

Data Transformation Feature Generation and Dimensionality Reduction

3.1 INTRODUCTION

In this chapter, we deal with linear and nonlinear transformation techniques, which are used to generate a set of features from a set of measurements or from a set of originally generated features. The goal is to obtain new features that encode the classification information in a more compact way compared with the original features. This implies a reduction in the number of features needed for a given classification task, which is also known as *dimensionality reduction* because the dimension of the new feature space is now reduced. The goal, of course, is to achieve this dimensionality reduction in some optimal sense so that the loss of information, which in general is unavoidable after reducing the original number of features, is as small as possible.

3.2 PRINCIPAL COMPONENT ANALYSIS

Principal component analysis (PCA) is one of the most popular techniques for dimensionality reduction. Starting from an original set of l samples (features), which form the elements of a vector $x \in \mathcal{R}^l$, the goal is to apply a linear transformation to obtain a new set of samples:

$$y = A^T x$$

so that the components of y are uncorrelated. In a second stage, one chooses the most significant of these components. The steps are summarized here:

1. Estimate the covariance matrix S . Usually the mean value is assumed to be zero, $E[x] = 0$. In this case, the covariance and autocorrelation matrices coincide, $R \equiv E[xx^T] = S$. If this is not the case, we subtract the mean. Recall that, given N feature vectors, $x_i \in \mathcal{R}^l$, $i = 1, 2, \dots, N$, the autocorrelation matrix estimate is given by

$$R \approx \frac{1}{N} \sum_{i=1}^N x_i x_i^T \quad (3.1)$$

2. Perform the eigendecomposition of S and compute the l eigenvalues/eigenvectors, λ_i , $a_i \in \mathcal{R}^l$, $i = 0, 2, \dots, l-1$.

3. Arrange the eigenvalues in descending order, $\lambda_0 \geq \lambda_1 \geq \dots \geq \lambda_{l-1}$.
4. Choose the m largest eigenvalues. Usually m is chosen so that the gap between λ_{m-1} and λ_m is large. Eigenvalues $\lambda_0, \lambda_1, \dots, \lambda_{m-1}$ are known as the m *principal components*.
5. Use the respective (column) eigenvectors a_i , $i = 0, 1, 2, \dots, m-1$ to form the transformation matrix

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{m-1} \end{bmatrix}$$

6. Transform each l -dimensional vector x in the original space to an m -dimensional vector y via the transformation $y = A^T x$. In other words, the i th element $y(i)$ of y is the *projection* of x on a_i ($y(i) = a_i^T x$).

As pointed out in [Theo 09, Section 6.3], the total variance of the elements of x , $\sum_{i=0}^{l-1} E[x^2(i)]$ (for zero mean), is equal to the sum of the eigenvalues $\sum_{i=0}^{l-1} \lambda_i$. After the transformation, the variance of the i th element, $E[y^2(i)]$, $i = 0, 2, \dots, l-1$, is equal to λ_i . Thus, selection of the elements that correspond to the m largest eigenvalues retains the maximum variance.

To compute the principal components, type

$$[eigenval, eigenvec, explain, Y, mean_vec] = pca_fun(X, m)$$

where

X is an $l \times N$ matrix with columns that are the data vectors,

m is the number of the most significant principal components taken into account,

$eigenval$ is an m -dimensional column vector containing the m largest eigenvalues of the covariance matrix of X in descending order,

$eigenvec$ is an $l \times m$ -dimensional matrix, containing in its columns the eigenvectors that correspond to the m largest eigenvalues of the covariance matrix of X ,

$explain$ is an l -dimensional column vector whose i th element is the percentage of the total variance retained along (in the MATLAB terminology *explained* by) the i th principal component,

Y is an $m \times N$ matrix containing the projections of the data points of X to the space spanned by the m vectors of $eigenvec$,

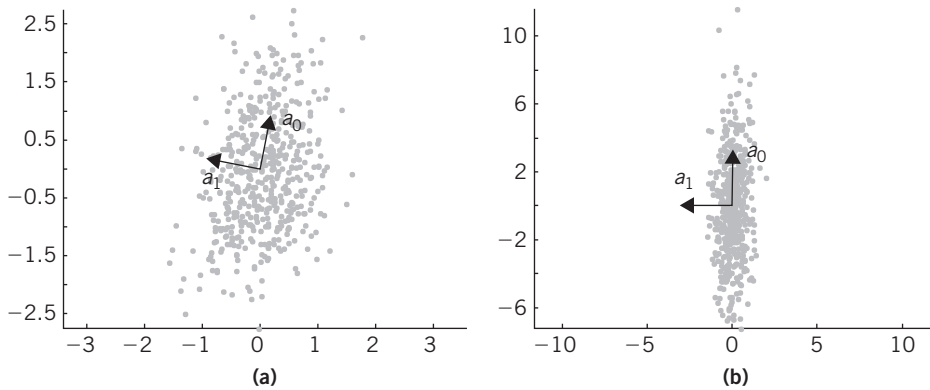
$mean_vec$ is the mean vector of the column vectors of X .

Example 3.2.1

1. Generate a set X_1 of $N = 500$ 2-dimensional vectors from a Gaussian distribution with zero mean and covariance matrix

$$S_1 = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 1.0 \end{bmatrix}$$

Perform PCA on X_1 ; that is, compute the two eigenvalues/eigenvectors of the estimate \hat{S}_1 of S_1 obtained using the vectors of X_1 . Taking into account that the i th eigenvalue “explains” the variance along the direction of the i th eigenvector of \hat{S}_1 , compute the percentage of the total variance explained

**FIGURE 3.1**

Data points of X_1 (a) and X_2 (b) considered in [Example 3.2.1](#), together with the (normalized) eigenvectors of \hat{S}_1 and \hat{S}_2 , respectively.

by each of the two components as the ratio $\frac{\lambda_i}{\lambda_0 + \lambda_1}$, $i = 0, 1$. Plot the data set X_1 as well as the eigenvectors of \hat{S}_1 . Comment on the results.

2. Similarly generate a data set X_2 , now with the covariance matrix $S_2 = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 9.0 \end{bmatrix}$. Repeat the previous experiment.

Solution. Take the following steps:

Step 1. To generate the data set X_1 , type

```
randn('seed',0) %For reproducibility of the results
S1=[.3 .2; .2 1];
[1,1]=size(S1);
mv=zeros(1,1);
N=500;
m=2;
X1=mvnrnd(mv,S1,N)';
```

To apply PCA on X_1 and to compute the percentage of the total variance explained by each component, type

```
[eigenval,eigenvec,explained,Y,mean_vec]=pca_fun(X1,m);
```

To plot the points of the data set X_1 together with the (normalized) eigenvectors of \hat{S}_1 , type (see [Figure 3.1\(a\)](#))

```
figure(1), hold on
figure(1), plot(X1(1,:),X1(2,:), 'r.')
```

```
figure(1), axis equal
figure(1), line([0; eigenvec(1,1)], [0; eigenvec(2,1)])
figure(1), line([0; eigenvec(1,2)], [0; eigenvec(2,2)])
```

The percentages of the total variance explained by the first and second components are 78.98% and 21.02%, respectively. This means that if we project the points of X_1 along the direction of the principal eigenvector (that corresponds to the largest eigenvalue of \hat{S}_1), we retain 78.98% of the total variance of X_1 ; 21.02% of the total variance, associated with the second principal component, will be “lost.”

Step 2. To generate the data set X_2 , repeat the previous code, where now X_1 and S_1 are replaced by X_2 and S_2 , respectively. In this case, the percentages of the total variance explained by the first and second components are 96.74% and 3.26%, respectively. This means that if we project the points of X_2 along the direction of the eigenvector that corresponds to the largest eigenvalue of \hat{S}_1 , we retain almost all the variance of X_2 in the 2-dimensional space (see [Figure 3.1\(b\)](#)). Explain this using physical reasoning. ■

The goal of the next example is to demonstrate that projecting in a lower-dimensional space, so as to retain most of the variance, does not necessarily guarantee that the classification-related information is preserved.

Example 3.2.2

1. **a.** Generate a data set X_1 consisting of 400 2-dimensional vectors that stem from two classes. The first 200 stem from the first class, which is modeled by the Gaussian distribution with mean $m_1 = [-8, 8]^T$; the rest stem from the second class, modeled by the Gaussian distribution with mean $m_2 = [8, 8]^T$. Both distributions share the covariance matrix

$$S = \begin{bmatrix} 0.3 & 1.5 \\ 1.5 & 9.0 \end{bmatrix}$$

- b.** Perform PCA on X_1 and compute the percentage of the total amount of variance explained by each component.
- c.** Project the vectors of X_1 along the direction of the first principal component and plot the data set X_1 and its projection to the first principal component. Comment on the results.
2. Repeat on data set X_2 , which is generated as X_1 but with $m_1 = [-1, 0]^T$ and $m_2 = [1, 0]^T$.
3. Compare the results obtained and draw conclusions.

Solution. Take the following steps:

Step 1(a). To generate data set X_1 and the vector y_1 , whose i th coordinate contains the class label of the i th vector of X_1 , type

```
randn('seed',0) %For reproducibility of the results
S=[.3 1.5; 1.5 9];
[1,1]=size(S);
mv=[-8 8; 8 8]';
N=200;
```

```
X1=[mvnrnd(mv(:,1),S,N); mvnrnd(mv(:,2),S,N)]';
y1=[ones(1,N), 2*ones(1,N)];
```

Step 1(b). To compute the eigenvalues/eigenvectors and variance percentages required in this step, type

```
m=2;
[eigenval,eigenvec,explained,Y,mean_vec]=pca_fun(X1,m);
```

Step 1(c). The projections of the data points of X_1 along the direction of the first principal component are contained in the first row of Y , returned by the function *pca_fun*. To plot the data vectors of X_1 as well as their projections, type

```
%Plot of X1
figure(1), hold on
figure(1), plot(X(1,y==1),X(2,y==1),'r.',X(1,y==2),X(2,y==2),'bo')
%Computation of the projections of X1
w=eigenvec(:,1);
t1=w'*X(:,y==1);
t2=w'*X(:,y==2);
X_proj1=[t1;t1].*((w/(w'*w))*ones(1,length(t1)));
X_proj2=[t2;t2].*((w/(w'*w))*ones(1,length(t2)));
%Plot of the projections
figure(1), plot(X_proj1(1,:),X_proj1(2,:),'k.',...
X_proj2(1,:),X_proj2(2,:),'ko')
figure(1), axis equal
%Plot of the eigenvectors
figure(1), line([0; eigenvec(1,1)], [0; eigenvec(2,1)])
figure(1), line([0; eigenvec(1,2)], [0; eigenvec(2,2)])
```

The percentages of the total amount of variance explained by the first and second components are 87.34% and 12.66%, respectively. That is, the projection to the direction of the first component retains most of the total variance of X_1 .

Step 2(a). To generate X_2 , the code for X_1 is executed again, with $m_1 = [-1, 0]^T$ and $m_2 = [1, 0]^T$.

Step 2(b)–(c). The codes in the (b)–(c) branches of [step 1](#) are executed for X_2 . The percentages of the total amount of variance explained by the two principal components are 90.19% and 9.81%, respectively.

Step 3. In the two previous cases approximately 90% of the total variance of the data sets is retained after projecting along the first principal component. However, there is no guarantee that class discrimination is retained along this direction. Indeed, in the case of X_1 the data in the two classes, after projection along the first principal eigenvector, remain well separated. However, this is not the case for data set X_2 (see [Figure 3.2](#)). ■

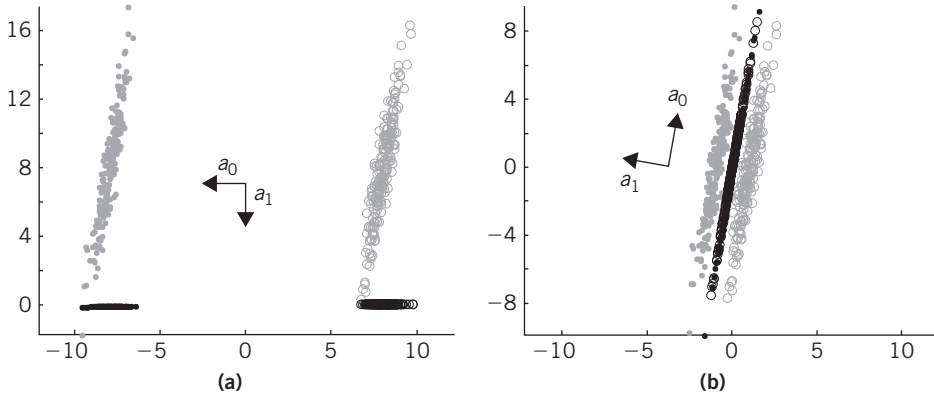


FIGURE 3.2

Data points (gray) of X_1 (a) and X_2 (b) considered in [Example 3.2.2](#), along with the (normalized) eigenvectors of \hat{S}_1 and \hat{S}_2 , respectively. The data points of the two classes of X_1 , after projection along the first principal eigenvector (black), remain well separated. This is not the case for X_2 .

Exercise 3.2.1

Take the following steps:

1. Generate a data set X_1 consisting of 400 3-dimensional vectors that stem from two classes. The first half of them stem from the first class, which is modeled by the Gaussian distribution with mean $m_1 = [-6, 6, 6]^T$; the rest stem from the second class, modeled by the Gaussian distribution with mean $m_2 = [6, 6, 6]^T$. Both distributions share the covariance matrix

$$S = \begin{bmatrix} 0.3 & 1.0 & 1.0 \\ 1.0 & 9.0 & 1.0 \\ 1.0 & 1.0 & 9.0 \end{bmatrix}$$

Perform PCA on X_1 and compute the percentage of the total amount of variance explained by each principal component. Project the vectors of X_1 on the space spanned by the first two principal components Y_1 and Y_2 . Plot the data in the X_{11} - X_{12} , X_{11} - X_{13} , X_{12} - X_{13} , Y_1 - Y_2 , Y_1 - Y_3 , Y_2 - Y_3 subspaces (six MATLAB figures in total).

2. Generate a data set X_2 as in [step 1](#), now with $m_1 = [-2, 0, 0]^T$ and $m_2 = [2, 0, 0]^T$. Repeat the process as described in [step 1](#).
3. Compare the results obtained from each data set and draw conclusions.

3.3 THE SINGULAR VALUE DECOMPOSITION METHOD

Given an $l \times N$ matrix, X , there exist square *unitary* matrices U and V of dimensions $l \times l$ and $N \times N$, respectively, so that

$$X = U \begin{bmatrix} \Lambda^{\frac{1}{2}} & O \\ O & 0 \end{bmatrix} V^T$$

where Λ is a square $r \times r$ matrix, with $r \leq \min\{l, N\}$ (r is equal to the rank of X). Since matrices U and V are unitary, their column vectors are, by definition, orthonormal and $UU^T = I$ and $VV^T = I$. Matrix $\Lambda^{\frac{1}{2}}$ is given by

$$\Lambda^{\frac{1}{2}} = \begin{bmatrix} \sqrt{\lambda_0} & & & \\ & \sqrt{\lambda_1} & & \\ & & \ddots & \\ & & & \sqrt{\lambda_{r-1}} \end{bmatrix}$$

where λ_i , $i = 0, 1, \dots, r-1$, are the r nonzero eigenvalues of XX^T , which are the same with the eigenvalues of X^TX [Theo 09, Section 6.4], and known as the *singular values* of X . Equivalently, we can write

$$X = \sum_{i=0}^{r-1} \sqrt{\lambda_i} u_i v_i^T \quad (3.2)$$

where u_i , v_i , $i = 0, 1, \dots, r-1$, are the corresponding eigenvectors of XX^T and X^TX , respectively. Moreover, u_i and v_i , $i = 0, \dots, r-1$ are the first r column vectors of U and V , respectively. The rest of the column vectors of U and V correspond to zero eigenvalues.

If we retain $m \leq r$ terms in the summation of Eq. (3.2), that is,

$$\hat{X} = \sum_{i=0}^{m-1} \sqrt{\lambda_i} u_i v_i^T \quad (3.3)$$

then \hat{X} is the best approximation (in the Frobenius sense) of X of rank m [Theo 09, Section 6.4].

To compute the Singular Value Decomposition (SVD) of a matrix X , type

$$[U, s, V, Y] = \text{svd_fun}(X, m)$$

where

X is an $l \times N$ matrix whose columns contain the data vectors,

m is the number of the largest singular values that will be taken into account,

U is an $l \times l$ matrix containing the eigenvectors of XX^T in descending order,

s is an r -dimensional vector containing the singular values in descending order,

V is an $N \times N$ matrix containing the eigenvectors of X^TX in descending order,

Y is an $m \times N$ matrix containing the projections of the data points of X on the space spanned by the m leading eigenvectors contained in U .

More on SVD can be found in [Theo 09, Section 6.4].

Exercise 3.3.1

1. Consider the data set X_1 of Exercise 3.2.1. Perform singular value decomposition using `svd_fun`. Then project the vectors of X_1 on the space spanned by the m leading eigenvectors contained in U (that correspond to Y_1 and Y_2). Finally, plot the data in the X_{11} - X_{12} , X_{11} - X_{13} , X_{12} - X_{13} , Y_1 - Y_2 , Y_1 - Y_3 , Y_2 - Y_3 spaces (six MATLAB figures in total).

2. Repeat for the data set X_2 of [Exercise 3.2.1](#) and compare the results. Observe that the results obtained for the SVD case are similar to those obtained in [Exercise 3.2.1](#) for the PCA case (why?).

Example 3.3.1. Generate a data set of $N = 100$ vectors of dimensionality $l = 2000$. The vectors stem from the Gaussian distribution with a mean equal to the l -dimensional zero vector and a diagonal covariance matrix, S , having all of its nonzero elements equal to 0.1 except $S(1, 1)$ and $S(2, 2)$, which are equal to 10,000. Apply PCA and SVD on the previous data set and draw your conclusions.

Solution. To generate matrix X , containing the vectors of the data set, type

```
N=100;
l=2000;
mv=zeros(1,l);
S=0.1*eye(l);
S(1,1)=10000;
S(2,2)=10000;
randn('seed',0)
X=mvnrnd(mv,S,N)';
```

Note that the data exhibit significant spread along the first two axes, that is, along the vectors

$$e_1 = [1, \overbrace{0, \dots, 0}^{l-1}]^T \text{ and } e_2 = [0, 1, \overbrace{0, \dots, 0}^{l-2}]^T.$$

To run PCA and SVD on X and to measure the execution time of each method, type

```
%PCA
t0=clock;
m=5;
[eigenval,eigenvec,explain,Y]=pca_fun(X,m);
time1=etime(clock,t0)
'----'

%SVD
t0=clock;
m=min(N,l);
[U,S,V,Y]=svd_fun(X,m);
time2=etime(clock,t0)
```

From the previously obtained results, two conclusions can be drawn.

First, both methods identify (approximately) e_1 and e_2 as the most significant directions. To verify this, compare the first two columns of *eigenvec* produced by PCA with the first two columns of U , that is, $U(:, 1:2)$, produced by SVD, by typing

```
[eigenvec(:,1:2) U(:,1:2)]'
```

Second, provided that enough computer memory is available, PCA will take orders of magnitude more time than SVD (if not enough memory is available, PCA will not run at all). The difference in

the performance of the two methods lies in the fact that in PCA we perform eigendecomposition on the $l \times l$ covariance matrix while in SVD we perform eigendecomposition on the $N \times N$ $X^T X$ and then, with a simple transformation, compute the eigenvectors of XX^T (which may be viewed as a scaled approximation of the autocorrelation matrix). Moreover, it has to be emphasized that, in general for such cases where $N < l$, the obtained estimate of the autocorrelation matrix is not a good one. Such cases, where $N < l$, arise in image-processing applications, in Web mining, in microarray analysis, and the like. ■

3.4 FISHER'S LINEAR DISCRIMINANT ANALYSIS

In PCA, the dimensionality reduction is performed in an unsupervised mode. Feature vectors are projected on the subspace spanned by the dominant eigenvectors of the covariance (autocorrelation) matrix. In this section, computation of the subspace on which one projects, in order to reduce dimensionality, takes place in a supervised mode. This subspace is also determined via the solution of an eigendecomposition problem, but the corresponding matrix is different.

In the 2-class case, the goal is to search for a *single* direction, w , so that the respective projections y of the l -dimensional feature vectors $x \in \mathcal{R}^l$ maximize Fisher's discriminant ratio.

Fisher's discriminant ratio of a *scalar* feature y in a 2-class classification task is defined as

$$FDR = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

where μ_1, μ_2 are the mean values of y , and σ_1^2, σ_2^2 are the variances of y in the two classes, respectively. In other words, after the projection on w the goal is for the mean values of the data points in the two classes to be as far apart as possible and for the variances to be as small as possible. It turns out that w is given by the maximum eigenvector of the matrix product $S_w^{-1} S_b$ [Theo 09, Section 5.8], where for two equiprobable classes

$$S_w = \frac{1}{2} (S_1 + S_2)$$

is known as the *within-class scatter matrix*, with S_1, S_2 being the respective covariance matrices. S_b is known as the *between-class scatter matrix*, defined by

$$S_b = \frac{1}{2} (m_1 - m_0)(m_1 - m_0)^T + \frac{1}{2} (m_2 - m_0)(m_2 - m_0)^T$$

where m_0 is the overall mean of the data x in the original \mathcal{R}^l space and m_1, m_2 are the mean values in the two classes, respectively [Theo 09, Section 5.6.3]. It can be shown, however, that in this special 2-class case the eigenanalysis step can be bypassed and the solution directly given by

$$w = S_w^{-1} (m_1 - m_2)$$

In the c -class case, the goal is to find the $m \leq c - 1$ directions (m -dimensional subspace) so that the so-called J_3 criterion, defined as

$$J_3 = \text{trace}\{S_w^{-1} S_b\}$$

is maximized. In the previous equation

$$S_w = \sum_{i=1}^c P_i S_i, S_b = \sum_{i=1}^c P_i (m_i - m_0)(m_i - m_0)^T$$

and the P_i 's denote the respective class a priori probabilities. This is a generalization of the FDR criterion in the multiclass case with different a priori probabilities. The m directions are given by the m dominant eigenvectors of the matrix product $S_w^{-1} S_b$.

It must be pointed out that the rank of the S_b matrix is $c - 1$ at the most (although it is given as a sum of c matrices, only $c - 1$ of these terms are independent [Theo 09, Section 5.6.3]. This is the reason that m was upper-bounded by $c - 1$; only the $c - 1$ largest eigenvalues (at most) are nonzero. In some cases, this may be a drawback because the maximum number of features that this method can generate is bounded by the number of classes [Theo 09, Section 5.8].

Example 3.4.1

1. Apply linear discriminant analysis (LDA) on the data set X_2 generated in the second part of Example 3.2.2.
2. Compare the results obtained with those obtained from the PCA analysis.

Solution. Take the following steps:

Step 1. To estimate the mean vectors of each class using the available samples, type

```
mv_est(:,1)=mean(X2(:,y2==1))';
mv_est(:,2)=mean(X2(:,y2==2))';
```

To compute the within-scatter matrix S_w , use the *scatter_mat* function, which computes the within class (S_w), the between class (S_b), and the mixture class (S_m) [Theo 09, Section 5.6.3] for a c -class classification problem based on a set of data vectors. This function is called by

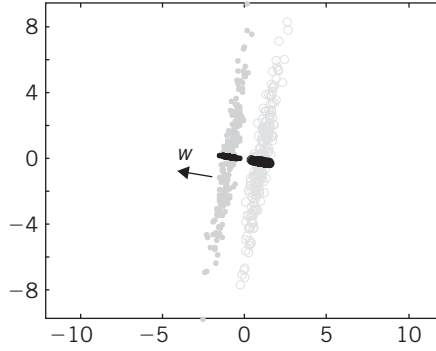
```
[Sw,Sb,Sm]=scatter_mat(X2,y2);
```

Since the two classes are equiprobable, the direction w along which Fisher's discriminant ratio is maximized is computed as $w = S_w^{-1}(m_1 - m_2)$. In MATLAB terms this is written as

```
w=inv(Sw)*(mv_est(:,1)-mv_est(:,2))
```

Finally, the projection of the data vectors of X_2 on the direction w as well as the plot of the results is carried out through the following statements (see Figure 3.3)

```
%Plot of the data set
figure(1), plot(X(1,y==1),X(2,y==1),'r.',...
X(1,y==2),X(2,y==2),'bo')
figure(1), axis equal
%Computation of the projections
t1=w'*X(:,y==1);
```

**FIGURE 3.3**

Points of the data set X_2 (gray) and their projections (black) along the direction of w , from [Example 3.4.1](#).

```
t2=w'*X(:,y==2);
X_proj1=[t1;t1].*((w/(w'*w))*ones(1,length(t1)));
X_proj2=[t2;t2].*((w/(w'*w))*ones(1,length(t2)));
%Plot of the projections
figure(1), hold on
figure(1), plot(X_proj(1,y==1),X_proj(2,y==1),'y.',...
X_proj(1,y==2),X_proj(2,y==2),'co')
```

Step 2. Comparing the result depicted in MATLAB figure 1, which was produced by the execution of the previous code, to the corresponding result obtained by the PCA analysis, it is readily observed that the classes remain well separated when the vectors of X_2 are projected along the w direction that results from Fisher's discriminant analysis. In contrast, classes were heavily overlapped when they were projected along the principal direction provided by PCA. ■

Example 3.4.2

- 1a.** Generate a data set of 900 3-dimensional data vectors, which stem from two classes—the first 100 vectors from a zero-mean Gaussian distribution with covariance matrix

$$S_1 = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

The rest grouped in 8 groups of 100 vectors. Each group stems from a Gaussian distribution. All of these distributions share the covariance matrix

$$S_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

while their means are

- $m_1^2 = [a, 0, 0]^T$
- $m_2^2 = [a/2, a/2, 0]^T$
- $m_3^2 = [0, a, 0]^T$
- $m_4^2 = [-a/2, a/2, 0]^T$
- $m_5^2 = [-a, 0, 0]^T$
- $m_6^2 = [-a/2, -a/2, 0]^T$
- $m_7^2 = [0, -a, 0]^T$
- $m_8^2 = [a/2, -a/2, 0]^T$

where $a = 6$ (m_i^2 denotes the mean of the i th Gaussian distribution of the second class).

Take the following steps:

- 1b.** Plot the 3-dimensional data set and view it from different angles to get a feeling of how the data are spread in the 3-dimensional space (use the Rotate-3D MATLAB utility).
- 1c.** Perform Fisher's discriminant analysis on the previous data set. Project the data on the subspace spanned by the eigenvectors that correspond to the nonzero eigenvalues of the matrix product $S_w^{-1}S_b$. Comment on the results.
- 2.** Repeat [step 1](#) for a 3-class problem where the data are generated like those in [step 1](#), with the exception that the last group of 100 vectors, which stem from the Gaussian distribution with mean m_8^2 , is labeled class 3.

Solution. Take the following steps:

Step 1(a). To generate a 3×900 -dimensional matrix whose columns are the data vectors, type

```
%Initialization of random number generator
randn('seed',10)
%Definition of the parameters
S1=[.5 0 0; 0 .5 0; 0 0 .01];
S2=[1 0 0; 0 1 0; 0 0 .01];
a=6;
mv=[0 0 0; a 0 0; a/2 a/2 0; 0 a 0; -a/2 a/2 0;...
    -a 0 0; -a/2 -a/2 0; 0 -a 0; a/2 -a/2 0]';
N=100;
% Generation of the data set
X=[mvnrnd(mv(:,1),S1,N)];
for i=2:9
    X=[X; mvnrnd(mv(:,i),S2,N)];
end
X=X';
c=2; %No of classes
y=[ones(1,N) 2*ones(1,8*N)]; %Class label vector
```

Step 1(b). To plot the data set X in the 3-dimensional space, type

```
figure(1), plot3(X(1,y==1),X(2,y==1),X(3,y==1),'r.',...
X(1,y==2),X(2,y==2),X(3,y==2),'b.')
figure(1), axis equal
```

With the Rotate-3D button of MATLAB figure 1, you can view the data set from different angles. It is easy to notice that the variation of data along the third direction is very small (because of the small values of $S_1(3,3)$ and $S_2(3,3)$). The data set in the 3-dimensional space may be considered as lying across the $x-y$ plane, with a very small variation along the z axis.

Clearly, the projection of the data set in the $x-y$ plane retains the separation of the classes, but this is not the case with the projections on the $x-z$ and $y-z$ planes. In addition, observe that there is no single direction (1-dimensional space) w that retains the separation of the classes after projecting X on it.

Step 1(c). To perform Fisher's discriminant analysis, first compute the scatter matrices S_w and S_b ; then perform eigendecomposition on the matrix $S_w^{-1}S_b$; finally, project the data on the subspace spanned by the eigenvectors of $S_w^{-1}S_b$ that correspond to the nonzero eigenvalues. The following MATLAB code may be used:

```
% Scatter matrix computation
[Sw,Sb,Sm]=scatter_mat(X,y);
% Eigendecomposition of Sw^(-1)*Sb
[V,D]=eig(inv(Sw)*Sb);
% Sorting the eigenvalues in descending order
% and rearranging accordingly the eigenvectors
s=diag(D);
[s,ind]=sort(s,1,'descend');
V=V(:,ind);
% Selecting in A the eigenvectors corresponding
% to non-zero eigenvalues
A=V(:,1:c-1);
% Project the data set on the space spanned by
% the column vectors of A
Y=A'*X;
```

Here we used the code for the multiclass case with $c = 2$. Since the number of classes is equal to 2, only one eigenvalue of $S_w^{-1}S_b$ is nonzero (0.000234). Thus, Fisher's discriminant analysis gives a single direction (1-dimensional space) along which the data will be projected.

To plot the projections of X on the subspace spanned by the eigenvector of $S_w^{-1}S_b$, which corresponds to the nonzero eigenvalue, type

```
figure(2), plot(Y(y==1),0,'ro',Y(y==2),0,'b.')
figure(2), axis equal
```

Observe that the projections of the data of the two classes coincide. Thus, in this case Fisher's discriminant analysis cannot provide a smaller subspace where the class discrimination is retained. This happens because the number of classes is equal to 2 and so the dimensionality of the reduced subspace is at most 1, which is not sufficient for the current problem.

Step 2(a). To generate a 3×900 -dimensional matrix whose columns are data vectors, repeat the code given in [step 1\(a\)](#), replacing the last two lines with

```
% Definition of the number of classes
c=3;
% Definition of the class label of each vector
y=[ones(1,N) 2*ones(1,7*N) 3*ones(1,N)];
```

Step 2(b). To plot the data set X in the 3-dimensional space, type


```
figure(1), plot3(X(1,y==1),X(2,y==1),X(3,y==1),'r.',X(1,y==2),...
X(2,y==2),X(3,y==2),'b.',X(1,y==3),X(2,y==3),X(3,y==3),'g.')
figure(1), axis equal
```

Step 2(c). Adopt the MATLAB code of [step 1\(c\)](#) for the current data set. In this case, since there are three classes, we may have at most two nonzero eigenvalues of $S_w^{-1}S_b$. Indeed, the nonzero eigenvalues now turn out to be 0.222145 and 0.000104.

To plot the projections of X on the space spanned by the eigenvectors of $S_w^{-1}S_b$ that correspond to the nonzero eigenvalues (2-dimensional space), type

```
figure(3), plot(Y(1,y==1),Y(2,y==1),'ro',...
Y(1,y==2),Y(2,y==2),'b.',Y(1,y==3),Y(2,y==3),'gx')
figure(3), axis equal
```

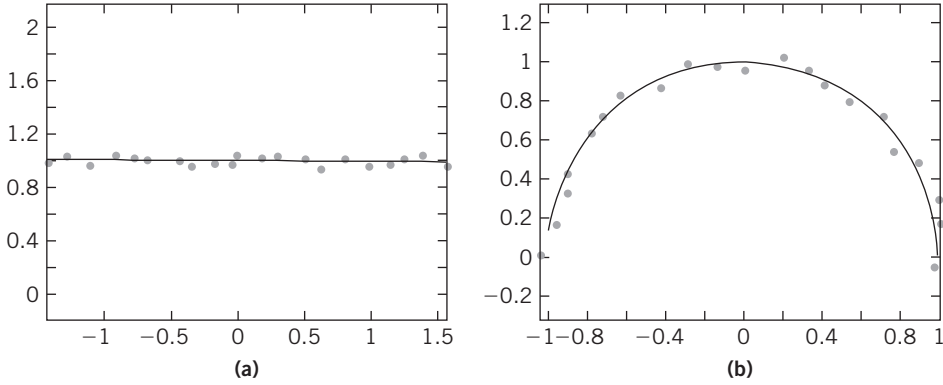
In this case, observe that the projection in the 2-dimensional subspace retains the separation among the classes at a satisfactory level.

Finally, keep in mind that there are data sets where the dimensionality reduction from projection in any subspace of the original space may cause substantial loss of class discrimination. In such cases, nonlinear techniques may be useful. 

3.5 THE KERNEL PCA

The three methods considered so far for dimensionality reduction are *linear*. A subspace of low dimension is first constructed as, for example, the span of the m dominant directions in the original \mathcal{R}^l , $l > m$ space.

The choice of dominant directions depends on the method used. In a second stage, all vectors of interest in \mathcal{R}^l are (linearly) projected in the low-dimensional subspace. Such techniques are appropriate whenever our data in \mathcal{R}^l lie (approximately) on a linear manifold (e.g., hyperplane). However, in many

**FIGURE 3.4**

(a) The 2-dimensional data points lying (approximately) on a line (linear manifold). (b) The 2-dimensional data points lying (approximately) on a semicircle (nonlinear manifold).

cases the data are distributed around a lower-dimensional manifold, which is not linear (e.g., around a circle or a sphere in a 3-dimensional space).

Figures 3.4(a,b) show two cases where data in the 2-dimensional space lie (approximately) on a linear and a nonlinear manifold, respectively. Both manifolds are 1-dimensional since a straight line and the circumference of a circle can be parameterized in terms of a *single* parameter.

The kernel PCA is one technique for dimensionality reduction when the data lie (approximately) on a nonlinear manifold. According to the method, data are first mapped into a high-dimensional space via a *nonlinear* mapping:

$$x \in \mathcal{R}^l \mapsto \phi(x) \in H$$

PCA is then performed in the new space H , chosen to be an RKHS. The inner products can be expressed in terms of the kernel trick, as discussed in Section 2.5.

Although a (linear) PCA is performed in the RKHS space H , because of the nonlinearity of the mapping function $\phi(\cdot)$, the method is equivalent to a nonlinear function in the original space. Moreover, since every operation can be expressed in inner products, the explicit knowledge of $\phi(\cdot)$ is not required. All that is necessary is to adopt the kernel function that defines the inner products. More details are given in [Theo 09, Section 6.7.1].

To use the kernel PCA, type

$$[s, V, Y] = \text{kernel_PCA}(X, m, \text{choice}, \text{para})$$

where

X is an $l \times N$ matrix whose columns contain the data vectors,

m is the number of (significant) principal components that will be considered,

choice is the type of kernel function to be used ('pol' for polynomial, 'exp' for exponential),

para is a 2-dimensional vector containing the parameters for the kernel function; for polynomials it is $(x^T y + \text{para}(1))^{\text{para}(2)}$ and for exponentials it is $\exp(-(x - y)^T (x - y) / (2\text{para}(1)^2))$,

s is an N -dimensional vector that contains the computed eigenvalues after applying the kernel PCA, V is an $N \times N$ matrix whose columns are the eigenvectors corresponding to the principal components of the Gram matrix, \mathcal{K} , which is involved in the kernel PCA [Theo 09, Section 6.7.1], Y is an $m \times N$ dimensional matrix that contains the projections of the data vectors of X on the subspace spanned by the m principal components.

Example 3.5.1. This example illustrates the rationale behind the kernel PCA. However, since kernel PCA implies, first, a mapping to a higher-dimensional space, visualization of the results is generally not possible. Therefore, we will “cheat” a bit and use a mapping function $\phi(\cdot)$ that does not correspond to a kernel function $k(\cdot, \cdot)$. (After all, the mapping to an RKHS is required only for the computational tractability needed to compute inner products efficiently.) However, this function allows transformation of a 2-dimensional space, where our data points lie around a nonlinear manifold, into another 2-dimensional space, where the data points are mapped around a linear manifold.

Consider a data set X consisting of 21 2-dimensional points of the form $x_i = (x_i(1), x_i(2)) = (\cos \theta_i + s_i, \sin \theta_i + s'_i)$, where $\theta_i = (i - 1) * (\pi/20)$, $i = 1, \dots, 21$ and s_i, s'_i are random numbers that stem from the uniform distribution in $[-0.1, 0.1]$ (see Figure 3.5(a)). These points lie around the semicircle modeled by $x^2(1) + x^2(2) = 1$, which is centered at the origin and is positive along the x_2 axis.

The mapping function $\phi(\cdot)$ is defined as

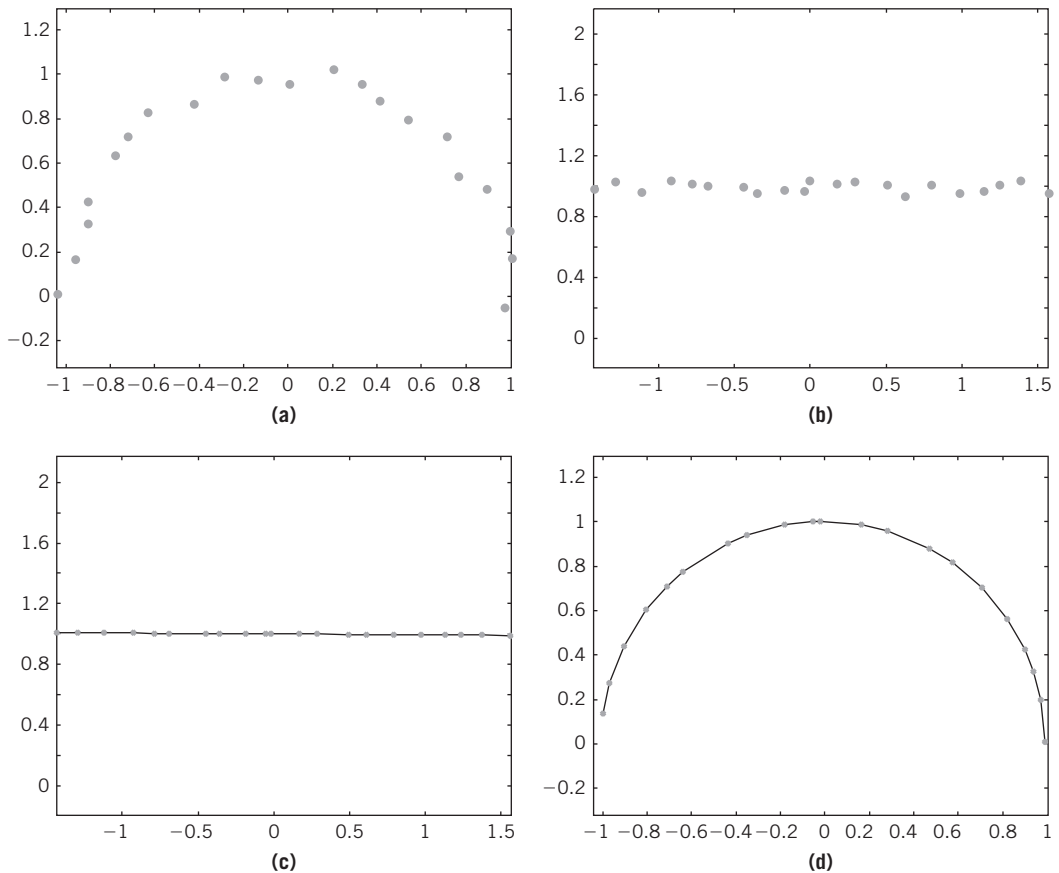
$$\phi\left(\begin{bmatrix} x(1) \\ x(2) \end{bmatrix}\right) = \begin{bmatrix} \tan^{-1}\left(\frac{x(2)}{x(1)}\right) \\ \sqrt{x^2(1) + x^2(2)} \end{bmatrix}$$

By applying $\phi(\cdot)$ on the data set X , we get the set $Y = \{y_i = \phi(x_i), i = 1, \dots, 21\}$, which is illustrated in Figure 3.5(b). Note that the points of Y lie around a linear manifold (straight line) in the transformed domain. Then we apply linear PCA on Y and keep only the first principal component, since almost all of the total variance of the data set (99.87%) is retained along this direction.¹ Let $Z = \{z_i = [z_i(1), z_i(2)]^T, i = 1, \dots, 21\}$ be the set containing the projections of y_i 's on the first principal component in the transformed space (see Figure 3.5(c)). Mapping z_i 's back to the original space via the inverse function of $\phi(\cdot)$, which is given by

$$\phi^{-1}\left(\begin{bmatrix} z(1) \\ z(2) \end{bmatrix}\right) = \begin{bmatrix} z(2) \cos z(1) \\ z(2) \sin z(1) \end{bmatrix} \equiv \begin{bmatrix} x'(1) \\ x'(2) \end{bmatrix}$$

the points $(x'(1), x'(2))$ are obtained that lie on the semicircle defined by $x^2(1) + x^2(2) = 1$, with $x(2) > 0$ (see Figure 3.5(d)).

¹Linear PCA requires subtraction of the mean of the data vectors (which is performed in the *pca_fun* function). In our case, this vector equals $[0, 1]^T$. After PCA, this vector is added to each projection of the points along the direction of the first principal component.

**FIGURE 3.5**

Example 3.5.1: (a) Data set in the original space (around a semicircle). (b) Data set in the transformed space (around a straight line). (c) Direction corresponding to the first principal component and the images (projections) of the points on it in the transformed space. (d) Images of the points in the original space.

Remark

- Observe that the nonlinear mapping transformed the original manifold to a linear one. Thus, the application of the (linear) PCA on the transformed domain is fully justified. Of course, in the general case one should not expect to be so “lucky”—that is, to have the transformed data lying across a linear manifold.

In the next example we consider kernel PCA in the context of a classification task. More specifically, the goal is to demonstrate the potential of the kernel PCA to transform a nonlinear classification problem, in the (original) l -dimensional space, into a linear one in an $m (< l)$ dimensional space. If this is achieved, the original classification problem can be solved in the transformed space by a linear classifier.

Example 3.5.2

1. Generate two data sets X and X_{test} , each one containing 200 3-dimensional vectors. In each, the first $N_1 = 100$ vectors stem from class 1, which is modeled by the uniform distribution in $[-0.5, 0.5]^3$, while the rest, $N_2 = 100$, stem from class -1 and lie around the sphere with radius $r = 2$ and centered at the origin. The N_2 points for each data set are generated as follows.

Randomly select a pair of numbers, $x(1)$ and $x(2)$, that stem from the uniform distribution in the range $[-2, 2]$, and check if $x^2(1) + x^2(2)$ is less than r^2 . If this is not the case, choose a different pair. Otherwise, generate two points of the sphere as $(x(1), x(2), \sqrt{r^2 - x^2(1) - x^2(2)} + \varepsilon_1)$ and $(x(1), x(2), -\sqrt{r^2 - x^2(1) - x^2(2)} + \varepsilon_2)$, where ε_1 and ε_2 are random numbers that stem from the uniform distribution in the interval $[-0.1, 0.1]$.

Repeat this procedure $N_2/2$ times to generate N_2 points. Also generate the vectors y and y_{test} which contain the class labels of the points of X and X_{test} , respectively. Then plot the data sets.

2. Perform kernel PCA on X using the exponential kernel function with $\sigma = 1$ and keep only the first two most significant principal components. Project the data points of X onto the subspace spanned by the two principal components and let Y be the set of these projections (plot Y).
3. Design a least squares (LS) classifier based on Y .
4. Evaluate the performance of the previous classifier based on X_{test} as follows: For each vector in $x \in X_{test}$, determine its projection onto the space spanned by the two most significant principal components, computed earlier, and classify it using the LS classifier generated in [step 3](#). Assign x to the class where its projection has been assigned. Plot the projections of the points of X_{test} onto the subspace spanned by the two principal components along with the straight line that realizes the classifier.
5. Repeat [steps 2](#) through [4](#) with $\sigma = 0.6$.

Solution. Take the following steps:

Step 1. To generate the points of X that belong to class 1, type

```
rand('seed',0)
noise_level=0.1;
n_points=[100 100]; %Points per class
l=3;
X=rand(l,n_points(1)) - (0.5*ones(l,1))*ones(1,n_points(1));
```

To generate the points of X that belong to class -1 , type

```
r=2; %Radius of the sphere
for i=1:n_points(2)/2
    e=1;
    while(e==1)
        temp=(2*r)*rand(1,1)-r;
        if(r^2-sum(temp.^2)>0)
```

```

        e=0;
    end
end
t=sqrt(r^2-sum(temp.^2))+noise_level*(rand-0.5);
qw=[temp t; temp -t]';
X=[X qw];
end

```

The data set X_{test} is generated similarly (use the value 100 as the seed for the *rand* function). To define the class labels of the data vectors, type

```

[1,N]=size(X);
y=[ones(1,n_points(1)) -ones(1,n_points(2))];
y_test=[ones(1,n_points(1)) -ones(1,n_points(2))];

```

To plot the data set X , type²

```

figure(1), plot3(X(1,y==1),X(2,y==1),X(3,y==1),'r.',...
X(1,y==-1),X(2,y==-1),X(3,y==-1),'b+')
figure(1), axis equal

```

X_{test} is plotted similarly. Clearly, the two classes are nonlinearly separable.

Step 2. To perform kernel PCA with kernel exponential and $\sigma = 1$, type

```

[s,V,Y]=kernel_PCA(X,2,'exp',[1 0]);

```

Note that Y contains in its columns the images of the points of X on the space spanned by the first two principal components, while V contains the respective principal components.

To plot Y , type

```

figure(2), plot(Y(1,y==1),Y(2,y==1),'r.',Y(1,y==-1),Y(2,y==-1),'b+')

```

Step 3. To design the LS classifier based on Y , type

```

w=SSErr([Y; ones(1,sum(n_points))],y,0);

```

Note that each column vector of Y has been augmented by 1. The resulting w is [33.8001, 2.4356, -0.8935].

Step 4. Type the following to generate the Y_{test} set, containing in its columns the projections of the vectors of X_{test} to the space spanned by the principal components:

```

[1,N_test]=size(X_test);

```

²Use the Rotate 3D button to observe the data set from different angles.

```

Y_test=[];
for i=1:N_test
    [temp]=im_point(X_test(:,i),X,V,2,'exp',[1 0]);
    Y_test=[Y_test temp];
end

```

To classify the vectors of X_{test} (Y_{test}) and compute the classification error, type

```

y_out=2*(w'*[Y_test; ones(1,sum(n_points))]>0)-1;
class_err=sum(y_out.*y_test<0)/sum(n_points);

```

Figure 3.6(a) shows the Y_{test} set together with the line that corresponds to the linear classifier. This is produced by typing

```

figure(6), plot(Y_test(1,y==1),Y_test(2,y==1),'r.',...
Y_test(1,y==-1),Y_test(2,y==-1),'b+')
figure(6), axis equal
% Plotting the linear classifier (works only if w(1)~=0)
y_lin=[min(Y_test(2,:)) max(Y_test(2,:))];
x_lin=[(-w(3)-w(2)*y_lin(1))/w(1) (-w(3)-w(2)*y_lin(2))/w(1)];
figure(6), hold on
figure(6), line(x_lin,y_lin)

```

Step 5. For this step, repeat the codes given in steps 2 through 4. Now in the call of the kernel PCA function (step 2), [1, 0] is replaced by [0.6, 0] (see also Figure 3.6(b)).

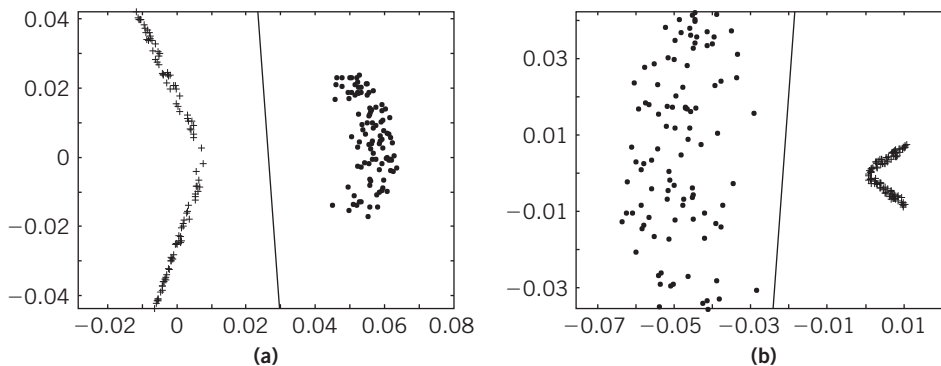


FIGURE 3.6

Y_{test} set produced in step 4 and linear classifier determined in step 3 of Example 3.5.2 for $\sigma = 1$ (a) and $\sigma = 0.6$ (b).

The previous steps having been performed, three conclusions can be drawn:

- First, kernel PCA *may* lead to mappings in lower-dimensional spaces, where the involved classes can be linearly separated, even though this is not the case in the original space. This cannot be achieved with linear PCA.
- Second, the choice of the kernel function parameters is critical. To verify this, try [step 5](#) with $\sigma = 3$. In this case, the arrangement of the classes in the transformed space looks very similar to the arrangement in the original space.
- Third, for this problem the two classes remain linearly separable even in the 1-dimensional space as defined by the first principal component. ■

Example 3.5.3

1. Generate two data sets X_1 and X_2 as follows:

- X_1 consists of 300 2-dimensional points. The first $N_1 = 100$ points stem from class 1, which is modeled by the uniform distribution in the region $[-2.5, -1.5] \times [-0.5, 0.5]$; the next $N_2 = 100$ points stem from class 2, which is modeled by the uniform distribution in the area $[0.5, 1.5] \times [-2.5, -1.5]$; and the final, $N_3 = 100$ points stem from class 3 and lie around the circle C with radius $r = 4$ and centered at the origin.

More specifically, the last N_3 points are generated as follows: The points of the form $x_i = -r + \frac{2r}{N_2/2-1}i$, $i = 0, \dots, N_2/2 - 1$ in the x axis are selected (they lie in the interval $[-2, 2]$). For each x_i the quantities $v_i^1 = \sqrt{r^2 - x_i^2} + \varepsilon_i^1$ and $v_i^2 = -\sqrt{r^2 - x_i^2} + \varepsilon_i^2$ are computed, where $\varepsilon_i^1, \varepsilon_i^2$ stem from the uniform distribution in the interval $[-0.1, 0.1]^T$. The points (x_i, v_i^1) and (x_i, v_i^2) , $i = 0, \dots, N_2 - 1$, lie around C .

- X_2 consists of 300 2-dimensional points. Points $N_1 = 100$, $N_2 = 100$, and $N_3 = 100$ stem from classes 1, 2, and 3, respectively. They lie around the circles centered at the origin and having radii $r_1 = 2$, $r_2 = 4$, and $r_3 = 6$, respectively (the points of each class are generated by adopting the procedure used for generating the last N_3 points of the previous data set, X_1).

Generate the vectors y_1 and y_2 , which contain the class labels of the data points of the sets X_1 and X_2 , respectively. Plot X_1 and X_2 .

2. Perform kernel PCA on X_1 , using the exponential kernel function with $\sigma = 1$ and keep only the first two principal components. Determine and plot the set Y_1 , which contains the projections of the data points of X_1 onto the subspace spanned by the two principal components. Repeat the steps for X_2 and draw conclusions.

Solution. Take the following steps:

Step 1. To generate the data set X_1 , type

```
rand('seed',0)
noise_level=0.1;
%%%
n_points=[100 100 100];
```

```

l=2;
% Production of the 1st class
X1=rand(l,n_points(1))- [2.5 0.5]*ones(1,n_points(1));
% Production of the 2nd class X1=[X1
rand(l,n_points(2))- [-0.5 2.5]*ones(1,n_points(2))];
% Production of the 3rd class
c1=[0 0];
r1=4;
b1=c1(1)-r1;
b2=c1(1)+r1;
step=(b2-b1)/(n_points(2)/2-1);
for t=b1:step:b2
    temp=[t c1(2)+sqrt(r1^2-(t-c1(1))^2)+noise_level*(rand-0.5);...
    t c1(2)-sqrt(r1^2-(t-c1(1))^2)+noise_level*(rand-0.5)'];
    X1=[X1 temp];
end

```

To generate the vector of the labels y_1 , type

```
y1=[ones(1,n_points(1)) 2*ones(1,n_points(2)) 3*ones(1,n_points(3))];
```

To plot the data set X_1 , type

```

figure(1), plot(X1(1,y1==1),X1(2,y1==1),'r.',...
X1(1,y1==2),X1(2,y1==2),'bx',...
X1(1,y1==3),X1(2,y1==3),'go')

```

Step 2. To perform kernel PCA on X_1 with the exponential kernel function ($\sigma = 1$), type

```

m=2;
[s,V,Y1]=kernel_PCA(X1,m,'exp',[1 0]);

```

To plot Y_1 , type

```

figure(2), plot(Y1(1,y1==1),Y1(2,y1==1),'r.',...
Y1(1,y1==2),Y1(2,y1==2),'bx',...
Y1(1,y1==3),Y1(2,y1==3),'go')

```

Work similarly for X_2 .

In Y_1 the classes are linearly separable, but this is not the case in Y_2 . That is, kernel PCA does not necessarily transform nonlinear classification problems into linear ones. ■

Exercise 3.5.1

In this exercise, we see how the original space is transformed using kernel PCA. To ensure visualization of the results, we consider a 2-dimensional problem, which will be mapped to the 2-dimensional space spanned by the first two principal components that result from the kernel PCA. More specifically, go through the following steps:

Step 1. Generate a data set X , which contains 200 2-dimensional vectors. $N_1 = 100$ vectors stem from class 1, which is modeled by the uniform distribution in $[-0.5, 0.5]^2$; $N_2 = 100$ vectors stem from class -1 and lie around the circle with radius 1 and centered at the origin. The N_2 points are produced as the last N_3 points of the X_1 data set in [Example 3.5.3](#).

Step 2. Repeat [step 1](#) from [Example 3.5.2](#), with $\sigma = 0.4$.

Step 3. Repeat [step 2](#) from [Example 3.5.2](#).

Step 4. Consider the points x of the form $(-2 + i * 0.1, -2 + j * 0.1)$, $i, j = 0, \dots, 40$ (these form a rectangular grid in the region $[-2, 2]^2$) and their images in the space spanned by the two principal components, determined in [step 2](#). Classify the image of each point x using the LS classifier designed in [step 3](#) and produce two figures. The first corresponds to the transformed space and an image point is plotted with a green o if it is classified to the first class (+1) and with a cyan x if it is classified to the second class (-1). The second figure corresponds to the original space and the corresponding points x are drawn similarly. Observe how the implied nonlinear transformation deforms the space.

Step 5. Repeat [steps 2](#) through [4](#), with $\sigma = 1$.

Hint

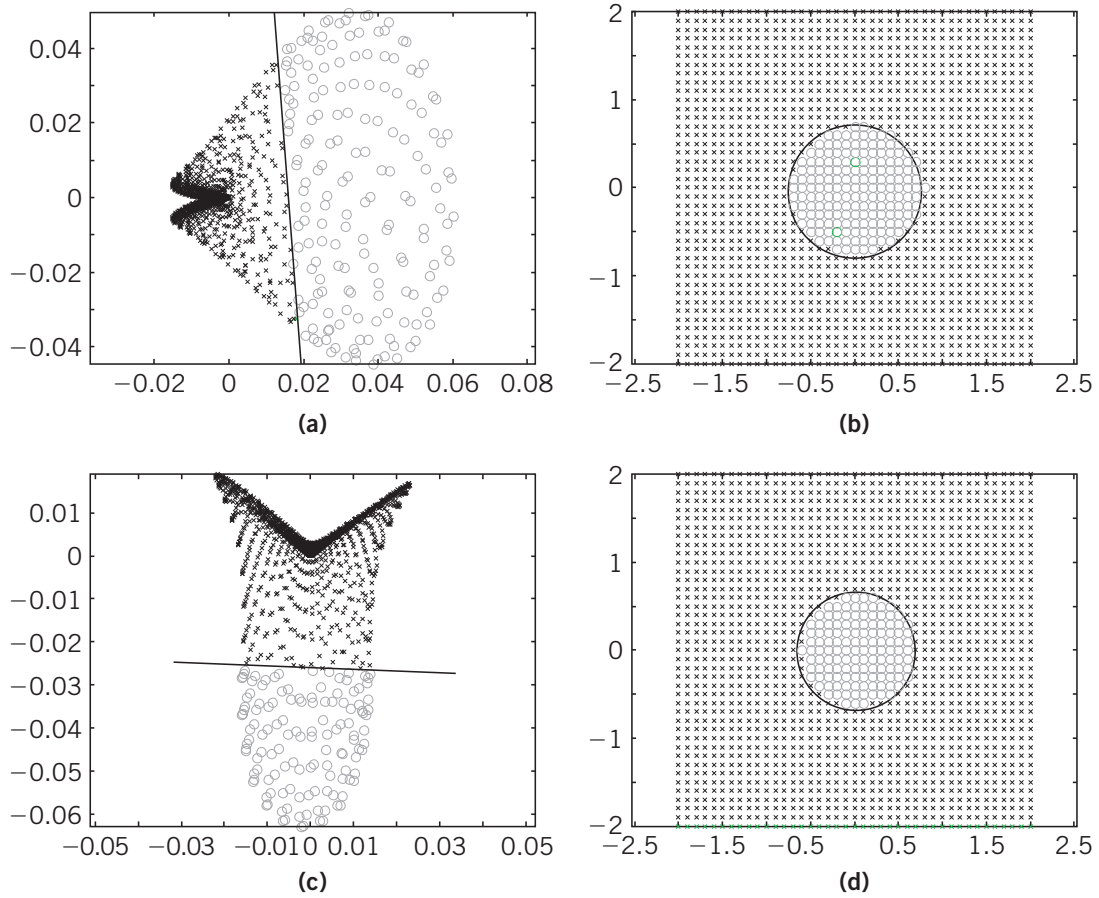
To generate X , work as in [Example 3.5.3](#). To define the class labels of the data vectors and to plot X , work as in [Example 3.5.2](#). To perform kernel PCA and define the linear classifier, also work as in [Example 3.5.2](#). To carry out [step 4](#), use the MATLAB function `plot_orig_trans_kPCA` by typing

```
m=2;
choice='exp';
para=[0.4 0];
reg_spec=[-2 0.1 2; -2 0.1 2];
fig_or=5;
fig_tr=6;
plot_orig_trans_kPCA(X,V,m,choice,para,w,reg_spec,fig_or,fig_tr)
```

The results are shown in [Figure 3.7](#). Observe how the points inside the circle (denoted by o) in the original space are expanded in the transformed space. Also notice the difference between the transformed spaces for $\sigma = 0.4$ and $\sigma = 1$, respectively ([Figure 3.7\(a, c\)](#)).

3.6 LAPLACIAN EIGENMAP

The Laplacian eigenmap method belongs to the family of so-called graph-based methods for dimensionality reduction. The idea is to construct a graph, $G(V, E)$, where nodes correspond to the data points x_i , $i = 1, 2, \dots, N$. If two points are close, the corresponding nodes are connected via an edge, which is weighted accordingly. The closer two points are the higher the value of the weight of the respective edge.

**FIGURE 3.7**

(a) Linear classifier in the space spanned by the first two principal components resulting from the kernel PCA, using exponential kernel function with $\sigma = 0.4$, from [Exercise 3.5.1](#). (b) Equivalent of the linear classifier in the original space. (c) Linear classifier in the space spanned by the first two principal components using the exponential kernel function with $\sigma = 1$, from [Exercise 3.5.1](#). (d) Equivalent of the linear classifier in the original space. Observe the influence of the value of σ .

Closeness is determined via a threshold value. For example, if the squared Euclidean distance between two points, $\|x_i - x_j\|^2$, is less than a user-defined threshold, say e , the respective nodes are connected and are said to be *neighbors*. The corresponding weight can be defined in different ways. A common choice for the weights $W(i, j)$ between the nodes i and j is, for some user-defined variable σ^2 ,

$$W(i, j) = \begin{cases} \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right), & \text{if } \|x_i - x_j\|^2 < e \\ 0 & \text{otherwise} \end{cases}$$

Thus, the graph encodes the *local information* in the original, high-dimensional, space \mathcal{R}^l .

It is further assumed that the data $x_i \in \mathcal{R}^l$ lie on a *smooth* manifold of dimension m . The value of m is assumed to be known. For example, the original data may live in the 3-dimensional space \mathcal{R}^3 , but lie around a sphere. The latter is a 2-dimensional smooth manifold since two parameters suffice to describe the surface of a sphere.

The goal of the method is to obtain an m -dimensional representation of the data so that the *local neighborhood information* in the manifold is *optimally* retained. In this way, the local geometric structure of the manifold is reflected in the obtained solution.

The Laplacian eigenmap turns out to be equivalent to an eigenvalue/eigenvector problem of the so-called *Laplacian matrix*. This matrix encodes the local information as described by the weights of the edges in the graph [Theo 09, Section 6.7.2].

To use the Laplacian eigenmap, type

$$y = \text{lapl_eig}(X, e, \text{sigma2}, m)$$

where

e is the value of the threshold,

sigma2 is the σ^2 parameter,

y is an $m \times N$ matrix whose i th column defines the projection of the i th data vector to the m -dimensional subspace.

Example 3.6.1. Generate a 3-dimensional Archimedes spiral as a pack of 11 identical 2-dimensional Archimedes spirals, one above the other. A 2-dimensional spiral is described in polar coordinates by the equation $r = a\theta$, where a is a user-defined parameter. In our case, the points of a 3-dimensional spiral are generated as follows: For θ , take the values from θ_{init} to θ_{fin} with step θ_{step} and compute

$$\begin{aligned} r &= a\theta \\ x &= r \cos \theta \\ y &= r \sin \theta \end{aligned}$$

The 11 points of the form (x, y, z) , where $z = -1, -0.8, -0.6, \dots, 0.8, 1$, are points of the spiral. Use $a = 0.1$, $\theta_{init} = 0.5$, $\theta_{fin} = 2.05 * \pi$, $\theta_{step} = 0.2$.

Plot the 3-dimensional spiral so that all points of the same 2-dimensional spiral are plotted using the same symbol and all groups of 11 points of the form (x, y, z) , where x and y are fixed and z takes the values $-1, -0.8, -0.6, \dots, 0.8, 1$, are plotted using the same color.

1. Perform the Laplacian eigenmap on the points of the previous data set for manifold dimension $m = 2$. Plot the results.
2. Perform linear PCA on the same data set, using the first two principal components. Plot the results.
3. Compare the results obtained in [steps 1 and 2](#).

Solution. To generate and plot the 3-dimensional spiral, call the function `spiral_3D` by typing

```
a=0.1;
init_theta=0.5;
```

```

fin_theta=2.05*pi;
step_theta=0.2;
plot_req=1; % Request for plot the spiral
fig_id=1;   % Number id of the figure
% Producing the 3D spiral
[X,color_tot,patt_id]=spiral_3D(a,init_theta,...
fin_theta,step_theta,plot_req,fig_id);
[l,N]=size(X);

```

Use Rotate 3D to see the spiral from different viewpoints.

Do the following:

Step 1. To perform the Laplacian eigenmap, call the function *lapl_eig* by typing

```

e=0.35;
sigma2=sqrt(0.5);
m=2;
y=lapl_eig(X,e,sigma2,m);

```

To plot the results, type

```

figure(2), hold on
for i=1:N
    figure(2), plot(y(1,i),y(2,i),patt_id(i),'Color',color_tot(:,i))
end

```

Step 2. To perform linear PCA, call the function *pca_fun* by typing:

```

[eigenval,eigenvec,explain,Y]=pca_fun(X,m);

```

To plot the results, type

```

figure(3), hold on
for i=1:N
    figure(3), plot(Y(1,i),Y(2,i),patt_id(i),'Color',color_tot(:,i))
end

```

Step 3. Observing MATLAB figure 1 on the computer screen, notice how the color of each 2-dimensional spiral varies from red to yellow to green to cyan to blue. On the mapping produced by the Laplacian eigenmap method (MATLAB figure 2), the following two comments are in order: Each 2-dimensional spiral is “stretched” so that the points are placed on a line segment (horizontal direction); in each vertical direction the succession of the colors observed in MATLAB figure 1 is the same as that observed in MATLAB figure 2. Thus, for the given choice of parameters for the Laplacian eigenmap, the method successfully “unfolds” the 3-dimensional spiral.

The linear PCA (MATLAB figure 3) also succeeds in “stretching” each 2-dimensional spiral so that the points are placed on a line segment (horizontal direction). However, the succession of colors

in the vertical direction is not the same as that observed in MATLAB figure 1 (green, yellow, red, cyan, blue). This indicates that after the projection produced by PCA, points that are distant from each other in the 3-dimensional space lie close in the 2-dimensional representation (see, for example, the red and cyan points in the original 3-dimensional space and in the reduced 2-dimensional space).

Finally, the results obtained by the Laplacian eigenmap are sensitive to the choice of parameters. If, for example, the previous experiment is performed for $e = 0.5$ and $\sigma^2 = 1$, the Laplacian eigenmap fails to completely “unfold” the spiral (in this case, red is close to green). However, this unfolding, although not perfect, is still better than the one produced by linear PCA. In general, extensive experimentation is required before choosing the right values for the parameters involved in the Laplacian eigenmap method. ■

Exercise 3.6.1

1. Generate a “cut cylinder” of radius $r = 0.5$ in the 3-dimensional space as a pack of 11 identical “cut circles,” one above the other, as in [Example 3.6.1](#). For the j th circle, with center $c_j = [c_{j1}, c_{j2}]^T$, the following points are selected:

- $(x_i, c_{j2} + \sqrt{r^2 - (x_i - c_{j1})^2})$ for x_i ranging from $c_{j1} - r$ to $c_{j1} + r$, with step $(2r)/(N - 1)$.
- $(x_i, c_{j2} - \sqrt{r^2 - (x_i - c_{j1})^2})$ for x_i ranging from $c_{j1} + r$ down to $(c_{j1} - r)/4$, with the previously chosen step, where N is the number of points on which x_i is sampled in the range $[c_{j1} - r, c_{j1} + r]$.

Plot the cut cylinder so that all points of the same 2-dimensional cut circle are plotted using the same symbol, and all groups of 11 points of the form (x, y, z) , where x and y are fixed and z takes the values $-1, -0.8, -0.6, \dots, 0.8, 1$ are plotted using the same color.

2. Perform the Laplacian eigenmap on the points of the previous data set for manifold dimension $m = 2$. Plot the results.
3. Perform linear PCA on the same data set using the first two principal components and plot the results.
4. Compare the results obtained in [steps 2 and 3](#).

Hints

1. To generate and plot the 3-dimensional cut cylinder, call the function `cut_cylinder_3D` by typing

```
r=0.5;
center=[0 0];
N=25;
plot_req=1; %Request for plot the cylinder
fig_id=1; %Number id of the figure
% Producing the 3D cut cylinder
[X,color_tot,patt_id]=cut_cylinder_3D(r,center,N,plot_req,...
fig_id);
[1,N]=size(X);
```

Use `Rotate 3D` to observe the cut cylinder from different viewpoints.

2. Apply [step 1](#) of [Example 3.6.1](#), using the same parameters for the Laplacian eigenmap method.

3. Apply [step 2](#) of [Example 3.6.1](#).
4. Observe that, once again, the Laplacian eigenmap gives better results compared with those of the linear PCA. However, this is not the case when the cut cylinder has a radius equal to 5. (Why? Compare the height of the cylinder with its radius.)