it includes several choices for the stress. With this function, one can use weights, specify the type of starting configuration, and request replicates for different random initial configurations.

3.2 Manifold Learning

Some recently developed approaches tackle the problem of nonlinear dimensionality reduction by assuming that the data lie on a submanifold of Euclidean space M. The common goal of these methods is to produce coordinates in a lower dimensional space, such that the neighborhood structure of the submanifold is preserved. In other words, points that are neighbors along the submanifold are also neighbors in the reduced parameterization of the data. (See Figure 3.5 and Example 3.5 for an illustration of these concepts.)

These new methods are discussed in this section. We will first present locally linear embedding, which is an unsupervised learning algorithm that exploits the local symmetries of linear reconstructions. Next, we cover isometric feature mapping, which is an extension to classical MDS. Finally, we present a new method called Hessian eigenmaps, which addresses one of the limitations of isometric feature mapping.

All of these methods are implemented in MATLAB, and the code is freely available for download. (We provide the URLs in Appendix B.) Because of this, we will not be including all of the code, but will only show how to use the existing implementations of the techniques.

3.2.1 Locally Linear Embedding

Locally linear embedding (LLE) was developed by Roweis and Saul [2000]. The method is an eigenvector-based method, and its optimizations do not involve local minima or iterative algorithms. The technique is based on simple geometric concepts. First, we assume that the data are sampled in

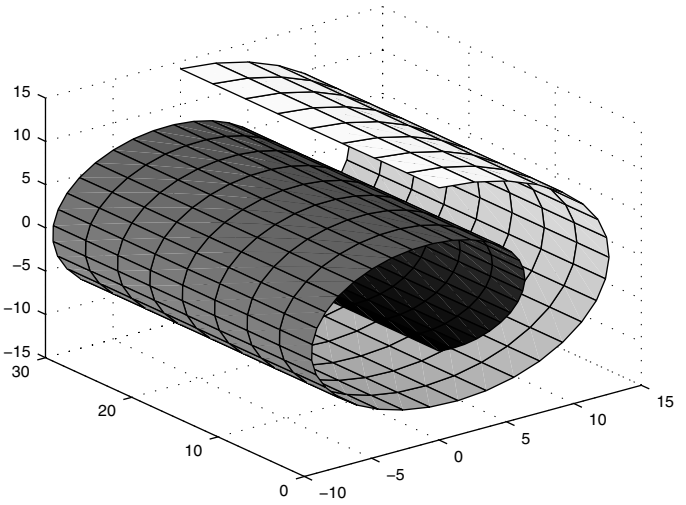

FIGURE 3.5
This shows the submanifold for the Swiss roll data set. We see that this is really a 2-D manifold (or surface) embedded in a 3-D space.

sufficient quantity from a smooth submanifold. We also assume that each data point and its neighbors lie on or close to a locally linear patch of the manifold M.

The LLE algorithm starts by characterizing the local geometry of the patches by finding linear coefficients that reconstruct each data point by using only its k nearest neighbors, with respect to Euclidean distance. There will be errors in the reconstruction, and these are measured by

$$\varepsilon(W) = \sum_i \left| \mathbf{x}_i - \sum_j W_{ij} \mathbf{x}_j \right|^2, \tag{3.4}$$

where the subscript j ranges over those points that are in the neighborhood of $\mathbf{x}_i$. The weights are found by optimizing Equation 3.4 subject to the following constraint:

$$\sum_j W_{ij} = 1 .$$

The optimal weights are found using least squares, the details of which are omitted here.

Once the weights W_{ij} are found, we fix these and find a representation $\mathbf{y}_i$ of the original points in a low-dimensional space. This is also done by optimizing a cost function, which in this case is given by

$$\Phi(\mathbf{y}) = \sum_i \left| \mathbf{y}_i - \sum_j W_{ij}\mathbf{y}_j \right|^2. \quad (3.5)$$

This defines a quadratic form in $\mathbf{y}_i$, and it can be minimized by solving a sparse eigenvector problem. The d eigenvectors corresponding to the smallest nonzero eigenvalues provide a set of orthogonal coordinates centered at the origin. The method is summarized below.

Locally Linear Embedding

1. Determine a value for k and d.
2. For each data point $\mathbf{x}_i$, find the k nearest neighbors.
3. Compute the weights W_{ij} that optimally reconstruct each data point $\mathbf{x}_i$ from its neighbors (Equation 3.4).
4. Find the d-dimensional points $\mathbf{y}_i$ that are optimally reconstructed using the same weights found in step 3 (Equation 3.5).

Note that the algorithm requires a value for k, which governs the size of the neighborhood, and a value for d. Of course, different results can be obtained when we vary these values. We illustrate the use of LLE in Example 3.5.

3.2.2 Isometric Feature Mapping - ISOMAP

Isometric feature mapping (ISOMAP) was developed by Tenenbaum, de Silva and Langford [2000] as a way of enhancing classical MDS. The basic idea is to use distances along a geodesic path (presumably measured along the manifold M) as measures of dissimilarity. As with LLE, ISOMAP assumes that the data lie on an unknown submanifold M that is embedded in a p-dimensional space. It seeks a mapping $f: X \rightarrow Y$ that preserves the intrinsic metric structure of the observations. That is, the mapping preserves the distances between observations, where the distance is measured along the geodesic path of M. It also assumes that the manifold M is globally isometric to a convex subset of a low-dimensional Euclidean space.

In Figure 3.6, we show an example that illustrates this idea. The Euclidean distance between two points on the manifold is shown by the straight line connecting them. If our goal is to recover the manifold, then a truer indication of the distance between these two points is given by their distance on the manifold, i.e., the geodesic distance along the manifold between the points.

The ISOMAP algorithm has three main steps. The first step is to find the neighboring points based on the interpoint distances d_{ij}. This can be done by either specifying a value of k to define the number of nearest neighbors or a radius ε. The distances are typically taken to be Euclidean, but they can be any valid metric. The neighborhood relations are then represented as a

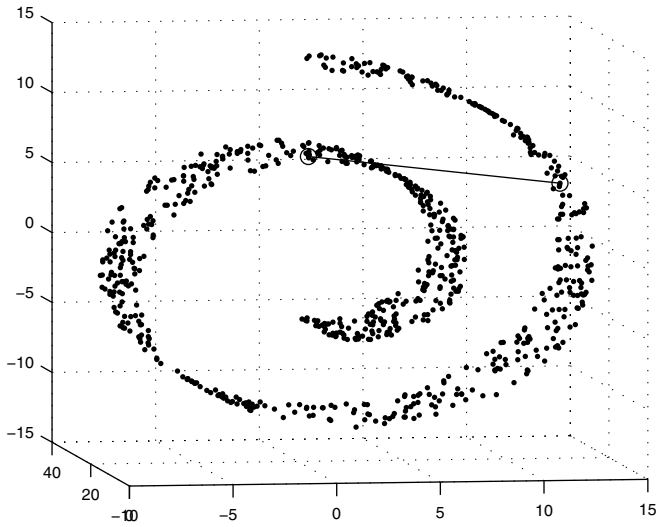

FIGURE 3.6
This is a data set that was randomly generated according to the Swiss roll parametrization [Tenenbaum, de Silva and Langford, 2000]. The Euclidean distance between two points indicated by the circles is given by the straight line shown here. If we are seeking the neighborhood structure as given by the submanifold M, then it would be better to use the geodesic distance (the distance along the manifold or the roll) between the points. One can think of this as the distance a bug would travel if it took the shortest path between these two points, while walking on the manifold.

weighted graph, where the edges of the graph are given weights equal to the distance d_{ij}. The second step of ISOMAP provides estimates of the geodesic distances between all pairs of points i and j by computing their shortest path distance using the graph obtained in step one. In the third step, classical MDS is applied to the geodesic distances and an embedding is found in d-dimensional space as described in the first section of this chapter.

Procedure - ISOMAP

1. Construct the neighborhood graph over all observations by connecting the ij-th point if point i is one of the k nearest neighbors of j (or if the distance between them is less than ε). Set the lengths of the edges equal to d_{ij}.
2. Calculate the shortest paths between points in the graph.
3. Obtain the d-dimensional embedding by applying classical MDS to the geodesic paths found in step 2.

The input to this algorithm is a value for k or ε and the interpoint distance matrix. It should be noted that different results are obtained when the value for k (or ε) is varied. In fact, it could be the case that ISOMAP will return fewer than n lower-dimensional points. If this happens, then the value of k (or ε) should be increased.

3.2.3 Hessian Eigenmaps

We stated in the description of ISOMAP that the manifold M is assumed to be convex. Donoho and Grimes [2003] developed a method called ***Hessian eigenmaps*** that will recover a low-dimensional parametrization for data lying on a manifold that is locally isometric to an open, connected subset of Euclidean space. This significantly expands the class of data sets where manifold learning can be accomplished using isometry principles. This method can be viewed as a modification of the LLE method. Thus, it is also called ***Hessian locally linear embedding*** (HLLE).

We start with a parameter space $\Theta \subset \mathrm{R}^d$ and a smooth mapping $\psi: \Theta \rightarrow \mathrm{R}^p$, where R^p is the embedding space and $d < p$. Further, we assume that Θ is an open, connected subset of R^d, and ψ is a locally isometric embedding of Θ into R^p. The manifold can be written as a function of the parameter space, as follows

$$M = \psi(\Theta).$$

One can think of the manifold as the enumeration $m = \psi(\theta)$ of all possible measurements as one varies the parameters θ for a given process.

We assume that we have some observations m_i that represent measurements over many different choices of control parameters θ_i (i = 1,...,n). These measurements are the same as our observations $\mathbf{x}_i$.[6] The HLLE description given in Donoho and Grimes [2003] considers the case where all data points lie exactly on the manifold M. The goal is to recover the underlying parameterization ψ and the parameter settings θ_i, up to a rigid motion.

We now describe the main steps of the HLLE algorithm. We leave out the derivation and proofs because we just want to provide the general ideas underlying the algorithm. The reader is referred to the original paper and the MATLAB code for more information and implementation details.

The two main assumptions of the HLLE algorithm are:

1. In a small enough neighborhood of each point m, geodesic distances to nearby points m' (both on the manifold M) are identical to Euclidean distances between associated parameter points θ and θ'.

[6] We use different notation here to be consistent with the original paper.

This is called ***local isometry***. (ISOMAP deals with the case where M assumes a globally isometric parameterization.)

2. The parameter space Θ is an open, connected subset of R^d, which is a weaker condition than the convexity assumption of ISOMAP.

In general, the idea behind HLLE is to define a neighborhood around some m in M and obtain local tangent coordinates. These local coordinates are used to define the Hessian of a smooth function f: $M \rightarrow R$. The function f is differentiated in the tangent coordinate system to produce the tangent Hessian. A quadratic form $\mathcal{H}(f)$ is obtained using the tangent Hessian, and the isometric coordinates θ can be recovered by identifying a suitable basis for the null space of $\mathcal{H}(f)$.

The inputs required for the algorithm are a set of n p-dimensional data points, a value for d, and the number of nearest neighbors k to determine the neighborhood. The only constraint on these values is that $\min(k,p) > d$. The algorithm estimates the tangent coordinates by using the SVD on the neighborhood of each point, and it uses the empirical version of the operator $\mathcal{H}(f)$. The output of HLLE is a set of n d-dimensional embedding coordinates.

Example 3.5

We generated some data from an S-curve manifold to be used with LLE and ISOMAP and saved it in a file called **scurve.mat**.[7] We show both the true manifold and the data randomly generated from this manifold in Figure 3.7.

```
load scurve
% The scurve file contains our data matrix X.
% First set up some parameters for LLE.
K = 12;
d = 2;
% Run LLE - note that LLE takes the input data
% as rows corresponding to dimensions and
% columns corresponding to observations. This is
% the transpose of our usual data matrix.
Y = lle(X,K,d);
% Plot results in scatter plot.
scatter(Y(1,:),Y(2,:),12,[angle angle],'+','filled');
```

We show the results in Figure 3.8. Note by the colors that neighboring points on the manifold are mapped into neighboring points in the 2-D embedding. Now we use ISOMAP on the same data.

```
% Now run the ISOMAP - we need the distances for input.
% We need the data matrix as n x p.
X = X';
dists = squareform(pdist(X));
```

[7] Look at the file called **example35.m** for more details on how to generate the data.

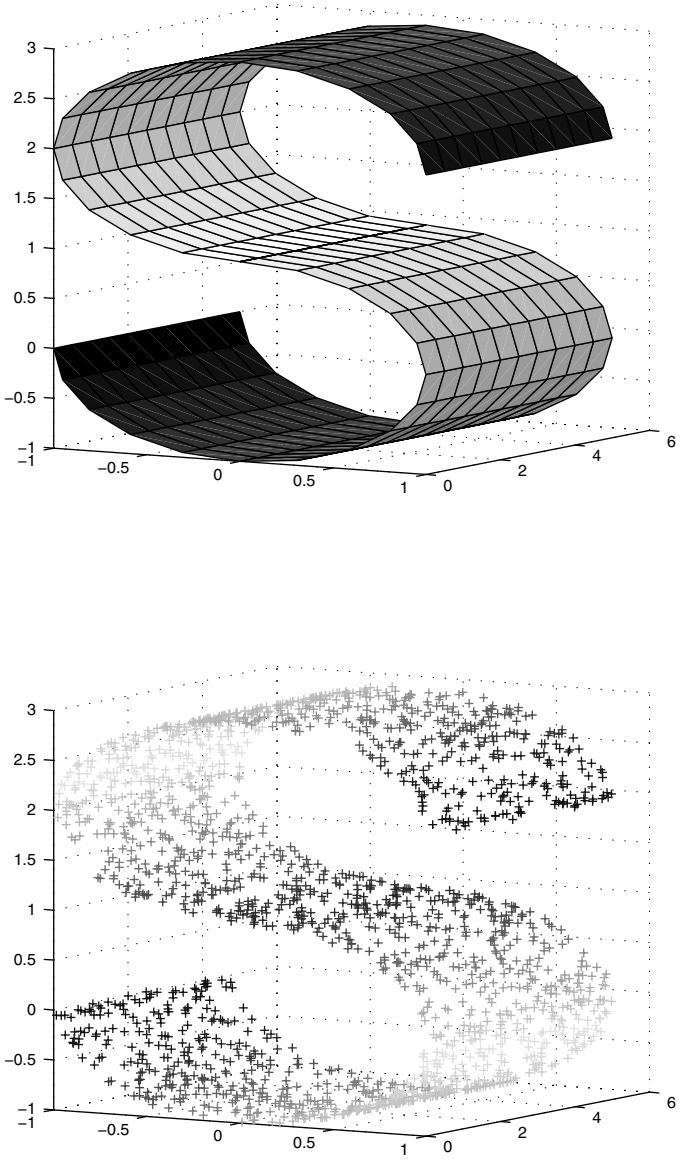

FIGURE 3.7
The top panel shows the true S-curve manifold, which is a 2-D manifold embedded in 3-D. The bottom panel is the data set randomly generated from the manifold. The gray scale values are an indication of the neighborhood. See the associated color figure following page 144.

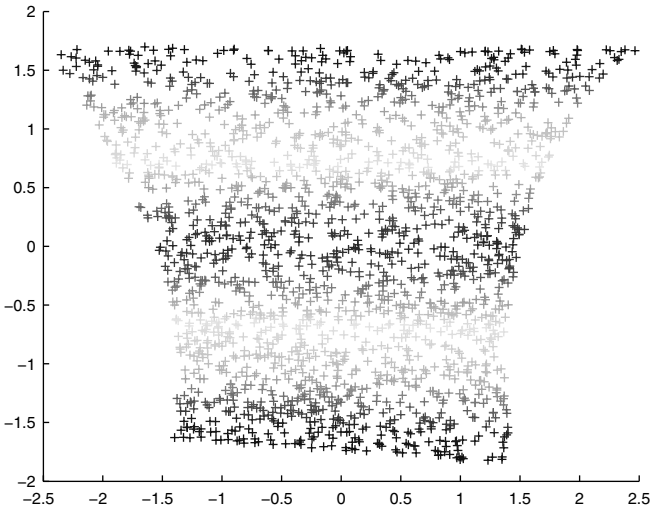

FIGURE 3.8
This is the embedding recovered from LLE. Note that the neighborhood structure is preserved. See the associated color figure following page 144.

```
options.dims = 1:10;    % These are for ISOMAP.
options.display = 0;
[Yiso, Riso, Eiso] = isomap(dists, 'k', 7, options);
```

Constructing a scatter plot of this embedding to compare with LLE is left as an exercise to the reader. As stated in the text, LLE and ISOMAP have some problems with data sets that are not convex. We show an example here using both ISOMAP and HLLE to discover such an embedding. These data were generated according to the code provided by Donoho and Grimes [2003]. Essentially, we have the Swiss roll manifold with observations removed that fall within a rectangular area along the surface.

```
% Now run the example from Grimes and Donoho.
load swissroll
options.dims = 1:10;
options.display = 0;
dists = squareform(pdist(X'));
[Yiso, Riso, Eiso] = isomap(dists, 'k', 7, options);
% Now for the Hessian LLE.
Y2 = hlle(X,K,d);
scatter(Y2(1,:),Y2(2,:),12,tt,'+');
```

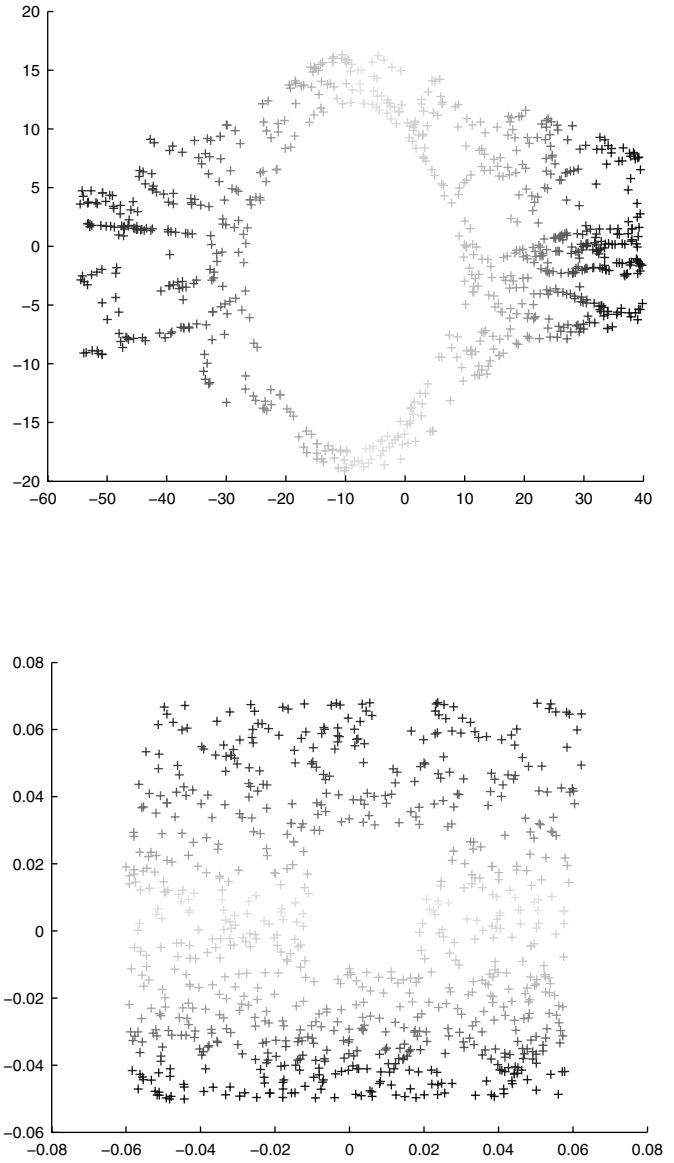

FIGURE 3.9
The top panel contains the 2-D coordinates from ISOMAP. We do see the hole in the embedding, but it looks like an oval rather than a rectangle. The bottom panel shows the 2-D coordinates from HLLE. HLLE was able to recover the correct embedding. See the associated color figure following page 144.

We see in Figure 3.9 that ISOMAP was unable to recover the correct embedding. We did find the hole, but it is distorted. HLLE was able to find the correct embedding with no distortion.
❑

We note a few algorithmic complexity issues. The two locally linear embedding methods, LLE and HLLE, can be used on problems with many data points (large n), because the initial computations are performed on small neighborhoods and they can exploit sparse matrix techniques. On the other hand, ISOMAP requires the calculation of a full matrix of geodesic distances for its initial step. However, LLE and HLLE are affected by the dimensionality p, because they must estimate a local tangent space at each data point. Also, in the HLLE algorithm, we must estimate second derivatives, which can be difficult with high-dimensional data.

3.3 Artificial Neural Network Approaches

We now discuss two other methods that we categorize as ones based on artificial neural network (ANN) ideas: the self-organizing map and the generative topographic mapping. These ANN approaches also look for intrinsically low-dimensional structures that are embedded nonlinearly in a high-dimensional space. As in MDS and the manifold learning approaches, these seek a single global low-dimensional nonlinear model of the observations. Both of these algorithms try to fit a grid or predefined topology (usually 2-D) to the data, using greedy algorithms that first fit the large-scale linear structure of the data and then make small-scale nonlinear refinements.

There are MATLAB toolboxes available for both self-organizing maps and generative topographic maps. They are free, and they come with extensive documentation and examples. Because the code for these methods can be very extensive and would not add to the understanding of the reader, we will not be showing the code in the book. Instead, we will show how to use some of the functions in the examples.

3.3.1 Self-Organizing Maps - SOM

The ***self-organizing map*** or SOM is a tool for the exploration and visualization of high-dimensional data [Kohonen, 1998]. It derives an orderly mapping of data onto a regular, low-dimensional grid. The dimensionality of the grid is usually $d = 2$ for ease of visualization. It converts complex, nonlinear relationships in the high-dimensional space into simpler geometric relationships such that the important topological and metric relationships are conveyed. The data are organized on the grid in such a way that observations that are close together in the high-dimensional space are also closer to each

other on the grid. Thus, this is very similar to the ideas of MDS, except that the positions in the low-dimensional space are restricted to the grid. The grid locations are denoted by $\mathbf{r}_i$.

There are two methods used in SOM: incremental learning and batch mapping. We will describe both of them, and then illustrate some of the functions in the SOM Toolbox[8] in Example 3.6. We will also present a brief discussion of the various methods for visualizing and exploring the results of SOM.

The ***incremental*** or ***sequential learning method*** for SOM is an iterative process. We start with the set of observations $\mathbf{x}_i$ and a set of p-dimensional model vectors $\mathbf{m}_i$. These ***model vectors*** are also called ***neurons***, ***prototypes***, or ***codebooks***. Each model vector is associated with a vertex ($\mathbf{r}_i$) on a 2-D lattice that can be either hexagonal or rectangular and starts out with some initial value ($\mathbf{m}_i(t=0)$). This could be done by setting them to random values, by setting them equal to a random subset of the original data points or by using PCA to provide an initial ordering.

At each training step, a vector $\mathbf{x}_i$ is selected and the distance between it and all the model vectors is calculated, where the distance is typically Euclidean. The SOM Toolbox handles missing values by excluding them from the calculation, and variable weights can also be used. The best-matching unit (BMU) or model vector is found and is denoted by $\mathbf{m}_c$. Once the closest model vector $\mathbf{m}_c$ is found, the model vectors are updated so that $\mathbf{m}_c$ is moved closer to the data vector $\mathbf{x}_i$. The neighbors of the BMU $\mathbf{m}_c$ are also updated, in a weighted fashion. The update rule for the model vector $\mathbf{m}_i$ is

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t)h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)],$$

where t denotes time or iteration number. The learning rate is given by $\alpha(t)$ and $0 < \alpha(t) < 1$, which decreases monotonically as the iteration proceeds.

The neighborhood is governed by the function $h_{ci}(t)$, and a Gaussian centered at the best-matching unit is often used. The SOM Toolbox has other neighborhood functions available, as well as different learning rates α (t). If the Gaussian is used, then we have

$$h_{ci}(t) = \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_c\|^2}{2\sigma^2(t)}\right),$$

where the symbol $\|\bullet\|$ denotes the distance, and the $\mathbf{r}_i$ are the coordinates on the grid. The size or width of the neighborhood is governed by $\sigma(t)$, and this value decreases monotonically.

The training is usually done in two phases. The first phase corresponds to a large learning rate and neighborhood radius $\sigma(t)$, which provides a large-scale approximation to the data. The map is fine tuned in the second phase,

[8] See Appendix B for the website where one can download the SOM Toolbox.

where the learning rate and neighborhood radius are small. Once the process finishes, we have the set of prototype or model vectors over the 2-D coordinates on the map grid.

The ***batch training method*** or Batch Map is also iterative, but it uses the whole data set before adjustments are made rather than a single vector. At each step of the algorithm, the data set is partitioned such that each observation is associated with its nearest model vector. The updated model vectors are found as a weighted average of the data, where the weight of each observation is the value of the neighborhood function $h_{ic}(t)$ at its BMU c.

Several methods exist for visualizing the resulting map and prototype vectors. These methods can have any of the following goals. The first is to get an idea of the overall shape of the data and whether clusters are present. The second goal is to analyze the prototype vectors for characteristics of the clusters and correlations among components or variables. The third task is to see how new observations fit with the map or to discover anomalies. We will focus on one visualization method called the U-matrix that is often used to locate clusters in the data [Ultsch and Siemon, 1990].

The U-matrix is based on distances. First, the distance of each model vector to each of its immediate neighbors is calculated. This distance is then visualized using a color scale. Clusters are seen as those map units that have smaller distances, surrounded by borders indicating larger distances. Another approach is to use the size of the symbol to represent the average distance to its neighbors, so cluster borders would be shown as larger symbols. We illustrate the SOM and the U-matrix visualization method in Example 3.6.

Example 3.6

We turn to the **oronsay** data set to illustrate some of the basic commands in the SOM Toolbox. First we have to load the data and put it into a MATLAB data structure that is recognized by the functions. This toolbox comes with several normalization methods, and it is recommended that they be used before building a map.

```
% Use the oronsay data set.
load oronsay
% Convert labels to cell array of strings.
for i = 1:length(beachdune)
    mid{i} = int2str(midden(i));
    % Use next one in exercises.
    bd{i} = int2str(beachdune(i));
end
% Normalize each variable to have unit variance.
D = som_normalize(oronsay,'var');
% Convert to a data structure.
sD = som_data_struct(D);
% Add the labels - must be transposed.
```

```
sD = som_set(sD,'labels',mid');
```

We can visualize the results in many ways, most of which will be left as an exercise. We show the U-matrix in Figure 3.10 and include labels for the codebook vectors.

```
% Make the SOM
sM = som_make(sD);
sM = som_autolabel(sM,sD,'vote');
% Plot U matrix.
som_show(sM,'umat','all');
% Add labels to an existing plot.
som_show_add('label',sM,'subplot',1);
```

Note that the larger values indicate cluster borders, and low values indicate clusters. By looking at the colors, we see a couple of clusters - one in the lower left corner and one in the top. The labels indicate some separation into groups.

❑

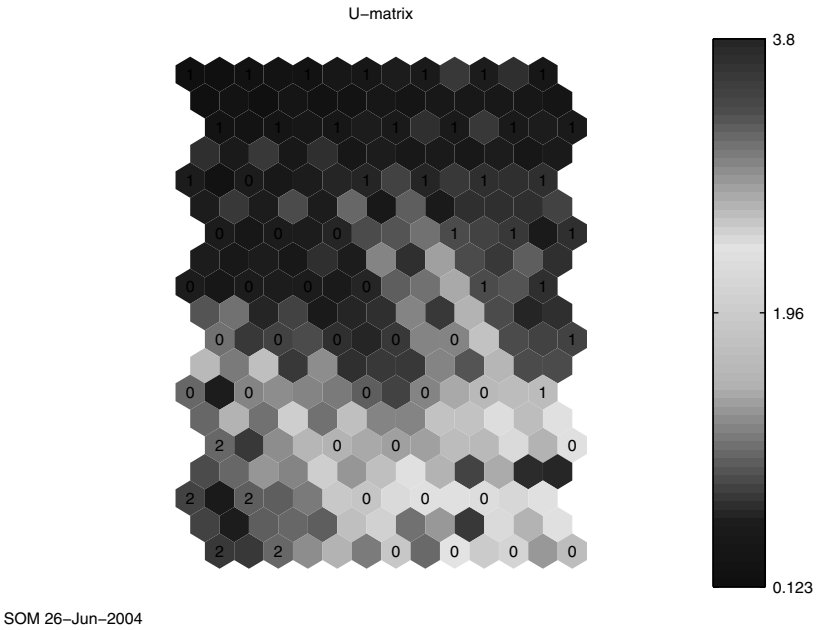

FIGURE 3.10
This is the SOM for the **oronsay** data set. We can see some cluster structure here by the colors. One is in the upper part of the map, and the other is in the lower left corner. The labels on the map elements also indicate some clustering. Recall that the classes are midden (0), *Cnoc Coig* (1), and *Caisteal nan Gillean* (2). See the associated color figure following page 144.

3.3.2 Generative Topographic Maps - GTM

The SOM has several limitations [Bishop, Svensén and Williams, 1996]. First, the SOM is based on heuristics, and the algorithm is not derived from the optimization of an objective function. Preservation of the neighborhood structure is not guaranteed by the SOM method, and there could be problems with convergence of the prototype vectors. The SOM does not define a density model, and the use of the codebook vectors as a model of the distribution of the original data is limited. Finally, the choice of how the neighborhood function should shrink during training is somewhat of an art, so it is difficult to compare different runs of the SOM procedure on the same data set. The ***generative topographic mapping*** (GTM) was inspired by the SOM and attempts to overcome its limitations.

The GTM is described in terms of a latent variable model (or space) with dimensionality d [Bishop, Svensén and Williams, 1996, 1998]. The goal is to find a representation for the distribution $p(\mathbf{x})$ of p-dimensional data, in terms of a smaller number of d latent variables $\mathbf{m} = (m_1, \ldots, m_d)$. As usual, we want $d < p$, and often take $d = 2$ for ease of visualization. We can achieve this goal by finding a mapping $\mathbf{y}(\mathbf{m};\mathbf{W})$, where $\mathbf{W}$ is a matrix containing weights, that takes a point $\mathbf{m}$ in the latent space and maps it to a point $\mathbf{x}$ in the data space. This might seem somewhat backward from what we considered previously, but we will see later how we can use the inverse of this mapping to view summaries of the observations in the reduced latent space.

We start with a probability distribution $p(\mathbf{m})$ defined on the latent variable space (with d dimensions), which in turn induces a distribution $p(\mathbf{y} \mid \mathbf{W})$ in the data space (with p dimensions). For a given $\mathbf{m}$ and $\mathbf{W}$, we choose a Gaussian centered at $\mathbf{y}(\mathbf{m};\mathbf{W})$ as follows:

$$p(\mathbf{x}|\mathbf{m},\mathbf{W},\beta) = \left(\frac{\beta}{2\pi}\right)^{p/2} \exp\left\{\frac{-\beta}{2}\|\mathbf{y}(\mathbf{m};\mathbf{W}) - \mathbf{x}\|^2\right\},$$

where the variance is β^{-1},[9] and $\|\bullet\|$ denotes the inner product. Other models can be used, as discussed in Bishop, Svensén and Williams [1998]. The matrix $\mathbf{W}$ contains the parameters or weights that govern the mapping. To derive the desired mapping, we must estimate β and the matrix $\mathbf{W}$.

The next step is to assume some form for $p(\mathbf{m})$ defined in the latent space. To make this similar to the SOM, the distribution is taken as a sum of delta functions centered on the nodes of a regular grid in latent space:

$$p(\mathbf{m}) = \frac{1}{K}\sum_{k=1}^{K}\delta(\mathbf{m} - \mathbf{m}_k),$$

[9] We note that this is different than the usual notation, familiar to statisticians. However, we are keeping it here to be consistent with the original derivation.

where K is the total number of grid points or delta functions. Each point $\mathbf{m}_k$ in the latent space is mapped to a corresponding point $\mathbf{y}(\mathbf{m}_k;\mathbf{W})$ in the data space, where it becomes the center of a Gaussian density function. This is illustrated in Figure 3.11.

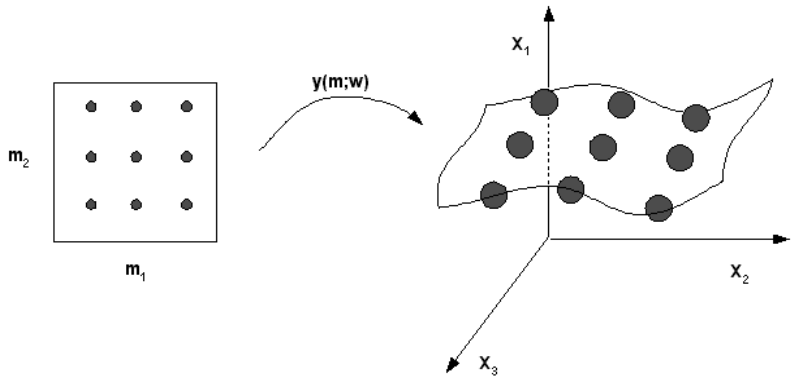

FIGURE 3.11
This illustrates the mapping used in GTM. Our latent space is on the left, and the data space is on the right. A Gaussian is centered at each of the data points, represented by the spheres.

We can use maximum likelihood and the Expectation-Maximization (EM) algorithm (see Chapter 6) to get estimates for β and $\mathbf{W}$. Bishop, Svensén and Williams [1998] show that the log-likelihood function for the distribution described here is given by

$$L(\mathbf{W},\ \beta) = \sum_{i=1}^{n} \ln\left\{\frac{1}{K}\sum_{k=1}^{K} p(\mathbf{x}_i|\mathbf{m}_k,\ \mathbf{W},\ \beta)\right\}. \tag{3.6}$$

They next choose a model for $\mathbf{y}(\mathbf{m};\mathbf{W})$, which is given by

$$\mathbf{y}(\mathbf{x};\mathbf{W}) = \mathbf{W}\phi(\mathbf{x}),$$

where the elements of $\phi(\mathbf{m})$ have M fixed basis functions $\phi_j(\mathbf{m})$, and $\mathbf{W}$ is of size $p \times M$. For basis functions, they choose Gaussians whose centers are distributed on a uniform grid in latent space. Note that the centers for these basis functions are ***not*** the same as the grid points $\mathbf{m}_i$. Each of these Gaussian basis functions ϕ has a common width parameter σ. The smoothness of the manifold is determined by the value of σ, the number of basis functions M, and their spacing.

Looking at Equation 3.6, we can view this as a missing-data problem, where we do not know which component k generated each data point $\mathbf{x}_i$. The

EM algorithm for estimating β and $\mathbf{W}$ consists of two steps that are done iteratively until the value of the log-likelihood function converges. In the E-step, we use the current values of the parameters to evaluate the posterior probabilities of each component k for every data point. This is calculated according to the following

$$\tau_{ki}(\mathbf{W}_{\text{old}},\ \beta_{\text{old}}) = \frac{p(\mathbf{x}_i|\mathbf{m}_k,\ \mathbf{W}_{\text{old}},\ \beta_{\text{old}})}{\sum_{c=1}^{K} p(\mathbf{x}_i|\mathbf{m}_c,\ \mathbf{W}_{\text{old}},\ \beta_{\text{old}})}, \tag{3.7}$$

where the subscript 'old' indicates the current values.

In the M-step, we use the posterior probabilities to find weighted updates for the parameters. First, we calculate a new version of the weight matrix from the following equation

$$\mathbf{\Phi}^T\mathbf{G}_{\text{old}}\mathbf{\Phi}\mathbf{W}_{\text{new}}^T = \mathbf{\Phi}^T\mathbf{T}_{\text{old}}\mathbf{X}, \tag{3.8}$$

where $\mathbf{\Phi}$ is a K x M matrix with elements $\Phi_{kj} = \phi_j(\mathbf{m}_k)$, $\mathbf{X}$ is the data matrix, $\mathbf{T}$ is a $K \times n$ matrix with elements τ_{ki}, and $\mathbf{G}$ is a $K \times K$ diagonal matrix where the elements are given by

$$G_{kk} = \sum_{i=1}^{n} \tau_{ki}(\mathbf{W},\beta).$$

Equation 3.8 is solved for $\mathbf{W}_{\text{new}}$ using standard linear algebra techniques. A time-saving issue related to this update equation is that $\mathbf{\Phi}$ is constant, so it only needs to be evaluated once.

We now need to determine an update for β that maximizes the log-likelihood. This is given by

$$\frac{1}{\beta_{\text{new}}} = \frac{1}{np}\sum_{i=1}^{n}\sum_{k=1}^{K}\tau_{kn}(\mathbf{W}_{\text{old}},\ \beta_{\text{old}})\left\|\mathbf{W}_{\text{old}}\phi(\mathbf{m}_k - \mathbf{x}_i)\right\|^2. \tag{3.9}$$

To summarize, the algorithm requires starting points for the matrix $\mathbf{W}$ and the inverse variance β. We must also specify a set of points $\mathbf{m}_i$, as well as a set of basis functions $\phi_j(\mathbf{m})$. The parameters $\mathbf{W}$ and β define a mixture of Gaussians with centers $\mathbf{W}\phi(\mathbf{m}_k)$ and equal covariance matrices given by $\beta^{-1}\mathbf{I}$. Given initial values, the EM algorithm is used to estimate these parameters. The E-step finds the posterior probabilities using Equation 3.7, and the M-step updates the estimates for $\mathbf{W}$ and β using Equations 3.8 and 3.9. These steps are repeated until the log-likelihood (Equation 3.6) converges.

So, the GTM gives us a mapping from this latent space to the original p-dimensional data space. For EDA purposes, we are really interested in going the other way: mapping our p-dimensional data into some lower-dimensional space. As we see from the development of the GTM, each datum $\mathbf{x}_i$ provides a posterior distribution in our latent space. Thus, this posterior *distribution* in latent space provides information about a *single* observation. Visualizing all observations in this manner would be too difficult, so each distribution should be summarized in some way. Two summaries that come to mind are the mean and the mode, which are then visualized as individual points in our latent space. The mean for observation $\mathbf{x}_i$ is calculated from

$$\overline{\mathbf{m}}_i = \sum_{k=1}^{K} \tau_{ki} \mathbf{m}_k .$$

The mode for the i-th observation (or posterior distribution) is given by the maximum value τ_{ki} over all values of k. The values $\overline{\mathbf{m}}_i$ or modes are shown as symbols in a scatterplot or some other visualization scheme.

Example 3.7

We again turn to the **oronsay** data to show the basic functionality of the GTM Toolbox.[10] The parameters were set according to the example in their documentation.

```
load oronsay
% Initialize parameters for GTM.
noLatPts = 400;
noBasisFn = 81;
sigma = 1.5;
% Initialize required variables for GTM.
[X,MU,FI,W,beta] = gtm_stp2(oronsay,noLatPts,...
    noBasisFn,sigma);
lambda = 0.001;
cycles = 40;
[trndW,trndBeta,llhLog] = gtm_trn(oronsay,FI,W,...
    lambda,cycles,beta,'quiet');
```

The function **gtm_stp2** initializes the required variables, and **gtm_trn** does the training. Each observation gets mapped into a probability distribution in the 2-D map, so we need to find either the mean or mode of each one to show as a point. We can do this as follows:

```
% Get the means in latent space.
mus = gtm_pmn(oronsay,X,FI,trndW,trndBeta);
% Get the modes in latent space.
```

[10] See Appendix B for information on where to download the GTM Toolbox.

```
modes = gtm_pmd(oronsay,X,FI,trndW);
```

We now plot the values in the lower-dimensional space using symbols corresponding to their class.

```
ind0 = find(midden == 0);
ind1 = find(midden == 1);
ind2 = find(midden == 2);
plot(mus(ind0,1),mus(ind0,2),'k.',mus(ind1,1),...
    mus(ind1,2),'kx',mus(ind2,1),mus(ind2,2),'ko')
```

The resulting plot is shown in Figure 3.12, where we can see some separation into the three groups.

❑

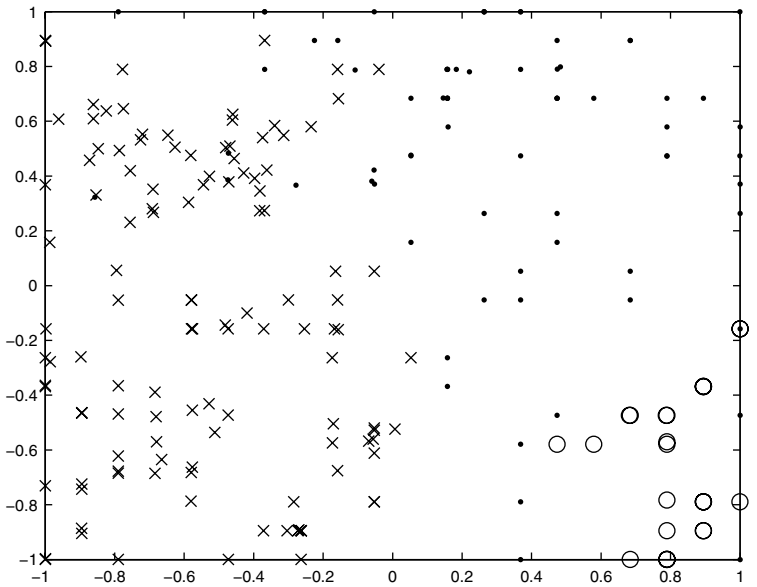

FIGURE 3.12
This shows the map obtained from GTM, where the distribution for each point is summarized by the mean. Each mean is displayed using a different symbol: Class 0 is '.'; Class 1 is 'x'; and Class 2 is 'o'. We can see some separation into groups from this plot.

3.4 Summary and Further Reading

In this chapter, we discussed several methods for finding a nonlinear mapping from a high-dimensional space to one with lower dimensionality.

The first set of methods was grouped under the name multidimensional scaling, and we presented both metric and nonmetric MDS. We note that in some cases, depending on how it is set up, MDS finds a linear mapping. We also presented several methods for learning manifolds, where the emphasis is on nonlinear manifolds. These techniques are locally linear embedding, ISOMAP, and Hessian locally linear embedding. ISOMAP is really an enhanced version of classical MDS, where geodesic distances are used as input to classical MDS. HLLE is similar in spirit to LLE, and its main advantage is that it can handle data sets that are not convex. Finally, we presented two artificial neural network approaches called self-organizing maps and generative topographic maps.

We have already mentioned some of the major MDS references, but we also include them here with more information on their content. Our primary reference for terminology and methods was Cox and Cox [2001]. This is a highly readable book, suitable for students and practitioners with a background in basic statistics. The methods are clearly stated for both classical MDS and nonmetric MDS. Also, the authors provide a CD-ROM with programs (running under DOS) and data sets so the reader can use these tools.

Another book by Borg and Groenen [1997] brings many of the algorithms and techniques of MDS together in a way that can be understood by those with a two-semester course in statistics for social sciences or business. The authors provide the derivation of the methods, along with ways to interpret the results. They do not provide computer software or show how this can be done using existing software packages, but their algorithms are very clear and understandable.

A brief introduction to MDS can be obtained from Kruskal and Wish [1978]. This book is primarily focused on applications in the social sciences, but it would be useful for those who need to come to a quick understanding of the basics of MDS. Computational and algorithmic considerations are not stressed in this book. There are many overview papers on MDS in the various journals and encyclopedias; these include Mead [1992], Siedlecki, Siedlecka and Sklansky [1988], Steyvers [2002], and Young [1985].

Some of the initial work in MDS was done by Shepard in 1962. The first paper in the series [Shepard, 1962a] describes a computer program to reconstruct a configuration of points in Euclidean space, when the only information that is known about the distance between the points is some unknown monotonic function of the distance. The second paper in the series [Shepard, 1962b] presents two applications of this method to artificial data. Kruskal continued the development of MDS by introducing an objective function and developing a method for optimizing it [Kruskal, 1964a, 1964b]. We highly recommend reading the original Kruskal papers; they are easily understood and should provide the reader with a nice understanding of the origins of MDS.

Because they are relatively recent innovations, there are not a lot of references for the manifold learning methods (ISOMAP, LLE, and HLLE).

However, each of the websites has links to technical reports and papers that describe the work in more depth (see Appendix B). A nice overview of manifold learning is given in Saul and Roweis [2002], with an emphasis on LLE. Further issues with ISOMAP are explored in Balasubramanian and Schwartz [2002]. More detailed information regarding HLLE can be found in a technical report written by Donoho and Grimes [2002].

There is one book dedicated to SOM written by Kohonen [2001]. Many papers on SOM have appeared in the literature, and a 1998 technical report lists 3,043 works that are based on the SOM [Kangas and Kaski, 1998]. A nice, short overview of SOM can be found in Kohonen [1998]. Some recent applications of SOM have been in the area of document clustering [Kaski, et al., 1998; Kohonen, et al., 2000] and the analysis of gene microarrays [Tamayo, et al., 1999]. Theoretical aspects of the SOM are discussed in Cottrell, Fort and Pages [1998]. The use of the SOM for clustering is described in Kiang [2001] and Vesanto and Alhoniemi [2000]. Visualization and EDA methods for SOM are discussed in Mao and Jain [1995], Ultsch and Siemon [1990], Deboeck and Kohonen [1998], and Vesanto [1997; 1999]. GTM is a recent addition to this area, so there are fewer papers, but for those who want further information, we recommend Bishop, Svensén and Williams [1996, 1997a, 1997b, and 1998], Bishop, Hinton and Strachan [1997], and Bishop and Tipping [1998].

Exercises

3.1 Try the classical MDS approach using the **`skull`** data set. Plot the results in a scatterplot using the text labels as plotting symbols (see the **`text`** function). Do you see any separation between the categories of gender? [Cox and Cox, 2001]. Try PCA on the **`skull`** data. How does this compare with the classical MDS results?

3.2 Apply the SMACOF and nonmetric MDS methods to the **`skull`** data set. Compare your results with the configuration obtained through classical MDS.

3.3 Use **`plot3`** (similar to **`plot`**, but in three dimensions) to construct a 3-D scatterplot of the data in Example 3.1. Describe your results.

3.4 Apply the SMACOF method to the **`oronsay`** data set and comment on the results.

3.5 Repeat the Examples 3.2, 3.4 and problem 3.4 for several values of d. See the **`help`** on **`gplotmatrix`** and use it to display the results for $d > 2$. Do a scree-like plot of stress versus d to see what d is best.

3.6 The Shepard diagram for nonmetric MDS is a plot where the ordered dissimilarities are on the horizontal axis. The distances (shown as points) and the disparities (shown as a line) are on the vertical. With large data sets, this is not too useful. However, with small data sets, it

can be used to see the shape of the regression curve. Implement this in MATLAB and test it on one of the smaller data sets.

3.7 Try using **`som_show(sM)`** in Example 3.6. This shows a U-matrix for each variable. Look at each one individually and add the labels to the elements: **`som_show(sM,'comp',1)`**, etc. See the SOM Toolbox documentation for more information on how these functions work.

3.8 Repeat Example 3.6 and problem 3.7 using the other labels for the **`oronsay`** data set. Discuss your results.

3.9 Repeat the plot in Example 3.7 (GTM) using the modes instead of the means. Do you see any difference between them?

3.10 Do a help on the Statistics Toolbox (version 5) function **`mdscale`**. Apply the methods (metric and nonmetric MDS) to the **`skulls`** and **`oronsay`** data sets.

3.11 Apply the ISOMAP method to the **`scurve`** data from Example 3.5. Construct a scatterplot of the data and compare to the results from LLE.

3.12 Apply the LLE method to the **`swissroll`** data of Example 3.5. Construct a scatterplot and compare with HLLE and ISOMAP.

3.13 What is the intrinsic dimensionality of the **`swissroll`** and **`scurve`** data sets?

3.14 Where possible, apply MDS, ISOMAP, LLE, HLLE, SOM, and GTM to the following data sets. Discuss and compare the results.

a. BPM data sets

b. gene expression data sets

c. **`iris`**

d. **`pollen`**

e. **`posse`**

f. **`oronsay`**

g. **`skulls`**

3.15 Repeat Example 3.2 with different starting values to search for different structures. Analyze your results.