

## Pattern Analysis

A pattern is a general concept and means a form, a template, a model, or, more abstractly, a set of rules or a data structure. Pattern recognition is the detection of underlying patterns in data. By pattern analysis we mean analysis of data on the basis of patterns, involving pattern recognition, classification, modeling and statistics.

An important class of patterns is those related to images. Images are an interesting form of biomedical data, and looking for patterns in images can give useful information. Also, the analysis of images provide nice, comprehensive examples of pattern analysis algorithms.

In this chapter we cover some pattern analysis algorithms. Pattern analysis is a broad field, with many applications and strong links to information processing, computer science, biometrics and biostatistics. The size of bioinformatic data and databases excludes most manual operations on this data, and the successful extraction of useful information relies heavily on the effectiveness of automatic browsing, searching, and linking. Combining of browsing bioinformatic data files with pattern analysis algorithms, such as automatic classification, has great potential and can lead to very interesting findings.

### 4.1 Feature Extraction

A feature is a mapping from a pattern space or image space to a feature space i.e., a space of numbers or vectors. Examples of features are the positions of image fragments, the areas of parts of images, lengths of contours of objects in images, the numbers of occurrences of certain strings in sequences, and the coefficients of series expansions (Taylor, Fourier, etc.) of functions associated to images or patterns related to experiments performed.

Feature extraction is one of the initial steps of pattern analysis. The definition of features for a given situation is a crucial element in the art of construction of pattern analysis algorithms. If the features defined correspond well to the type of information one is looking for, then it is likely that the

whole pattern analysis system will perform satisfactorily. However, it can also happen that the relations between the defined features and patterns we are after are so ambiguous that the pattern analysis system will eventually fail.

In the systems of measurements that are performed in molecular biology, biochemistry or genetics, it often happens that experiments and the analysis of them provide hundreds, thousands, or even more features, and the problem is to reduce their dimensionality or to look for a hierarchy in the data.

## 4.2 Classification

The classification problem for patterns involves setting discrimination rules, based on the features and a knowledge of the classes of the patterns. In the sequel, we will treat terms “classes” and “states of an experiment” as synonymous. The simplest formulation of the classification task is as follows. There are two possible states of the experiment coded, for example, as 0 for normal and 1 for disease. The feature extraction system has already been designed and we have  $x_1, x_2, \dots, x_n$ , which are feature vectors known to correspond to state 0, and  $x_{n+1}, x_{n+2}, \dots, x_{n+m}$ , which are feature vectors known to correspond to state 1. The feature space is  $k$ -dimensional, i.e.,  $x \in R^k$ . The problem is to find a scalar function (a classifier)  $f(x)$  defined on the feature space such that  $f(x_i) = 0$  for all  $i = 1, \dots, n$  and  $f(x_i) = 1$  for all  $i = n + 1, \dots, n + m$ . If we succeed in the construction of the classifier, it will allow automated classification of experimental states on the basis of extracted features.

A more general formulation of the classification problem involves more than two classes. Also, it may be either impossible or unsuitable to obtain a perfect discrimination. As a result of noise in the data, the knowledge about the classes may be erroneous, and so it can be more reasonable to allow for some classification errors.

### 4.2.1 Linear Classifiers

Let us code the states of the experiment by  $-1$  and  $1$  instead of  $0$  and  $1$ . This change of coding is motivated by our aim of using a “sign” function, which returns values  $-1$  and  $1$ . By a linear classifier function or linear discriminant function, we mean a function

$$f(x) = \text{sign}(w^T x + w_0). \quad (4.1)$$

In the above “sign” the a sign function, which returns  $-1$  or  $1$  depending on whether the argument is negative or positive;  $x$  is a feature vector, which is a  $k$ -dimensional, column vector,  $w^T$  is a  $k$ -dimensional row vector of weights,  $w_0$  is a scalar, and the superscript  $T$  denotes, as usual, vector transposition. The function (4.1) could also be called an affine classifier owing to the occurrence of the offset term  $w_0$ . The geometric locus in the space  $x \in R^k$

$$L = \{x : w^T x + w_0 = 0\} \quad (4.2)$$

is called a separating hyperplane.

With the data  $x_1, x_2, \dots, x_n$  and  $x_{n+1}, x_{n+2}, \dots, x_{n+m}$ , belonging to two classes as described above, the problem of the construction of the linear classifier (4.1) can be formulated as follows. Find a vector  $w \in R^k$  and a scalar  $w_0$  such that  $w^T x_i + w_0 < 0$  for all  $i = 1, \dots, n$  and  $w^T x_i + w_0 > 0$  for all  $i = n+1, \dots, n+m$ . Using matrix and vector notation, these  $n+m$  inequalities can be written as

$$M \begin{bmatrix} w \\ w_0 \end{bmatrix} < 0 \quad (4.3)$$

where  $[w^T \ w_0]^T$  is a  $k+1$ -dimensional column vector, and  $M$  is a matrix defined as follows

$$M = \begin{bmatrix} x_1^T & 1 \\ \vdots & \vdots \\ x_n^T & 1 \\ -x_{n+1}^T & -1 \\ \vdots & \vdots \\ -x_{n+m}^T & -1 \end{bmatrix}. \quad (4.4)$$

As we can see, the problem of constructing a linear classifier reduces to the problem of looking for a vector  $[w^T \ w_0]^T$  which satisfies the system of linear inequalities (4.3). The system (4.3) is homogeneous, which means that if  $[w^T \ w_0]^T$  solves it, then any other vector  $\alpha[w^T \ w_0]^T$  obtained by multiplication by a positive constant  $\alpha > 0$  is also a solution. So, equivalently to (4.3) we can analyze

$$M \begin{bmatrix} w \\ w_0 \end{bmatrix} \leq -\mathbf{1}, \quad (4.5)$$

where  $\mathbf{1}$  means a vector with all entries equal to one.

One method for solving (4.3) or (4.5) with respect to  $[w^T \ w_0]^T$  is by using the linear programming algorithm mentioned in Chap. 5. One can easily define a linear programming problem with the property that its solution solves the system of inequalities (4.3). One possibility is as follows:

$$\max z \quad (4.6)$$

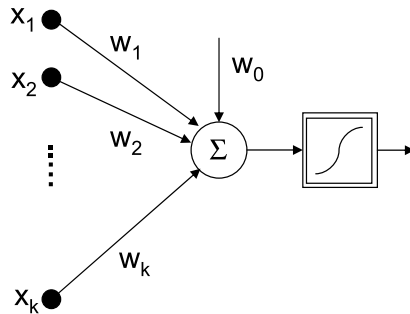
subject to

$$M \begin{bmatrix} w \\ w_0 \end{bmatrix} \leq -\mathbf{1}z \quad (4.7)$$

and

$$0 \leq z \leq 1. \quad (4.8)$$

In the above formulation, the variables of the linear programming problem are  $w$ ,  $w_0$ , and  $z$ , (one scalar variable  $z$  has been added). If the optimal solution



**Fig. 4.1.** Graphical representation of a model of an artificial neuron

to (4.6)–(4.8) is  $z_{\text{opt}} = 1$ , then a solution to (4.3) exists. If  $z_{\text{opt}} = 0$ , then the system (4.3) is infeasible.

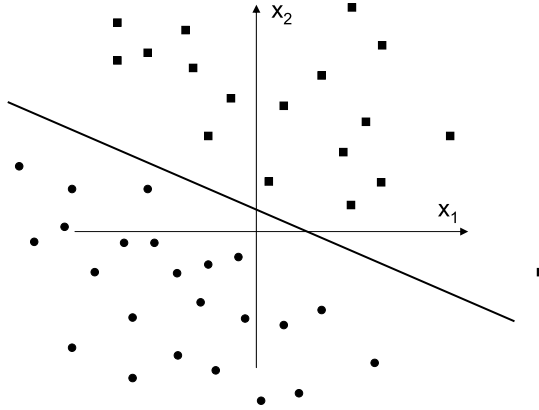
The method of choosing weights  $w_1, \dots, w_k$  by solving linear programming problem, as shown above, is very efficient. However, many other methods are also in use. One group of approaches to adjusting the weights in classifiers uses iterative procedures [67], where weights are modified step by step and the procedure stops when solution to (4.3) is achieved. Such procedures are called training of the classifier. For single classifiers (4.1), they are only toy algorithms, but become important when classifiers are organized in larger structures, namely artificial neural networks, as we outline below.

#### 4.2.2 Linear Classifier Functions and Artificial Neurons

The linear classifier functions (4.1) are closely related to artificial neurons. A model of an artificial neuron can be represented graphically as shown in Fig. 4.1. Signals (the elements of the vector  $x$ )  $x_1, \dots, x_k$  are multiplied by weights  $w_1, \dots, w_k$ , summed up with offset  $w_0$ . This step of signal transformation is the same as in the linear classifier (4.1). The threshold element “sign” which appears in (4.1) is, in the neuron in Fig. 4.1 replaced by a smooth function, for example a sigmoid [67]. The function in the output block of the artificial neuron is called the neuron activation function. This function is chosen to be a smooth approximation of a thresholding element, i.e., a sigmoid, logistic, arctan function, etc. The smoothness makes neuron activation functions more physically sound and, more importantly, makes it possible to construct training algorithms based on derivatives.

#### 4.2.3 Artificial Neural Networks

As discussed above, linear classifiers or single neurons can perform linear discrimination; in other words the separation can only be done by means

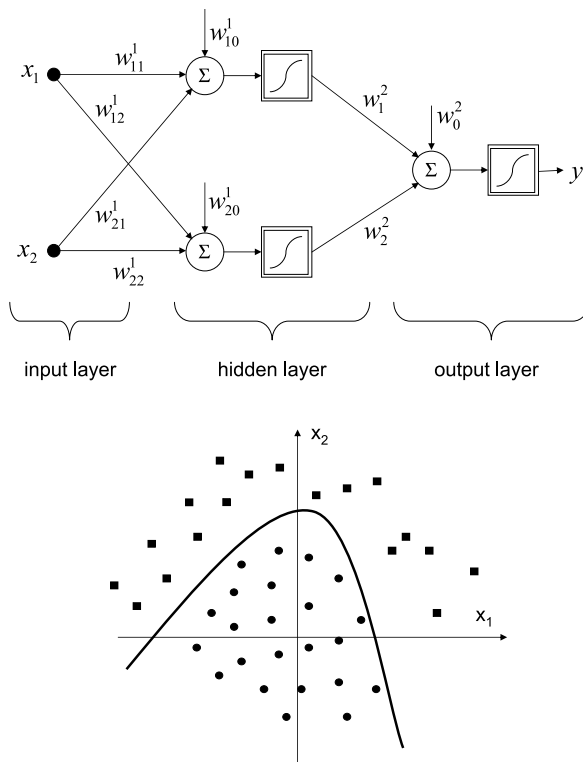


**Fig. 4.2.** An example of two classes which can be separated linearly. The feature vectors are two-dimensional. The two classes are marked by circles and squares

of lines, planes, or hyperplanes. An example of a linear separation of feature vectors belonging to two classes, depicted by circles and squares, is shown in Fig. 4.2.

However, one may wish to design discriminant systems which allow more complicated boundaries between classes. This aim can be achieved by combining several neurons into a network, as shown in Fig. 4.3. The neural network presented in the upper part of Fig. 4.3 is called a multilayer perceptron, or hidden-layer perceptron. This is a simple example, where the input vector  $x$  has two components  $x_1, x_2$  and the total number of neurons in the network is three. This neural network is organized into three layers. The first, input layer is built from the input signals  $x_1, x_2$ . The second, hidden layer contains two neurons, which, as their inputs, take sums of the input signals with different weights,  $w_{11}^1$  and  $w_{12}^1$  with an offset  $w_{10}^1$  for the first neuron, and  $w_{11}^1$  and  $w_{12}^1$  with an offset  $w_{20}^1$  for the second neuron. The superscript 1 indexes the first layer. The outputs from the neurons in the second layer are fed into the last, third layer, which has only one neuron, with an output signal  $y$ . In the lower plot in Fig. 4.3 we present the shape of a separation line which can be obtained with the use of the neural net in the upper plot. This line separates two different classes determined by states of the experiment, marked by circles and squares. Such a shape of the separation line cannot be obtained with a single-neuron classifier.

Artificial neural networks of the type shown in the upper part of Fig. 4.3 can have more than one hidden layer, as well as more neurons in each of the layers. The crucial task is the training algorithms for artificial neural networks. A well-known recursive algorithm for adjusting the values of the weights is called back propagation [67].



**Fig. 4.3.** *Upper plot:* a neural net with three layers, two input signals  $x_1$  and  $x_2$ , and one output signal  $y$ . *Lower plot:* the shape of a separation line, which can be obtained with the use of neural net from the upper plot. The line separates two classes marked by circles and squares

#### 4.2.4 Support Vector Machines

We now return to the problem of the construction of linear classification functions (4.1). In this subsection, we introduce an approach using supporting vector machines (SVMs) (SVM). The idea of SVMs involves designing a linear classifier which is optimal in the sense of its distances to points belonging to separated classes. Before explaining this idea in more detail, let us recall the fact from the multidimensional analytical geometry, namely that the distance between a hyperplane  $H = \{x : w^T x + w_0 = 0\}$  and a point  $y \in R^k$  is given by the formula

$$d(H, y) = \frac{|w^T y + w_0|}{\|w\|}. \quad (4.9)$$

In Fig. 4.4 we have shown a graphical representation of the situation where the feature space is two-dimensional, i.e., the vector  $x$  has two components  $x_1, x_2$  and there are two classes, marked by circles and squares. This figure shows

two plots, with identical locations of some feature vectors corresponding to two classes. There are infinitely many possible linear discriminant functions for separating the two classes. The left and right plots present two separating lines related to two different linear discriminant functions. In the left-hand plot we presented a “randomly chosen” separating line, obtained, for example, by some recursive procedure for modification of the weights. In the right-hand plot, we present a separating line related to a special discriminant function  $f^*(x) = \text{sign}(w^{*T}x + w_0^*)$ . Denoting the feature vectors in Fig. 4.4 by  $x_1, x_2, \dots, x_n$  (those corresponding to circles), and  $x_{n+1}, x_{n+2}, \dots, x_{n+m}$  (corresponding to squares), the special property of this separating line can be explained as follows. The separating line in the right-hand plot,  $L^* = \{x : w^{*T}x + w_0^* = 0\}$ , has the property that (i) it separates the two classes and (ii) it maximizes the minimal distance  $d(L^*, x_k)$  between  $L^*$  and the points  $x_1, x_2, \dots, x_{n+m}$ :

$$w^*, w_0^* \leftarrow \max_{w, w_0} \min_{1 \leq k \leq n+m} d(L, x_k). \quad (4.10)$$

The conditions (i) and (ii) determine uniquely the parameters  $w^*, w_0^*$ . Using (4.9) and (4.10) and recalling the idea of construction of the matrix  $M$  in (4.4), one can derive that the parameters  $w^*, w_0^*$  can be obtained from the solution to the quadratic programming problem

$$\min w^T w, \quad (4.11)$$

subject to the constraints

$$M \begin{bmatrix} w \\ w_0 \end{bmatrix} \leq -\mathbf{1}. \quad (4.12)$$

The symbols  $M$  and  $\mathbf{1}$  have the same meaning as in (4.3)–(4.5). The quadratic programming problem is also mentioned in Chap. 5 in (5.56) and (5.57).

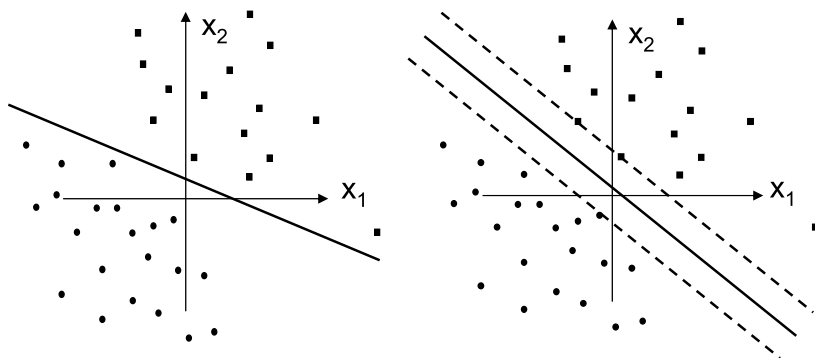
There are many examples where the optimal discriminant function

$$f^*(x) = \text{sign}(w^{*T}x + w_0^*)$$

shown in the right-hand plot in Fig. 4.4 has better properties than a “randomly chosen” discriminant function, such as the one in the left hand plot in Fig. 4.4. One may also wish to extend the method to discriminating to more than two classes and to more complicated shapes of the separating lines or surfaces. An appropriate methodology can be designed by developing the ideas sketched above, [45, 46]. Classifiers based on an optimal separation of the kind shown in the right plot in Fig. 4.4 and described in (4.9)–(4.12) are called supporting vector machines.

## 4.3 Clustering

Clustering involves a situation where there is a need to identify classes solely on the basis of feature vectors. We infer classes by using the hypothesis that



**Fig. 4.4.** In the *left-hand plot* a “randomly chosen” separating line is presented. In the *right-hand plot* we show the separating line which maximizes the minimal distance  $d(L^*, x_k)$  between  $L^*$  and points  $x_1, x_2, \dots, x_{n+m}$

separate classes correspond to regions where data points occur with increased density. We call such regions of increased density clusters. An example is shown in Fig. 4.5. In the plot in figure 4.5 we can see clearly that the data points, representing some features or patterns, tend to be concentrated around two points, forming two data clusters. So it may be reasonable to hypothesize that these two clusters are related to two different classes in the data.

We shall present two algorithms for clustering, the  $K$ -means algorithm and the hierarchical clustering algorithm. Both of these approaches are related to methods presented also in other chapters of this book. The  $K$ -means algorithm can be interpreted in terms of analyzing mixtures by using EM methods (Chap. 2), and hierarchical clustering is closely related to inferring trees (Chap. 7). In order to decide whether the data points are densely or sparsely located, one needs to use some distance measure. The most natural is the Euclidean distance, but other distances can also be used.

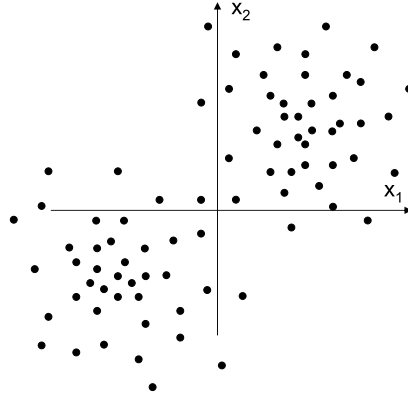
### 4.3.1 K-means Clustering

The idea behind the algorithm for  $K$ -means clustering is very simple, and similar to the idea the EM algorithm for estimating the parameters of mixtures of distributions.

Concerning the construction of the algorithm, we do the following:

- (I) We assume that the number  $K$  of clusters is known, and we make two more assumptions, as follows.





**Fig. 4.5.** Example of a pattern of feature vectors. The classes are not predefined. The hypothesis of the existence of two classes is made based on the fact that points tend to be concentrated around two centers, forming two data clusters

- (II) Each of the clusters has a center point  $x_i^C$ ,  $i = 1, \dots, K$ , with the coordinates equal to the mean of the coordinates of the data points that belong to that cluster.
- (III) For each of the data points  $x$ , we decide which of the clusters does it belong to by computing the distances between that point and centers of all clusters  $d(x, x_i^C)$ ,  $i = 1, \dots, K$ . We take the index  $i$  of  $x_i^C$  that minimizes the distance  $d(x, x_i^C)$  to indicate the cluster containing  $x$ .

On the basis of the assumptions (I)–(III), the following design of the clustering algorithm is quite obvious. We choose randomly some initial values for the centers of the clusters  $x_i^C$ ,  $i = 1, \dots, K$ , and then iterate the following two steps until convergence is obtained:

- Step 1.* Assign each data point to a cluster, on the basis of the criterion in (III).
- Step 2.* On the basis of the assignment in step 1, update values for centers of clusters as defined in (II).

The above algorithm is both simple and an efficient tool for searching for clusters.

### 4.3.2 Hierarchical Clustering

The drawback of the K-means clustering algorithm is the need to know the number of clusters in advance. There are several methods to overcome this difficulty. The natural approach is to perform clustering for different numbers of clusters and try to estimate number of clusters by some method of assessing the quality of the clustering.

It is also worthwhile to consider another approach, hierarchical clustering. In this approach, a tree or a treelike structure is constructed on the basis of pairwise distances between feature vectors. Clusters are obtained by “cutting” the tree at some level, and the number of clusters is controlled by deciding at which level of the hierarchy of the tree the splitting is performed. The construction of the tree is based on neighbor joining, an idea described also in Chap. 7. There are several variants of the hierarchical-clustering algorithm [67]. Here we shall describe the basic idea and some of the possible modifications.

Assume that the data points (feature vectors) are  $x_1, x_2, \dots, x_n$  and define a matrix  $D^0$  of distances between them

$$D^0 = [d(x_i, x_j)]. \quad (4.13)$$

In the above,  $d(x_i, x_j)$  denotes the Euclidean distance between the feature vectors  $x_i$  and  $x_j$ . The idea of neighbor joining is (i) to find a pair of feature vectors  $x_{i*}$  and  $x_{j*}$  with a minimal distance

$$i^*, j^* \leftarrow \min_{i,j} d(x_i, x_j), \quad (4.14)$$

and (ii) to join  $x_{i*}$  and  $x_{j*}$ . Joining  $x_{i*}$  and  $x_{j*}$  is often realized by replacing  $x_{i*}$  and  $x_{j*}$  by their mean:

$$x_{i*}, x_{j*} \rightarrow \frac{1}{2}(x_{i*} + x_{j*}) = y. \quad (4.15)$$

After joining  $x_{i*}$  and  $x_{j*}$  we update the matrix of distances between feature vectors, i.e.,  $D^0 \rightarrow D^1$ ; the entries of  $D^1$  which need updating are the distances  $d(x_i, y)$  between the new vector  $y$  and the feature vectors  $x_i$  that were not involved in the joining operation. Performing the above-described operations sequentially leads to the construction of a neighbor-joining tree for the vectors  $x_1, x_2, \dots, x_n$ .

Sequential joining of vectors, as defined in (4.15), leads to the formation of clusters. By keeping track of the indexes of the vectors that have been joined to each other, we know which vectors belong to which cluster. Since we are focused on clusters rather than on a tree, we may make some modifications to the algorithm described by (4.13)–(4.15). The sequential application of the replacement rule (4.15) leads to defining vectors  $y$ , which can be interpreted as centers of clusters. Then, one can use a modification of the rule (4.15), defined by

$$x_{i*}, \text{cluster}(x_{j1}, \dots, x_{jm}) \rightarrow \text{cluster}(x_{j1}, \dots, x_{jm}, x_{i*}) \quad (4.16)$$

and

$$y = \text{cluster\_center}(x_{j1}, \dots, x_{jm}, x_{i*}) = \frac{x_{j1} + \dots + x_{jm} + x_{i*}}{m + 1}, \quad (4.17)$$

for merging  $x_{i*}$  with cluster  $(x_{j1}, \dots, x_{jm})$  and for computing the center of the cluster. When deciding about merging vectors with clusters, we can use the distances between the vectors and the centers of clusters.

Other variants of the hierarchical-clustering algorithm are also possible, for example one may use using other definitions of the distance function or other rules for merging vectors with existing clusters [235, 67]. Some possible definitions of distances are the euclidean, correlation, Pearson or Spearman, and Manhattan distances. The rules used most often for defining clusters are single-linkage clustering, where the distance between two clusters  $i$  and  $j$  is the minimum of distances between members of clusters  $i$  and  $j$ ; complete-linkage clustering, where the distance between two clusters is the maximum of the distances between their members; and average-linkage clustering, where the distance between two clusters is the mean value of the distances between members of the clusters.

## 4.4 Dimensionality Reduction, Principal Component Analysis

A need for dimensionality reduction arises when the number of features is large. Experimental results in molecular biology and biochemistry often lead to the creation of a large number of measurement data points. Examples are gene expression intensities in DNA microarrays, proteomic spectra, and data concerning conformations of large molecules such as proteins. In such situations the number of features (measurements) obtained in each experiment is much bigger than the number of experiments. One expects that only some of the measurements will be correlated with the state of the experiment under study.

Two cases are possible. The first possibility is that the state of the experiment (e.g. diseased versus healthy) is known. A related problem is to select the subset of features most suitable for differentiating between experimental states. Some aspects of this problem are discussed in this book, in Chap. 11. The second possibility is that inference must be done solely on the basis of the set of feature vectors, without any knowledge about the underlying structure. A well-established methodology for this problem is principal component analysis [131]. The searching for principal components in the data is based on analysis of the variances along different directions in the feature space. The method of principal component analysis (PCA) can also be applied to the situation where the classes of experimental states are known. Below we present some of the computational aspects of these applications of principal component analysis.

#### 4.4.1 Singular-Value Decomposition (SVD)

We start from a theorem on SVD of a real matrix. Let us define a real  $m \times n$  matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}.$$

The SVD theorem, [103, 131], states that  $A$  can always be represented as follows:

$$A = U \Sigma V^T \quad (4.18)$$

where  $U$  and  $V^T$  are (nonsingular) real orthogonal transformation matrices, of dimensions  $m \times m$  and  $n \times n$  respectively and the superscript  $T$  represents matrix transposition. The  $m \times n$ -dimensional matrix  $\Sigma$  is composed of the following blocks:

$$\Sigma = \begin{bmatrix} \Xi_{r \times r} & O_{r \times (n-r)} \\ O_{(m-r) \times r} & O_{(m-r) \times (n-r)} \end{bmatrix},$$

where  $O_{k \times l}$  denotes  $k \times l$ -dimensional matrix with all entries equal to zero, and  $\Xi_{r \times r}$  is a diagonal matrix

$$\Xi_{r \times r} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix},$$

with real elements  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ , and where  $r = \text{rank}(A)$ . Clearly,  $r \leq \min(n, m)$ .

The numbers  $\sigma_1, \sigma_2, \dots, \sigma_r$  are called the singular values of the matrix  $A$  and the first  $r$  columns of the matrix  $U$  are called the principal directions of the matrix  $A$ . More precisely, the first  $r$  columns of  $U$  are the principal directions for the columns of the matrix  $A$  and the first  $r$  rows of the matrix  $V^T$  are the principal directions for the rows of the matrix  $A$ . The singular values and orthogonal matrices  $U$  and  $V^T$  are related to eigenvalues and eigenvectors of the Grammian matrices  $AA^T$  and  $A^T A$ , which can be clearly seen from (4.18). Recalling that the orthogonality of  $U$  and  $V$  implies that  $U^T U = I_{m \times m}$  and  $V^T V = I_{n \times n}$ , where  $I_{k \times k}$  denotes the  $k \times k$  identity matrix, we have the following equalities for the Grammian matrices  $AA^T$  and  $A^T A$

$$AA^T = U \Sigma \Sigma^T U^T = U \Xi_{m \times m}^2 U^T \quad (4.19)$$

$$A^T A = V \Sigma^T \Sigma V^T = V \Xi_{n \times n}^2 V^T. \quad (4.20)$$

In the above expressions,  $\Xi_{k \times k}^2$  (where  $k$  equals either  $m$  or  $n$ ) stands the diagonal matrix

$$\Xi_{k \times k}^2 = \begin{bmatrix} \Xi_{r \times r}^2 & O_{r \times (k-r)} \\ O_{(k-r) \times r} & O_{(k-r) \times (k-r)} \end{bmatrix},$$

where

$$\Xi_{r \times r}^2 = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r^2 \end{bmatrix}.$$

Since the expressions on the right-hand sides of (4.19) and (4.20) are Jordan canonical forms, the columns of the orthogonal matrix  $U$  are eigenvectors of the Grammian matrix  $AA^T$  and the columns of the orthogonal matrix  $V$  are eigenvectors of the Grammian matrix  $A^T A$ . One can also see that nonzero eigenvalues of both  $AA^T$  and  $A^T A$  are equal to the squares of the singular values  $\sigma_1^2, \sigma_2^2, \dots, \sigma_r^2$  of the matrix  $A$ .

#### 4.4.2 Geometric Interpretation of SVD

The representation (4.18) has several interesting geometric interpretations. One geometric interpretation is as follows. Let us understand  $A$  as a linear operator mapping  $n$ -dimensional vectors  $x \in R^n$  to  $m$ -dimensional vectors  $y = Ax$ ,  $y \in R^m$ . The representation (4.18) implies that for every linear operator of rank  $r$ , one can find two orthogonal bases, in the domain and image spaces  $R^n$  and  $R^m$ , respectively, such that the first  $r$  vectors of the orthogonal basis in the domain space  $R^n$  are mapped to first  $r$  vectors of the orthogonal basis in the image space  $R^m$ . The orthogonal basis in the domain space is given by the rows of the matrix  $V^T$ , and the orthogonal basis in the image space by the columns of the matrix  $U$ .

Another important geometric interpretation of the decomposition (4.18), which will be used in this book in several contexts, is related to expressing the principal directions of the matrix in terms of solutions to optimization problems and to computing projections onto subspaces. Let us interpret the matrix  $A$  as a set of  $n$  column vectors, each belonging to the space  $R^m$ :

$$A = [a_1 \ a_2 \ \dots \ a_n], \quad (4.21)$$

$$a_k = \begin{bmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{mk} \end{bmatrix} \in R^m, k = 1, 2, \dots, n.$$

We now ask a somewhat imprecise question: Which direction in the space  $R^m$  is most representative for the vectors  $a_k$ ,  $k = 1, 2, \dots, n$ ? Precisely, we call a vector  $c \in R^m$  the most representative for the set of vectors  $a_k$ ,  $k = 1, 2, \dots, n$ , if:

(1)  $c$  is a linear combination of vectors  $a_k$ ,

$$c = \sum_{k=1}^n \beta_k a_k, \quad (4.22)$$

with scalar coefficients  $\beta_k$ ;

(2) the coefficients  $\beta_k$ ,  $k = 1, 2, \dots, n$  are normalized to 1, i.e.,  $\sum_{k=1}^n \beta_k^2 = 1$ ; and

(3) the vector  $c$  is the longest possible under conditions (1) and (2).

By “longest possible” we mean the one with the largest Euclidean norm. Conditions (1)–(3) lead to the following maximization problem:

$$\max \|c\|$$

under the constraints

$$c = \sum_{k=1}^n \beta_k a_k, \quad \sum_{k=1}^n \beta_k^2 = 1.$$

Noting that  $\max \|c\|$  is equivalent to  $\max \|c\|^2$  and introducing the vector of coefficients

$$b = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad (4.23)$$

we can write the above maximization problem as

$$\max b^T A^T A b \quad (4.24)$$

under the constraints

$$b^T b = 1. \quad (4.25)$$

A necessary conditions for optimality for the constrained optimization problem (4.24), (4.25) (see Chap. 5) are that  $(b, \lambda)$  is the stationary point of the Lagrange functional

$$L(b, \lambda) = b^T A^T A b + \lambda(1 - b^T b). \quad (4.26)$$

In the above  $\lambda$  is a scalar Lagrange multiplier. Stationarity is verified by comparing the gradients of  $L(b, \lambda)$  with respect to  $\lambda$  and  $b$ , with zero and a zero vector. The condition

$$\frac{\partial L}{\partial \lambda} = 0 \quad (4.27)$$

is equivalent to (4.25), and

$$\frac{\partial L}{\partial b} = 0 \quad (4.28)$$

leads to

$$A^T Ab - \lambda b = 0, \quad (4.29)$$

or

$$\lambda b = A^T Ab. \quad (4.30)$$

This is an eigenvalue–eigenvector problem. So the problem of maximizing (4.24) with the constraint (4.25) leads to computing the eigenvector  $b$  and the eigenvalue  $\lambda$  of the symmetric matrix  $A^T A$ . The solution is nonunique since in general there are  $n$  (nonnegative, real) eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_n$  of  $A^T A$ . Nonuniqueness is a consequence of using only necessary optimality conditions. However, among those satisfying (4.30), the optimal  $(b, \lambda)$  can easily be identified. Substituting (4.30) in (4.24) results in

$$\max b^T A^T Ab = \max \lambda b^T b = \max \lambda = \lambda_{\max}. \quad (4.31)$$

So the solution  $(\lambda, b)$  is the maximal eigenvalue  $\lambda_{\max}(A^T A)$  and the corresponding eigenvector  $b$ . From (4.30), (4.20), and (4.18) we now see that the eigenvector  $b$  corresponding to  $\lambda_{\max}(A^T A)$  is the first column of the matrix  $V$ .

Let us return to the representative direction  $c$ . The relation (4.22) can be represented in vector notation as

$$c = Ab.$$

Multiplying both sides of the above equation by  $AA^T$  and recalling (4.30) and (4.25) we obtain

$$\lambda_{\max} c = AA^T c.$$

The conclusion is that the representative direction  $c$ , defined by conditions (1)–(3) above is, up to some scaling factor, the first principal direction of the matrix  $A$ . It turns out that the first principal direction (the first column of the matrix  $U$ ) of the matrix  $A$  has the interpretation given by conditions (1)–(3) above. Repeating the above with the matrix  $A$  understood as a set of  $m$  rows, rather than a set of  $n$  columns as in (4.21), we obtain an interpretation of the first row of the matrix  $V^T$  as the first principal direction for the rows of the matrix  $A$ .

After establishing the meaning of the first principal direction of the matrix  $A$ , one can ask about other principal directions (the other columns of the matrix  $U$ ). The second, third, and further principal directions of the matrix  $A$ , can again be interpreted as representative directions determined by some sets of vectors, in the following sense. Let us represent all vectors  $a_k$ ,  $k = 1, 2, \dots, n$  as sums of two components, parallel and orthogonal to  $c$ :

$$a_k = \hat{a}_k + \tilde{a}_k. \quad (4.32)$$

In the above  $\hat{a}_k$  is parallel to  $c$ , which means that  $\hat{a}_k = \rho_k c$  for some scalar value  $\rho_k$ , and  $\tilde{a}_k$  is orthogonal to  $c$ , which means that their scalar product

equals zero, i.e.,  $c^T \tilde{a}_k = 0$ . Taking the scalar products of both sides of (4.32) with  $c$ , we obtain  $\rho_k = c^T a_k / c^T c$ , and consequently

$$\hat{a}_k = \frac{c^T a_k}{c^T c} c$$

and

$$\tilde{a}_k = a_k - \frac{c^T a_k}{c^T c} c. \quad (4.33)$$

By defining the row vector

$$\rho^T = [\rho_1 \ \rho_2 \ \dots \ \rho_n]$$

we can express the relation (4.33) in matrix–vector notation, as follows:

$$\tilde{A} = A - c \rho^T \quad (4.34)$$

where  $\tilde{A}$  is defined as

$$\tilde{A} = [\tilde{a}_1 \ \tilde{a}_2 \ \dots \ \tilde{a}_n]. \quad (4.35)$$

The matrix  $\tilde{A}$  is composed of the residual vectors  $\tilde{a}_k$  (4.33), and it can be verified (we give it as Exercise 6) that, if the matrix  $A$  has a set of singular values  $\sigma_1 > \sigma_2 \ \dots \ \sigma_r > 0$  then matrix  $\tilde{A}$  will have singular values  $\sigma_2 > \sigma_3 \ \dots \ \sigma_r > 0$ , or in other words the largest singular value  $\sigma_1 = \sigma_{\max}$  is replaced by zero. Now, solving the problem (4.24)–(4.25) with  $A$  replaced by  $\tilde{A}$  will lead to computing the second singular value and the second principal direction of  $A$ , and so forth. This leads to the following representation of the matrix  $A$ :

$$A = \sum_{k=1}^r c_k \rho_k^T$$

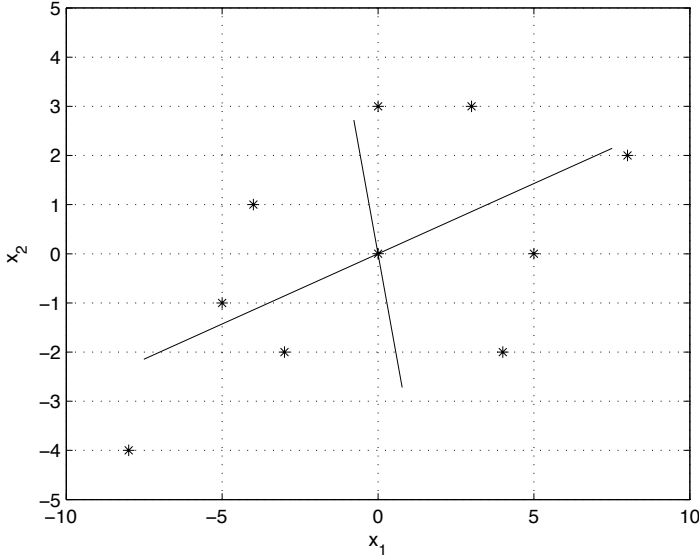
where two sets of orthogonal vectors  $c_k$  and  $\rho_k^T$ ,  $k = 1, 2, \dots, r$  are called loadings and scores, respectively. The above representation also follows directly from the form of the singular-value decomposition (4.18).

The principal components and the singular-value decomposition also have a very important statistical interpretation in terms of variances of random variables. Let us consider a set of  $n$  random variables  $X_1, X_2, \dots, X_n$ . For each of them,  $m$  realizations are given (i.e., measured), which are denoted as follows:  $x_{11}, x_{21}, \dots, x_{m1}, x_{12}, x_{22}, \dots, x_{m2}, \dots, x_{1n}, x_{2n}, \dots, x_{mn}$ . We form the matrix of data (measurements)  $X$ ,

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}. \quad (4.36)$$

Assume that the realizations are centered, which means that for each column  $k$  we have  $\sum_{i=1}^m x_{ik} = 0$ . The total sampling variance  $\text{Var}(X)$  of the data (4.36) is then defined as





**Fig. 4.6.** Scatterplot of ten pairs of realizations of random vectors  $x_1, x_2$ . The vectors  $v_1^T$  and  $v_2^T$ , the principal directions of the matrix  $X$ , are marked by line segments. The longer segment is for the first principal direction, and the shorter segment is for the second principal direction

$$\text{Var}(X) = \frac{1}{m-1} \text{trace}(X^T X), \quad (4.37)$$

where  $\text{trace}(A)$  means the trace (sum of diagonal elements) of the matrix  $A$ . We can see that the sampling variance as defined by (4.37) is invariant with respect to orthogonal transformations of the data, that is, if  $Y = XW^T$ , where  $W^T$  is an  $m \times m$ -dimensional orthogonal matrix, then  $\text{Var}(Y) = \text{Var}(X)$ . From (4.20) it follows that the total sampling variance can be expressed in terms of the squares of the singular values of matrix  $X$ ,  $\sigma_1, \sigma_2, \dots, \sigma_r$ , where  $r = \text{rank}(A)$ :

$$\text{Var}(X) = \frac{1}{m-1} \sum_{k=1}^r \sigma_k^2.$$

Let us consider the SVD decomposition of the matrix  $X$

$$X = U \Sigma V^T$$

and its equivalent form

$$XV = \Sigma U. \quad (4.38)$$

Denoting by  $U_r$  the matrix composed of the first  $r$  columns of  $U$ , and by  $V_r$  the matrix composed of the first  $r$  columns of  $V$  we can express the relation (4.38) as follows:

$$XV_r = \Sigma U_r = [\sigma_1 u_1 \ \sigma_2 u_2 \ \dots \ \sigma_r u_r],$$

where  $u_1, u_2, \dots, u_r$  are the columns of the matrix  $U_r$  (the principal directions of  $X$ ). Since  $\text{Var}(XV_r) = \text{Var}(\Sigma U_r) = \text{Var}(X)$  and the columns of  $U_r$  are orthogonal, we can characterize each of principal directions  $u_1, u_2, \dots, u_r$  in terms of how much of total variance they include. It is common to say that the first principal direction or component captures  $(\sigma_1^2 / \sum_{k=1}^r \sigma_k^2) \cdot 100\%$  of total variance, the first two principal components capture  $(\sigma_1^2 + \sigma_2^2) / \sum_{k=1}^r \sigma_k^2 \cdot 100\%$ , and the first  $j$  principal components capture

$$\frac{\sum_{k=1}^j \sigma_k^2}{\sum_{k=1}^r \sigma_k^2} \cdot 100\% \quad (4.39)$$

of the total variance.

As an example, let us consider the following data matrix

$$X = \begin{bmatrix} 3 & -3 & -8 & 4 & 0 & -4 & 5 & 8 & 0 & -5 \\ 2 & -2 & -4 & -2 & 0 & 1 & 0 & 2 & 3 & -1 \end{bmatrix}^T, \quad (4.40)$$

consisting of ten realizations of two random variables  $X_1$  and  $X_2$ , written as two columns of the matrix  $X$ . The superscript  $T$  stands for transposition. Columns of  $X$  are mean-centered. A scatterplot for pairs of realizations of  $X_1$  and  $X_2$  is shown by asterisks in Fig. 4.6. The SVD decomposition of  $X$  is

$$X = U \Sigma V^T, \quad (4.41)$$

where

$$U = \begin{bmatrix} -0.237 & 0.364 & 0.559 & -0.168 & 0.000 & 0.196 & -0.279 & -0.503 & -0.084 & 0.307 \\ 0.220 & -0.194 & -0.269 & -0.567 & 0.000 & 0.392 & -0.270 & -0.082 & 0.527 & 0.095 \\ 0.563 & -0.291 & 0.726 & -0.017 & 0.000 & -0.021 & 0.075 & 0.197 & 0.116 & -0.113 \\ -0.211 & -0.534 & 0.032 & 0.655 & 0.000 & 0.262 & -0.225 & -0.196 & 0.246 & 0.143 \\ 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.229 & 0.364 & -0.058 & 0.256 & 0.000 & 0.801 & 0.178 & 0.172 & -0.170 & -0.122 \\ -0.308 & -0.243 & 0.106 & -0.210 & 0.000 & 0.171 & 0.836 & -0.184 & 0.118 & 0.125 \\ -0.528 & -0.049 & 0.222 & -0.159 & 0.000 & 0.147 & -0.169 & 0.754 & 0.036 & 0.157 \\ -0.053 & 0.510 & 0.079 & 0.265 & 0.000 & -0.189 & 0.141 & 0.073 & 0.772 & -0.065 \end{bmatrix} \quad (4.42)$$

$$\Sigma = \begin{bmatrix} 15.621 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5.657 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \quad (4.43)$$

and

$$V^T = \begin{bmatrix} -0.962 & -0.275 \\ 0.962 & -0.274 \end{bmatrix}.$$

In the above  $r = n$ . The nonzero entries of  $\Sigma$  are the singular values of  $X$ , and the rows of  $V^T$ ,

$$v_1^T = [-0.962 \ -0.275]$$

and

$$v_2^T = [0.962 \ -0.274],$$

are two principal directions of the rows of  $X$ . The vectors given by the principal directions, with lengths scaled by the corresponding singular values, are also shown in Fig. 4.6. For rectangular matrices, such as  $X$  in (4.40), the matrix of singular values  $\Sigma$  always contains zero rows or columns, as does  $\Sigma$  in (4.43). So, instead of the decomposition (4.18) it may be reasonable to use an “economy” SVD, where zero columns or rows of the matrix  $\Sigma$  are skipped and the corresponding rows or columns of the matrix  $U$  or  $V^T$  are removed. For example, we have the following economy-size decomposition for (4.41):

$$X = U^0 \Sigma^0 V^T, \quad (4.44)$$

where

$$U^0 = \begin{bmatrix} -0.237 & 0.364 \\ 0.220 & -0.194 \\ 0.563 & -0.291 \\ -0.211 & -0.534 \\ 0.000 & 0.000 \\ 0.229 & 0.364 \\ -0.308 & -0.243 \\ -0.528 & -0.049 \\ -0.053 & 0.510 \end{bmatrix},$$

and

$$\Sigma^0 = \begin{bmatrix} 15.621 & 0 \\ 0 & 5.657 \end{bmatrix}.$$

The columns removed from the matrix  $U$  have no influence on the product representation of the matrix  $X$ .

In cases where data sets to be analyzed are large, using economy-size SVD can save a lot of computational time and memory space.

#### 4.4.3 Partial-Least-Squares (PLS) Method

Here we assume that the input measurement data (also called the explaining or predictor variables) given by (4.36) are accompanied by measurements of a scalar output (also called the dependent variable)  $Y$ . So our data structure is now

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}. \quad (4.45)$$

In a general setting we would assume a vector-valued output. However, here we shall confine the presentation to a scalar output, as given in (4.45). Now we

try to form a linear combination of columns of the matrix  $X$  (the vectors  $x_1, \dots, x_n$ ), with coefficients  $\beta_k$ ,  $k = 1, 2, \dots, n$ , normalized to 1,  $\sum_{k=1}^n \beta_k^2 = 1$ , such that the resulting vector

$$c = \sum_{k=1}^n \beta_k x_k$$

maximizes the covariance or, equivalently, the scaled scalar product  $c^T y / (m - 1)$ . Using a vector notation analogous to (4.24) and (4.25), we can state this maximization problem as

$$\max \frac{1}{m-1} y^T X b$$

with the constraint

$$b^T b = 1,$$

where  $b$  is a vector of parameters  $\beta_k$  as in (4.23). Using the technique of constrained optimization (Chap. 5), we obtain, analogously as to (4.26)–(4.29), the following optimal vector,

$$b = \frac{X^T y}{\sqrt{y^T X X^T y}}$$

and the first PLS direction (component),

$$c = \frac{X X^T y}{\sqrt{y^T X X^T y}}.$$

The second, third, and further PLS components are obtained by projections onto the direction given by  $c$  in the above expression and analyzing the residual vectors, analogously to (4.33)–(4.35).

## 4.5 Parametric Transformations

Transformations are workhorses in all areas of applied mathematics. Some examples already shown are generating functions and characteristic functions, discussed in Chap. 2. The characteristic function is actually the Fourier transform of the probability density function of a random variable. One- and two-dimensional Fourier transforms are also widely applied in pattern analysis, for example for noise reduction and extraction of image features. Fourier transformation is also used in bioinformatics, for example for analysis of repetitive structure of sequences. A DNA sequence is changed to numerical symbols by some method, then a Fourier transformation is applied to the numerical sequence obtained, and the resulting spectrum is used to search for special geometric-like or repetitive patterns. There are numerous textbooks (e.g., [66])

devoted to transforms including the Fourier, Laplace, Laurent, Hankel, and Hilbert transforms.

However, in this section, we focus on transformations which are not as widely known as the above, but underline interesting relations between computer-science algorithms and pattern analysis methods. They come under different names, such as parametric transforms, Hough transforms, and geometric hashing, but share the idea of using a process of pattern scanning in conjunction with addressing and operating on a data structure to record occurrences of data objects. The contents of this data structure can be then used to obtain useful information about the objects under study. Clearly, this idea has some similarity to the method of hashing and hash tables, presented in Chap 3.

In this section we discuss some of these approaches from the pattern analysis perspective. Owing to their flexibility they have large potential to serve in numerous procedures for browsing databases for correlations, similarities of different types, etc. Later we also show some applications of these methods in genomics and in protein docking.

#### 4.5.1 Hough Transform

The Hough transform [132] provides a method for detecting parametric curves in images and estimating the values of their parameters. Most often, Hough transforms use contours in a binary as input data and apply a duality between points on the curve and the parameters of the curve. The Hough transform can also be understood as a feature extraction technique based on interpreting the contents of a digital image by using a feature space. The basic example is the detection of straight lines in images, as presented in Fig. 4.7. The task is to detect occurrences of straight lines in the image. We assume that the image to be analyzed is binary, as shown in the left plot in Fig. 4.7, and so it contains a number of discrete image points. The equation of a straight line in the image space  $x, y$  is

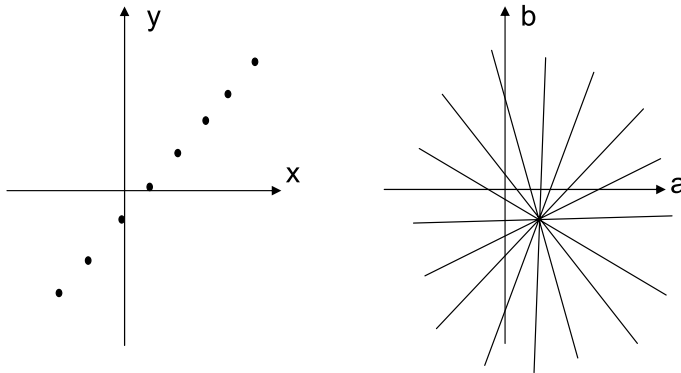
$$y = ax + b. \quad (4.46)$$

To accomplish the aim of detecting straight lines in the image, we create a parameter space (a plane) with coordinates  $a, b$ , as shown in the right-hand part of Fig. 4.7. For each of the points in the image  $x_i, y_i$ , we draw a corresponding line in the parameter space  $a, b$ ,

$$y_i = ax_i + b. \quad (4.47)$$

Because all points in the image space  $x, y$  are collinear, all lines in the parameters space  $a, b$  intersect in one point. The occurrence of the point of intersection,  $a^*, b^*$  of many lines in the parameter space  $a, b$  indicates detection of a line in the image space  $y = a^*x + b^*$ .

In practical situations, the parameter space is discretized and consists of a finite number of pixels. The discretized parameter space is called an



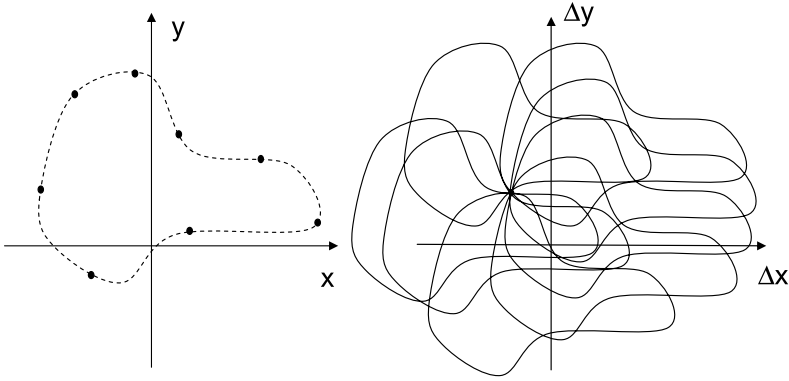
**Fig. 4.7.** The idea of the Hough transform. For each point  $x_i, y_i$  in the image space  $x, y$ , we draw a corresponding line  $y_i = ax_i + b$  in the parameter space  $a, b$ . If the points in the image space belong to a line, as shown in the *left-hand plot*, their corresponding lines in the parameter space intersect at one point, as depicted in the *right-hand plot*

accumulator array. Drawing lines corresponding to points found in the image is equivalent to incrementing memory locations in the accumulator array. The procedure of incrementing entries of accumulator array is often called voting. Detecting lines can be accomplished by browsing through the accumulator array and searching for local maxima.

The idea described above can also be used for detecting other parametric curves in images, for example circles and ellipses.

#### 4.5.2 Generalized Hough Transforms

Generalized Hough transforms extend the idea described above to the non-parametric curves. The most straightforward generalization is as follows. Assume we are searching for occurrences of a shape, such as the one shown by the dashed curve in the left plot in Fig. 4.8. The binary image to be analyzed consists of points, also depicted in the left plot in Fig. 4.8. The problem is, does the shape occur in the image? We are not allowing rotations of the target shape, so it is natural to define a parameter space with translations  $\Delta x$  and  $\Delta y$  along the axes as coordinates. The procedure for updating the accumulator array associated with this parameter space is very similar to the one described in the previous subsection. We browse through the image and, after detecting a point with coordinates  $x_i, y_i$ , we draw the target shape in the parameter space translated by the vector  $\Delta x = x_i, \Delta y = y_i$ . This is shown in the right plot in Fig. 4.8. Again, the intersection of many of the drawn shapes at one point indicates the occurrence of the target shape in the image.



**Fig. 4.8.** The idea of the generalized Hough transform. For each point  $x_i, y_i$  in the image space (left plot) we draw the target shape (dashed curve) in the parameter space translated by the vector  $\Delta x = x_i, \Delta y = y_i$ . This is shown in the right image. The intersection of many of drawn shapes at one point indicates the occurrence of the target shape in the image.

The algorithm described here does not allow us to detect rotated or rescaled target shapes. However, extensions that overcome this limitation have been proposed in several papers [17, 215, 228].

### 4.5.3 Geometric Hashing

Ideas similar to the above has been used to construct another algorithm called geometric hashing. [120, 292] Let us assume that we aim to search for occurrences in images of a pattern of points,  $x_1, x_2, \dots, x_n, x_i \in R^2$ . The first step of the algorithm is to compute a signature that is invariant under translations, rotations and scale changes. The signature is the set of points in  $R^2$  obtained by the following procedure. Go through all pairs of points  $x_i, x_j$ ,  $1 \leq i, j \leq n$ . For each pair  $x_i, x_j$ , (I) find a transformation  $T$  that maps  $x_i \rightarrow T(x_i) = (-1, 0)$  and  $x_j \rightarrow T(x_j) = (1, 0)$ , and (II) add all transformed points  $T(x_m)$ ,  $m \neq i, m \neq j$  to the signature.

Let us analyze a binary image given by another set of points  $y_1, y_2, \dots, y_m, y_i \in R^2$ . From the above definition it is clear that if the set  $\{y_1, y_2, \dots, y_m\}$  contains  $\{x_1, x_2, \dots, x_n\}$  possibly translated, rotated and rescaled, then the signature of  $\{y_1, y_2, \dots, y_m\}$  contains the signature of  $\{x_1, x_2, \dots, x_n\}$ . In the programs developed in practice the coordinates of the vectors of the signatures are discretized and stored with by use of data structures that are of the form of accumulator arrays.

## 4.6 Exercises

1. Assume the data points given in Table 4.1.

- a) Assume that the points with numbers 1–5 correspond to class 1 and those with numbers 6–10 to class 2. Find a linear discriminant function for classes 1 and 2 using the method of linear programming as described in (4.6)–(4.8). A linear-programming algorithm can be found in many software packages. Draw the data from the Table 4.1 in the plane  $x, y$ . Draw the separating line obtained by solving (4.6)–(4.8).
- b) For the same data and the same assumption that points with numbers 1–5 correspond to class 1 and those with numbers 6–10 to class 2, find the optimal linear discriminant function for classes 1 and 2 by solving the quadratic programming problem (4.11)–(4.12). Draw the data and the optimal separating line. Again, a quadratic programming algorithm can be found in many software packages.

**Table 4.1.** Table of data points to be used in exercises

No.	$x$	$y$
1	1	1.5
2	1.5	3
3	3	1
4	3	2
5	3.5	2
6	-0.5	-0.5
7	-0.5	2
8	-1	0.5
9	1	-1
10	2	-1

2. Separate the classes 1 and 2 defined in the previous exercise by using the artificial neural network shown in Fig. 4.3. There are many software packages that support the designing and training and artificial neural networks. One of them can be used to solve this exercise.
3. Decompose the data set from Table 4.1 into two classes using the K-means algorithm. Decompose the data set from Table 4.1 into three classes using the K-means algorithm.
4. Construct a neighbor-joining tree for the data set from Table 4.1 by using rules (4.13) and (4.15).
5. Build a hierarchical clustering tree by using an algorithm with the rules (4.16)–(4.17).
6. Prove that the largest singular value of the matrix  $\tilde{A}$  in (4.35) is the second largest singular value of the matrix  $A$  in (4.21).
7. The transformation  $T([x_i, y_i])$  mentioned in Sect. 4.5.3, is defined by  $T([x_1, y_1]) = [-1, 0]$  and  $T([x_2, y_2]) = [1, 0]$ , where  $[x_1, y_1]$  and  $[x_2, y_2]$  are given in Table 4.1. Compute  $T([x_i y_i])$  for all points in Table 4.1.
8. Develop a computer program for geometric hashing.



9. Study the problem of extending the geometric hashing algorithm described in Sect. 4.5.3 to the case of three-dimensional feature space [211].



<http://www.springer.com/978-3-540-24166-9>

Bioinformatics

Polanski, A.; Kimmel, M.

2007, XVII, 376 p., Hardcover

ISBN: 978-3-540-24166-9