7

Feature Extraction and Selection Methods

This Chapter provides background and algorithms for feature extraction and feature selection from numerical data. Both methods are performed to reduce the dimensionality of the original data. Feature extraction methods do it by generating new transformed features and selecting the informative ones while feature selection methods choose a subset of original features.

1. Introduction

Nowadays, we deal with large datasets that include up to billions of objects (examples, patterns) and up to several thousands of features. This Chapter provides an introduction to data preprocessing methods, which are concerned with the extraction and selection of features to reduce the dimensionality and improve the data for subsequent data mining analysis. **Feature selection** selects a subset of features among the set of all features from the original dataset. On the other hand, **feature extraction** generates new features based on the original dataset.

This Chapter describes both **supervised** and **unsupervised** feature extraction methods. These include dimensionality reduction and feature extraction via unsupervised **Principal Component Analysis**, unsupervised **Independent Component Analysis**, and **supervised Fisher's linear discriminant analysis**. The first two methods are linear transformations that optimally reduce dimensionality, in terms of the number of features, of the original unsupervised dataset. The Fisher's method also implements a linear transformation that optimally converts supervised datasets into a new space that includes fewer features, which are more suitable for classification. While the above methods are mainly used with numerical (time-independent) data, we also describe two groups of methods for preprocessing of **time-series** data. These include **Fourier transform** and **Wavelets** and their two-dimensional versions. We also discuss **Zernike moments** and **Singular Value Decomposition**.

The second part of the Chapter describes a wide variety of feature selection methods. The design of these methods is based on two components, namely, **selection criteria** and **search methods**.

2. Feature Extraction

Data preprocessing may include transformation (projection) of the original **patterns** (also called **examples** or **objects**) into the transformed pattern space, frequently along with **reduction of**

dimensionality of a pattern by extraction of only the most informative features. The transformation and reduction of pattern dimensionality may improve the recognition process through a consideration of only the most important data representation, possibly with uncorrelated-pattern elements retaining maximum information about the original data. These approaches may also lead to better generalization abilities of a subsequently designed model, such as a classifier.

Reduction of the original pattern dimensionality refers to a transformation of original n-dimensional patterns into other m-dimensional feature patterns ($m \leq n$). The pattern transformation and dimensionality reduction can be considered as a nonlinear transformation (mapping)

$$\mathbf{y} = \mathbf{F}(\mathbf{x}) \tag{1}$$

of n-dimensional original patterns $\mathbf{x}$ (vectors in the n-dimensional pattern space) into m-dimensional transformed patterns $\mathbf{y}$ (vectors in the m-dimensional transformed pattern space). The m-dimensional transforming function $\mathbf{F}(\mathbf{x})$ may be designed based on the available knowledge about a domain and data statistics. Elements y_i ($i = 1, 2, \cdots, m$) of the transformed patterns $\mathbf{y}$ are called **features** and the m-dimensional transformed patterns $\mathbf{y}$ are called **feature vectors**. Feature vectors represent data objects in the **feature space**. However, the general name pattern is also adequate in this context.

The projection and reduction of a pattern space may depend on the goal of processing. The purpose of transformation is to obtain a pattern representing data in the best form for a given processing goal. For example, one can choose features in order to characterize (model) a natural phenomenon that generates patterns. Another goal may be finding the best features for the classification (recognition) of objects.

Below we present an optimal linear transformation that guarantees the preservation of maximum information by the extracted feature vector.

The reasons for performing data transformation and dimensionality reduction of patterns are as follows:

– Removing redundancy in data
– Compression of data sets
– Obtaining transformed and reduced patterns containing only a relevant set of features that help to design classifiers with better generalization capabilities
– Discovering the intrinsic variables of data that help design a data model, and improving understanding of phenomena that generate patterns
– Projecting high-dimensional data (preserving intrinsic data topology) onto low-dimensional space in order to visually discover clusters and other relationships in data

2.1. Principal Component Analysis

Probably the most popular statistical method of linear pattern transformation and feature extraction is **Principal Component Analysis** (PCA). This linear transformation is based on the statistical characteristics of a given data set represented by the **covariance matrix** of data patterns, its **eigenvalues**, and the corresponding **eigenvectors**.

Principal Component Analysis (PCA) is a technique, developed in a biological context, to represent a linear regression analysis as fitting planes to data in the sense of least-squares error.

PCA determines an optimal linear transformation

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{2}$$

of a real-valued n-dimensional random data pattern $\mathbf{x} \in \mathbb{R}^n$ into another m-dimensional ($m \leq n$) transformed vector $\mathbf{y} \in \mathbb{R}^m$. The $m \times n$ linear transformation matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ is optimal from the point of view of obtaining the maximal information retention. PCA is realized through exploring statistical correlations among elements of the original patterns and finding (possibly reduced) data representation that retains the maximum nonredundant and uncorrelated intrinsic information of the original data. Exploration of the original data set, represented by the original n-dimensional patterns $\mathbf{x}^i$, is based on computing and analyzing a data covariance matrix, its eigenvalues, and the corresponding eigenvectors arranged in descending order. The arrangement of subsequent rows of a transformation matrix $\mathbf{W}$ as the normalized eigenvectors, corresponding to the subsequent largest eigenvalues of the data covariance matrix, will result in an optimal linear transformation matrix $\hat{\mathbf{W}}$. The elements of the m-dimensional transformed feature vector $\mathbf{y}$ will be uncorrelated and arranged in decreasing order according to decreasing information content. This allows for a straightforward reduction of dimensionality (and thus data compression) by discarding trailing feature elements with the lowest information content. Depending on the nature of an original data pattern, one can obtain a substantial reduction of feature vector dimensionality $m << n$ compared with the dimensionality of original data patterns. First, having determined the optimal transformation matrix $\hat{\mathbf{W}}$, one can reduce the decorrelated feature vector dimension and use reduced feature vectors for classification. Second, all original n-dimensional data patterns can be optimally transformed to data patterns in the feature space with lower dimensionality. This means that the original data will be compressed with the minimal information loss when the data are reconstructed (preserving the maximal information content of the original data).

A PCA-based linear transformation of an original data pattern can also be interpreted as a projection of original patterns into m-dimensional feature space with orthonormal bases (guaranteeing that one obtains decorrelation of feature vector elements).

We can think of a PCA as an **unsupervised learning** from data. Indeed, PCA does not use knowledge about a class associated with a pattern, but only discovers correlation among patterns and their elements, as well as ordered intrinsic directions where the data patterns change most (with maximum variance), as shown in Figure 7.1.

Despite the fact that PCA is an unsupervised method, it can also be used in classifier design for the projection and reduction of feature patterns. Here, PCA is applied solely to patterns in order to determine an optimal transformation of original patterns into a principal component space and possibly to reduce the dimensionality of the projected pattern. Once an optimal transformation

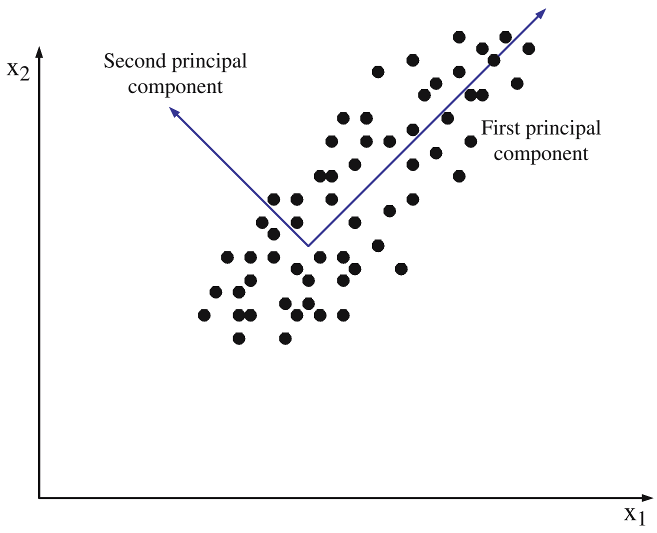

Figure 7.1. Principal components.

has been completed, the projected patterns will have the same class assignments as those in the original data set.

2.1.1. Statistical Characteristics of Data Required by PCA

Let us consider data objects characterized by n-dimensional column patterns $\mathbf{x} \in \mathbb{R}^n$ in n-dimensional pattern space whose elements take real values $x_i \in \mathbb{R}$. We assume that our knowledge about a domain is represented as a limited size sample (from a certain domain) of N random patterns $\mathbf{x}^i$ gathered as an unlabeled training data set T_{tra}:

$$T_{tra} = \{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\} \tag{3}$$

The entire training set data will be represented as an $N \times n$ data pattern matrix:

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^1)^T \\ (\mathbf{x}^2)^T \\ \vdots \\ (\mathbf{x}^N)^T \end{bmatrix} \tag{4}$$

One row of the data matrix contains one transposed pattern. If a data set contains patterns labeled by classes, for unsupervised PCA analysis we need to extract only patterns from this data set.

The data can be characterized by second-order statistics, namely, by the n-dimensional **mean** vector

$$\boldsymbol{\mu} = E[\mathbf{x}] = \left[E[x_1], E[x_2], \cdots, E[x_n]\right]^T \tag{5}$$

and the square $n \times n$-dimensional **covariance** matrix

$$\mathbf{R}_{xx} = \boldsymbol{\Sigma} = E\left[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\right] \tag{6}$$

where $E[\cdot]$ denotes the expectation operator and $\boldsymbol{\mu}$ is the mean vector of a pattern vector $\mathbf{x}$. The square, semipositive definite, symmetric ($r_{ij} = r_{ji}$), real-valued **covariance matrix $\mathbf{R}_{xx}$** describes correlations between elements of pattern vectors (treated as random variables). The PCA technique assumes that the original data patterns are zero-mean random vectors

$$\boldsymbol{\mu} = E[\mathbf{x}] = \mathbf{0} \tag{7}$$

If this condition is not satisfied, one can convert an original pattern $\mathbf{x}$ to the zero mean representation by the operation $\mathbf{x} - \boldsymbol{\mu}$. For zero mean patterns, the covariance matrix (equal to the correlation matrix) is defined as

$$\mathbf{R}_{xx} = \boldsymbol{\Sigma} = E[\mathbf{x}\mathbf{x}^T] \tag{8}$$

The true values $\boldsymbol{\mu}$ and $\mathbf{R}_{xx}$ for the mean vectors and the covariance matrix are in practice not available, since we usually do not know the exact probabilistic characteristics of patterns generated by nature. Our knowledge about a pattern-generation mechanism is included in a given data set T_{tra} containing a finite number of N patterns $\{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\}$. Under these circumstances, we find estimates for the mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^i \tag{9}$$

and the covariance matrix (unbiased estimate)

$$\hat{\mathbf{R}}_{xx} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathbf{x}^i - \boldsymbol{\mu})(\mathbf{x}^i - \boldsymbol{\mu})^T \tag{10}$$

based on a given limited sample. For zero-mean data, the covariance estimate becomes

$$\hat{\mathbf{R}}_{xx} = \frac{1}{N-1} \sum_{i}^{N} \mathbf{x}^i (\mathbf{x}^i)^T = \frac{1}{N-1} \mathbf{X}^T \mathbf{X} \tag{11}$$

where $\mathbf{X}$ is a whole $N \times n$ original data pattern matrix (data set).

The intrinsic characteristic of given data $\mathbf{X}$ can be found as a set of n eigenvalues λ_i and the corresponding eigenvectors $\mathbf{e}^i$ by solving the **eigenvalue problem**

$$\mathbf{R}_{xx}\mathbf{e}^i = \lambda_i \mathbf{e}^i, \quad i = 1, 2, \cdots, n \tag{12}$$

We consider the orthonormal eigenvectors, which is a legitimate approach since a covariance matrix $\mathbf{R}_{xx}$ is symmetric and real valued. This means that the eigenvectors are orthogonal $(\mathbf{e}^i)^T \mathbf{e}^j = 0 (i, j = 1, 2, \cdots, n, \ i \neq j)$ with unit length $\|\mathbf{e}^i\| = \sqrt{(\mathbf{e}^i)^T \mathbf{e}^i} = 1 (i = 1, 2, \cdots n)$.

In the PCA analysis, it is essential that the eigenvalues of the matrix $\mathbf{R}_{xx}$ are arranged in the decreasing order

$$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n \geq 0 \tag{13}$$

with $\lambda_1 = \lambda_{\max}$. The corresponding orthonormal eigenvectors $\mathbf{e}^i$ will be composed as the square $n \times n$ matrix

$$\mathbf{E} = [\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n] \tag{14}$$

with the i^{th} column representing one eigenvector $\mathbf{e}^i$ corresponding to the eigenvalue λ_i. The most dominant first eigenvector $\mathbf{e}^1$ in the first column of the matrix $\mathbf{E}$ corresponds to the first most dominant eigenvalue λ_1 of the covariance matrix $\mathbf{R}_{xx}$. The second most dominant eigenvector $\mathbf{e}^2$ in the second column corresponds to the second most dominant eigenvalue λ_2, etc.

The arrangement of eigenvalues and corresponding eigenvectors in descending order is essential for data dimensionality reduction. Only the first m principal components of projected feature vectors (those carrying the most information) and corresponding to the first m dominant eigenvalues should be considered.

The eigenvalue problem equation can be written in the matrix form

$$\mathbf{R}_{xx}\mathbf{E} = \mathbf{E}\boldsymbol{\Lambda} \tag{15}$$

where $\boldsymbol{\Lambda} = diag[\lambda_1, \lambda_1, \cdots, \lambda_n]$.

We can observe that for the orthonormal matrix $\mathbf{E}$ we have

$$\mathbf{E}^T\mathbf{E} = \mathbf{I} \tag{16}$$

where $\mathbf{I}$ is the $n \times n$ unit matrix. Consequently, we have

$$\mathbf{E}^{-1} = \mathbf{E}^T \tag{17}$$

In light of the above equality, we can write formulas for the so-called **orthogonal similarity transformation**

$$\mathbf{E}^{-1}\mathbf{R}_{xx}\mathbf{E} = \mathbf{E}^T\mathbf{R}_{xx}\mathbf{E} = \boldsymbol{\Lambda} \tag{18}$$

and consequently the **spectral factorization** of the covariance matrix $\mathbf{R}_{xx}$,

$$\mathbf{R}_{xx} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T \tag{19}$$

2.1.2. The Optimization Criterion of PCA

The goal of PCA is to find the optimal linear transformation $\mathbf{y} = \mathbf{W}\mathbf{x}$ of the original n-dimensional data patterns $\mathbf{x}$ into m-dimensional feature vectors $\mathbf{y}$, possibly with lower dimensionality ($m < n$). More formally, PCA can be considered as a static optimization problem, with a specifically defined optimization criterion, which will guarantee obtaining (through optimal projection) a transformed feature vector possessing the desired characteristics. In PCA, it is required that:

– optimal transformation is orthogonal (with orthonormal basis)
– elements of the transformed feature vector $\mathbf{y}$ are uncorrelated
– orthonormal basis of the linear projections shows, in decreasing order, the orthogonal intrinsic directions in data along which the data changes (variances) are maximal
– pattern reconstruction error will be minimal in the least-squares sense

For the orthonormal linear transformation $\mathbf{y} = \mathbf{W}\mathbf{x}$, with the $m \times n$-dimensional orthonormal tranformation matrix $\mathbf{W}$, an estimate of the reconstructed pattern is $\hat{\mathbf{x}} = \mathbf{W}^{-1}\mathbf{y}$. Since for the orthonormal matrices we have $\mathbf{W}^{-1} = \mathbf{W}^T$, thus

$$\hat{\mathbf{x}} = \mathbf{W}^{-1}\mathbf{y} = \mathbf{W}^T\mathbf{y} = \mathbf{W}^T\mathbf{W}\mathbf{x} \tag{20}$$

The criterion for the optimal, PCA-based, linear transformation is selected in order to guarantee obtaining a minimum of the reconstruction error metric. The reconstruction error-based criterion has the form

$$J_{\text{lse}}(\mathbf{W}) = E\left[\|\mathbf{x} - \hat{\mathbf{x}}\|^2\right] \tag{21}$$

where $\|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$ denotes the square of the Euclidean distance, denoted by the subscript *lse* indicating that the criterion used in optimization is based on the mean least squares error criterion. For practical computations, one can use the criterion

$$J_{\text{lse}}(\mathbf{W}) = \frac{1}{2}\sum_{i}^{N} \|\mathbf{x}^i - \hat{\mathbf{x}}^i\|^2 = \frac{1}{2}\sum_{i}^{N}\sum_{j}^{n}(x_j^i - \hat{x}_j^i)^2 \tag{22}$$

PCA seeks the optimal transformation matrix $\mathbf{W}$ that guarantees minimization of the mean squares reconstruction error (criterion $J_{\text{lse}}(\mathbf{W})$) for a given data set T_{tra}.

In order to better understand the goal of PCA, we provide a more detailed interpretation of the PCA criterion:

$$\begin{aligned} J(\mathbf{W}) &= E\left[\|\mathbf{x} - \hat{\mathbf{x}}\|^2\right] \\ &= E\left\{\text{trace}\left[(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T\right]\right\} \\ &= \text{trace}(\mathbf{R}_{xx}) - \text{trace}(\mathbf{W}\mathbf{R}_{xx}\mathbf{W}^T) \end{aligned} \tag{23}$$

In the above equations, we used the facts that $\text{trace}(\mathbf{W}) = \text{trace}(\mathbf{W}^T)$ and $\mathbf{W}\mathbf{W}^T = \mathbf{W}\mathbf{W}^{-1} = \mathbf{I}$, along with the following equalities:

$$\begin{aligned} \|\mathbf{x} - \hat{\mathbf{x}}\|^2 &= \text{trace}\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T\} \\ \text{trace}\{E[\mathbf{W}^T\mathbf{W}\mathbf{x}\mathbf{x}^T\mathbf{W}^T\mathbf{W}]\} &= \text{trace}\{\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{R}_{xx}\mathbf{W}^T\} \\ &= \text{trace}(\mathbf{W}\mathbf{R}_{xx}\mathbf{W}^T) \end{aligned} \tag{24}$$

We can observe that in the PCA criterion given in Equation (23), the second term $\mathrm{trace}(\mathbf{W}\mathbf{R}_{xx}\mathbf{W}^T) = J_{\mathrm{variance}}(\mathbf{W})$ and is equal to the variance of the projected feature vector $\mathbf{y}$, or consequently, equal to to the variance of the reconstructed pattern vector $\hat{\mathbf{x}}$:

$$J_{\mathrm{variance}}(\mathbf{W}) = \mathrm{trace}(\mathbf{W}\mathbf{R}_{xx}\mathbf{W}^T) = E\big[\mathrm{trace}(\mathbf{y}\mathbf{y}^T)\big] = \sum_{i=1}^{m} y_i^2 \tag{25}$$

$$J_{\mathrm{lse}}(\mathbf{W}) = \mathrm{trace}(\mathbf{W}^T\mathbf{W}\mathbf{R}_{xx}\mathbf{W}^T\mathbf{W}) = E[\mathrm{trace}(\hat{\mathbf{x}}\hat{\mathbf{x}}^T)] = \sum_{i=1}^{n} \hat{x}_i^2$$

The conclusion from the above criterion analysis is that minimization of the mean square error criterion $J_{\mathrm{lse}}(\mathbf{W})$ is in fact equivalent to maximization of the projected feature vector $\mathbf{y}$ variance (with the criterion J_{variance}). One can interpret PCA as minimization of the reconstruction error in the mean least squares sense, or equivalently as maximization of the resulting projection (feature vector) variance. We see that the optimal PCA-based linear transformation will result in a projection of the original patterns into the feature vectors, with elements located in the feature space in the directions with maximal variances (maximal variabilities).

2.1.3. PCA Theorem

For a given data set (a training set), let $T_{\mathrm{tra}} = \{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\}$, containing N n-dimensional zero-mean randomly generated patterns $\mathbf{x} \in \mathbb{R}^n$ with real-valued elements and the symmetric, real-valued $n \times n$ covariance matrix $\mathbf{R}_{xx} \in \mathbb{R}^{n\times n}$. Let the eigenvalues of the covariance matrix $\mathbf{R}_{xx}$ be arranged in decreasing order $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n \geq 0$ (with $\lambda_1 = \lambda_{\max}$). Assume that the corresponding orthonormal eigenvectors (orthogonal with unit length $||\mathbf{e}|| = 1$) $\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n$ compose the $n \times n$ orthonormal matrix

$$\mathbf{E} = [\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n] \tag{26}$$

with columns being orthonormal eigenvectors. Then the optimal linear transformation

$$\hat{\mathbf{y}} = \hat{\mathbf{W}}\mathbf{x} \tag{27}$$

transforms the original n-dimensional patterns $\mathbf{x}$ into m-dimensional $(m \leq n)$ feature patterns, minimizing the mean least squares reconstruction error criterion $J_{\mathrm{lse}}(\mathbf{W})$ given by Equation (25) (or maximizing the variance of projected patterns) and provides for the $m \times n$ optimal transformation matrix $\hat{\mathbf{W}}$, denoted also by $\mathbf{W}_{KL}$ (under the constraints $\mathbf{W}\mathbf{W}^T = \mathbf{I}$), as

$$\hat{\mathbf{W}} = \begin{bmatrix} (\mathbf{e}^1)^T \\ (\mathbf{e}^2)^T \\ \vdots \\ (\mathbf{e}^m)^T \end{bmatrix} \tag{28}$$

composed with m rows that are the first m orthonormal eigenvectors of the original data covariance matrix $\mathbf{R}_{xx}$.

The resulting optimal linear transformation $\mathbf{y} = \hat{\mathbf{W}}\mathbf{x}$, with the optimal transformation matrix $\hat{\mathbf{W}}$, is called the **Karhunen-Loéve** (KLT) or **Hotelling** transformation.

2.1.4. Properties of the Karhunen-Loéve Transformation

The optimal KLT transformation guarantees the minimum reconstruction error in the least squares sense, with the minimal value

$$\min J_{\mathrm{lse}}(\mathbf{W}) = \sum_{i=m+1}^{n} \lambda_i \tag{29}$$

The minimum value of the reconstruction error is equal to the sum of the trailing $n-m$ eigenvalues $\lambda_{m+1}, \lambda_{m+2}, \cdots, \lambda_n$ (from the ordered eigenvalues) of the covariance matrix $\mathbf{R}_{xx}$, where m is the possibly reduced length of the projected feature vector $\mathbf{y}$ ($m \leq n$).

Simultaneously, the transformation guarantees the maximum of the projected feature vector variance, with the maximum value

$$\max J_{\text{variance}}(\mathbf{W}) = \sum_{i=1}^{m} \lambda_i \tag{30}$$

equal to the sum of the first m eigenvalues of $\mathbf{R}_{xx}$. The orthonormal eigenvectors $\mathbf{e}^1$, $\mathbf{e}^2$, $\cdots$, $\mathbf{e}^n$ (rows of the optimal transformation matrix $\hat{\mathbf{W}}$), corresponding to the descending-order eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ of the data covariance matrix $\mathbf{R}_{xx}$, are called the **principal eigenvectors**. They show orthogonal directions (in descending order, corresponding to the principal eigenvectors and eigenvalues) in the pattern space where data change maximally (with maximal variance). The m principal eigenvectors (arranged as rows) compose the optimal transformation matrix $\hat{\mathbf{W}}$.

For a given n-dimensional random pattern $\mathbf{x}$, the optimal transformation $\mathbf{y} = \hat{\mathbf{W}}\mathbf{x}$ will produce the optimally projected m-dimensional feature vector $\mathbf{y} = [y_1, y_2, \cdots, y_m]^T$. The elements $y_1, y_2, \cdots, y_m$ of the feature vector are called the **principal components** of a pattern $\mathbf{x}$. The principal components are statistically uncorrelated with covariances

$$E[y_i y_j] = \mathbf{e}_i^T \mathbf{R}_{xx} \mathbf{e}_j = 0 \tag{31}$$

and with variances equal to the corresponding eigenvalues

$$E[y_i y_i] = E[y_i^2] = (\mathbf{e}^i)^T \mathbf{R}_{xx} \mathbf{e}^i = \lambda_i \tag{32}$$

The covariance matrix $\mathbf{R}_{yy} = E[\mathbf{y}\mathbf{y}^T]$ of the projected feature vectors $\mathbf{y}$ is diagonal, with eigenvalues of the original pattern covariance matrix $\mathbf{R}_{xx}$ on the main diagonal given in descending value order. The variances are arranged in descending variance value order $E[y_1^2] \leq E[y_2^2] \leq \cdots \leq E[y_m^2]$.

The i^{th} principal component y_i of the original pattern $\mathbf{x}$, corresponding to the i^{th} largest eigenvalue λ_i of the covariance matrix $\mathbf{R}_{xx}$, is obtained as an inner product

$$y_i = (\mathbf{e}^i)^T \mathbf{x} = e_1^i x_1 + e_2^i x_2 + \cdots + e_n^i x_n \tag{33}$$

of the i^{th} orthonormal eigenvector $\mathbf{e}^i$ (i^{th} row of $\hat{\mathbf{W}}$) and a given pattern $\mathbf{x}$. It is just a linear combination of the elements of pattern vector $\mathbf{x}$.

The first principal component $y_1 = (\mathbf{e}^1)^T \mathbf{x}$, corresponding to the first most dominant eigenvalue λ_1 (and first eigenvector $\mathbf{e}^1$) of the covariance matrix, is such that its variance

$$\text{variance}(y_1) = E[y_1^2] = E[(\mathbf{e}^1 \mathbf{x})^2] = \mathbf{e}^1 E[\mathbf{x}\mathbf{x}^T](\mathbf{e}^1)^T = \mathbf{e}^1 \mathbf{R}_{xx} (\mathbf{e}^1)^T = \lambda_1 \tag{34}$$

is maximal. The first most dominant principal component y_1 is along the first eigenvector direction $\mathbf{e}^1$, with maximum variance equal to the most dominant eigenvalue λ_1 of the covariance matrix.

Subsequently, the second principal component $y_2 = (\mathbf{e}^2)^T \mathbf{x}$, corresponding to the second dominant eigenvalue λ_2 (and the second eigenvector $\mathbf{e}^2$), is such that its variance

$$\text{variance}(y_2) = E[y_2^2] = \lambda_2 \tag{35}$$

is maximal. The second dominant principal component y_2 is along the second eigenvector direction $\mathbf{e}^2$, with the second maximum variance equal to the second dominant eigenvalue λ_2 of the covariance matrix. The direction of the second principal component is perpendicular to the direction of the first most dominant principal component. Similarly, the third dominant principal component $y_3 = (\mathbf{e}^3)^T\mathbf{x}$ is along the third eigenvector direction $\mathbf{e}^3$, with the third maximum variance equal to the third dominant eigenvalue λ_3 of the covariance matrix. The direction of the third principal component is perpendicular (orthogonal) to the direction of the first and second dominant principal component. Generally, the i^{th} principal component will be in the direction orthogonal to all prior principal components $y_1, y_2, \cdots, y_{i-1}$, with the maximal value of variance in this direction equal to λ_i. The m principal components form principal component space into which patterns $\mathbf{x}$ are optimally projected. Most information is contained along the first principal component.

2.1.5. Optimal KLT Transformation of the Original Patterns

Once we have defined the optimal KLT transformation matrix $\hat{\mathbf{W}}$, the optimal transformation of a given n-dimensional original pattern $\mathbf{x}$ into the m-dimensional optimal feature pattern $\mathbf{y}$ is given by $\mathbf{y} = \hat{\mathbf{W}}\mathbf{x}$. The inverse transformation can be obtained from

$$\hat{\mathbf{x}} = \hat{\mathbf{W}}^{-1}\mathbf{y} = \hat{\mathbf{W}}^T\mathbf{y} = \sum_{i=1}^{m} y_i\mathbf{e}^i \tag{36}$$

The optimal transformation of the entire original data set $\mathbf{X}$ is given by the formula

$$\mathbf{Y} = (\hat{\mathbf{W}}\mathbf{X}^T)^T = \mathbf{X}\hat{\mathbf{W}}^T \tag{37}$$

The $m \times m$ covariance matrix $\mathbf{R}_{yy} = E[\mathbf{y}\mathbf{y}^T]$ for the projected patterns $\mathbf{y}$ can be estimated as

$$\mathbf{R}_y = \frac{1}{N-1}\mathbf{Y}^T\mathbf{Y} = \hat{\mathbf{W}}\mathbf{R}_{xx}\hat{\mathbf{W}}^T = \text{diag}[\lambda_i] \tag{38}$$

The entire reconstructed pattern set is given by

$$\hat{\mathbf{X}} = \mathbf{Y}\hat{\mathbf{W}} \tag{39}$$

2.1.6. Dimensionality Reduction

PCA can be effectively used for feature extraction and dimensionality reduction. Instead of the entire n-dimensional original data pattern $\mathbf{x}$, one can form the m-dimensional $(m \leq n)$ feature vector $\mathbf{y} = [y_1, y_2, \cdots, y_m]^T$ containing only the first m most dominant principal components of $\mathbf{x}$, corresponding to the first m most dominant eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_m$ of the original data pattern covariance matrix. PCA is the best technique for linear feature extraction from the original set of patterns, in the sense of minimization of the reconstruction error. Projection of patterns from highly dimensional pattern space $(\mathbf{x} \in \mathbb{R}^n)$ onto reduced principal component space (with dimension $m << n$) will result in the square of the approximation error

$$\mathbf{e}^2 = \sum_{i=m+1}^{n} \lambda_i \tag{40}$$

which is equal to the sum of $n - m$ discarded eigenvalues. We also can say that the m principal components y_i are the **most expressive** features of a data set. PCA provides a way to reduce (compress) data representation by choosing the feature vector with lower dimensionality. The reduced-size feature vectors represent data in a new feature space, where the feature vector elements are uncorrelated and placed along the orthogonal directions of principal components with maximal variances. This characteristic might be desirable in classifier design for some types of data. However, PCA does not consider the improvement of classification in principal component space, as a transformation criterion. This mean that most expressive features obtained through principal components are well suited for data representation (the model) and compression. However, they might not be good for classification. The data in the feature space will be an approximation (a model) of the original data patterns.

The approximation error vector **e** is orthogonal to the reconstructed data pattern $\hat{\mathbf{x}}$. The least squares error between **x** and $\hat{\mathbf{x}}$ is

$$||\mathbf{e}|| = ||\mathbf{x} - \hat{\mathbf{x}}|| = \left[\sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \right]^{1/2} \tag{41}$$

The least mean squares error is equal to

$$E[||\mathbf{x} - \hat{\mathbf{x}}||^2] = \sum_{i=m+1}^{n} \lambda_i \tag{42}$$

and it is also equal to the sum of $n - m$ eigenvalues not used.

One method (criterion) for (data representation oriented) selection of a dimension of a reduced feature vector **y** is to choose a minimal number of the first m most dominant principal components $y_1, y_2, \cdots, y_m$ of **x** for which the mean square reconstruction error is less than the heuristically set error threshold ϵ. Another, more practical method may be to select the minimal number of the first m most dominant principal components for which a percentage V of a sum of unused eigenvalues of a sum of all eigenvalues

$$V = \frac{\sum_{i=m+1}^{n} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \, 100\% \tag{43}$$

and is less than a defined threshold ζ: $P < \zeta$.

One can try to use principal components for classification. Selection of the best principal components for classification purposes is outside the scope of this Chapter. In such a case, Fisher's Linear Discriminant Analysis (described in Section 2.2) should be used.

Example: Let us consider a data set (shown in Table 7.1) containing 10 two-feature patterns $\mathbf{x} \in \mathbb{R}^2$ from two classes $c_1 = 0$ and $c_2 = 1$ (five patterns in each class) drawn according to the Gaussian normal density distribution.

The patterns $\mathbf{x} \in \mathbb{R}^2$ from both classes of the original data set can be composed as a 10×2(n=2, $N = 10$) data pattern matrix $\mathbf{X}_{\text{orig}}$ (Table 7.2). The mean vector for the original patterns **x**

Table 7.1. Two-class data set.

$\mathbf{x_1}$	$\mathbf{x_2}$	**class**
1	2	1
2	2	1
2	3	1
3	1	1
3	2	1
6	8	2
7	8	2
8	7	2
8	8	2
7	9	2

Table 7.2. Data pattern matrix

$$\mathbf{X}_{\text{orig}} = \begin{bmatrix} 1 & 2 \\ 2 & 2 \\ 2 & 3 \\ 3 & 1 \\ 3 & 2 \\ 6 & 8 \\ 7 & 8 \\ 8 & 7 \\ 8 & 8 \\ 7 & 9 \end{bmatrix}$$

is $\boldsymbol{\mu} = [4.7,\ 5.0]^T$, and the zero mean data pattern can be obtained by extraction $\mathbf{x} - \boldsymbol{\mu}$, yielding the zero mean data pattern matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} -3.7 & -3.0 \\ -2.7 & -3.0 \\ -2.7 & -2.0 \\ -1.7 & -4.0 \\ -1.7 & -3.0 \\ 1.3 & 3.0 \\ 2.3 & 3.0 \\ 3.3 & 2.0 \\ 3.3 & 3.0 \\ 2.3 & 4.0 \end{bmatrix}$$

The 2×2 covariance matrix $\mathbf{R}_{xx}$ of the data patterns $\mathbf{X}$ is

$$\mathbf{R}_{xx} = \begin{bmatrix} 7.5667 & 8.1111 \\ 8.1111 & 10.4444 \end{bmatrix}$$

$\lambda_1 = 0.7678$, $\lambda_2 = 17.2433$, and $[0.6424\ 0.7624]^T$. Arranged in decreasing order, the eigenvalues and corresponding orthogonal eigenvectors of matrix $\mathbf{R}_{xx}$ are $\lambda_1 = 17.2433$, $\lambda_2 = 0.7678$, and $\mathbf{e}^1 = [0.6424\ 0.7664]^T$, $\mathbf{e}^2 = [-0.7664\, 0.6424]^T$. Matrix $\mathbf{E}$, composed with eigenvectors $\mathbf{e}^1$ and $\mathbf{e}^2$ as columns, is

$$\mathbf{E} = \begin{bmatrix} 0.6424 & -0.7664 \\ 0.7664 & 0.6424 \end{bmatrix}$$

We can easily see that the eigenvectors are orthonormal: orthogonal $((\mathbf{e})^1)^T \mathbf{e}^2 = 0$ with unit length $||\mathbf{e}^i|| = 1$. Finally, the optimal KLT transformation matrix $\hat{\mathbf{W}}$, with rows of eigenvectors corresponding to the decreasing-order eigenvalues of a covariance matrix, is

$$\hat{\mathbf{W}} = \begin{bmatrix} 0.6424 & 0.7664 \\ -0.7664 & 0.6424 \end{bmatrix}$$

The projected first vector $\mathbf{x}^1 = [-3.7,\ -3.0]^T$ from $\mathbf{X}$ gives, as a feature vector in the principal component space, $\mathbf{y}^1 = [-4.6760,\ 0.9084]^T$. Here, $y_1^1 = -4.6760$ is the first principal component of the pattern $\mathbf{x}^1$ along the direction of the first eigenvector $\mathbf{e}^1$. The data have maximal variance $\lambda_1 = 17.2433$ in this direction. Consequently, $y_2^1 = 0.9084$ is the second principal component of the pattern $\mathbf{x}^1$ along the direction of the second eigenvector $\mathbf{e}^2$ orthogonal to $\mathbf{e}^1$. The projection of all vectors from $\mathbf{X}$ onto principal components are $\mathbf{Y} = \mathbf{X}\hat{\mathbf{W}}^T$:

$$\mathbf{Y} = \begin{bmatrix} -4.6760 & 0.9084 \\ -4.0336 & 0.1421 \\ -3.2672 & 0.7844 \\ -4.1576 & -1.2667 \\ -3.3912 & -0.6243 \\ 3.1342 & 0.9309 \\ 3.7766 & 0.1645 \\ 3.6526 & -1.2443 \\ 4.4190 & -0.6019 \\ 4.5430 & 0.8069 \end{bmatrix}$$

The covariance matrix of $\mathbf{Y}$ is diagonal, with elements equal to λ_1 and λ_2:

$$\begin{bmatrix} 17.2433 & 0.0000 \\ 0.0000 & 0.7678 \end{bmatrix}$$

The first original pattern vector $\mathbf{x}^1 = [-3.7,\ -3.0]^T$ can be reconstructed, by the inverse operation $\hat{\mathbf{x}} = \hat{\mathbf{W}}^T\mathbf{y}$, from the principal component space vectors $\mathbf{y}^1 = [-4.6760,\ 0.9084]^T$:

$$\hat{\mathbf{x}}^1 = \begin{bmatrix} 0.6424 & -0.7664 \\ 0.7664 & 0.6424 \end{bmatrix} \begin{bmatrix} -4.6760 \\ 0.9084 \end{bmatrix} = \begin{bmatrix} -3.7 \\ -3.0 \end{bmatrix}$$

The reconstruction is exact, since the dimension of the original pattern and the feature patterns are equal $(m = n)$, and consequently we used all eigenvectors in the optimal transformation matrix $\hat{\mathbf{W}}$. Reconstruction of all the original pattern space vectors $\hat{\mathbf{X}}$, by the inverse operation $\hat{\mathbf{X}} = \mathbf{Y}\mathbf{W}$, gives

$$\hat{\mathbf{X}} = \begin{bmatrix} -3.7 & -3.0 \\ -2.7 & -3.0 \\ -2.7 & -2.0 \\ -1.7 & -4.0 \\ -1.7 & -3.0 \\ 1.3 & 3.0 \\ 2.3 & 3.0 \\ 3.3 & 2.0 \\ 3.3 & 3.0 \\ 2.3 & 4.0 \end{bmatrix}$$

The fully reconstructed original pattern vectors, with the mean vector $\hat{\mathbf{x}}_{\text{orig}} = \hat{\mathbf{x}} + \boldsymbol{\mu}$ added, are equal to the original patterns $\mathbf{X}_{\text{orig}}$:

$$\hat{\mathbf{X}}_{\text{orig}} = \begin{bmatrix} 1 & 2 \\ 2 & 2 \\ 2 & 3 \\ 3 & 1 \\ 3 & 2 \\ 6 & 8 \\ 7 & 8 \\ 8 & 7 \\ 8 & 8 \\ 7 & 9 \end{bmatrix}$$

Let us consider now how we can model considered zero mean data patterns $\mathbf{X}$ by only one latent variable $\mathbf{y} = [y_1] \in \mathbb{R}(m = 1)$ being the first principal component of two dimensional patterns $\mathbf{x}$. Here, we form the 1×2 optimal KLT transformation matrix $\hat{\mathbf{W}}$ by choosing as its sole row the first eigenvector $\mathbf{e}^1$ of the covariance matrix $\mathbf{R}_{xx}$, corresponding to the first most dominant eigenvalue λ_1:

$$\hat{\mathbf{W}} = [\mathbf{e}^1] = [0.6424 \quad 0.7664]$$

The projected first vector $\mathbf{x}^1 = [-3.7, \ -3.0]^T$ from $\mathbf{X}$ gives, as the feature vector in the principal component space, $\mathbf{y}^1 = [-4.6760]$. Here, $y_1^1 = -4.6760$ is the first principal component of the pattern $\mathbf{x}^1$ along the direction of the first eigenvector $\mathbf{e}^1$. The data have maximal variance λ_1 in this direction. The projection of all vectors from $\mathbf{X}$ onto the first principal component gives $(\mathbf{Y} = \mathbf{X}\hat{\mathbf{W}}^T)$

$$\mathbf{Y} = \begin{bmatrix} -4.6760 \\ -4.0336 \\ -3.2672 \\ -4.1576 \\ -3.3912 \\ 3.1342 \\ 3.7766 \\ 3.6526 \\ 4.4190 \\ 4.5430 \end{bmatrix}$$

The covariance of $\mathbf{Y} = [17.2433]$ for λ_1. The first original pattern vector, $\mathbf{x}^1 = [-3.7, \ -3.0]^T$, can be reconstructed, by the inverse operation $\hat{\mathbf{x}} = \hat{\mathbf{W}}^T\mathbf{y}$, from the principal component space vectors $\mathbf{y}^1 = [-4.6760]$:

$$\hat{\mathbf{x}}^1 = \begin{bmatrix} 0.6424 \\ 0.7664 \end{bmatrix} [-4.6760] = \begin{bmatrix} -3.0038 \\ -3.5836 \end{bmatrix}$$

The reconstructed error vector is $\mathbf{x}^1 - \hat{\mathbf{x}}^1 = [-0.6962, \ 0.5836]^T$. Reconstruction of all original pattern space vectors $\hat{\mathbf{X}}$, by the inverse operation $\hat{\mathbf{X}} = \mathbf{Y}\mathbf{W}$, gives

$$\hat{\mathbf{X}} = \begin{bmatrix} -3.0038 & -3.5836 \\ -2.5911 & -3.0913 \\ -2.0988 & -2.5039 \\ -2.6708 & -3.1863 \\ -2.1785 & -2.5989 \\ 2.0134 & 2.4020 \\ 2.4261 & 2.8943 \\ 2.3464 & 2.7993 \\ 2.8387 & 3.3866 \\ 2.9184 & 3.4817 \end{bmatrix}$$

The reconstructed error vectors are

$$\mathbf{X}-\hat{\mathbf{X}}=\begin{bmatrix} 0.6962 & 0.5836 \\ -0.1089 & 0.0913 \\ -0.6012 & 0.5039 \\ 0.9708 & -0.8137 \\ 0.4785 & -0.4011 \\ -0.7134 & 0.5980 \\ -0.1261 & 0.1057 \\ 0.9536 & -0.7993 \\ 0.4613 & -0.3866 \\ -0.6184 & 0.5183 \end{bmatrix}$$

The least mean squares error is equal to $\lambda_2 = 0.7678$ (its numerical estimate as the mean of the pattern error equals 0.7478).

2.2. Supervised Feature Extraction Based on Fisher's Linear Discriminant Analysis

Fisher's linear discriminant method, with a linear transformation of the original patterns, is the classic method of real-valued feature extraction and pattern **dimensionality reduction**. This linear transformation is obtained based on statistical characteristics extracted from a given data set represented by data pattern **scatter matrices** (proportional to covariance matrices). Fisher's linear transformation is constructed based on a given limited-size data set T_{tra}, containing N examples. Each example $(\mathbf{x}^i, c^i_{\text{target}})$ $(i = 1, 2, \cdots, N)$, representing one object of recognition, is constituted with an n-dimensional real-valued pattern $\mathbf{x} \in \mathbb{R}^n$ with corresponding target class c^i_{target}. We assume that a data set T_{tra} contains N_i $(\sum_i^l N_i = N)$ examples from each categorical class c_i, with the total number of classes denoted by l.

Fisher's linear discriminant analysis is a method of **supervised learning** from data, since it considers patterns labeled by target classes and reveals and uses measures of pattern scatter through the total data set, as well as within and between classes. A linear transformation of an original data pattern can also be interpreted as a projection of original patterns into a reduced m-dimensional feature space. Here, feature space reduction and feature extraction via transformation are included in one transformation.

Fisher's linear discriminant analysis, based on statistical data-analysis techniques, determines an optimal linear transformation

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{44}$$

of a real-valued n-dimensional data pattern $\mathbf{x}$ into another m-dimensional $(m \le n)$ transformed pattern $\mathbf{y}$. The $m \times n$ transformation matrix $\mathbf{W}$ is designed optimally from the point of view of maximal interclass separability of projected patterns. This design allows us to find a reduced compact data representation, in lower-dimensionality pattern space, with maximal separability between classes, thereby allowing us to design a better classifier.

Designing Fisher's linear transformation as a static optimization problem with specifically defined optimization criterion leads to obtaining an optimal transformation. The criterion J_{Fisher} is selected for the optimal Fisher linear transformation in order to ensure that we obtain a minimum interclass separability of transformed patterns. The evaluation of interclass separability is based on scatter matrices (proportional to covariance matrices) estimated for a given data set for each class and between classes. It is constructed to choose new features, by the transformation of original features to fewer new features, which will ensure a small within-class scatter and a large between-class scatter.

2.2.1. Two-class Data and Fisher's Projection onto a Line

First, we assume a two-class data set and analyze a linear transformation of n-dimensional patterns $\mathbf{x} \in \mathbb{R}^n$ into a **one-dimensional** feature space, with patterns $y \in \mathbb{R}$ containing one feature. This transformation has the linear form

$$y = \mathbf{w}^T \mathbf{x} \tag{45}$$

where $\mathbf{w} = [w_1, w_2, \cdots, x_n]^T$ is an n-dimensional transformation vector (containing n adjustable coefficients). We see that this is a linear projection of multidimensional patterns in the n-dimensional pattern space onto a line in the one-dimensional reduced feature space y. Here, y is an extracted and reduced data feature. Even for well-clustered and separated patterns from two classes in the original n-dimensional pattern space, such linear projection onto a line may result in a large loss of information and a confusing overlap of patterns from both classes. However, by rotating a line of projection (by changing a transformation coefficient w_i), we can find a line position that will give maximal separability of projected patterns from the two classes. This observation sets the foundation for Fisher's linear transformation (projection), which can be generalized also for multiclass data sets.

We will discuss Fisher's linear transformation as constructed based on a given limited-size data set T_{tra}. The set T_{tra} contains N examples (containing patterns labeled by classes). Each example $(\mathbf{x}^i, c^i_{\text{target}})(i = 1, 2, \cdots, N)$ is constituted with an n-dimensional real-valued pattern $\mathbf{x} \in \mathbb{R}^n$ with corresponding target class c^i_{target} $(i = 1, 2)$. We assume that a data set T_{tra} (a training set) contains N_i $(N_1 + N_2 = N)$ examples from each of two categorical classes c_1, c_2, with the total number of classes denoted by $l = 2$.

2.2.2. Statistical Characteristics of Patterns from an Original n-feature Data Set T_{tra}

First, we should provide measures of the statistical characteristics of patterns from an original n-feature data set T_{tra}. The mean for each class, which shows a separation of patterns from each class, can be estimated by

$$\boldsymbol{\mu}_i = \frac{1}{N_i} \sum_{\mathbf{x} \in xc_i} \mathbf{x}, \quad (i = 1, 2) \tag{46}$$

where $\mathbf{x} \in xc_i$ denotes a set of patterns from the class c_i. In order to calculate the scatter of patterns within one class c_i around this class mean, the $n \times n$-dimensional **within-class** c_i **scatter matrix** is defined as

$$\mathbf{S}_i = \sum_{\mathbf{x} \in xc_i} (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T, \quad (i = 1, 2) \tag{47}$$

This matrix is proportional to the covariance matrix; it is symmetric and positive semidefinite (and, for $N_i > n$, usually nonsingular). The summarizing measure for the scatter of patterns around means for all l classes, the so-called **within-class scatter matrix**, can be defined as

$$\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2 \tag{48}$$

The **total scatter matrix** (for all N_{all} patterns from T_{tra})

$$\mathbf{S}_t = \sum_{j=1}^{N} (\mathbf{x}^j - \boldsymbol{\mu})(\mathbf{x}^j - \boldsymbol{\mu})^T \tag{49}$$

illustrates a between-class scatter. The **total data mean** can be estimated by

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}^j = \frac{1}{N}(N_1 \boldsymbol{\mu}_1 + N_2 \boldsymbol{\mu}_2) \tag{50}$$

We find that the total scatter matrix can be decomposed into two matrices

$$\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b \tag{51}$$

where $\mathbf{S}_w$ is the within-class scatter matrix, and the $n \times n$ square matrix $\mathbf{S}_b$ is the **between-class scatter matrix**, defined as

$$\mathbf{S}_b = N_1(\boldsymbol{\mu}_1 - \boldsymbol{\mu})(\boldsymbol{\mu}_1 - \boldsymbol{\mu})^T + N_2(\boldsymbol{\mu}_2 - \boldsymbol{\mu})(\boldsymbol{\mu}_2 - \boldsymbol{\mu})^T \tag{52}$$

2.2.3. Statistical Characteristics of the Projected Original Data Set Patterns onto a Single Feature y Pattern Space

Similar statistical characteristics can be provided for projected original data set patterns onto a single feature y pattern space (with linear transformation $y = \mathbf{w}^T\mathbf{x}$). All projected patterns y form a projected data set $T_{\text{tra, proj}}$ with N cases $(y^i, c^i_{\text{target}})$ containing labeled single feature patterns $y \in \mathbb{R}$. The mean of projected patterns y in $T_{\text{tra, proj}}$ for each class c_1 and c_2 can be estimated from

$$\mu_{i,p} = \frac{1}{N_i}\sum_{y \,\in\, yc_i} y = \frac{1}{N_i}\sum_{\mathbf{x} \,\in\, xc_i} \mathbf{w}^T\mathbf{x} = \mathbf{w}^T\boldsymbol{\mu}_i \quad (i = 1, 2) \tag{53}$$

and the scatter of a feature y (from projected original data) for each class is

$$s^2_{i,p} = \frac{1}{N_i}\sum_{y \,\in\, yc_i} (y - \mu_{i,p})^2, \quad (i = 1, 2) \tag{54}$$

Note that the estimate of variance of the projected patterns for each class is $(1/N_i)s^2_{i,p}$. The total within-class scatter of y in an entire data set $T_{\text{tra, proj}}$ is defined by

$$s^2_{t,p} = s^2_{1,p} + s^2_{2,p} \tag{55}$$

We can see that the distance between projected means can be found to be $|\mu_{1,p} - \mu_{2,p}| = |\mathbf{w}^T(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)|$.

The design criterion for optimal interclass separability

$$J_{\text{Fisher}}(\mathbf{W}) = \frac{|(\mu_{1,p} - \mu_{2,p})^2|}{s^2_{1,p} + s^2_{2,p}} \tag{56}$$

guarantees a large value either for larger between-class scatter or for smaller within-class scatter.

The above criterion for a single feature, based on interclass separability, is called the Fisher **F-ratio**. It guarantees finding an optimal linear transformation $y = \mathbf{w}^T\mathbf{x}$ of n-feature patterns into one feature pattern y, maximizing the between-class variance while simultaneously minimizing the within-class variance.

Let us sketch a solution for an optimal linear transformation. We try to find an optimal transformation vector $\hat{\mathbf{w}}$ providing maximization of a criterion $J_{\text{Fisher}}(\mathbf{W})$. Derivations show that $(\mu_{1,p} - \mu_{2,p})^2 = \mathbf{w}^T\mathbf{S}_b\mathbf{w}$, with

$$\mathbf{S}_b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \tag{57}$$

and $s^2_{t,p} = s^2_{1,p} + s^2_{2,p} = \mathbf{w}^T\mathbf{S}_w\mathbf{w}$. Thus, the criterion J for interclass separability of projected patterns y as a function of $\mathbf{w}$ is given by

$$J_{\text{Fisher}}(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_b\mathbf{w}}{\mathbf{w}^T\mathbf{S}_w\mathbf{w}} \tag{58}$$

It is known that the optimal transform vector $\mathbf{w}$ that minimizes the criterion functional $J(\mathbf{w})$ must be a solution of the generalized eigenvalue problem

$$\mathbf{S}_b\mathbf{w} = \lambda\mathbf{S}_w\mathbf{w} \tag{59}$$

or, written in another form (after multiplying both sides of the criterion by $\mathbf{S}^{-1}$),

$$\mathbf{S}_w^{-1}\mathbf{S}_b\mathbf{w} = \lambda\mathbf{w} \tag{60}$$

For a nonsingular $\mathbf{S}_w$, we have a final solution for the optimal value of Fisher's linear transformation vector $\hat{\mathbf{w}}$ (Fukunaga, 1990):

$$\hat{\mathbf{w}} = \mathbf{S}_w^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{61}$$

2.2.4. Fisher Linear Discriminant – Classification

The main purpose of Fisher's linear transformation is to optimally convert patterns into a pattern space with a reduced dimensionality of projected pattern more suitable for the classification. However, we can design a classifier based on Fisher's linear transformation by using an optimal pattern projection formula that acts like a discriminant:

$$y = d(\mathbf{x}) = \hat{\mathbf{w}}^T\mathbf{x} \tag{62}$$

If we choose a threshold value $y_{\text{threshold}}$ for the projected pattern y, a new pattern $\mathbf{x}$ could be classified as belonging to class $c_i = 1$ if $y = d(\mathbf{x}) = \hat{\mathbf{w}}^T\mathbf{x}) \geq y_{\text{threshold}}$, and otherwise as belonging to class $c_i = 2$.

Let us consider the original l-class data set T_{tra} with n-dimensional class labeled patterns. We assume that for the class data set T_{tra} with n dimensions, the dimensionality of the original patterns is smaller than a number of classes $n > l$. We will discuss how to transform n-dimensional patterns from an original data set T_{tra} into reduced-size m-dimensional feature patterns (with $m > 1$), thereby ensuring maximal interclass separability in the projected space. Later we will show that $m > l$.

Here we have m linear transformations (discriminants)

$$y_i = \mathbf{w}_i^T\mathbf{x},\ (i = 1, 2, \cdots, m) \tag{63}$$

for all features y_i of projected patterns, with $\mathbf{w}_i$ being the transformation column vectors (for y_i discriminant). All m features of projected patterns form an m-dimensional projected feature pattern $\mathbf{y} = [y_1, y_2, \cdots, y_m]^T \in \mathbb{R}^m$. The linear transformation has the form

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{64}$$

where $\mathbf{W}$ is an $m \times n$-dimensional transformation matrix, with each row $\mathbf{w}_i$ being a transformation vector for a corresponding feature y_i $(i = 1, 2, \cdots, m)$.

One can generalize definitions for the statistical characteristics for both data sets, the original T_{tra} and the projected $T_{\text{tra, proj}}$, needed to find an optimal transform matrix $\hat{\mathbf{W}}$.

The $n \times n$ **within-class c_i scatter matrix** is defined as

$$\mathbf{S}_i = \sum_{\mathbf{x} \in xc_i} (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T, \quad (i = 1, 2, \cdots, l) \tag{65}$$

where $\boldsymbol{\mu}_i$ is the mean

$$\boldsymbol{\mu}_i = \frac{1}{N_i} \sum_{\mathbf{x} \in xc_i} \mathbf{x},\ i = 1, 2, \cdots, l \tag{66}$$

for patterns within each class. The **within-class scatter matrix** is defined as

$$\mathbf{S}_w = \sum_{i=1}^{l} \mathbf{S}_i \tag{67}$$

The **total scatter matrix** for all patterns is defined as

$$\mathbf{S}_t = \sum_{j=1}^{N} (\mathbf{x}^j - \boldsymbol{\mu})(\mathbf{x}^j - \boldsymbol{\mu})^T \tag{68}$$

where $\boldsymbol{\mu}$ is the estimate of the **total data mean**

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}^j = \frac{1}{N} \sum_{i=1}^{l} N_i \boldsymbol{\mu}_i \tag{69}$$

We can find the decomposition

$$\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b \tag{70}$$

where $\mathbf{S}_w$ is the within-class scatter matrix, and the $n \times n$ **between-class scatter matrix $\mathbf{S}_b$** is defined as

$$\mathbf{S}_b = \sum_{i=1}^{l} N_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T \tag{71}$$

Now, the projection of an n-dimensional pattern space into an $(l-1)$-dimensional discriminant space is given by $(l-1)$ functions

$$y_i = \mathbf{w}_i^T \mathbf{x}, \;\; i = 1, \cdots, l-1 \tag{72}$$

We can also write the matrix version of above equation, assuming that $\mathbf{y} \in \mathbb{R}^{(l-1)}$ and that $\mathbf{W}$ is the $n \times (l-1)$ matrix containing, as rows, the transpose of the transformation weigh vectors $\mathbf{w}_i$:

$$\mathbf{y} = \mathbf{W}^T \mathbf{x} \tag{73}$$

Projections of all the original patterns $\{\mathbf{x}^1, \cdots, \mathbf{x}^N\}$ will result in the set of corresponding projected patterns $\{\mathbf{y}^1, \cdots, \mathbf{y}^N\}$. One can find similar statistical characteristics (denoted by p) for the projected data set $T_{\text{tra, proj}}$ with m-dimensional projected patterns $\mathbf{y}$:

$$\boldsymbol{\mu}_{i,p} = \frac{1}{N_i} \sum_{\mathbf{y} \in yc_i} \mathbf{y}, \;\; (i = 1, 2, \cdots, l) \tag{74}$$

$$\boldsymbol{\mu}_p = \frac{1}{N} \sum_{j=1}^{N} \mathbf{y}^j = \frac{1}{N} \sum_{i=1}^{l} N_i \boldsymbol{\mu}_{i,p} \tag{75}$$

$$\mathbf{S}_{i,p} = \sum_{\mathbf{y} \in yc_i} (\mathbf{y} - \boldsymbol{\mu}_{i,p})(\mathbf{y} - \boldsymbol{\mu}_{i,p})^T, \;\; (i = 1, 2, \cdots, l) \tag{76}$$

$$\mathbf{S}_{w,p} = \sum_{i=1}^{l} \mathbf{S}_{i,p}, \;\; \mathbf{S}_{t,p} = \sum_{j=1}^{N} (\mathbf{y}^j - \boldsymbol{\mu}_p)(\mathbf{y}^j - \boldsymbol{\mu}_p)^T \tag{77}$$

It can be seen that

$$\mathbf{S}_{w,p} = \mathbf{W}^T \mathbf{S}_w \mathbf{W} \tag{78}$$

and

$$\mathbf{S}_{b,p} = \mathbf{W}^T \mathbf{S}_b \mathbf{W} \tag{79}$$

For multiple classes and multifeature patterns, the following scalar between-class separability criterion may be defined:

$$J(\mathbf{W}) = \frac{|\mathbf{S}_{b,p}|}{|\mathbf{S}_{w,p}|} = \frac{|\mathbf{W}^T \mathbf{S}_b \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_w \mathbf{W}|} \tag{80}$$

Here, $|\mathbf{S}_{b,p}|$ denotes a scalar representation of the between-class scatter matrix, and similarly, $|\mathbf{S}_{w,p}|$ denotes a scalar representation of the within-class scatter matrix for projected patterns.

An optimal transformation matrix $\hat{\mathbf{W}}$ can be found as a solution to the general eigenvalue problem

$$\mathbf{S}_b \mathbf{w}_i^T = \lambda_i \mathbf{S}_w \mathbf{w}_i^T, \quad (i = 1, 2, \cdots, n) \tag{81}$$

Multiplying both sides by $\mathbf{S}_w^{-1}$ gives

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{w}_i^T = \lambda_i \mathbf{w}_i^T, \quad (i = 1, 2, \cdots, n) \tag{82}$$

The solution of this problem is based on computing, and rearranging in decreased order, the eigenvalues and corresponding eigenvectors for the matrix $\mathbf{S}_w^{-1}\mathbf{S}_b$. Let us assume that the eigenvalues of the matrix $\mathbf{S}_w^{-1}\mathbf{S}_b$ are arranged in decreasing order $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n \geq 0$ (with $\lambda_1 = \lambda_{\max}$). Consequently, assume that the corresponding eigenvectors $\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n$ compose the $n \times n$ matrix

$$\mathbf{E} = [\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n] \tag{83}$$

with columns being eigenvectors. Then the optimal linear transformation matrix $\hat{\mathbf{W}}$ is composed with rows being the first m columns of matrix $\mathbf{E}$:

$$\hat{\mathbf{W}} = \mathbf{E}^T = \begin{bmatrix} (\mathbf{e}^1)^T \\ (\mathbf{e}^2)^T \\ \vdots \\ (\mathbf{e}^m)^T \end{bmatrix} \tag{84}$$

Thus, an optimal Fisher transformation matrix $\hat{\mathbf{W}}$ (denoted also by $\mathbf{W}_F$) is composed with m rows being the first m eigenvectors of the matrix $\mathbf{S}_w^{-1}\mathbf{S}_b$.

The linear Fisher transformation faces problems when the within-class scatter matrix S_w becomes degenerate (noninvertible). This can happen when the number of cases (objects) is smaller than the dimension of a pattern.

2.3. Sequence of PCA and Fisher's Linear Discriminant Projection

The PCA, with its resulting linear Karhunen-Loéve projection, provides feature extraction and reduction that are optimal from the point of view of minimizing the reconstruction error. However, PCA does not guarantee that selecting (reduced) principal components as a feature vector will be adequate for classification (will have discriminatory power). Nevertheless, the projection of high-dimensional patterns into lower-dimensional orthogonal principal-component feature vectors might ensure better classification for some data types.

On the other hand, the linear Fisher transformation is designed optimally from the point of view of maximal interclass separability of projected patterns. This transformation is a linear projection of the original patterns into a reduced **discriminative** feature space (suitable for classification).

However, we recall that the linear Fisher transformation faces problems when the within-class scatter matrix S_w becomes degenerate (noninvertible). This can happen when the number of cases (objects) in a data set is smaller than the pattern dimension. One solution to this problem is first to transform a data set with high-dimensional patterns into a lower dimensional feature space, for example by using a KLT transform, and then to apply a discriminative linear Fisher projection to the lower-dimensional patterns. This approach may lead to feature vectors that are more suitable for classification (and with better discriminatory power than the original patterns).

The entire projection procedure of the original high dimensional (n-dimensional) patterns $\mathbf{x}$ into the lower-dimensional (c-dimensional), more discriminative feature vectors can be decomposed into two subsequent linear transformations:

– PCA with the resulting Karhunen-Loéve projection into the m-dimensional principal component feature vectors $\mathbf{y}$
– A linear Fisher projection of m-dimensional principal component vectors $\mathbf{y}$ into c-dimensional Fisher discriminative feature vectors $\mathbf{z}$

Let us consider a given limited size data set T containing N cases labeled by associated classes. Each case $(\mathbf{x}^i, c^i_{\text{target}})(i = 1, 2, \cdots, N)$ is composed of an n-dimensional real-valued pattern $\mathbf{x} \in \mathbb{R}^n$ and a corresponding target class c^i_{target}. The total number of classes is l.

The selection of the projection dimensions m and c can be done in the following way. One can choose m such that for a number N of cases in a data set we have $m + l \leq N$. We know that it is impossible for m to be greater than $N - 1$ (since there is a maximum of $N - 1$ nonzero eigenvalues in the Karhunen-Loéve projection). We further constrain the final dimension of the reduced PCA feature vector to be less than the rank of the within-class scatter matrix $\mathbf{S}_w$ in order to make $\mathbf{S}_w$ nondegenerate (invertible). However, m cannot be smaller than the number of classes l. Since there are at most $l - 1$ nonzero eigenvalues of the matrix $\mathbf{S}_w^{-1}\mathbf{S}_b$ (where $\mathbf{S}_b$ is the between-class scatter matrix), one can choose $c \leq l - 1$ as a resulting dimension in the discriminative Fisher feature space. This will be the final dimension of a sequence of two projections. The relations between the dimensions of each projection are $c + 1 \leq l \leq m \leq N - l$.

The algorithm for the Karhunen-Loéve-Fisher transformations is as follows.
Given: A data set T, containing N cases $(\mathbf{x}^i, c^i_{\text{target}})$ labeled by associated categorical classes (with a total of l classes)

1. Extract from data set T only the pattern portion represented by the $n \times N$ matrix $\mathbf{X}$ with n-dimensional patterns $\mathbf{x}$ as rows.
2. Select a dimension m for the feature vectors containing principal components (already projected by the Karhunen-Loéve transformation), satisfying the inequalities $l \leq m \leq N - l$.
3. Compute, for the data in matrix $\mathbf{X}$, the $m \times n$-dimensional optimal linear Karhunen-Loéve transformation matrix $\mathbf{W}_{KL}$.
4. Transform (project) each original pattern $\mathbf{x}$ onto a reduced-size m-dimensional pattern $\mathbf{y}$ by using the formula $\mathbf{y} = \mathbf{W}_{KL}\mathbf{x}$ (or, for all projected patterns, by using the formula $\mathbf{Y} = \mathbf{X}\mathbf{W}_{KL}^T$).
5. Select a dimension c for the final reduced feature vectors $\mathbf{z}$ (projected by Fisher's transformation), satisfying the inequality $c + 1 \leq l$.
6. Compute, for the projected data in matrix $\mathbf{Y}$, the $c \times m$-dimensional optimal linear Fisher transformation matrix $\mathbf{W}_F$.
7. Transform (project) each projected pattern $\mathbf{y}$ from $\mathbf{Y}$ into the reduced-size c-dimensional pattern $\mathbf{z}$ by using the formula $\mathbf{z} = \mathbf{W}_F\mathbf{y}$ (or, for all projected patterns, by the formula $\mathbf{Z} = \mathbf{Y}\mathbf{W}_F^T$).

Result: The $m \times n$-dimensional optimal linear Karhunen-Loéve transformation matrix is $\mathbf{W}_{KL}$. The $c \times m$ dimensional optimal linear Fisher's transformation matrix is $\mathbf{W}_F$. The projected pattern matrix is $\mathbf{Z}$.

2.4. Singular Value Decomposition as a Method of Feature Extraction

One of the most important matrix decompositions is the **Singular Value Decomposition** (SVD). SVD can be used both as a powerful method of extracting features from images and as a method of image compression. We know from linear algebra theory that any symmetric matrix can be transformed into a diagonal matrix by means of orthogonal transformation. Similarly, any rectangular $n \times m$ real image represented by an $n \times m$ matrix $\mathbf{A}$, where $m \leq n$, can be transformed into a diagonal matrix by singular value decomposition. SVD decomposes a rectangular matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ into two orthogonal matrices $\boldsymbol{\Psi}_r$ of dimension $n \times n$ and $\boldsymbol{\Phi}_r$ of dimension $m \times m$, a pseudodiagonal matrix $\boldsymbol{\Lambda}$ of dimension $r \times r$, and a pseudodiagonal matrix containing singular values of the transposed matrix. Here, $\boldsymbol{\Lambda} = \mathrm{diag}\{\sigma_1, \sigma_2, \ldots, \sigma_p\}$, where $p = \min(m, n)$. The real nonnegative numbers $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$ are called the singular values of matrix $\mathbf{A}$.

The following equalities hold:

$$\boldsymbol{\psi}_r^T \boldsymbol{\psi}_r = \boldsymbol{\psi}_r \boldsymbol{\psi}_r^T = \mathbf{I} \quad \text{and} \quad \boldsymbol{\phi}_r^T \boldsymbol{\phi}_r = \boldsymbol{\phi}_r \boldsymbol{\phi}_r^T = \mathbf{I} \tag{85}$$

Now, assume that the rank of matrix $\mathbf{A}$ is $r \leq m$. The matrices $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$ are nonnegative and symmetric and have identical eigenvalues λ_i. For $m \leq n$, there are at most $r \leq m$ nonzero eigenvalues. The SVD transform decomposes matrix $\mathbf{A}$ into the product of two orthogonal matrices $\boldsymbol{\psi}$ of dimension $n \times r$, and $\boldsymbol{\phi}$ of dimension $m \times r$ and a diagonal matrix $\boldsymbol{\Lambda}^{\mathbf{1/2}}$ of dimension $r \times r$. The SVD of a matrix $\mathbf{A}$ (for example, representing an image) is given by

$$\mathbf{A} = \boldsymbol{\psi} \boldsymbol{\Lambda}^{1/2} \boldsymbol{\phi}^T = \sum_{i=1}^{r} \sqrt{\lambda_i} \psi_i \phi_i^T. \tag{86}$$

where the matrices $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ have r orthogonal columns $\psi_i \in \mathbb{R}^n$, $\phi_i \in \mathbb{R}^m$ $(i = 1, \cdots, r)$, respectively (representing the orthogonal eigenvectors of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$, respectively). The square matrix $\boldsymbol{\Lambda}^{1/2}$ has diagonal entries defined by

$$\boldsymbol{\Lambda}^{1/2} = \mathrm{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \cdots, \sqrt{\lambda_r}) \tag{87}$$

where $\sigma_i = \sqrt{\lambda_i} (i = 1, 2, \cdots, r)$ are the singular values of the matrix $\mathbf{A}$. Each λ_i, $(i = 1, 2, \cdots, r)$ is the nonzero eigenvalue of $\mathbf{A}\mathbf{A}^T$ (and of $\mathbf{A}^T\mathbf{A}$). Since the columns of $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ are the eigenvectors of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$, respectively, the following equations must therefore be satisfied:

$$(\mathbf{A}\mathbf{A}^T - \lambda_i \mathbf{I}_{n \times n}) \psi_i = \mathbf{0}, \quad i = 1, 2, \cdots, n \tag{88}$$

$$(\mathbf{A}^T\mathbf{A} - \lambda_i \mathbf{I}_{m \times m}) \phi_i = \mathbf{0}, \quad i = 1, 2, \cdots, m \tag{89}$$

Note that in both cases, we assume $\lambda_i = 0$ if $i > r$. We also include additional ψ_i such that $\mathbf{A}^T \psi_i = 0$, for $i = r+1, \cdots, n$, and additional ϕ_i such that $\mathbf{A}\phi_i = 0$, for $i = r+1, \cdots, m$. The above equations are also defined for $i = 1, 2, \cdots, r$. In this case the SVD transform is unitary, with unitary matrices $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$.

Having decomposed matrix $\mathbf{A}$ (an image) as $\mathbf{A} = \boldsymbol{\psi} \boldsymbol{\Lambda}^{1/2} \boldsymbol{\phi}^T$, and since $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ have orthogonal columns, the **singular value decomposition transform** (SVD transform) of the image $\mathbf{A}$ is defined as

$$\boldsymbol{\Lambda}^{1/2} = \boldsymbol{\psi}^T \mathbf{A} \boldsymbol{\phi} \tag{90}$$

The r singular values $\sqrt{\lambda_i} (i = 1, 2, \cdots, r)$ from the main diagonal of the matrix $\boldsymbol{\Lambda}^{1/2}$ represent in condensed form the matrix $\mathbf{A}$. If the matrix $\mathbf{A}$ represents an $n \times m$ image, then r singular values can be considered as extracted features from an image.

Unlike the PCA, the SVD is purely a matrix processing technique and not a statistical technique. However, SVD may relate to statistics if we consider processing a matrix related to statistical observations. If the singular values λ_i are arranged in decreasing order of magnitude, i.e., $\lambda_1 > \lambda_2 > \cdots > \lambda_r$, the error in approximation of the image is minimized in the least squares error sense. The nonzero diagonal singular values are unique for a given image. They can be used as features for textural image modeling, compression, and possibly classification purposes. The SVD features have many excellent characteristics, such as stability and rotational and translation invariances.

The SVD transform provides a means of extraction of the most expressive features for minimization of reconstruction error. Let us assume that, for the $n \times m (m \leq n)$ image represented by the matrix $\mathbf{A}$ of rank $r \leq m$, we find the SVD decomposition $\mathbf{A} = \boldsymbol{\psi} \boldsymbol{\Lambda}^{1/2} \boldsymbol{\phi}^T$, corresponding to r eigenvalues λ_i of $\mathbf{A}\mathbf{A}^T$. Now, assume that we will represent the original image $\mathbf{A}$ by the reduced number $k \leq r$ of features $\sqrt{\lambda_i} (i = 1, 2, \cdots, k)$. Assume that eigenvalues $\lambda_i (i = 1, 2, \cdots, r)$ are arranged in decreasing order. The reconstructed $n \times m$ image $\mathbf{A}_k$ based on k eigenvalues of $\mathbf{A}\mathbf{A}^T$ can be obtained by

$$\mathbf{A}_k = \sum_{i=1}^{k} \sqrt{\lambda_i} \psi_i \phi_i^T, \quad k \leq r \tag{91}$$

The reconstruction matrix $\mathbf{A}_k$ of rank $k \leq r$ is the best approximation, in the least squares error sense, of the original matrix $\mathbf{A}$. The reconstruction least squares error is computed by

$$e_k^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij} - a_{k,ij}|^2 \tag{92}$$

or equivalently by

$$e_k^2 = \sum_{i=k+1}^{r} \lambda_i \tag{93}$$

We see that the least squares reconstruction error is equal to the sum of $r - k$ trailing eigenvalues of $\mathbf{A}\mathbf{A}^T$.

Despite the expressive power of the SVD transform image features, it is difficult to say arbitrarily how powerful the SVD features could be for the classification of images.

2.5. Independent Component Analysis

Independent component analysis (ICA) is a computational and statistical method for discovering intrinsic independent factors in the data (sets of random variables, measurements, or signals). ICA is an unsupervised data processing method that exploits higher-order statistical dependencies among data. It discovers a generative model for the observed multidimensional data (given as a database). In the ICA model, the observed data variables x_i are assumed to be linear mixtures of some unknown independent sources s_i (latent variables, intrinsic variables) $x_i = h_1 s_1 + \cdots + h_m s_m$. A mixing system is assumed to be unknown.

Independent variables are assumed to be nongaussian and mutually statistically independent. Latent variables are called the independent components or sources of the observed data. ICA tries to estimate unknown **mixing matrix** and **independent components** representing processed data.

2.5.1. The Cocktail-party Problem

One interesting example of independent component analysis is the blind source separation problem known as "the cocktail party." Let us consider a room where two people are speaking simultaneously. Two microphones in different locations record speech signals $x_1(t)$ and $x_2(t)$. Each recorded signal is a weighted sum of speech signals generated by the two speakers $s_1(t)$ and $s_2(t)$. The mixing of speech signals can be expressed as a linear equation:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t) \tag{94}$$

where a_{11}, a_{12}, a_{21}, and s_{22} are parameters. The estimation of two original speech signals $s_1(t)$ and $s_2(t)$ using only the recorded signals $x_1(t)$ and $x_2(t)$ is called the **cocktail-party problem**. For known parameters a_{ij}, one could solve the linear mixing equation using classical linear algebra methods. However, the cocktail-party problem is more difficult, since the parameters a_{ij} are unknown. Independent component analysis can be used to estimate the mixing parameters a_{ij} based on information about their independence. This, in turn, allows us to separate the two original source signals $s_1(t)$ and $s_2(t)$ from their recorded mixtures $x_1(t)$ and $x_2(t)$.

Figure 7.2 shows an example of two original source signals and their linear mixtures.

2.5.2. ICA: Data and Model

ICA can be considered as an extension of Principal Component Analysis (PCA). We know that PCA finds the linear transformation of data patterns such that transformed patterns will have uncorrelated elements. The transformation matrix is formed with the ordered **eigenvectors** (corresponding to ordered **eigenvalues** in descending order) of the patterns covariance matrix. Discovered uncorrelated orthogonal principal components are optimal in the sense of minimizing mean-squares reconstruction error, and they show the directions in which data change the most. PCA decorrelates patterns but does not assure that uncorrelated patterns will be statistically independent.

ICA provides data representation (through the linear transformation) based on discovered statistically independent **latent variables** (independent components). The observed data signal can

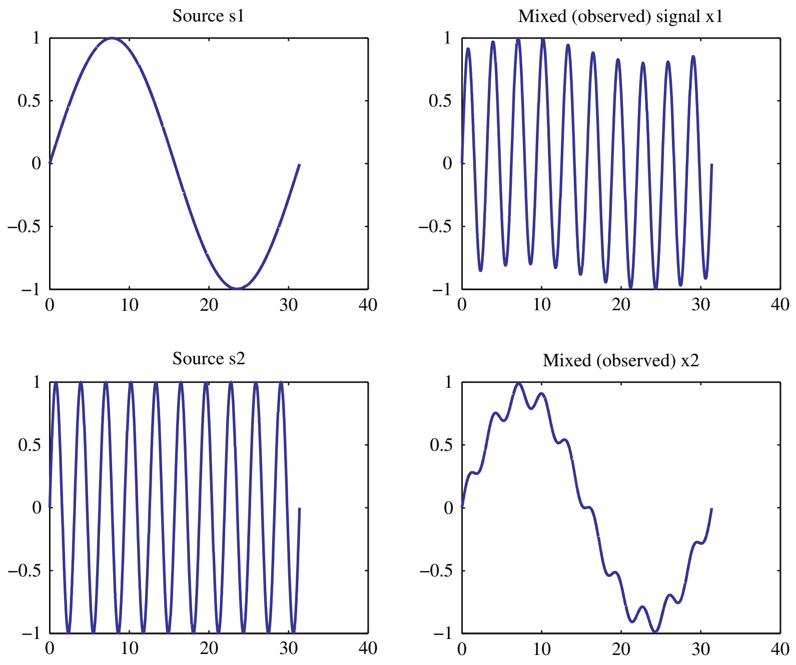

Figure 7.2. Mixing and unmixing.

be expressed as a linear mixture of statistically independent components (with highly nongaussian distributions). ICA is optimal in the sense of maximal statistical independence of sources.

The ICA model assumes that the n observed sensory signals x_i are given as the n-dimensional column pattern vectors $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T \in \mathbb{R}^n$. ICA is applied based on a sample from the given domain of observed patterns. This sample is given as a set T of N pattern vectors $T = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, which can be represented as an $n \times N$ **data matrix X** of measured N data patterns $\mathbf{x}_i (i = 1, \cdots, N)$. Columns of matrix $\mathbf{X}$ are composed with patterns $\mathbf{x}_i$. Generally, data pattern elements are considered to be random variables.

The **ICA model** for the x_i is given as linear mixtures of m source-independent variables s_j,

$$x_i = \sum_{j=1}^{m} h_{i,j} s_j = h_{i1}s_1 + h_{i2}s_2 + \cdots + h_{im}s_m, \quad i = 1, 2, \cdots, n \tag{95}$$

where x_i is the observed variable, s_j is the j^{th} independent component (source signal), and $h_{i,j}$ are mixing coefficients. The source variable constitutes the m-dimensional column source vector (source pattern, independent component pattern) $\mathbf{s} = [s_1, s_2, \cdots, s_m]^T \in \mathbb{R}^m$. Without loss of generality, we can assume that both the observable variables and the independent components have zero mean. The observed variables x_i can always be centered by subtracting the sample mean.

The ICA model can be presented in the matrix form

$$\mathbf{x} = \mathbf{H}\mathbf{s} \tag{96}$$

where $\mathbf{H} \in \mathbb{R}^{n \times m}$ is the $n \times m$ unknown mixing matrix whose row vector $[h_{i,1}, h_{i,2}, \cdots, h_{i,m}]$ represents the mixing coefficients for observed signal x_i. We can also write

$$\mathbf{x} = \sum_{i=1}^{m} \mathbf{h}_i s_i \tag{97}$$

where $\mathbf{h}_i$ denotes the i^{th} column of the mixing matrix $\mathbf{H}$.

In statistical independent component analysis, a data model is a generative type of model. This model expresses how the observed data are generated by a process of mixing of the independent components s_i. The independent components (sources) are latent variables that are not directly observable. The mixing matrix $\mathbf{H}$ is also assumed to be unknown.

The task of ICA is to estimate both the mixing matrix $\mathbf{H}$ and the sources $\mathbf{s}$ (independent components) based on the set of observed pattern vectors $\mathbf{x}$ (see Figure 7.3).

The ICA model can be discovered based on the $n \times N$ data matrix $\mathbf{X}$. The ICA model for the set of patterns from matrix $\mathbf{X}$ can be written as

$$\mathbf{X} = \mathbf{H}\ \mathbf{S} \tag{98}$$

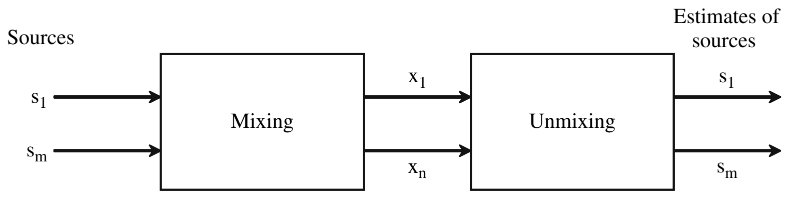

Figure 7.3. Mixing and unmixing.

where $\mathbf{S}$ is the $m \times N$ matrix whose columns contain m-dimensional independent component vectors $\mathbf{s}_i = [s_{i,1}, s_{i,2}, \cdots, s_{i,m}]^T$ discovered from the observation vectors. We can also write

$$\mathbf{X}^T = \mathbf{S}^T \ \mathbf{H}^T \tag{99}$$

ICA seems to be an underdetermined system. However, despite some constraints and ambiguities, one can discover independent components based on the principles of statistical independence of sources.

The following ambiguities are associated with ICA:

– Energies (variances) of independent components cannot be discovered because a scalar multiplier in s_i results in the same vector $\mathbf{x}$ as a scaling of the i^{th} column in matrix $\mathbf{H}$.
– It is not possible to determine the order of independent components because exchanging two sources results in the same $\mathbf{x}$ as swapping the corresponding columns in $\mathbf{H}$.

These ambiguities arise from the fact that both $\mathbf{s}$ and $\mathbf{H}$ are unknown. The ICA assumes that the components s_i are statistically independent and have nongaussian distributions. Once the $n \times m$ mixing matrix $\mathbf{H}$ has been estimated, we can compute its $m \times n$ inverse matrix (demixing, **separation matrix**) $\mathbf{B} = \mathbf{H}^{-1}$ (for $m = n$), or pseudo-inverse $\mathbf{B} = \mathbf{H}^{+}$ (for $m \leq n$), then the independent component vector for the observation vector $\mathbf{x}$ can be computed by

$$\mathbf{s} = \mathbf{B}\mathbf{x} \tag{100}$$

The inverse ICA model for the set of patterns T (matrix $\mathbf{X}$) can be written as

$$\mathbf{S} = \mathbf{B} \ \mathbf{X} \tag{101}$$

where $\mathbf{S}$ is the $m \times N$ matrix in which columns contain m-dimensional independent component vectors discovered from the observation vectors. We can also write

$$\mathbf{S}^T = \mathbf{X}^T \ \mathbf{B}^T \tag{102}$$

The extracted independent components s_i are as independent as possible, but can be evaluated by an information-theoretic cost criterion such as minimum Kulback-Leibler divergence.

ICA closely relates to the technique of **blind source separation** (BSS) or blind signal separation. In BSS, source signals (i.e., unknown independent components) are extracted from the observed patterns with very little knowledge about the mixing matrix or the statistics of the source signals.

2.5.3. Preprocessing

ICA can be preceded by necessary preprocessing, including centering and whitening, in order to simplify operations and make them better conditioned.

Centering of $\mathbf{x}$ is the process of subtracting its mean vector $\boldsymbol{\mu} = E\{\mathbf{x}\}$:

$$\mathbf{x} = \mathbf{x} - E\{\mathbf{x}\} \tag{103}$$

This process makes $\mathbf{x}$ (and consequently also $\mathbf{s}$) a zero-mean variable.

Once the mixing matrix $\mathbf{H}$ has been estimated for the centered data, then we can compute the final estimation by adding the mean vector of $\mathbf{s}$ back to the centered estimates of $\mathbf{s}$. The mean vector of $\mathbf{s}$ is given by $\mathbf{H}^{-1}\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is the mean that has been computed in preprocessing.

The second frequent preprocessing step in ICA (provided for centered signals) is **whitening**. The whitening is a linear transformation of measured patterns $\mathbf{x}$ that produces decorrelated white patterns $\mathbf{y}$, possibly with reduced dimensionality compared with n. Speaking more formally, whitening of the sensor signal vector $\mathbf{x}$ is a linear transformation

$$\mathbf{y} = \mathbf{W}\mathbf{x} \quad \text{so} \quad E\{\mathbf{y}\mathbf{y}^T\} = \mathbf{I}_l \tag{104}$$

where $\mathbf{y} \in \mathbb{R}^l$ is the l-dimensional ($l \leq n$) whitened vector, $\mathbf{W}$ is the $l \times n$ **whitening matrix** and $\mathbf{I}_l$ is the $l \times l$ identity matrix.

First, let us assume that $l = n$ (the dimension of whitened signal equals dimension of the original pattern $\mathbf{x}$).

Generally, the observed sensor signals x_i are mutually correlated, and their covariance matrices $\mathbf{R}_{\mathbf{xx}} = E\{\mathbf{x}\mathbf{x}^T\}$ are of full rank (not diagonal). The purpose of whitening is to transform the observed vector $\mathbf{x}$ linearly to a new vector $\mathbf{y}$ (which is white) whose elements are uncorrelated and whose variances equal unity. This mean that the covariance matrix of $\mathbf{y}$ is equal to the identity matrix:

$$\mathbf{R}_{yy} = E\{\mathbf{y}\mathbf{y}^T\} = E\{\mathbf{W}\mathbf{x}\mathbf{x}^T\mathbf{W}^T\} = \mathbf{W}\mathbf{R}_{\mathbf{x}\,\mathbf{x}}\mathbf{W}^T = \mathbf{I}_l \tag{105}$$

This transformation (which is always possible) is called **sphering**. We can notice that the matrix $\mathbf{W} \in \mathbb{R}^{l\times n}$ is not unique. Whitening also allows dimensionality reduction because it considers only the $l \leq n$ largest eigenvalues and the corresponding l eigenvectors of the covariance matrix of $\mathbf{x}$. Since the covariance matrix of observed vectors $\mathbf{x}$ is usually symmetric positive definite, whitening can therefore be realized using, for example, the eigenvalue decomposition of the covariance matrix $E\{\mathbf{x}\mathbf{x}^T\} = \mathbf{R}_{\mathbf{xx}} \in \mathbb{R}^{n\times n}$ of the observed vector $\mathbf{x}$:

$$\mathbf{R}_{\mathbf{xx}} = E\{\mathbf{x}\mathbf{x}^T\} = \mathbf{E}_{\mathbf{x}}\mathbf{\Lambda}_{\mathbf{x}}\mathbf{E}_{\mathbf{x}}^T \tag{106}$$

Here, $\mathbf{E}_{\mathbf{x}} \in \mathbb{R}^{n\times n}$ is the orthogonal matrix of eigenvectors of $\mathbf{R}_{\mathbf{xx}} = E\{\mathbf{x}\mathbf{x}^T\}$ and $\mathbf{\Lambda}$ is the $n \times n$ diagonal matrix of the corresponding eigenvalues

$$\mathbf{\Lambda}_{\mathbf{x}} = \mathrm{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n) \tag{107}$$

Note that $E\{\mathbf{x}\mathbf{x}^T\}$ can be estimated from the available sample $\mathbf{x}_1, \cdots, \mathbf{x}_N$ of the observed vector $\mathbf{x}$. For $l = n$, we consider all $l = n$ eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ (and all $l = n$ corresponding eigenvectors of the covariance matrix $\mathbf{R}_{\mathbf{xx}}$).

The whitening matrix can be computed as

$$\mathbf{W} = \mathbf{E}_{\mathbf{x}}\mathbf{\Lambda}_{\mathbf{x}}^{-1/2}\mathbf{E}_{\mathbf{x}}^T \tag{108}$$

Thus, the whitening operation can be realized using the formula

$$\mathbf{y} = \mathbf{E}_{\mathbf{x}}\mathbf{\Lambda}_{\mathbf{x}}^{-1/2}\mathbf{E}_{\mathbf{x}}^T\ \mathbf{x} = \mathbf{W}\mathbf{x} \tag{109}$$

here the matrix $\mathbf{\Lambda}^{-1/2}$ is computed as $\mathbf{\Lambda}^{-1/2} = \mathrm{diag}(\lambda_1^{-1/2}, \cdots, \lambda_n^{-1/2})$. One can now find that for whitened vectors we have $E\{\mathbf{y}\mathbf{y}^T\} = \mathbf{I}_l$.

Recalling that $\mathbf{y} = \mathbf{W}\mathbf{x}$ and $\mathbf{x} = \mathbf{H}\ \mathbf{s}$, we can find from the above equation that

$$\mathbf{y} = \mathbf{E}_{\mathbf{x}}\mathbf{\Lambda}_{\mathbf{x}}^{-1/2}\mathbf{E}_{\mathbf{x}}^T\ \mathbf{H}\ \mathbf{s} = \mathbf{H}_w\mathbf{s} \tag{110}$$

We can see that whitening transforms the original mixing matrix $\mathbf{H}$ into a new one, $\mathbf{H}_w$:

$$\mathbf{H}_w = \mathbf{E}_{\mathbf{x}}\mathbf{\Lambda}_{\mathbf{x}}^{-1/2}\mathbf{E}_{\mathbf{x}}\mathbf{H} \tag{111}$$

An important feature of whitening is producing the new mixing matrix $\mathbf{H}_w$, which is orthogonal ($\mathbf{H}_w^{-1} = \mathbf{H}_w^T$):

$$E\{\mathbf{y}\mathbf{y}^T\} = \mathbf{H}_w E\{\mathbf{s}\mathbf{s}^T\}\mathbf{H}_w^T = \mathbf{H}_w \mathbf{H}_w^T = \mathbf{I}_l \tag{112}$$

If during the whitening process $l = n$ we consider all $l = n$ eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ (and all $l = n$ corresponding eigenvectors of the covariance matrix $\mathbf{R}_{\mathbf{xx}}$), the dimension of the matrix $\mathbf{W}$ is $n \times n$, and there is no reduction of the size of the observed transformed vector $\mathbf{y}$. For the fully dimensional matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$, whitening provides only decorrelation of observed vectors and orthogonalization of the mixing matrix. We can see that whitening for $l = n$ can reduce the number of parameters to be estimated by ICA processing. Instead of having to estimate the n^2 parameters that are the elements of the original mixing matrix $\mathbf{H}$, we only need to estimate the new, orthogonal mixing matrix $\mathbf{H}_w$. In this orthogonal matrix, we have to estimate only $n(n-1)/2$ parameters.

Whitening allows us to reduce the dimensionality of the whitened vector by considering only the $l(l \leq n)$ largest eigenvalues and the corresponding l eigenvectors of the covariance matrix $\mathbf{E}_{\mathbf{xx}} = \mathbf{R}_{\mathbf{xx}}$. This will result in obtaining a reduced whitening matrix $\mathbf{W}$ of dimension $l \times n$ and a reduced l-dimensional whitened vector $\mathbf{y} \in \mathbb{R}^l$. Consequently, the dimension of the new mixing matrix $\mathbf{H}_w$ will be reduced to $l \times n$. This is similar to the dimensionality reduction of patterns in PCA. Then the resulting dimension of the matrix $\mathbf{W}$ is $l \times n$, and there is a reduction of the size of the observed transformed vector $\mathbf{y}$ from n to l.

For the set of N original patterns arranged in matrix $\mathbf{X}$, whose columns are patterns $\mathbf{x}$, the whitening is given by

$$\mathbf{Y} = \mathbf{W}\ \mathbf{X} \tag{113}$$

where $\mathbf{Y}$ is the $N \times l$ matrix, whose columns are constituted by prewhitened vectors $\mathbf{y}$. The dewhitening operation is given by

$$\tilde{\mathbf{X}} = \mathbf{D}\mathbf{Y} \tag{114}$$

where the dewhitening matrix is defined as $\mathbf{D} = \mathbf{W}^{-1}$ for $l = n$ and as $\mathbf{D} = \mathbf{W}^{+}$ for $l = n$ (where $\mathbf{W}^{+}$ denotes the pseudoinverse of the rectangular matrix).

The output vector of the whitening process $\mathbf{y}$ can be considered as an input to the ICA algorithm: an input to the unmixing operation (separation, finding an independent components vector).

We can see that instead of estimating the mixing matrix $\mathbf{H}$, we can just estimate the orthogonal matrix $\mathbf{W}_w$, which simplifies the computation.

Whitening results in a new mixing matrix $\mathbf{H}_w$ (and in the ICA model $\mathbf{y} = \mathbf{H}_w\mathbf{s}$) being found, as well as a set of N whitened patterns $\mathbf{Y}$. Assume that the mixing matrix $\mathbf{H}_w$ and $\mathbf{Y}$ have been found. Recall that $\mathbf{y} = \mathbf{H}_w\mathbf{s}$; therefore, the unmixing operation for one vector $\mathbf{y}$ is given by

$$\mathbf{s} = \mathbf{B}_w\mathbf{y} \tag{115}$$

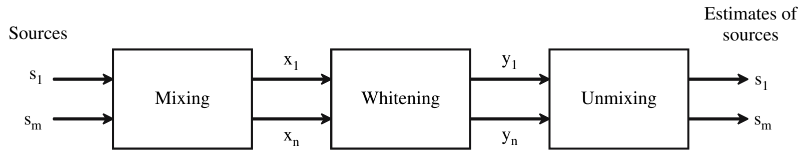

Figure 7.4. Whitening and unmixing.

where the demixing matrix is computed as inverse $\mathbf{B}_w = \mathbf{H}_w^{-1}$ (for $l = n$) or as pseudo-inverse $\mathbf{B} = \mathbf{H}_w^{+}$ (for $l \leq n$) of $\mathbf{H}_w$.

For a set of N whitened patterns $\mathbf{Y}$, we can compute a set of N corresponding independent component vectors arranged as the $m \times N$ matrix $\mathbf{S}$ whose columns are constituted by m-dimensional independent component vectors $\mathbf{s}_i = [s_{i,1}, s_{i,2}, \cdots, s_{i,m}]^T$:

$$\mathbf{S} = \mathbf{B}_w\mathbf{Y} \tag{116}$$

Since $\mathbf{Y} = \mathbf{WX}$, therefore we can also write

$$\mathbf{S} = \mathbf{B}_w\mathbf{WX} \tag{117}$$

Discovered independent components $\mathbf{s}$ are only estimates. Thus, inverse operations will also be just estimates. Once the mixing matrix $\mathbf{H}_w$ has been estimated, one can estimate the original mixing matrix $\mathbf{H}$, denoted by $\mathbf{H}_{\text{esty}}$, by applying the inverse (for $l = n$) (or pseudo-inverse) of the whitening operation on the new mixing matrix ($\mathbf{H}_w = \mathbf{W}\ \mathbf{H}$):

$$\mathbf{H}_{\text{esty}} = \mathbf{W}^{-1}\mathbf{H}_w \tag{118}$$

An approximation (**reconstruction**) of the original observed vector $\mathbf{x}$ from a given $\mathbf{s}$ can be computed as

$$\tilde{\mathbf{x}} = \mathbf{H}_{\text{esty}}\mathbf{s} \tag{119}$$

For the set of N independent component vectors arranged as columns of the $m \times N$ matrix $\mathbf{S}$, we can provide an estimation of the ICA model:

$$\tilde{\mathbf{X}} = \mathbf{H}_{\text{esty}}\mathbf{S} \tag{120}$$

Whitening can be realized using the PCA transformation. Here, the eigenvalues of the pattern covariance matrix $\mathbf{R}_{\mathbf{xx}}$ are arranged in decreasing order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, and the corresponding eigenvectors, arranged in decreasing order, constitute rows of the whitening transformation matrix $\mathbf{W}$.

The dimensionality reduction of whitened patterns can be realized by considering only the first l eigenvectors in constructing the whitening matrix.

The ICA estimates of the **demixing** (separation) **matrix B** and **independent component vectors S** are based on a set of patterns $\mathbf{X}(\mathbf{S} = \mathbf{BX})$. Estimations can be realized using different criteria and algorithms. In a computationally efficient algorithm, the following maximization criterion has been used:

$$J(\tilde{\mathbf{s}}) = \sum_{i=1}^{m} \left| E\{\tilde{s}_i^4\} - 3[E\{\tilde{s}_i^2\}]^2 \right| \tag{121}$$

where m is number of independent and m-dimensional principal component vectors. The above equation corresponds to Fourth-order cumulant kurtosis. Based on the gradient operation of J, the independent component separation matrix guarantees that the independency characteristic will be preserved:

$$p(s_1, s_2, \cdots, s_m) = \Pi_{i=1}^{m} p_i[s_i] \tag{122}$$

where $p[]$ is the **probability density function**.

Let us consider two scalar-valued random variables y_1 and y_2. The variables y_1 and y_2 are **independent** if information about the value of y_1 does not provide any information about the value of y_2 (and vice versa). Practically, independence can be explained by probability densities. Let us denote by $p(y_1, y_2)$ the joint probability density function (pdf) of y_1 and y_2. Additionally, let us denote by $p_1(y_1)$ the pdf of y_1 when it is considered alone:

$$p_1(y_1) = \int p(y_1, y_2)dy_2 \tag{123}$$

and similarly for y_2. Now, one can state that y_1 and y_2 are independent if and only if the joint pdf can be expressed as

$$p(y_1, y_2) = p_1(y_1)p_2(y_2) \tag{124}$$

This definition can be generalized for n random variables (the joint probability density must be a product of n pdfs).

By virtue of that definition, for two functions, h_1 and h_2, we have

$$E\{h_1(y_1)h_2(y_2)\} = E\{h_1(y_1)\}E\{h_2(y_2)\} \tag{125}$$

We can write generally for n independent random variables that

$$p(y_1, y_2, \cdots, y_n) = p_1(y_1)p_2(y_2)\ldots p_n(y_n) \tag{126}$$

Consequently, we can write, for any functions f_i,

$$E\{f_1(y_1), \cdots, f_n(y_n)\} = E\{f_1(y_1)\}\cdots E\{f_n(y_n)\} \tag{127}$$

It is known that uncorrelated variables are only partly independent. A weaker definition of variable independence is **uncorrelatedness**. Two random variables $\mathbf{v}_1$ and v_2 are uncorrelated, if their covariance is zero:

$$E\{y_1 y_2\} - E\{y_1\}E\{y_2\} = 0 \tag{128}$$

Independent variables are uncorrelated (independency implies uncorrelatedness); however, uncorrelated variables do not have to be independent. The majority of ICA methods provide uncorrelated estimates of the independent components.

ICA assumes that independent components must be nongaussian. The basis for estimating the ICA model is this assumption about nongaussianity.

The most natural measure of nongaussianity is kurtosis or the fourth-order cumulant. The **kurtosis** of random variable y defined as

$$\text{kurt}(y) = E\{y^4\} - 3(E\{y^2\})^2 \tag{129}$$

Assuming that y has unit variance, the above equation simplifies to $\text{kurt}(y) = E\{y^4\} - 3$. We see that kurtosis is a normalized version of the fourth moment $E\{y^4\}$. Since for a Gaussian variable y, the fourth moment is $3(E\{y^2\})$, therefore the kurtosis value is zero for a Gaussian random variable. For most nongaussian random variables, kurtosis is nonzero. Random variables that have a negative kurtosis are called **subgaussian**, and those with positive kurtosis are called **supergaussian**.

Usually nongaussianity is measured by the absolute (or square) value of kurtosis. This measure is equal to zero for a Gaussian variable, and greater than zero for most nongaussian random variables.

Kurtosis has been frequently used as a measure of nongaussianity in ICA and related fields. In practical computations, kurtosis can be estimated by using the fourth moment of the sample data.

The linearity property helps in the simplification of kurtosis:

$$\begin{aligned}\text{kurt}(x_1 + x_2) &= \text{kurt}(x_1) + \text{kurt}(x_2)\\ \text{kurt}(\alpha x_1) &= \alpha^4 \text{kurt}(x_1)\end{aligned} \tag{130}$$

where x_1 and x_2 are two independent random variables, and α is a coefficient. Kurtosis values can be very sensitive to outliers, and thus they may be not robust measures of nongaussianity. As a remedy for this weakness, negentropy can be used for robust approximations of kurtosis.

A **negentropy** (which is based on the information-theoretic quantity of differential entropy) can be considered as a robust measure of nongaussianity. Thus, it can be used as a performance criterion in the optimal solution (estimation) of the ICA model.

We recall that entropy of a random variable can be considered as the degree of information given by observation of the variable. For more unpredicted, "random" (unstructured, disarrayed) variables, the entropy is larger. Formally, entropy is expressed as the coding length of the random variable. **Entropy** H_{entr} for a discrete random variable Y, with e_i as the possible values of Y, is defined as

$$E_{\text{entr}}(Y) = -\sum_i P(Y = e_i) \log P(Y = e_i) \tag{131}$$

For a continuous-valued random variable $\mathbf{y}$ with known probability density $f(\mathbf{y})$, the entropy (called **differential entropy**) H_{dentr} is defined as

$$H_{\text{dentr}}(\mathbf{y}) = -\int f(\mathbf{y}) \log f(\mathbf{y}) \ d\mathbf{y} \tag{132}$$

Information theory shows that a Gaussian variable has the largest entropy among all random variables of equal variance. This property predisposes entropy to be a good measure of nongaussianity. The Gaussian distribution is the "most random" (the least structured) among all distributions.

A good measure of nongaussianity is differential entropy, called **negentropy**:

$$J(\mathbf{y}) = H_{\text{entr}}(\mathbf{y}_{\text{gauss}}) - H_{\text{dentr}}(\mathbf{y}) \tag{133}$$

where $\mathbf{y}_{\text{gauss}}$ is a Gaussian random variable with the same covariance matrix as $\mathbf{y}$. Negentropy has nonnegative values, and it is zero for the Gaussian distribution of y. Furthermore, negentropy is invariant for invertible linear transformations. Computation of negentropy requires an estimation of the probability density of y, which makes the use of pure definition impractical.

For the random variable y (zero mean with unit variance), a computationally feasible approximation of negentropy using higher-order moments can be expressed as

$$J(y) \approx \frac{1}{12} E\{y^3\}^2 + \frac{1}{48} \text{kurt}(y)^2 \tag{134}$$

The applicability of this approximation may be limited due to the nonrobustness associated with kurtosis. Better approximations of negentropy are given by

$$J(y) \approx \sum_{i=1}^{r} a_i \Big[E\{G_i(y)\} - E\{G_i(\nu)\}\Big]^2 \tag{135}$$

where $G_i (i = 1, \cdots, r)$ are chosen r nonquadratic functions. Here a_i are positive constants and ν is a zero mean and unit variance Gaussian variable. Variable y is assumed to be of zero mean and unit variance. The above measure can be used to test nongaussianity. It is always nonnegative and is equal to zero if y has a Gaussian distribution.

If we use only one nonquadratic function G, the approximation results in

$$J(y) \approx \left[E\{G(y)\} - E\{G(\nu)\}\right]^2 \tag{136}$$

for practically any nonquadratic function G. For $G(y) = y^4$, one can obtain exactly a kurtosis-based approximation. The following choices of G are recommended:

$$G_1(u) = \frac{1}{a_1} \log(\cosh\ a_1 u), \quad G_2(u) = -\exp(-u^2/2) \tag{137}$$

where $1 \leq a_1 \leq 2$ is a constant.

Measures of nongaussianity can be used as performance criteria for ICA estimation.

Two fundamental applications of ICA are blind source separation and feature extraction.

In blind source separation, the observed values of $\mathbf{x}$ correspond to the realization of an m-dimensional discrete-time signal $\mathbf{x}(t)$, $(t = 1, 2, \cdots)$. The components $s_i(t)$ are called source signals, which are usually original, uncorrupted signals or noise sources. Frequently such sources are statistically independent from each other, and thus the signals can be recovered from linear mixtures x_i by finding a transformation in which the transformed signals are as independent as possible, as in ICA.

In feature extraction based on ICA, the $i^{\text{th}} s_i$ independent component represents the i^{th} feature in the observed data pattern vector $\mathbf{x}$.

2.5.4. Feature Extraction using ICA

In feature extraction, which is based on independent component analysis, one can consider an i^{th} independent component s_i as the i^{th} feature of the recognized object represented by the observed pattern $\mathbf{x}$. The feature pattern can be formed from m independent components of the observed data pattern. A procedure of pattern formation using ICA follows:

1. Extract n-element original feature patterns $\mathbf{x}$ from the recognized objects. Compose the original data set T containing N cases $\{\mathbf{x}_i^T, c_i\}$ that contain patterns and the corresponding class c_i. The original patterns are represented as the $n \times N$ pattern matrix $\mathbf{X}$ (composed with patterns as columns) and the corresponding categorical classes (represented as column $\mathbf{c}$).
2. Subtract a mean vector from each pattern.
3. Perform linear feature extraction by projection (and reduction) of the original patterns $\mathbf{x}$ from matrix $\mathbf{X}$ through ICA.

   (a) Whiten the data set $\mathbf{X}$, including dimensionality reduction. Obtain l-element decorrelated whitened patterns $\mathbf{x}_w$ gathered in an $l \times N$ matrix $\mathbf{X}_w$.
   (b) Estimate (for whitened patterns) the $m \times n$ unmixing matrix $\mathbf{B}$.
   (c) Estimate m independent components forming an m-element independent component pattern $\mathbf{x}_s$ for each pattern $\mathbf{x}_w$. This result can be realized through projections of patterns $\mathbf{x}_w$ into an m-element pattern $\mathbf{x}_s$ (in independent component space). This projection can be realized using the discovered unmixing matrix $\mathbf{B}$. The independent component patterns will have decorrelated and independent elements.
   (d) Form the final $m \times N$ pattern set $\mathbf{X}_s$ with N ICA patterns $\mathbf{x}_s$ as columns.

4. Construct the final class-labeled data set as the matrix $\mathbf{X}_{sf}$ which is formed by adding class column vector $\mathbf{c}$ to matrix $\mathbf{X}_s$.

ICA does not guarantee that the independent components selected first, as a feature vector, will be the most relevant for classification. In contrast to PCA, ICA does not provide an intrinsic order for the representation features of a recognized object (for example, an image). Thus, one cannot reduce an ICA pattern just by removing its trailing elements (which is possible for PCA patterns). Selecting features from independent components is possible through the application of **rough set** theory (see Chapter 5). Specifically, defined in rough sets, the computation of a reduct can be used to select some independent components-based features (attributes) constituting a reduct. These reduct-based independent component features will describe all concepts in a data set. The rough set method is used for finding reducts from discretized ICA patterns. The final pattern is formed from ICA patterns based on the selected reduct.

2.6. Vector Quantization and Compression

Data compression is particularly important for images, time series and speech, where the amount of data to be handled and stored is very large. When data sets contain a lot of redundant information, or when specific applications do not require high precision of data representation, then **lossy compression** might be a viable option among techniques of data storing or transmitting. Vector quantization is a powerful technique of lossy data representation (approximation) that is widely used in data compression and in classification.

Vector quantization(VQ) is a technique of representing (encoding) input data patterns by using a smaller finite set of **codevectors (template vectors, reference vectors)** that is a good approximation of the input pattern space. When an input data pattern is processed, it can be encoded (approximated) by the nearest codevector.

The objective of vector quantization for a given training data set T is to design (discover) the optimal **codebook**, containing a predetermined number of reference code vectors, which guarantee minimization of the chosen distortion metric for all encoded patterns from the data set. Each code vector in the codebook has an associated integer index used for referencing.

Once the optimal set of codevectors in the codebook has been computed, then it can be used for the encoding and decoding of input patterns (for example in data transmission). This process may result in substantial data compression.

Vector quantization belongs to **unsupervised learning** techniques. It uses an unlabeled training set in the design phase. The fundamental role played by unsupervised quantization involves **distortion measures**. Vector quantization approximates data and naturally will cause some information loss. One distortion metric is used to measure a distance (similarity) between an input pattern $\mathbf{x}$ and the codevector $\mathbf{w}$. It can also be used for the selection, for a given $\mathbf{x}$, of the nearest codevector $\mathbf{w}_j$ from a codebook. The other metric is used to determine the average distortion measure for all training patterns and a given codebook. This metric measures the quality of the entire codebook with respect to a given data set T, and it is instrumental in the design of an optimal ("best") codebook. The optimal codebook depends on the distortion metrics used. Examples of distortion metrics are discussed in the next sections.

The design process for vector quantization comprises the discovery of an optimal set of codevectors (a codebook) that best approximates a given training set of patterns. An optimal vector quantizer yields the minimum of the average distortion for a given data set. The processing of an input pattern using VQ relates to finding the reference vector that is closest to a pattern, according to the selected similarity metric.

2.6.1. Vector Quantization as a Clustering Process

Vector quantization, which is based on the optimal design of a codebook, can be seen as **clustering** (see Chapter 9). It corresponds to a partition of the input pattern space into encoding regions/clusters. Here, for each discovered cluster (encoding region), one can find the vector that is a representation of the cluster. For example, the mean vector of a cluster (a cluster **centroid**) can be selected as a representation of the entire cluster. Such a centroid vector can be considered as the codevector associated with a given cluster. This codevector can be added as one entry (codevector) in the codebook. A set of all indexed cluster centroids is treated as the codebook of codevectors. A codebook represents, in an approximate way, a given training set. Based on a given codebook, a training set of patterns, and selected distortion measures, one can find the optimal partition of the input pattern space into a number of encoding regions (clusters) with codevectors representing entire regions. For an optimal partition and an optimal codebook, the average distortion measure as an optimization criterion will be minimal. This means that a distortion measure between any input pattern **x** and its quantized representation as codevector **w** is minimal.

The procedure of deciding to which cluster an input pattern belongs can be realized by using selected distortion measures between a pattern and codevector. Here, a specific role is played by the Euclidean distance (or the squared Euclidean distance)

$$d(\mathbf{x},\mathbf{y}) = \| \mathbf{x}-\mathbf{y} \|_2 = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{138}$$

The optimal partition, which uses the Euclidean distance as a similarity measure (a distortion measure between **x** and **w**) in order to find the closest codevector to the input pattern, produces a special vector quantizer called the **Voronoi quantizer**. The Voronoi quantizer partitions the input pattern space into Voronoi cells (clusters, encoding regions). One Voronoi cell is represented by one corresponding codevector in a codebook. A cell contains all those input patterns **x** that are closer to the codevector $\mathbf{w}_j$ (in the sense of Euclidean distance) than to any other codevector. The data set patterns that fall inside a Voronoi cell will be assigned to one corresponding codevector.

This partition of the input pattern space is called **Voronoi tessellation**. The "rigid" boundaries between Voronoi cells are perpendicular bisector planes of lines that join codevectors in neighboring cells. Topologically, the Voronoi partition resembles a honeycomb.

A codebook contains all the codevectors representing a data set. The set of all encoding regions is called the partition of the input pattern space.

2.6.2. Vector Quantization – Design and Processing

Design of an optimal codebook is a minimization process. Let us consider a given data set T (training set) containing N n-dimensional continuous-valued input patterns $\mathbf{x} \in \mathbb{R}^n$. Vector quantization is the process of categorization of input patterns **x** into M distinct **encoding regions (cells, clusters)** $C_1, C_2, \cdots, C_M$. Each encoding region C_i is represented by one n-dimensional real-valued codevector $\mathbf{w}_i \in \mathbb{R}^n (i = 1, 2, \cdots, M)$. A set of codevectors will be denoted by $W = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_M\}$ or, in $M \times n$ matrix form, as

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_M^T \end{bmatrix} \tag{139}$$

The indexed set of M codevectors $\mathbf{w}_i$ $(i = 1, 2, \cdots, M)$ is called the **codebook** $W_c = \{(i, \mathbf{w}_i)\}_1^M$ of the quantization. Table 7.3 shows the form of the codebook.

The vector quantization for a given data set contains two steps:

1. Designing the optimal (i.e., having minimum average distortion) codebook $W_c = \{(i,\ \mathbf{w}_i)\}_{i=1}^M$, which is an indexed set of M reference codevectors representing M entirely distinct cells in the input pattern space. A codebook of codevectors is designed for a given training set T of n-dimensional patterns $\mathbf{x} \in \mathbb{R}^n$.

Table 7.3. The code book.

Index	Reference codevector w
1	$\mathbf{w}_1$
2	$\mathbf{w}_2$
$\vdots$	$\vdots$
M	$\mathbf{w}_M$

2. Processing input patterns: encoding/decoding.

Vector quantization can also be seen as a combination of two functions: vector **encoding** and vector **decoding**. During the encoding process, every input pattern vector $\mathbf{x}$ is compared with each of the M codevectors in a codebook, according to some distortion measure $d(\mathbf{x}, \mathbf{w}_i)$(i=1, 2, ..., M). The codevector $\mathbf{w}_j$ that yields the minimum distortion is selected as a quantized representation of the input pattern, and the index j associated with this codevector is generated for transmission or storage. In other words, the representative codevector for the pattern $\mathbf{x}$ (an encoding vector) is determined to be the closest codevector $\mathbf{w}_j$ in the sense of a given pattern distortion metric (for example, the Euclidean distance). This codevector is a unique representation of all input patterns from the encoding region represented by the considered codevector.

If indexed codevectors (a codebook) are known in advance, then instead of representing the pattern $\mathbf{x}$ by the whole corresponding codevector $\mathbf{w}_j$, one can provide only the the codevector index j (the encoding region number j into which the pattern falls). This encoding region number can then be stored or transmitted instead of the pattern (as the compressed representation of the pattern).

In the decoding phase, a decoder has a copy of the codebook and uses the received index to find the corresponding codeword. The decoder then outputs the codevector $\mathbf{w}_j$ as a substitute for the original vector $\mathbf{x}$. Figure 7.5 shows the basic quantization model.

In other words, determining an encoding region for a given input pattern $\mathbf{x}$ is provided by choosing the "winning" codebook vector $\mathbf{w}_j$, representing the j^{th} encoding region C_j, which has the smallest value of the distortion metric used to determine the distance of the input pattern $\mathbf{x}$ to the codevector $\mathbf{w}$:

$$\text{winning } j^{\text{th}} \text{ codevector}: \ \mathbf{w}_j \min_{i=1,2,\cdots,M} d(\mathbf{x}, \mathbf{w}_i) \tag{140}$$

The pattern $\mathbf{x}$ thus belongs to the same region C_j as the "winning" closest codebook vector $\mathbf{w}_j$.

One of the most important applications of competitive learning is data compression through vector quantization. The idea of applying competitive learning for vector quantization, which is based on a winner selection determined by the distortion (distance) measure, is quite natural and straightforward.

Vector quantization is understood as the process of dividing the original input pattern space into a number of distinct regions (Voronoi cells). To each region, a codevector $\mathbf{w}_i$ ($i = 1, 2, \cdots, M$))

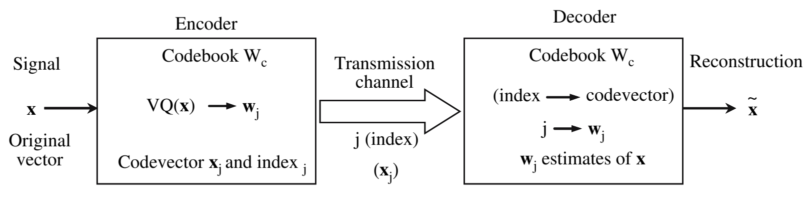

Figure 7.5. Encoding and decoding.

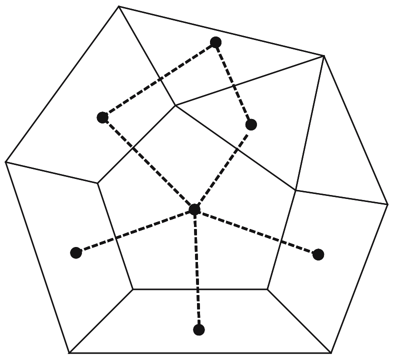

Figure 7.6. Voronoi cells and Voronoi tesselation.

is assigned. This codevector represents all the patterns from the entire region (cell), as indicated in Figure 7.6.

In other words, a vector quantization maps N n-dimensional patterns $\mathbf{x}$ from the input pattern space $\mathbb{R}^n$ into a finite set of M codevectors $\mathbf{w}_i \in \mathbb{R}^n$ belonging to a codebook. The nearest-neighbors region (**Voronoi cell**) is associated with each $\mathbf{w}_i$ codevector. A region is defined as

$$C_i = \{\mathbf{x} \in \mathbb{R}^n : d(\mathbf{x}, \mathbf{w}_i) \leq d(\mathbf{x}, \mathbf{w}_j), \quad \text{for all} \quad i \neq j\} \tag{141}$$

The set of Voronoi regions partitions the entire space $\mathbb{R}^n$ such that

$$\bigcup_i C_i = \mathbf{R}^n \quad \text{and} \quad C_i \cap C_j = \emptyset \quad \text{for} \quad i \neq j \tag{142}$$

Vector quantization is a technique that partitions the input pattern space into M distinct regions (cells), each with a codevector representing the entire region. Vector quantization maps any input pattern vector $\mathbf{x} \in \mathbb{R}^n$ from an n-dimensional input pattern space $\mathbb{R}^n$ into a finite set of M n-dimensional codevectors $\mathbf{w}_i \in \mathbb{R}^n$ $(i = 1, 2, \cdots, M)$:

$$VQ : \mathbf{x} \in \mathbb{R}^n \ \rightarrow W \tag{143}$$

where $W = \{\mathbf{w}_i\}_1^M$ is the set of M codevectors. The set of indexed codevectors $\mathbf{w}_c = \{i, \mathbf{w}_i\}_1^M$ constitutes a **codebook** of vector quantization.

Since the codebook has M entries, the quantity R, called the **rate** of the quantizer, is defined as

$$R = \log_2 \ M \tag{144}$$

measured in bits per input pattern vector. Since the input pattern has n components, it follows that

$$r = \frac{R}{n} \tag{145}$$

is the rate in bits per vector component.

In the topological interpretation, the knowledge of the mapping function VQ determines a partition of input patterns into M subsets (cells):

$$C_i = \{\mathbf{x} \in \mathbb{R}^n : VQ(x) = \mathbf{w}_i\}, \ (i = 1, \cdots, M) \tag{146}$$

A cell C_i in $\mathbb{R}^n$ is associated with each codevector $\mathbf{w}_i$ of the vector quantizer, that is, $C_i = \{\mathbf{x} \in \mathbb{R}^n : VQ(x) = \mathbf{w}_i\}$. It is sometime called the inverse image of a codevector $\mathbf{w}_i$ under mapping VQ and is denoted by $C_i = VQ^{-1}(\mathbf{w}_i)$.

From the definition of a cell, it follows that

$$\bigcup_i C_i = \mathbb{R}^n \text{ and } C_i \cap C_j = \emptyset \text{ for } i \neq j \tag{147}$$

which indicates that the cells form a partition of $\mathbb{R}^n$.

When the input pattern $\mathbf{x}$ is quantized and represented as a codevector $\mathbf{w}$ ($\mathbf{w} = VQ(\mathbf{x})$), a quantization error results and a distortion measure can be defined between $\mathbf{x}$ and $\mathbf{w}$. As with the measurement of a distance between two vectors (see Appendix A), several distortion measures have been proposed for vector quantization algorithms, including **Euclidean distance**, the **Minkowski norm**, the **weighted-squares distortion**

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} a_i |x_i - y_i|^2, \; a_i \geq 0 \tag{148}$$

the **Mahalanobis distortion**, and the **general quadratic distortion**, defined as

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y}) \, \mathbf{B}(\mathbf{x} - \mathbf{y})^T = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} \, (x_i - y_i)(x_j - y_j) \tag{149}$$

where $\mathbf{B}$ is an $n \times n$ positive definite symmetric matrix.

These distortion measures are computed for two vectors, and they depend on the error vector $(\mathbf{x} - \mathbf{y})$. They are called **difference distortion measures**.

It is also possible to define the following average distortion measure for a given set $T = \{\mathbf{x}_i\}_{i=1}^{N}$ of N input pattern vectors, a given set of M codevectors $W = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_M\}$, and a given partition $P(W)$ of the input vector space on cells:

$$d_a = d_a\big(W, P(W)\big) = \frac{1}{N} \sum_{j=1}^{N} \min_{\mathbf{w} \in W} d(\mathbf{x}_j, \mathbf{w}) \tag{150}$$

where $d(\mathbf{x}_j, \mathbf{w})$ is the distortion measure for two vectors.

One can also define an average distortion measure as

$$D_a = \frac{1}{N} \sum_{i=1}^{N} d\big(\mathbf{x}_i, VQ(\mathbf{x}_i)\big) = \frac{1}{N} \sum_{i=1}^{N} \| \, \mathbf{x}_i - VQ(\mathbf{x}_i) \, \|_2^2 \tag{151}$$

or as

$$D_a = \frac{1}{M} D_i \tag{152}$$

where D_i is the total distortion for cell C_i:

$$D_i = \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{w}_i) \tag{153}$$

where $VQ(\mathbf{x}_i)$ denotes a mapping of the input pattern $\mathbf{x}_i$ into a codevector $\mathbf{w} = VQ(\mathbf{x}_i)$.

The goal of designing an optimal quantizer is to find a codebook of M codevectors such that the average distortion measure in encoding all patterns from the training set is minimal. To design an M-level codebook, the n-dimensional input pattern space is partitioned into M distinct regions or cells $\{C_i \; : \; i = 1, 2, \cdots, M\}$ and a codevector $\mathbf{w}_i$ is associated with each cell C_i. Figure 7.7 shows the partition.

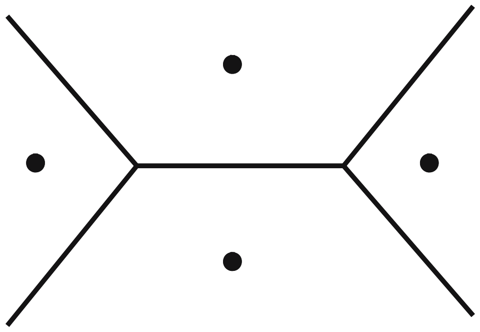

Figure 7.7. Partition of a pattern space into encoding regions (cells).

The quantizer then assigns the $\mathbf{w}_i$ if an input vector $\mathbf{x}$ belongs to C_i. Two necessary conditions (criteria) must be fulfilled in order to obtain an optimal codebook: the nearest neighbor condition and the centroid condition.

The first condition is that the partition of the input pattern space should be obtained by using a minimum-distortion or **nearest-neighbor selection rule**:

$$VQ(\mathbf{x}) = \mathbf{w}_i \quad \text{iff} \quad d(\mathbf{x}, \mathbf{w}_i) \le d(\mathbf{x}, \mathbf{w}_j), \ j \ne i, \ 1 \le j \le M \tag{154}$$

Here, the quantizer selects for $\mathbf{x}$ the codevector $\mathbf{w}_j$ that results in the minimum distortion with respect to $\mathbf{x}$. In other words, a cell C_i should consist of all patterns that are closer to $\mathbf{w}_i$ than any of the other codevectors:

$$\{\mathbf{x} \in T \ : \| \mathbf{x} - \mathbf{w}_i \|_2^2 \le \| \mathbf{x} - \mathbf{w}_j \|_2^2\}, \quad j = 1, \cdots M \tag{155}$$

The nearest neighbor conditions permits the realization of an optimal partition.

The second necessary condition to achieve an optimal codebook is that each codevector $\mathbf{w}_i$ is chosen to minimize the average distortion in cell C_i. That is, for each cell there exists a minimum distortion codevector for which

$$E\big[d(\mathbf{x}, \mathbf{w}) \mid \mathbf{x} \in C_i\big] = \min_j \Big(E\big[d(\mathbf{x}, \mathbf{w}_j)\big] \mid \mathbf{x} \in C_i\Big) \tag{156}$$

where E denotes the expectation with respect to the underlying probability distribution

$$E\big[d(\mathbf{x}, \mathbf{w}) \mid \mathbf{x} \in C_i\big] = \int_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{w}) \ p(\mathbf{x}) \ d\mathbf{x} \tag{157}$$

The codevector $\mathbf{w}_i$ is called the **centroid** of the cell C_i:

$$\mathbf{w}_i = \mathrm{cent}(C_i) \tag{158}$$

One can also say that codevector $\mathbf{w}_i$ should be the average of all those training patterns that are in the encoding region (a cell):

$$\mathbf{w}_i = \frac{\sum_{\mathbf{x} \in C_i} \mathbf{x}}{\sum_{\mathbf{x} \in C_i} 1}, \ i = 1, \cdots, M \tag{159}$$

Computing the centroid for a particular region or cell depends mainly on the definition of the distortion measure. The cells are known as **nearest neighbor cells** or Voronoi cells. Generally, the design of an optimal codebook requires knowledge of the probability distribution of the source pattern vectors. However, in most cases the distribution is not known and the codebook determination usually involves training from examples. Usually, a set of N training vectors $T = \{x_j\}_{j=1}^N$ is given that is representative of the data the quantizer is more likely to encounter in practice.

We briefly describe two vector quantization (clustering) algorithms:

– the generalized Lloyd algorithm (LBG centroid algorithm), and
– the iterative cluster splitting algorithm.

The idea of the **LBG centroid algorithm** is as follows. The algorithm starts with an initial codebook selected. For the given codebook (with its set of codevectors) the minimum distortion is iteratively improved until a local minimum is reached. In each iteration, a partition is realized, and then for a given partition the optimal set of codevectors is found. This can be done in two steps. In the first step, for each iteration, each pattern is mapped to the nearest codevector (according to the defined distortion measure between the two patterns) in the current codebook. In the second step, all the codevectors of the code book are recalculated as the centroids of new encoding regions of the new partitions. The algorithm continues as long as improvement is achieved. For a given set of codevectors and a data set of patterns, the partition of the input pattern space that minimizes the average distortion measure is achieved by mapping each pattern $\mathbf{x}_i$ $(i = 1, \cdots, N)$ to the codevector $\mathbf{w}_j$ $(j = 1, \cdots, M)$, for which the distortion $d(\mathbf{x}_i, \mathbf{w}_j)$ is the minimum over all $\mathbf{w}_j$. The centroid vector quantization clustering algorithm (which uses a squared Euclidean distortion metric) can be described as follows:

Algorithm:
Given: A training set $T = \{\mathbf{x}_i\}_1^N$ of N unlabeled pattern vectors, given number M of codebook reference vectors (quantization levels), and distortion measure threshold $\epsilon \geq 0$.

1. Select randomly an initial setting of the M-level codebook containing a set of M codevectors $\hat{W}^0 = \{\hat{\mathbf{w}}_i^0\}_{i=1}^M$.
2. Select a distortion measure threshold $\epsilon \geq 0$.
3. Set the previous average distortion measure $d_a^0 = 0$.
4. Set the iteration step number $k = 1$.
5. Given $\hat{W}^k = \{\hat{\mathbf{w}}_i^k\}_{i=1}^M$, find the minimum distortion partition $P(\hat{W}^k) = \{C_i\}_{i=1}^M$ of the training set patterns (based on the nearest-neighbor condition)

$$\mathbf{x}_j \in C_i \quad \text{if} \quad d(\mathbf{x}_j, \mathbf{w}_i) \leq d(\mathbf{x}_j, \mathbf{w}_l) \quad \text{for} \quad \text{all} \quad l = 1, 2, \cdots, M \tag{160}$$

   where $d(\mathbf{x}, \mathbf{w})$ is considered to be the distortion measure. If the distortion measures are the same for a few codevectors, then the assigning of patterns to cells is random.
6. Compute the average distortion for a given codebook of codevectors $\hat{W}^k$ and partition $P(W^k)$ at step k:

$$d_a^k = d_a^k\big[\hat{W}^k, P(\hat{W}^k)\big] = \frac{1}{N}\sum_{i=1}^{N} d\big(\mathbf{x}_i, VQ(\mathbf{x}_i)\big) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - VQ(\mathbf{x}_i)\|_2^2 \tag{161}$$

7. If the average distortion measure d_a^k at iteration step k relative to d_a^{k-1} is below a certain threshold

$$\frac{\left|d_a^{k-1} - d_a^k\right|}{d_a^k} \leq \epsilon \tag{162}$$

   then stop the iteration, with $\hat{W}^k$ being the final codebook of the codevector set. Otherwise, proceed to the next step.

8. Find the optimal codebook (codevectors) (based on the centroid condition) $\hat{W}^{k+1}\big(P(\hat{W}^k)\big) = \{\hat{\mathbf{w}}^{k+1}(C_i) \, : \, i = 1, 2, \cdots, M\}$ for $P(\hat{W}^k)$. For the squared-error distortion measure, $\hat{\mathbf{w}}^{k+1}(C_i)$ is the Euclidean center of gravity (or centroid) for a cell C_i and given by the equation

$$\hat{\mathbf{w}}^{k+1}(C_i) = \frac{1}{\|C_i\|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j \tag{163}$$

Here $\hat{\mathbf{w}}^{k+1}(C_i)$ is averaged in a componentwise manner, where $\|C_i\|$ denotes the number of training vectors in the cell C_i. If $\|C_i\| = 0$, set $\hat{\mathbf{w}}^{k+1}(C_i) = \mathbf{w}_i^k$ (the previous codevector).

Set $k = k + 1$. Continue the iteration from step 5.

In the above algorithm, the initial codebook vector set $\hat{W}^0$ must be guessed. Frequently, the M vectors randomly chosen from the training set T are selected as initial values for the codevectors.

Alternatively, an initial codebook vector set can be obtained by computing the centroid of the training set vectors and dividing this vector into two vectors. Each of these two vectors is then split into two new codevectors, and this process continues until the initial M level codebook vector set has been created. Each vector $\mathbf{w}_i$ is split by adding a fixed perturbation vector ϵ, thereby producing two new codevectors $\mathbf{w}_i + \epsilon$ and $\mathbf{w}_i - \epsilon$.

The top-down **cluster** (region) **splitting algorithm** starts with a single cluster including all patterns of the training set T, with one codevector $\mathbf{w}_1$ computed as a mean of all patterns belonging to the first cluster. Then new clusters are created (one at a time) by splitting the existing clusters. This can be done by "splitting" the codevector $\mathbf{w}_1$ into two close codevectors $\mathbf{w}_1 + \epsilon$ and $\mathbf{w}_1 - \epsilon$, where ϵ is an n-dimensional vector with small values ϵ of all elements. Such new codevectors are considered in the following iterations. For a given codevectors, a two-step procedure is realized: finding the minimum average distortion partitions, and finding the optimal set of code vectors (codebook) for a given partition. The cluster splitting process continues until the required number of clusters is reached.

The performance of VQ is frequently measured as the **signal-to-distortion ratio** (SDR):

$$SDR = 10 \, \log_{10} \frac{\sigma^2}{D_{\text{average}}} \; [db] \tag{164}$$

computed in dB. Here σ^2 is the source variance and D_{average} is the average squared-error distortion. Larger values correspond to better performance.

Example: We provided vector quantization of the black and white version of the Halinka image (Figure 7.8(a)) composed with $I \times J = 720 \times 680$ pixels of 256 gray levels. All gray levels are represented by an integer value included in [0, 255].

The image shown in Figure 7.8(a) was divided into $N = I/(n_b) \times J/(n_b) = 180 \times 170 = 30600$ blocks, each of $n_b \times n_b = 4 \times 4$ pixels, where $n_b = 4$ is the number of rows in a block (equal to the number of columns). Each block has been unfolded (row by row) into the resulting $n = n_b \times n_b = 4 \times 4 = 16$-element block vector $\mathbf{x}_b$ (a column vector). A sequence of n-element ($n = 16$) block vectors, constituted with a sequence of considered image blocks (from left to right, row by row) forms the unsupervised training set T_{tra} containing $N = 30600$ block vectors. The number of codevectors in the codebook was set to $M = 64$.

For the training set T_{tra} and a codebook size equal to $M = 64$, the optimal codebook was found using the generalized Lloyd algorithm. Figure 7.8(b) shows the reconstructed black and white version of the Halinka image.

Figure 7.8. Original (a) and reconstructed (b) Halinka images.

In image compression, the so-called **peak signal-to-noise ratio** (PSNR) is frequently used to evaluate the quality of the resulting images after the quantization process. The *PSNR* is defined as

$$PSNR = 10\ \log_{10} \frac{\mathrm{MAX}^2}{\frac{1}{IJ}\sum_{i=0}^{i=I-1}\sum_{j=0}^{i=J-1}\left(f(i,j)-\tilde{f}(i,j)\right)^2} \tag{165}$$

where $f(i,j)$ and $\tilde{f}(i,j)$ are, respectively, the gray level of the original image and of the reconstructed one and I and J are the number of rows and columns of an image. Here, MAX denotes the maximum value of an image intensity. For example, for an 8-bit image, $\mathrm{MAX} = 2^8 - 1 = 255$.

2.6.3. Complexity Issues

The computational and storage costs impose a very real and practical limitation on the applicability of vector quantization. The computational cost refers to the amount of computation needed per unit of time, and the storage cost refers to the memory size required to store the codebook for a particular application.

With a basic vector quantizer designed, each n-dimensional input vector $\mathbf{x}$ can be encoded by computing the distortion measure between the vector $\mathbf{x}$ and each of the codebook codevectors, keeping track of the one with minimum distortion and continuing until every codevector has been tested. For a quantizer with a codebook of size M, the number of distortion computations is M, and for a training set containing N patterns, each distortion computation requires N multiply-add operations for the squared-error distortion (other distortion measures can have higher computational demands). Therefore, the computational cost κ for encoding each input vector $\mathbf{x}$ is

$$\kappa = M\ N \tag{166}$$

If we encode each codevector into $R = \log_2 M$ bits, since $R = rn$ (where $r = \frac{R}{n}$ is a rate in bits per vector component) then

$$\kappa = N2^{rn} \tag{167}$$

The computational cost grows exponentially with the number of pattern dimensions and the number of bits per dimension. The storage cost can be measured assuming one storage location per vector component

$$\mu = n2^{rn} \tag{168}$$

Again, the storage cost grows exponentially in the number of dimensions and the number of bits per dimension.

2.7. Learning Vector Quantization

Kohonen proposed the so-called **Learning Vector quantization (LVQ)** algorithm (and its neural network implementation), which combines **supervised learning** and vector quantization. The algorithm guarantees the performance of vector quantization, but in addition uses the idea of **competitive learning** to select the winning neuron, whose weights will be adjusted and punish-reward learning utilized for the direction of weight adjustment. In a nutshell, assuming that Euclidean distance is used for pattern similarity measures, LVQ provides fine tuning (moving) of the rigid boundaries of the Voronoi tessellation regions. This technique utilizes additional information about regions, which comes from the class labels associated with data set patterns. The learning vector quantization algorithm assumes that the training set $T = \{(\mathbf{x}_i,\ C_i)\}_{i=1}^{N}$ of N n-element input pattern vectors $\mathbf{x} \in \mathbb{R}^n$ labeled by corresponding l categorical classes C_i is available for supervised learning. Generally, it is also assumed that the probability of classes is known as well as the probability density of patterns $p(\mathbf{x})$.

From this perspective, the LVQ algorithm can be seen as a classifier design based on a given supervised training set. However, the LVQ algorithm with Kohonen learning also provides vector quantization of input patterns from the population represented by a given, representative, labeled training set. This means that the LVQ algorithm guarantees the mapping of input patterns from the input pattern space $\mathbb{R}^n$ into one of the reference code vectors from the limited-size codebook $\left(W_c = \{(i,\ \mathbf{w}_i)\}_{i=1}^{M}\right)$. This is an approximation of input pattern vectors by their quantized values.

In the LVQ algorithm, for the purpose of precise vector quantization approximation, usually several reference code vectors of the codebook are assigned to represent the labeling patterns of each class C_i from the training set

$$W_{C_i} = \{\mathbf{w}_j\} \text{ for all } j,\ \mathbf{w}_j \text{ representing class } C_i \tag{169}$$

The pattern $\mathbf{x}$ is considered to belong to the same class to which the nearest reference code vector from the codebook $\mathbf{w}_j$ belongs. For vector $\mathbf{x}$ and for the Euclidean distance metric, we can write

$$j^{\text{th}} \text{ nearest reference vector } \mathbf{w}_j \min_{i=1,2,\cdots,M} ||\mathbf{x} - \mathbf{w}_i|| \tag{170}$$

Kohonen proposed a competitive supervised learning algorithm that approximately minimizes misclassification errors of vector quantization, stated as the **nearest-neighbors classification**.

During proposed supervised learning with a punish-reward idea of weights adjustment, the optimal reference vectors $\mathbf{w}_i$ $(i = 1, 2, \cdots, M)$ of the codebook may be found as the asymptotic values of the following competitive learning technique.

First, for a given input pattern $\mathbf{x}$ belonging to the class C_l and a previous value $\{\mathbf{w}_j^k\}_{i=1}^{M}$ of the codebook reference code vectors, the code vector that is nearest to the input pattern $\mathbf{x}$ is selected as the winner of the competition:

$$j^{\text{th}} \text{ nearest code vector } \mathbf{w}_j \min_{i=1,2,\cdots,M} ||\mathbf{x} - \mathbf{w}_i|| \tag{171}$$

This reference code vector belongs to a certain class C_r. Then only that j^{th} code vector $\mathbf{w}_j$ nearest to $\mathbf{x}$ will be adjusted in the following way:

$$\begin{aligned} \mathbf{w}_j^{k+1} &= \mathbf{w}_j^k + \alpha(k)[\mathbf{x} - \mathbf{w}_j] \quad \text{if} \quad C_l = C_r \\ \mathbf{w}_j^{k+1} &= \mathbf{w}_j^k - \alpha(k)[\mathbf{x} - \mathbf{w}_j] \quad \text{if} \quad C_l \neq C_r \\ \mathbf{w}_i^{k+1} &= \mathbf{w}_i^k \quad \text{if} \quad i \neq j \end{aligned} \tag{172}$$

where $0 < \alpha(k) < 1$ is the learning rate. The above weights adjustment is based on the "winner-take-all" and the punish-reward idea. Only the reference code vector $\mathbf{w}_j$ nearest to the pattern $\mathbf{x}$

is adjusted. If the class C_r of the reference code vector $\mathbf{w}_j$ is the same as the class C_l of the input pattern $\mathbf{x}$, then this winning reference code vector is rewarded by a positive adjustment in the direction of improving the match with vector $\mathbf{x}$:

$$\mathbf{w}_j^{k+1} = \mathbf{w}_j^k + \alpha(k)[\mathbf{x} - \mathbf{w}_j] \tag{173}$$

If the class of the winning reference code vector and the class of the input pattern are different, than the reference code vector is punished:

$$\mathbf{w}_j^{k+1} = \mathbf{w}_j^k - \alpha(k)[\mathbf{x} - \mathbf{w}_j] \tag{174}$$

Reference code vectors other than the j^{th} reference code vectors nearest to the pattern $\mathbf{x}$ remain unchanged. This learning scheme for vector quantization, which adjusts only the reference vector nearest to $\mathbf{x}$, is called **LVQ1**.

The main reason for the punish-reward learning scheme is to minimize the number of misclassifications. This scheme also ensures that the reference code vectors will be pulled away from the pattern overlap areas when misclassifications persist.

Vector quantization through supervised learning allows more fine tuning of the decision surfaces between classes (the borders between Voronoi cells represented by reference vectors). This outcome is understandable in the light of the idea of pattern classification, where the decision surface between pattern classes is most important and not the precision of the pattern distribution within classes.

In the above learning rule, an $\alpha(k)$ is a learning rate. The learning rate values are not critical for learning (as long as the number of learning steps is sufficiently large). The value may be set as a small constant or as a decreasing function with learning time (with a starting value for examples smaller than 0.1). Frequently a linear decreasing function of learning time is chosen $-\alpha(k) = \alpha_{\max}\left(1 - \frac{k}{N_l}\right)$ – where N_l is the maximal number of learning steps.

2.7.1. Learning Vector Quantization Neural Network

The feedforward static competitive learning vector quantization neural network LVQ1 consists of an input and an output layer fully connected through weights (see Figure 7.9). The weightless input layer is connected via weights with the output layer. The weightless n neurons of the input layer receive only the n-dimensional input pattern $\mathbf{x} \in \mathbb{R}^n$. The number of input layer neurons equals the size of the input pattern $\mathbf{x} \in \mathbb{R}^n$. The subsequent output layer with M neurons is fully connected with the input layer neurons via weights. The number of output layer neurons is equal

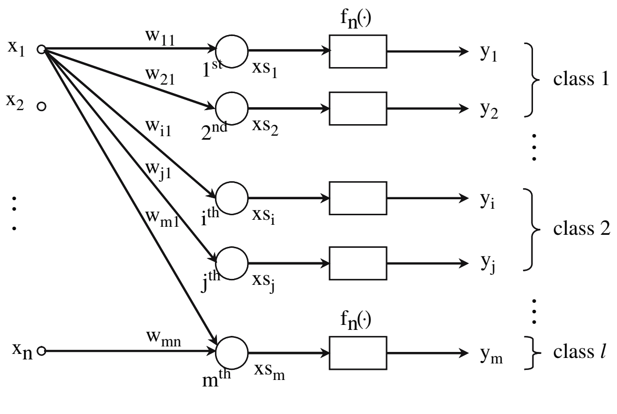

Figure 7.9. LVQ neural network.

to the size (or number of reference code vectors) of codebook M. The connection weights are described by the $n \times M$ weights matrix

$$\mathbf{W} = [\mathbf{w}_i], \quad i = 1, 2, \cdots, M \tag{175}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \cdots, w_{iM}]$ gives the vector weights of the i^{th} output neuron. The weights vectors $\mathbf{w}_i$ of each i^{th} output neuron represent the reference code vector of the codebook. Each output neuron represents a certain class of the training set.

The type of output activation function of the output neurons is not critically important and may be assumed to be an identity function $y = f_h(xs) = xs$, where xs is the intermediate value of an output neuron. Each i^{th} $(i = 1, 2, \cdots, M)$ output neuron, with value y_i, has a known class $C_{y_i} \in \{C_1, C_2, \cdots, C_l\}$ that represents it.

Several output neurons $i, j, \cdots, s$ representing the distinct reference vectors $\{\mathbf{w}_i, \mathbf{w}_j, \cdots, \mathbf{w}_s\}$ may be assigned to the same class C_g. Generally, we have $M \geq M$ in order to provide a better approximation of the input pattern space by reference vectors.

The processing model for immediate outputs $\mathbf{xs}$ and outputs $\mathbf{y}$ of the output neurons can be written as

$$\begin{aligned} \mathbf{xs} &= \mathbf{Wx} \\ \mathbf{y} &= F(\mathbf{xs}) \end{aligned} \tag{176}$$

or written in the scalar form

$$\begin{aligned} xs_i &= \mathbf{w}_i\mathbf{x} = \sum_{l=1}^{n} w_{il}x_l, \;\; i = 1, 2, \ldots, M \\ y_i &= f_h(xs_i) = xs_i \;\; i = 1, 2, \cdots, M \end{aligned} \tag{177}$$

where $\mathbf{W}$ is the $M \times n$ weights matrix.

2.7.2. Learning Algorithm for LVQ1 Neural Network

Vector quantization by the LVQ1 neural network requires

- Supervised learning from the class-labeled patterns of training set T. This learning is in the design of the optimal codebook $W_c = \{(i, \;\mathbf{w})\}_{i=1}^{M}$ composed of reference code vectors indexed by i, that is, $W = \{\mathbf{w}_i\}_{i=1}^{M}$, which are the asymptotic values of the learned vector weights of the output neurons.
- Processing of the input patterns by the already-designed neural network. For a given input pattern $\mathbf{x}$, the most responsive output neurons will be declared as the winners and will denote the reference vector, which will represent input pattern.

Given: The training set $T = \{\mathbf{x}_i, C_{x_i}\}_{i=1}^{L}$ containing N class-labeled patterns $\mathbf{x} \in \mathbb{R}^n$, the number of learning steps N_l, and the learning rate function $\alpha(k)$ (for example, $\alpha(k) = \alpha_{\max}\left(1 - \frac{k}{N_l}\right)$).

1. Select randomly M input patterns from the training set T and assign them as the starting values of the codebook reference vectors

$$\mathbf{w}_i^0 = \mathbf{x}_i, \quad i = 1, 2, \cdots, M \tag{178}$$

The remaining patterns of the training sets will be used for learning.

2. Present a randomly selected input pattern $\mathbf{x}$ to the network. This pattern will represent a known class C_l.

3. Adjust the learning rate $\alpha(k)$ for the k^{th} learning steps.
4. Select the j^{th} winning neuron. The winner is the neuron whose weights vector (representing the current codebook reference vector $\mathbf{w}_j = \mathbf{x}_{c,j}$) best matches (is nearest) the input vector $\mathbf{x}_i$ according to the minimal Euclidean distance:

$$j^{\text{th}} \quad \text{nearest reference vector} \quad \mathbf{w}_j \quad \min_{i=1,2,\ldots,M} ||\mathbf{x} - \mathbf{w}_i|| \tag{179}$$

   This reference vector $\mathbf{w}_j$ belongs to a certain class C_r.
5. Adjust only the j^{th} winning neuron's weights $\mathbf{w}_j$, related to the j^{th} reference vector of the codebook, by an amount depending on whether the class C_l of input pattern $\mathbf{x}$ is in the same as the class C_r of the j^{th} winning neuron:

   (a)

$$\begin{aligned} \mathbf{w}_j^{k+1} &= \mathbf{w}_j^k + \alpha(k)[\mathbf{x} - \mathbf{w}_j] \quad \text{if} \quad C_l = C_r \\ \mathbf{w}_j^{k+1} &= \mathbf{w}_j^k - \alpha(k)[\mathbf{x} - \mathbf{w}_j] \quad \text{if} \quad C_l \neq C_r \end{aligned}$$

   (b)

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k \quad \text{if} \quad i \neq j \tag{180}$$

   where $0 < \alpha(k) < 1$ is the learning rate. The above weights adjustments is based on the "Winner-Take-All" and punish-reward ideas.

6. Test stopping condition. This condition may be specified by reaching a fixed maximal number of iterations $k \leq N_l$ or by the learning rate reaching a minimal value $\alpha_{\min}$. If the stopping condition is not met, then continue iterations from step 2. If the stopping condition is satisfied, stop the iterations.

Result: A weight vector $\mathbf{w}_i (i = 1, 2, \cdots, M)$ that constitutes the reference vectors of the codebook.

Modification of the learning rate is frequently realized after all the patterns from the training sets have been learned (after learning of one epoch).

During the learning computation of the network output neurons, intermediate outputs xs_i $(i = 1, 2, \cdots, M)$ are not required.

The inputs are processed using the following algorithm.

Given: The network architecture with an n-neuron input layer fully connected via weights with M neurons of the output layer. The weights vectors of the output neurons represent M stored reference code vectors of the codebook. The learned weights matrix $\mathbf{W} = [\mathbf{w}_i](i = 1, 2, \cdots, M)$ whose vectors $\mathbf{w}_i$ represent the reference vectors of a codebook is also given.

For the given input pattern $\mathbf{x}$, the LVQ1 neural network provides the following forward computations.

1. The input-layer neurons receive the input pattern $\mathbf{x}$.
2. The intermediate values xs of the output neurons are computed by the matrix equations

$$\mathbf{xs} = \mathbf{Wx} \tag{181}$$

   where

$$xs_i = \sum_{j=1}^{n} w_{ij} x_j, \quad i = 1, 2, \cdots, M \tag{182}$$

3. The final values of the output neurons are computed through the output activation functions f_h

$$y_i = f_h(xs_i) \tag{183}$$

or in the vector form

$$\mathbf{y} = \mathbf{F}_h(\mathbf{xs}) \tag{184}$$

where $\mathbf{F}_h(\cdot) = [f_h(xs_1), f_h(xs_2), \cdots, f_h(xs_M)]^T$ is the M-dimensional output activation vector function.

As a result of processing, many output neurons may have nonzero values, depending on the distance from the input pattern to the neuron's weight vector. However, an output neuron may have the dominating value. This neuron can be considered as indicating the input vector class (the cluster to which the input vector belong). To specify the reference vector best representing an input vector, one can identify the neuron with the most responsive output value:

$$j^{\text{th}} \text{ code vector } \; y_j = \max_{i=1,2,\ldots,M} y_i, \tag{185}$$

2.7.3. Practical Design Issues for LVQ Vector Quantizers

There is no existing rigorous proven design method for optimal LVQ vector quantizers. Good design still relies on heuristics and simulation-based findings. The goal of LVQ-type vector quantization is not to best approximate the probability density function of the class pattern samples but to directly define between-class borders based on the nearest-neighbor technique. The performance and accuracy of an LVQ quantizer can be evaluated based on its ability to perform accurate and generalized classification or to quantize input patterns for reduction purposes with minimal distortion. The accuracy of an LVQ may depend in the design phase on the following:

– An optimal number of reference vectors assigned to each class defined by training set
– Initial values of the codebook reference vectors
– Concrete, detailed implementation of a learning algorithm: an appropriate learning rate and its decreasing rule, an appropriate stopping criterion, and an order of selecting the input pattern from the training set, as well as a mix of different LVQ types of algorithms in the multistep hybrid sequence of learning
– A stopping rule and generalization through crossvalidation

2.8. The Fourier Transform

Fourier analysis is one of the most important data processing methods. This section focuses on the role of **Fourier transform** in feature extraction and pattern formation in timeseries.

An appropriate transform of data or functions into another space may result in better understanding and representation, with simpler and computationally more efficient processing algorithms. The Fourier transform maps a function, or a signal, into the **frequency domain**, providing information about the periodic frequency components of a function or of a signal (generated by a function).

The Fourier transform decomposes a function into a **spectrum** of its frequency components, and the inverse transform synthesizes a function from its spectrum of frequency components. In experimental domains, the transform of a signal can be thought of as that signal in the "frequency domain." In other words, the Fourier transform decomposes a function or a signal into sine waves with different frequencies. The Fourier transform localizes a function or a signal in frequency space. However, it does not localize such frequencies in time. The results of the Fourier transform are Fourier coefficients $F(u)$ (a spectrum) related to different frequencies. Multiplication of **spectral coefficients** by a sinusoid of corresponding frequency results in reconstruction (an inverse transform) of the sinusoidal component of the original function or signal.

The Fourier transform allows us to form patterns of timeseries or of images in frequency domain. Fourier transform processing comes from the observation that a periodic continuous function $f(t)$ with period 2π can be represented by the infinite sum of sine and cosine functions

$$a_0 + \sum_{k=1}^{\infty} (a_k \cos(kt) + b_k \sin(kt)) \tag{186}$$

where parameters a_0, a_k, b_k are defined as

$$\begin{aligned} a_0 &= \frac{1}{2\pi} \int_0^{2\pi} f(t)\, dt \\ a_k &= \frac{1}{\pi} \int_0^{2\pi} f(kt) \sin(kt)\, dt \\ b_k &= \frac{1}{\pi} \int_0^{2\pi} f(kt) \cos(kt)\, dt \end{aligned} \tag{187}$$

Fourier transform is a linear mapping of functions to other functions (in a frequency domain). In other words, the Fourier transform decomposes a function into a continuous spectrum of its frequency components. The inverse Fourier transform reconstructs (synthesizes) a function from its spectrum of frequency components back into the original domain. In empirical domains, the Fourier transform provides representation of a signal in the time domain into the "frequency domain."

For a continuous function $f(t)$ of a real variable t, the continuous **Fourier transform** $FT(f(t)) = F(u)$ is defined as

$$FT(f(t)) = F(u) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi ut}\, dt \tag{188}$$

where $j = \sqrt{-1}$ is the imaginary unit and u is the **frequency variable**. Given $F(u)$, we can obtain an original function $f(t)$ by using the **inverse Fourier transform**:

$$f(t) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ut}\, du \tag{189}$$

We can also write

$$F(j\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt \tag{190}$$

where $\omega = 2\pi u$ is the angular frequency (in radians) and u is the oscillatory frequency (in HZ). Here, the inverse Fourier transform is given by

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(j\omega) e^{j\omega t}\, d\omega \tag{191}$$

The term frequency variable comes from the fact that, using Euler's formula, the exponential term $e^{-j2\pi ut}$ may be written in the form

$$e^{-j2\pi ut} = \cos 2\pi ut - j \sin 2\pi ut \tag{192}$$

$F(u)$ is composed of an infinite sum of sine and cosine terms, and each value of u determines the frequency of its corresponding sine-cosine pairs.

We can also write

$$F(u) = \int_{-\infty}^{\infty} f(t)(\cos\ 2\pi ut - j\sin\ 2\pi ut)dt \tag{193}$$

The Fourier transform (even for a real function) is generally complex, that is

$$F(u) = R(u) + j\,I(u) \tag{194}$$

where $R(u)$ and $I(u)$ are, respectively, the real and imaginary components of $F(u)$. The above equation is often more convenient to express in the exponential form

$$F(u) =\mid F(u) \mid\ e^{j\phi(u)} \tag{195}$$

where

$$\mid F(u) \mid = [R^2(u) + I^2(u)]^{1/2} \tag{196}$$

The $|F(u)|$ (magnitude) is called the **Fourier spectrum** (frequency spectrum) of function $f(t)$. The term

$$\phi(u) = \tan^{-1}\left[\frac{I(u)}{R(u)}\right] \tag{197}$$

is called the **phase angle** of the Fourier transform. The square of Fourier spectrum values

$$P(u) =\mid F(u) \mid^2 = R^2(u) + I^2(u) \tag{198}$$

is called the **power spectrum** (spectral density) of function $f(t)$.

2.8.1. Basic Properties of the Fourier Transform

Time – Frequency Duality.

$$F(t) \Leftrightarrow f(-u) \tag{199}$$

Linearity.

$$F(af(t) + bg(t)) = aF(f(t)) + bF(g(t)) \tag{200}$$

Symmetry. The Fourier transform is symmetric, since $F(u) = FT(f(t))$ implies $F(-u) = FT(f(-t))$:

$$\begin{gathered}\text{if } f(x) \text{ is real, then } F(-u) = F(u)^* \\ \text{if } f(x) \text{ is imaginary, then } F(-u) = -F(u)^* \\ \text{if } f(x) \text{ is even, then } F(-u) = F(u) \\ \text{if } f(x) \text{ is odd, then } -F(-u) = -F(u)\end{gathered} \tag{201}$$

where $*$ indicates a complex conjugate operation.

Orthogonality. Functions $\frac{1}{\sqrt{2\pi}}e^{j\omega t}$ form an orthogonal basis

$$\int_{-\infty}^{\infty} \left(\frac{1}{\sqrt{2\pi}}e^{jat}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-jbt}\right) dt = \delta(a-b) \tag{202}$$

where δ is Kronecker delta. The Fourier transform can be considered as a transformation of coordinate bases in this space.

Scaling.

$$f(at) \Leftrightarrow \frac{1}{|a|}F\left(\frac{u}{a}\right) \tag{203}$$

Time Shift.

$$f(t-t_0) \Leftrightarrow e^{-j2\pi u t_0}F(u) \tag{204}$$

Convolution.

$$f(t) \otimes g(t) \Leftrightarrow F_f(u)F_g(u) \tag{205}$$

where $\otimes$ denotes the convolution operation. Here, we have

$$f(t) \otimes g(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau \tag{206}$$

Releigh Property.

$$\int_{-\infty}^{+\infty} |f(t)|^2 dt = \int_{-\infty}^{+\infty} |F(u)|^2 du \tag{207}$$

2.8.2. Discrete Data and the Discrete Fourier Transform

In real life processing we deal mostly with discrete data and we apply Discrete Fourier Transform.

The **sampling rate** or sampling **frequency** determines the number of samples per second taken from a continuous signal in order to create a discrete signal. The inverse of the sampling frequency is the **sampling period** or **sampling time**, which is the time between samples.

The sampling frequency is measured in Hertz. The Hertz (denoted by Hz) is the unit of frequency. One Hz denotes one sample of signal per second, 50 Hz means 50 samples per second, etc. The unit may be applied to any periodic event.

2.8.3. The Discrete Fourier Transform

Frequently a continuous function $f(t)$ is given as a finite discrete set of N uniformly spaced samples of the function values

$$\{f(t_0), f(t_0+\Delta t), f(t_0+2\Delta t), \cdots, f(t_0+(N-1)\Delta t)\} \tag{208}$$

denoted also as

$$\{f(k), k=0,1,\cdots,N-1\}, \quad \{f(k)\}_{k=0}^{N-1} \tag{209}$$

in N sampling points of variable t in the range of values $[t_0, t_0+(N-1)\Delta t]$. Here, Δt is the time increment between samples.

For the discrete sampled function $f(k)$, the **discrete Fourier transform** (DTF) is defined as

$$F(u) = \sum_{k=0}^{N-1} f(k) e^{-j2\pi uk/N}, \text{ for } u = 0, 1, 2, \cdots, N-1 \tag{210}$$

and the **inverse of the discrete Fourier transform** is given by

$$f(k) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{j2\pi uk/N}, \text{ for } k = 0, 1, 2, \cdots, N-1 \tag{211}$$

The discrete values of samples $u = 0, 1, \cdots, N-1$ in the discrete Fourier transform correspond to the samples of the continuous Fourier transform given for the values 0, Δu, $2\Delta u$, $\cdots$, $(N-1)\Delta u$. Hence, $F(u)$ represents $F(u\,\Delta u)$. The relation between Δt and Δu is given by

$$\Delta u = \frac{1}{N\,\Delta t} = \frac{1}{T} \tag{212}$$

We can see that the following equalities hold in the time and frequency domains.

Time Domain.

– T : total sampling time (sampling interval, total sampling length of a time signal)
– N : total number of discrete samples taken (sample size)
– Δt : time increment between samples (time spacing) $\Delta t = \frac{T}{N}$
– length : $T = (N-1)\Delta t$
– period : $T = N\Delta t$

Frequency Domain.

– N : number of components in the spectrum
– f_s : the sampling frequency (sampling rate) $f_s = \frac{1}{\Delta t} = \frac{N}{T}$
– $\Delta u = \Delta f$: the **frequency increment** (**frequency resolution**) $\Delta u = \frac{1}{T} = f_s/N = \frac{1}{N\,\Delta t}$
– f_p : the frequency period (spectrum period) $f_p = Nf_s = \frac{N}{\Delta t}$ in Hz.
– $f_{\max}$: the maximum frequency $f_{\max} = \frac{1}{2} f_s$

The time domain values are measured in seconds, and the frequency domain values in Hz. Both FT and DFT deal with discrete functions in time and frequency. The DFT transforms a series of N values in time $f(t_i)$, $(i = 0, 1, \cdots, N-1)$, into a series of N components $F(f_i)$, $(i = 0, 1, \cdots, N-1)$, in the frequency space, where f_i is a discrete frequency in the spectrum. For samples equally spaced in time, with total sampling time T [seconds] and time increment between samples Δt (signal sampled every Δt seconds for $(N-1)\Delta t$ seconds), the N discrete samples taken create a time signal of length

$$T = (N-1)\Delta t \tag{213}$$

which is equal to the total sampling period T in seconds. For a signal with N samples, the frequency spectrum has N components. The numbers of samples N in time is usually taken to be a power of $2 (N = 2^r)$.

The **sampling frequency** is

$$f_s = \frac{1}{\Delta t} = \frac{N}{T} \tag{214}$$

The spacing interval for frequencies (the frequency resolution)

$$\Delta f = \frac{1}{N\Delta t} = \frac{1}{T} \tag{215}$$

is determined for given N and Δt. The highest frequency is determined as

$$f_{\max} = \frac{N}{2}\Delta f = \frac{1}{2}f_s(Hz) \tag{216}$$

In DFT, a computing aliasing situation might happen when false spectral lines appear in a spectrum from frequencies above the measured bandwidth. Application of the **Nyquist frequency** criterion helps to prevent such situations. The Nyquist frequency (critical frequency) is half the sampling frequency for a signal. In principle, a Nyquist frequency equal to the signal bandwidth is sufficient to allow perfect reconstruction of the signal from the samples. The critical $1/2f_s$ frequency point is known as the Nyquist frequency. The Nyquist criterion states that for signal sampling at some sampling frequency f_s, it is possible to obtain proper (without aliasing) frequency information only for frequencies less than $f_s/2$. In other words, only half of the N outputs of DFT can be used.

Example: Assume that we sample a signal for $T = 2$ seconds, at a sampling rate of $f_s = 100\,\text{Hz}$. The number of samples taken in 2 seconds is $N = Tf_s = 2 \times 100 = 200$. The time interval between samples is $\Delta t = 1/f_s = 1/100 = 0.01$ seconds. The number of useful samples that has been obtained is $N/2 = 100$. The frequency resolution is $\Delta f = \frac{1}{T} = 0.5\,Hz$. The maximum frequency for which the DFT output is reliable is $f_{\max} = f_s/2 = (100\,Hz)/2 = 50\,Hz$ (or $f_{\max} = \frac{N}{2}\Delta f = 50\,Hz$).

The real part of the transform is even whenever the signal is real. Hence, the values for $k < N/2$ are negative frequency results. Negative frequency values are mirror images of the positive values around the middle frequency $f_p/2$ called the **folding frequency**. For N points in time space, there are N points in the frequency domain. For real signals, there are N complex values in the transform with N real and complex parts. Around the folding frequency, the real part is even and the imaginary part is odd. Including the component for frequency $f = 0$, there are in fact only $N/2+1$ points in the frequency domain.

For the signal $f(t)$, with N samples, with sampling time Δt and corresponding spacing interval for frequencies $\Delta u = \frac{1}{N\Delta t}$, the frequency components of the spectrum in HZ are

$$f = -\frac{N\,\Delta u}{2}, \cdots, 0, \Delta u, , \cdots, \left(\frac{N}{2} - 1\right)\Delta u \tag{217}$$

The Fourier spectrum of the signal $f(t)$ is periodic, and the spectrum period is $f_p = N/\Delta t = Nf_s$ in Hz. Since the signal spectrum is symmetric in pattern formation, one can only consider spectral values for $N/2+1$ frequencies:

$$f = 0, \Delta u, 2\Delta u, \cdots, \left(\frac{N}{2} - 1\right)\Delta u \tag{218}$$

2.8.4. Fast Fourier Transform

A **fast Fourier transform** (FFT) is a computationally efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. Due to reduced computation costs FFT plays an important role in digital signal processing and its applications.

For a sequence of discrete complex values $x_0, \cdots, x_{N-1}$, a DFT is defined by the formula

$$F(u) = \sum_{k=0}^{N-1} x_k e^{\frac{-j2\pi}{N}ku} \quad u = 0, \cdots, N-1 \tag{219}$$

A direct evaluation of the Fourier transform for this sequence would requires $O(N^2)$ arithmetical operations. An FFT algorithm computes the same result in only $O(n \log n)$ operations (where log is the base-2 logarithm). The FFT algorithm can also be easily adapted for computation of an inverse Fourier transform.

2.8.5. Properties of the Discrete Fourier Transform

The basic properties of the DFT are similar to that for the continuous Fourier transform. We can list some specific properties of the DFT:

– **Orthogonality**. Vectors $e^{j2\pi ku/N}$ form an orthogonal basis over the set of N-dimensional complex vectors:

$$\sum_{k=0}^{N-1} e^{j2\pi ku/N} \; e^{-j2\pi k'u/N} = \begin{cases} N & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases} \tag{220}$$

– **Autocorrelation** from the Fourier transform. It can be shown that the Fourier transform of the power spectrum is the autocorrelation function. In applications where the full autocorrelation function is needed, it may be faster to use this method than the direct computation.

2.8.6. Short Fourier Transform

The **discrete short-term Fourier transform** (SDFT) is the DFT performed for the short N-elements frame (usually windowed) of discrete complex (or just real) values $x(k)_{k=0}^{N-1}$. The result is an N-element array of complex values that is the discrete frequency spectrum of a signal spaced at interval $f_s = 1/(N\Delta t)$, where Δt is the spacing between time samples.

The SFT is widely used in speech processing, where a speech signal is divided into overlapping short sliding frames (containing 64, 128, 256, or 512 samples). It is believed that the parameters of the spectral model of the speech frame are stationary within a frame, and a sequence of frames represents variability and the natural sequential character of the speech signal. The SFT spectral components are extracted from windowed frame signals $x_w(k)$:

$$x_w(k) = x(k) \cdot w(k) \tag{221}$$

The popular Hamming window is defined as $w(k) = 0.54 - 0.46\cos(2\pi k/(n_w - 1))$, where n_w is the window size.

2.8.7. Spectrogram

In order to grasp the sequential character of longer signals (for example, speech signals), the subsequent windows from a sliding frame along the speech signal are arranged in a so-called **spectrogram**. A spectrogram shows the sequential character of a speech signal in the time-frequency coordinate system on a plane. Time runs along the horizontal axis, while frequency runs along the vertical axis. The amount of energy (the power spectrum of the window's short Fourier transform) in any region in the time-frequency plane is depicted by the darkness of shading. This spectrogram image is composed with subsequent power spectrum vectors (as subsequent columns) obtained by taking discrete short Fourier transforms of short subsequent windows of the original signal in the time domain. A spectrogram is a signal representation in the time domain that captures the sequential nature of processed signals. A spectrogram might be further processed, for example, by using image processing transforms. We can also compose a pattern vector representing a sequence of frames as a concatenation of spectrogram columns.

2.8.8. Pattern Forming for a Spectral DFT Signal

Since spectrum of signal is symmetric in pattern forming, one can only consider spectral values for $n_{dft} = N/2+1$ frequencies

$$f = \{0, f_s, 2f_s, \cdots, \left(\frac{N}{2}-1\right)f_s\} \tag{222}$$

which corresponds to the following indices of DFT values

$$n_{dft}, n_{dft}+1, \cdots, n_{dft}+N/2+1 \tag{223}$$

1. The **spectral patterns $\mathbf{x}_{sdft}$**.
   The spectral DFT pattern can be formed as

$$\mathbf{x}_{sdft} = \left[|F(n_{dft})|, |F(n_{dft}+1)|, \cdots, |F(n_{dft}+N/2+1)|\right]^T \tag{224}$$

   where $|F(i)|$ is an amplitude of DFT transform value for frequency related to index i of DFT.
2. The **power spectrum patterns $\mathbf{x}_{pdft}$**.
   The power spectrum pattern can be formed as

$$\mathbf{x}_{pdft} = \left[PF(n_{dft}), PF(n_{dft}+1), \cdots, PF(n_{dft}+N/2+1)\right]^T \tag{225}$$

   where $PF(i)$ is the power spectrum value for frequency related to index i of DFT.

2.9. Two-dimensional Fourier Transform and 2D Spectral Features

One of the most powerful feature extraction methods from images or sub-windows of images is based on the two-dimensional Fourier transform. The Fourier transform allows us to represent and interpret an image in the frequency domain. Let us begin with the general continuous two-dimensional Fourier transform.

2.9.1. The Continuous 2D Fourier Transform

The two-dimensional continuous Fourier transform (2DFT) for the two-dimensional continuous function $f(x, y)$ is defined as

$$F(u, v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)e^{-j2\pi(ux+vy)}\, dx\, dy \tag{226}$$

and the **inverse** of the Fourier transform is defined as

$$f(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u, v)e^{j2\pi(ux+vy)}\, du\, dv \tag{227}$$

The Fourier transform $F(u, v)$ (even for a real function) is generally complex. From Euler's formula, we have

$$e^{-j(ux+vy)} = \cos(ux+vy) - j\sin(ux+vy)$$

Here, x and y are spatial coordinates, u and v are the spatial frequencies, and $F(u, v)$ is the **frequency spectrum**.

For the continuous transform, (x, y) and (u, v) take on a real continuum of values. The spectrum $F(u, v)$ is complex and periodic.

2.9.2. The Discrete 2D Fourier Transform

For the two-dimensional continuous function sampled in the $2D$ grid of $M \times N$ points, with divisions of width Δx and Δy for the x- and y-axis, respectively, we can define the **two-dimensional discrete Fourier transform**. Here, the discrete function $f(x, y)$ represents discrete samples of the function $f(x_0 + x\Delta x, y_0 + y\Delta y)$ for $x = 0, 1, \cdots, M-1$ and $y = 0, 1, \cdots,$ N-1. Similarly, the discrete function $F(u, v)$ represents samples of the function $F(u\Delta u, v\Delta v)$ at $u = 0, 1, \cdots, M-1$ and $v = 0, 1, \cdots, N-1$. The sampling increments in the frequency domain are given by

$$\Delta u = \frac{1}{M\Delta x} \qquad \Delta v = \frac{1}{N\Delta y} \tag{228}$$

The two-dimensional discrete Fourier transform is given by the formula

$$\begin{aligned} F(u, v) &= \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})} \\ &= \frac{1}{M} \sum_{x=0}^{M-1} \left[\frac{1}{N} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{vy}{N})} \right] e^{-j2\pi(\frac{ux}{M})}; \\ & u = 0, 1, \cdots, M-1, \quad v = 0, 1, \cdots, N-1 \end{aligned} \tag{229}$$

The term in square brackets [] is the one-dimensional discrete Fourier transform of the x^{th} line (row) and can be computed using standard Fourier transform procedures (usually assuming $N = 2k$). Each line is replaced by its Fourier transform, and the one-dimensional discrete Fourier transform of each column is computed.

The **inverse** $2D$ **discrete Fourier transform** is given by the equation

$$\begin{aligned} f(x, y) &= \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})}; \\ x &= 0, 1, \cdots, M-1, \quad y = 0, 1, \cdots, N-1 \end{aligned} \tag{230}$$

The kernel function for the $2D$ discrete Fourier transform is

$$e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{231}$$

The Fourier transform $F(u, v)$ (even for a real function) is generally complex and consists of real and imaginary parts. Using Euler's formula, it can be expressed as

$$F(u, v) = Re(u, v) + j\,Im(u, v), \quad u = 0, \cdots, M-1, \quad v = 0, \cdots, N-1 \tag{232}$$

where

$$Re(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \left[\sum_{y=0}^{N-1} f(x, y) \cos\left(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\right) \right] \tag{233}$$

is the real part, and

$$Im(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \left[\sum_{y=0}^{N-1} -f(x, y) \sin\left(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\right) \right] \tag{234}$$

is the imaginary part of the transform. We can also express $F(u, v)$ in the exponential form

$$F(u, v) = |F(u, v)|e^{i\phi(u,v)} \tag{235}$$

where the norm of magnitude (amplitude) $|F(u, v)|$

$$|F(u, v)| = \sqrt{Re^2(u, u) + Im^2(u, v)} \tag{236}$$

is called the **Fourier spectrum** (frequency spectrum) of $f(x, y)$ and the term

$$\phi(u, v) = \tan^{-1}\left[\frac{Im(u, v)}{Re(u, v)}\right] \tag{237}$$

is the **phase spectrum** (phase angle). The square of the amplitude

$$P(u, v) = |F(u, v)|^2 = Re^2(u, v) + Im^2(u, v) \tag{238}$$

is called the **power spectrum** $P(u, v)$ of $f(x, y)$. The power spectrum $P(u, v)$ is also called the **spectral density**.

Let us consider the Fourier transform for images, which are defined on a finite support. In computing the Fourier transform of an image, we will consider an image as an $M \times N$ matrix, where M is a number of rows and N is a number of columns:

$$\begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,0) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{239}$$

where $f(x, y)$ denotes pixel brightness at the integer coordinates (x, y) of an image. If an image has width N and height M with the origin in a center, then

$$F(u, v) = \sum_{-M/2}^{M/2} \sum_{-N/2}^{N/2} f(x, y)e^{-j2\pi(ux+vy)} \tag{240}$$

Here, we assume that $f(x, y)$ is extended, with $f(x, y) = 0$ outside the image frame.

The $2D$ discrete Fourier transform is an important image processing tool that is used to decompose an image into its sine and cosine components. The input to 2DDFT is an image in the real domain, whereas output of the transformation represents the image in the Fourier or frequency space. In the Fourier space image, each point represents a specific frequency contained in the real domain image.

The Fourier transform is used in a wide range of applications, such as image analysis, filtering, recognition, compression, and image reconstruction.

2.9.3. Basic Properties of 2DDFT

There are several properties associated with the two-dimensional Fourier transform and the 2D inverse Fourier transform. Generally, the properties of 2DDFT are the same as those for one-dimensional DFT. These properties have an interesting interpretation when 2DDFT is applied to images. We can list some of the most important properties of 2DDT as applied to digital image processing. The Fourier transform is, in general, a complex function of real frequency variables. As such, the transform can be written in terms of its magnitude and phase.

The Fourier transform is **linear**. For 2DDFT of images, this means, that

– adding two images together results in adding the two Fourier transforms together
– multiplying an image by a constant multiplies the image's Fourier + transform by the same constant

Separability means that the Fourier transform of a two-dimensional function is the Fourier transform in one dimension of the Fourier transform in the other direction. This means that we can compute the two-dimensional Fourier transform by providing a one-dimensional Fourier transform of the rows and then taking a one-dimensional Fourier transform of the columns of the result.

Rotational Invariance. Rotation of an image results in the rotation of its Fourier transform.

Translation and Phase. Translation of an image does not change the magnitude of the Fourier transform but does change its phase.

Scaling. Changing the spatial unit of distance changes the Fourier transform. If the $2D$ signal $f(x, y)$ is scaled $(M_x\, x, M_y\, y)$ in its spatial coordinates (x, y), then $F(u, v)$ becomes $F(u/M_x, v/M_y)/|M_x M_y|$.

Periodicity and Conjugate Symmetry. The Fourier transform in discrete space is periodic in both space and in frequency. The periodicity of the Fourier transform can be explained by

$$\begin{gathered} F(u, -v) = F(u, N_p - v)\ ;\ F(-u, v) = F(M_p - u, v) \\ F(aM_p + u, bN_p + v) = F(u, v)\ ;\ F(-x, y) = F(M_p - x, y) \\ f(x, -y) = f(x, N_p - y)\ ;\ f(aM_p + x, bN_p + y) = f(x, y) \end{gathered} \tag{241}$$

where N_p and M_p are periods. If a 2D signal $f(x, y)$ is real, then the Fourier transform possess certain symmetries:

$$F(u, v) = {}^{*}F(-u, -v) \tag{242}$$

The symbol $(*)$ indicates complex conjugation of $F(u, v)$. For real signals,

$$|F(u, v)| = |F(-u, -v)| \tag{243}$$

If a 2D signal is real and even, then the Fourier transform is real and even.

Energy. According to Parseval's theorem, the energy in a 2D signal can be computed either in the spatial domain or in the frequency domain. For a continuous 2D signal with finite energy,

$$E = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} |f(x, y)|^2\, dx\, dy = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} |F(u, v)|^2\, du\, dv \tag{244}$$

For a discrete 2D signal with finite energy,

$$E = \sum_{-\infty}^{+\infty} \sum_{-\infty}^{+\infty} |f(x, y)|^2 = \frac{1}{4\pi^2} \int_{-\pi}^{+\pi} \int_{-\pi}^{+\pi} |F(u, v)|^2\, du\, dv \tag{245}$$

Convolution. For three given two-dimensional signals a, b, and c and their Fourier transforms F_a, F_b, and F_c,

$$c = a \otimes b \rightarrow F_a \cdot F_b \tag{246}$$

$$c = a \cdot b \rightarrow F_c = \frac{1}{4\pi^2} F_a \otimes F_b \tag{247}$$

where $\otimes$ denotes convolution operation. Convolution in the spatial domain is equivalent to multiplication in the Fourier (frequency) domain and vice versa. This property provides a method for the implementation of a convolution.

2.9.4. 2DDFT Patterns

A power spectrum of 2DDFT of an image can be used to form an image **spectral pattern**. In the first phase of feature extraction from an image, the 2DDFT can be used in order to convert the gray-scale image pixels into the corresponding spatial frequency representation. The 2DDFT complex features are extracted from an $M \times N$ pixel image by formulas:

$$F(u, v) = Re(u, v) + j\, Im(u, v), \quad u = 0, \cdots, M-1, \quad v = 1, \cdots, N-1 \tag{248}$$

where

$$\begin{aligned} Re(u, v) &= \frac{1}{MN} \sum_{x=-\frac{M}{2}}^{\frac{M}{2}-1} \left[\sum_{y=-\frac{N}{2}}^{\frac{N}{2}-1} f(x, y) \cos\left(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\right) \right] \\ Im(u, v) &= \frac{1}{MN} \sum_{x=-\frac{M}{2}}^{\frac{M}{2}-1} \left[\sum_{y=-\frac{N}{2}}^{\frac{N}{2}-1} -f(x, y) \sin\left(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\right) \right] \end{aligned} \tag{249}$$

and (x, y) and (u, v) denote the image pixel integer-valued coordinates.

In the above equation, $Re(u, v)$ denotes the real component and $Im(u, v)$ the imaginary component of the discrete Fourier transform of an image. For each pixel $f(u, v)$ of an original image, we can compute the real and imaginary part of $Re(u, v)$ and $Im(u, v)$ and then the real-valued power spectrum

$$P(u, v) = |F(u, v)|^2 = Re^2(u, v) + Im^2(u, v) \tag{250}$$

This power spectrum can be represented as an $m \times n$ image (a power spectrum map). In the most general situation, a two-dimensional transform takes a complex array. The most common application is for image processing, where each value in the array represents a pixel; the real value is the pixel value, and the imaginary value is 0. Two-dimensional Fourier transforms simply involve a number of one-dimensional Fourier transforms. More precisely, a two-dimensional transform is achieved by first transforming each row, replacing each row with its transform, and then transforming each column and replacing each column with its transform. Thus a 2D transform of a 1000×1000 image requires 2000 1D transforms. This conclusion follows directly from the definition of the Fourier transform of a continuous variable or the discrete Fourier transform of a discrete system.

2.9.5. Constructed Spatial 2DDFT Power Spectrum Features and Patterns

One of the natural ways of constructing a pattern from 2DDFT of 2D data (for example an image) is forming a column vector containing concatenated subsequent column of the power spectrum image (map) of the 2DDFT of 2D data. For $m \times n2D$ data the resulting pattern will have $m \times m$ elements.

In order to capture the spatial relation (a shape) of the components of the "power spectrum image" (map), a number of spatial features can be computed from the power spectrum array treated as an image.

We have assumed that the numerical characteristics (measures) of the shape of the spatial frequency spectrum, such as location, size, and orientation of peaks and entropy of the normalized spectrum in regions of spatial frequency, can be used as object features (pattern elements) suitable for recognition of an image. Some spatial features of the normalized "power spectrum" image require the computation for such an image of the covariance matrix and its eigenvalues and eigenvectors as well as principal components. Numerous subsequent spatial spectral features (computed in the frequency domain) can be extracted or created from the normalized "power spectrum image" $P(u, v)$:

$$P(u, v) = \frac{P(u, v)}{\sum_{u,v \neq 0} P(u, v)} \tag{251}$$

Energy of the major peak

$$f_1 = p(u_1, v_1) \times 100 \tag{252}$$

where u_1, v_1 are the frequency coordinates of the maximum peak of the normalized power spectrum. Here, f is a percentage of the total energy.

Laplacian of the major peak

$$\begin{aligned} f_2 &= \nabla^2 P(u_1, v_1) \\ &= P(u_1 + 1, v_1) + P(u_1 - 1, v_1) + P(u_1, v_1 + 1) + P(u_1, v_1 - 1) - 4P(u_1, v_1) \end{aligned} \tag{253}$$

Laplacian of the secondary peak

$$f_3 = \nabla^2 P(u_2, v_2) \tag{254}$$

where u_2, v_2 are the coordinates of the second largest peak in the $P(u, v)$ map.

Spread of the major peak
f_4 is the number of adjacent neighbors of u_1, v_1 with

$$P(u, v) \geq \frac{1}{2} k P(u_1, v_1) \tag{255}$$

where the neighbors are $u_1 \pm 1$, v_1, and u_1, $v_1 \pm 1$.

Squared frequency of the major peak in $P(u, v)$

$$f_5 = u_1^2 + v_1^2 \tag{256}$$

Relative orientation of the major and secondary peaks

$$f_6 = \left| \tan^{-1} \frac{v_1}{u_1} - \tan^{-1} \frac{v_2}{u_2} \right| \tag{257}$$

Isotropy of the normalized power spectrum $P(u, v)$

$$f_7 = \frac{|\delta_u - \delta_v|}{\left[(\delta_u - \delta_v)^2 - 4\delta_{uv}^2\right]^{\frac{1}{2}}} \tag{258}$$

where

$$\delta_u = \sum_u \sum_v u^2 p(u, v)$$
$$\delta_v = \sum_u \sum_v v^2 p(u, v)$$
$$\delta_{uv} = \sum_u \sum_v uvp(u, v)$$

Here, f_7 measures the elongation of the normalized power spectrum and is maximum for parallel line faces.

Circularity of the normalized power spectrum

$$f_8 = \frac{A_D}{A_C} \tag{259}$$

where

A_D = number of nonzero frequency components within a circle of radius $\sqrt{\lambda_1}$
A_C = number of distinct frequency components within a circle of radius $\sqrt{\lambda_1}$
λ_1 = maximum eigenvalue of the covariance matrix of $p(u, v)$

Major peak horizontal frequency

$$f_9 = u_1 \tag{260}$$

Major peak vertical frequency

$$f_{10} = v_1 \tag{261}$$

Secondary peak horizontal frequency

$$f_{11} = u_2 \tag{262}$$

Secondary peak vertical frequency

$$f_{12} = v_2 \tag{263}$$

Squared distance between the major and secondary peak

$$f_{13} = (u_1 - u_2)^2 + (v_1 - v_2)^2 \tag{264}$$

Principal component magnitude (squared)

$$f_{14} = \lambda_1 \tag{265}$$

Principal component direction

$$f_{15} = \cos^{-1}(\phi_1) \tag{266}$$

where $\boldsymbol{\phi} = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix}$ is a normalized eigenvector for eigenvalue λ_1

Ratio of the minor to major principal axis

$$f_{16} = \left(\frac{\lambda_2}{\lambda_1}\right)^{\frac{1}{2}} \tag{267}$$

where λ_2 is the minimum eigenvalue of the covariance matrix of $p(u, v)$

Moment of inertia, quadrant I

$$f_{17} = \sum_{u>0}\sum_{v>0}(u^2 - v^2)^{\frac{1}{2}}p(u, v) \tag{268}$$

Moment of inertia, quadrant II

$$f_{18} = \sum_{u<0}\sum_{v>0}(u^2 - v^2)^{\frac{1}{2}}p(u, v) \tag{269}$$

Here, in f_{17} and f_{18}, the power spectrum is normalized within quadrants I and II, respectively.

Moment ratio

$$f_{19} = \frac{f_{18}}{f_{17}} \tag{270}$$

Percentage energy, quadrant I

$$f_{20} = \sum_{u>0}\sum_{v>0}p(u, v) \tag{271}$$

Percentage energy, quadrant II

$$f_{21} = \sum_{u<0}\sum_{v>0}p(u, v) \tag{272}$$

Ratio of nonzero components

$$f_{22} = \frac{n_1}{n_2} \tag{273}$$

where n_i denotes a number of nonzero frequency components in quadrant i

Laplacian of the major peak phase

$$f_{23} = \nabla^2\phi(u_1, v_1) \tag{274}$$

Laplacian of the secondary peak phase

$$f_{24} = \nabla^2\phi(u_2, v_2) \tag{275}$$

Relative entropy of the normalized power spectrum (R_1)

$$f_{25} = \frac{\left[-\sum_{u,v\in R_1} P_1(u,v)\log P_1(u,v)\right]}{\log K_1} \tag{276}$$

where

$P_1(u,v) = \frac{P(u,v)}{\sum_{u,v\in R_i} P(u,v)}$
$K_i =$ number of distinct frequencies in R_i

$$R_i = \left\{u,\ v : \frac{i-1}{4}u_m < |u| < \frac{i}{4}u_m \quad \text{and} \quad \frac{i-1}{4}v_m < |v| < \frac{i}{4}v_m\right\}$$

where u_m, v_m are the maximum frequency components for the local spectrum

Relative entropy (R_2)

$$f_{26} = \frac{\left[-\sum_{u,v\in R_2} P_2(u,v)\log P_2(u,v)\right]}{\log K_2} \tag{277}$$

Relative entropy (R_3)

$$f_{27} = \frac{\left[-\sum_{u,v\in R_3} P_3(u,v)\log P_3(u,v)\right]}{\log K_3} \tag{278}$$

Relative entropy (R_4)

$$f_{28} = \frac{\left[-\sum_{u,v\in R_4} P_4(u,v)\log P_4(u,v)\right]}{\log K_4} \tag{279}$$

Histogram subpattern $\mathbf{x}_h$:

$$f_{29}, \cdots, f_{29} + n_h \times N.$$

For an $M \times N$ "image" of normalized $P(u,v)$, the following histogram features can be extracted and presented as the $n_h \times N$-element vector $\mathbf{x}_h$:

(a) For each column of an image (treated as a matrix), the column elements are binned into n_h equally spaced containers and the number of elements in each container is computed. The result forms a histogram matrix $\mathbf{H}_p$ of dimension $n_h \times N$.
(b) Now we can consider the histogram subpattern as an $n_h \times N$-element vector $\mathbf{x}_h$, obtained through columnwise concatenation of the matrix $\mathbf{H}_p$.

Magnitude of complex Zernike moments (defined later in the Chapter) of order from $(1,1)$ through (p,q) for normalized $P(u,v)$.

The 2DDFT provides a powerful spectral representation of images in the frequency domain, with significant predispositions toward features of image patterns used in image recognition. A spectral pattern of an image can be formed as concatenated columns of a "normalized power spectrum image." One can also form a spectral pattern as any subset of the spatial spectral features defined above.

2.10. Wavelets

Wavelets are another powerful technique for processing timeseries and images. The foundation of wavelets comes from orthogonality, function decomposition, and multiresolution approximation. **Wavelets** (wavelet analysis, the wavelet transform) provide representation (approximation) of a function (or a signal) by a fast-decaying oscillating waveform (known as the **mother wavelet**). This waveform can be scaled and translated in order to best match the function or input signal.

A **wavelet** is a special kind of oscillating waveform of substantially limited duration with an average value equal to zero. Wavelets can be considered to be one step ahead of the Fourier transform.

The Fourier transform decomposes a function or signal into sine waves with different frequencies (or, in other words, is the sum over all time of the signal $f(t)$ multiplied by a complex exponential). However, it does not localize these frequencies in time. The results of the Fourier transform are Fourier coefficients $F(\omega)$:

$$FT\, f(t) = F(\omega) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{+\infty} f(t) e^{-j\omega t}\, dt \tag{280}$$

(a spectrum) related to different frequencies. Multiplication of spectral coefficients by a sinusoid of corresponding frequency results in the reconstruction (an inverse transform) of the sinusoidal component of the original function or signal.

Wavelets break up a function or a signal into a shifted and scaled instance of the mother wavelet. The **continuous wavelet transform** is the sum over all time of a function or a signal multiplied by the shifted, scaled instance of the wavelet function. The results of the wavelet transform are coefficients C:

$$C(\text{position}, \text{scale}) = \int_{-\infty}^{+\infty} f(t)\Psi(\text{position}, \text{scale}, t)\, dt \tag{281}$$

which are a function of position and scale. In the **inverse wavelet transform**, multiplication of each coefficient by the corresponding shifted and scaled wavelet results in constituent wavelets of the original function or signal. Wavelets provide localization both in frequency and in time (or in space). A function $f(t)$ (or signal $x(k)$) can be more easily analyzed or described when expressed as a linear decomposition

$$f(t) = \sum_r a_r \psi_r(t) \tag{282}$$

where r is an integer index (for a finite or infinite sum). The $a_r \in \mathbb{R}$ are **expansion coefficients**, and the $\psi_r(t)$ are real-valued functions of t (the expansion set). For a unique expansion, the set of functions $\psi_r(t)$ is called a basis. Especially important expansions of a function can be obtained for the orthogonal basis when we have

$$< \psi_r(t), \psi_l(t) >= \int \psi_r(t)\, \psi_l(t)\, dt = 0, \;\; r \neq l \tag{283}$$

$$a_r =< f(t), \psi_r(t) >= \int f(t)\psi_r(t)\, dt \tag{284}$$

For the best-known Fourier transform, the orthogonal basis functions $\psi_k(t)$ are $\sin(k\omega t)$ and $\cos(k\omega t)$ function of $k\omega t$. This transformation maps a one-dimensional function of the continuous variable into one-dimensional sequence of coefficients.

The main difference between wavelets and the Fourier transform is that wavelets are localized in both time and frequency, whereas the Fourier transform is localized in frequency space. The Short-time Fourier Transform (STFT) could be called a prelude to wavelets, since it is also time and frequency localized. However, wavelets provide better tools for muliresolution in frequency and in time. For two-parameter wavelet transformation, with the wavelet expansion function $\psi_{j,k}(t)$ forming the orthogonal basis, we have

$$f(t) = D_{\psi}^{-1} \int \int \frac{1}{j^2} < f(t), \psi_{j,k} > \psi_{j,k} dj\, dk \tag{285}$$

and the corresponding discrete version

$$f(t) = \sum_{k} \sum_{j} a_{j,k} \psi_{j,k}(t) \tag{286}$$

where j and k are integer indices and D_{Ψ}^{-j} is the scaling factor. The set of coefficients $a_{j,k}(j, k \in \mathbb{Z})$ is the discrete wavelet transform $DW\, f(t)$ of a function $f(t)$.

The wavelet expansion maps a one-dimensional function into a two-dimensional array of coefficients (discrete wavelets transform), allowing localization of the signal in both time and frequency simultaneously.

Wavelets are defined by the wavelet function $\psi(t)$ (wavelet) and the scaling function $\phi(t)$ (also called the **father wavelet**) in the time domain. The wavelet function acts as a band-pass filter, and scaling it for each level halves its bandwidth, whereas scaling the function filters the lowest level of the transform and ensures that the entire spectrum is covered.

2.10.1. A Wavelet Function

The wavelet transform is a method of approximating a given function $f(t) \in L^2(\mathbb{R})$ or a signal (as a sequence of values) using the functions $\psi(t)$ and $\phi(t)$. The function $\psi(t)$ is a scalable approximation curve localized on a definite time (or space) interval. The function $\psi(t)$ is called a **mother function** (**wavelet function**, or **generating wavelet**). The mother wavelet must satisfy the following admissibility conditions:

$$c_{\psi} = \int_{0}^{\infty} \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < \infty$$

or

$$\int_{-\infty}^{\infty} \psi(t) dt = 0 \tag{287}$$

where $\Psi(\omega) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{\infty} \psi(t) e^{-j\omega t}\, dt$ is the Fourier transform of $\psi(t)$.

The second derivative of the Gaussian function is an example of the wavelet function:

$$\psi(t) = (1 - t^2) e^{-\frac{t^2}{2}} \tag{288}$$

Two-dimensional parametrization, with a dilation parameter a and a translation parameter b, yields the possibility of scaling and shifting the wavelet function over a certain time (or space) domain. Wavelets constitute a family of functions designed from **dilations** and **translations** of a single function – the mother wavelet. For signal expansion, the mother wavelet is a band-pass filter. Incorporating continuous variation of the dilation parameter a and the translation parameter b, we can find a family of continuous wavelets

$$\psi_{a,b}(t) = |a|^{-\frac{1}{2}} \psi\left(\frac{t-b}{a}\right), \quad a, b \in \mathbb{R}, \ a \geq 0 \tag{289}$$

where a and b may vary over $\mathbb{R}$. This is a band-pass filter with two parameters: the dilation and translation parameters a and b. The dilation parameter a determinates the frequency of information (interpreted as changing the bandwidth of the filter). Varying the dilation parameter a generates different spectra of $\psi_{a,b}(t)$. For a smaller dilation parameter a, the wavelet is narrowed; for increased a, the wavelet stretches in time. The translation parameter b determinates the time information (location in time) or space information (location in space). It localizes the wavelet curve on a specific time interval with center at $t = b$. For this reason, wavelets are time (or space) and frequency localized. These properties are essential for wavelets. Wavelets functions have finite energy, and they are locally concentrated.

2.10.2. Continous Wavelet Transform

For a given function $f(t)$, the continuous wavelet transform is defined as

$$WT\,f(a,b) = |a|^{-\frac{1}{2}} \int_{-\infty}^{\infty} f(t)\overline{\psi\left(\frac{t-b}{a}\right)}\,dt \tag{290}$$

where a and b are dilation and translation parameters, and $\psi(\cdot)$ is a wavelet function. One can see that the wavelet transform is the scalar product of two functions, namely, $f(t)$ and $\psi_{a,b}$ in $L^2(\mathbb{R})$:

$$WT\,f(a,b) =< f(t), \psi_{a,b}(t) > \tag{291}$$

A function can be characterized by its wavelet coefficients $< f(t), \psi_{a,b}(t) >$. A function $f(t)$ can be reconstructed from its wavelet transform $WT\,f(a,b)$ by the inverse transform

$$f(t) = \frac{1}{D_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{a^2} WT\,f(a,b)\psi_{a,b}(t)\,da\,db \tag{292}$$

where D_ψ is a scaling factor representing an average energy of the wavelet function

$$D_\psi = 2\pi \int_{-\infty}^{\infty} \frac{|\psi(\omega)|^2}{|\omega|}\,d\omega \tag{293}$$

2.10.3. Discrete Wavelet Transform

In the discrete case, parameters a and b take only discrete values. The **discrete wavelets** are obtained, after sampling parameters a and b as $a = a_0^j$ and $b = kab_0 = ka_0^j b_0$ (where $j, k \in \mathbb{Z}$, $i, j = \mp 1, \mp 2, \cdots$)), as

$$\psi_{j,k}(t) = |a_0|^{-\frac{j}{2}} \psi(a_0^{-j}t - kb_0), \quad j, k \in \mathbb{Z} \tag{294}$$

where $\psi_{j,k}(t)$ constitutes basis for $L^2(\mathbb{R})$. The selection of a_0 and b_0 depends on an application.

The discrete wavelet transform is defined for sampled parameters by the equation

$$DWT\,f(j,k) = |a_0|^{-\frac{j}{2}} \int_{-\infty}^{\infty} f(t)\psi(a_0^{-j}t - kb_0)\,dt \tag{295}$$

For $a_0 = 2$ $(a = 2^j)$ and $b_0 = 1$ $(b = 2^j k)$, functions $\psi_{j,k}(t)$ form an orthogonal wavelet base for $L^2(\mathbb{R})$:

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}} \psi(2^{-j}t - k) \tag{296}$$

For a function $f(t) \in L^2(\mathbb{R})$, the discrete wavelet expansion of $f(t)$ is represented by

$$f(t) = \sum_j \sum_k b_{j,k} \psi_{j,k}(t) \tag{297}$$

where the expansion coefficients $b_{j,k}$ are the inner product of $f(t)$ and $\psi_{j,k}(t)$, i.e.,

$$b_{j,k} =< \psi_{j,k}(t), f(t) > \tag{298}$$

2.10.4. Multiresolution Analysis of a Function

A continuous wavelet transform is redundant. This redundancy can be avoided in a discrete transform by using the fast wavelet transform. The idea of multiresolution analysis of a function $f(t)$ is to construct a ladder of close subspaces of $\mathbb{Z}$, $\{V_n : n \in \mathbb{Z}\}$ (nested subspaces), for representing functions with successive resolution. This result can be realized with a basis **scaling function** $\phi(t)$ in $L^2(\mathbb{R})$ (low pass or smoothing function):

$$\phi_k(t) = \phi(t-k), \;\; k \in \mathbb{Z} \tag{299}$$

with a spanned subspace V_0 of $L^2(\mathbb{R})$ for this function for all integers from $-\infty$ to ∞, and with

$$f(t) = \sum_k a_k \phi_k(t) \;\text{ forany }\; f(t) \in V_0 \tag{300}$$

Subspaces in $L^2(\mathbb{R})$ are

$$\{0\} \subset \cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset L^2(\mathbb{R}) \tag{301}$$

These representations satisfy the following conditions:

1. A subspace V_n is contained in V_{n+1}:

$$V_n \subset V_{n+1}, \;\; n \in \mathbb{Z} \tag{302}$$

   A space containing high resolution signals will also contain lower resolution signals.
2. An intersection of all subspaces V_n for all $n \in \mathbb{Z}$ is null $\bigcap V_n = \{0\}$, and the union of subspaces is $\overline{\bigcup V_n}$ is dense in $L^2(\mathbb{R})$.
3. A subspace V_n is invariant under integral translations

$$f(t) \in V_n \Leftrightarrow f(t-k) \in V_n, \;\; k \in \mathbb{Z} \tag{303}$$

4. There exists a scaling function $\phi(t) \in V_0$ that, with its translated version $\phi(t-k)$, forms an orthonormal basis in V_0 so that following conditions hold:

$$\int \phi(t)\, dt = 1, \;\; \text{normalization}$$

$$\int \phi(t)\overline{\phi(t-k)}\, dt = \delta(k), \;\; \text{orthogonality} \tag{304}$$

The subspaces satisfy the natural scaling condition

$$f(t) \in V_n \Leftrightarrow f(2t) \in V_{n+1} \tag{305}$$

which means that elements in a space V_{n+1} are scaled version of elements in the next subspace. The scaling function can also be dilated and translated (similarly to the wavelet function):

$$\phi_{j,k} = |a|^{-\frac{1}{2}} \phi(a^{-1} - k), \;\; a = 2 \tag{306}$$

However, the scaling function is not orthogonal to its dilation. Nesting conditions show that $\phi(t)$ can be expressed in terms of the weighted sum of shifted $\phi(\cdot)$:

$$\phi(t) = \sum_k h_k \phi_{-1,k}(t) \tag{307}$$

where h_k $(k = 0, 1, \cdots, K-1)$ is a set of scaling coefficients $h_k =< \phi(t), \phi_{-1,k}(t) >$, and $\sum_{k\in\mathbb{Z}} |h_k|^2 = 1$:

$$\phi_{-1,k} = 2^{-\frac{1}{2}} \phi(2^{-1}t - k) \tag{308}$$

2.10.5. Construction of the Wavelet Function from the Scaling Function

Based on an idea of multiresolution and the property of the orthonormal complement W_i of subspaces V_i (contained in V_{i-1}), we find that wavelet function $\psi(t)$ can be expressed as a linear combination of the basis scaling functions $\phi_{-1,k}(t)$:

$$\psi(t) = \sum_k g_k \phi_{-1,k}(t) \tag{309}$$

where $g_k = (-1)^k h_{-k+1}$ or, in matrix notation,

$$\psi(t) = \boldsymbol{\phi}_{-1}\mathbf{g} \tag{310}$$

with $\mathbf{g} = [(-1)^{K-1} h_{-K+2}, \cdots, h_{-1}, h_0, h_1]^T$.

The coefficients h_k satisfy the orthonormality condition $\sum_{k=0}^{K-1} h_k \overline{h_{k+2n}} = \delta(n)$ and

$$\sum_{k=0}^{K-1} h_k = \sqrt{2}, \quad \sum_k (-1)^k h_k = 0 \tag{311}$$

A function $f(t) \in L^2(\mathbb{R})$ can be analyzed using multiresolution idea with scaling and wavelet functions $\phi_{j,k}(t)$ and $\psi_{j,k}(t)$ $(j, k = -1, 0, 1, 2, \cdots)$, respectively. These functions constitute the orthonormal bases of the approximation spaces W_j and V_j. Based on functions $\phi_{-1,k}(k)$ and $\psi_{-1,k}$, one can decompose a space V_{-1} into two subspaces V_0 and W_0. Similarly, the subspace V_0 can be decomposed into V_1 and W_1, and so forth. In general, we have the following decomposition:

$$\begin{aligned} &\phi_{j,k}(t) \;\; V_j \to V_{j+1} \\ &\psi_{j,k}(t) \;\; V_j \to W_{j+1} \end{aligned} \tag{312}$$

Each expanded subspace has a different resolution specified by index j. Thus, we may have the multiresolution expansion of a given space, with the resulting wavelet expansion of a function $f(t)$ in this space.

For signals, scaling functions encode the low spatial (or time) frequency information, whereas wavelets encode signals in different frequency bands to a certain level of frequency.

Let us consider a function uniquely determined by N discrete samples

$$\{f(1), f(2), \cdots, f(N)\}$$

It can be shown that it is possible to expand this function as a series of N orthogonal basis functions.

2.10.6. Discrete Wavelet Transform: Wavelet and Scaling Functions

For discrete parameters a and b, a **discrete wavelets transformation** decomposes a function into an expansion (using dilations and translations) of two functions: a scaling function $\phi(t)$ and a wavelet function $\psi(t)$. The basis sets for a scaling function (nonnormalized) are

$$\phi_{L,k}(t) = \phi(2^L t - k), \quad k = 1, 2, \cdots, K_L, \quad K_L = N2^{-L} \tag{313}$$

where L is an expansion level, and for the wavelet function

$$\psi_{j,k}(t) = \psi(2^j t - k), \quad j = 1, 2, \cdots, L; \quad k = 1, 2, \cdots, K; \quad K = N2^{-j} \tag{314}$$

where the level of expansion L satisfies $0 < L \le \log_2(N)$.

An L-level discrete wavelet transform of function $f(t)$ described by N samples contains:

1. a set of parameters $\{a_{L,k}\}$ defined by the inner products of $f(t)$ with $N2^{-j}$ translations of the scaling function $\phi(t)$ at L different widths

$$\{a_{L,k}(t)\} = \{< f(t), \phi_{L,k}(t) >; \quad k = 1, 2, \cdots, K_L, \quad K_L = N2^{-L}\} \tag{315}$$

2. a set of parameters $\{b_{j,k}\}$ defined by the inner products of $f(t)$ with $N2^{-L}$ translations of the wavelet function $\psi_{j,k}(t)$ at a single width

$$\{b_{j,k}(t)\} = \{< f(t), \psi_{j,k}(2^j t - k) >; \quad j = 1, 2, \cdots, L, \\ k = 1, 2, \cdots, K; \quad K = N2^{-j}\}$$

The reconstruction of a function $f(t)$ based on wavelet transform coefficients can be obtained by the inverse transform

$$f(t) = \sum_{k}^{K_L} a_{L,k}\phi_{L,k}(t) + \sum_{j}^{L}\sum_{k}^{K} b_{j,k}\psi_{j,k}(t) \tag{316}$$

The number of parameters of L-level wavelet transform is equal to

$$\sum_{j}^{L} 2^{-j} + N2^{-L} = N(1 - 2^L + 2^L) = N \tag{317}$$

which is the same as the corresponding number of coefficients of a Fourier transform.

2.10.7. Haar Wavelets

One of the simplest orthogonal wavelets is generated from the Haar scaling function and wavelet. The **Haar transform** uses square pulses to approximate the original function. The basis functions for Haar wavelets at some level all look like a unit pulse, shifted along the x-axis. Haar scales are all of unit pulses.

The Haar wavelet is defined as follows

$$\psi(t) = \begin{cases} 1, & \text{if} \quad t \in [0, 0.5) \\ -1, & \text{if} \quad t \in [0.5, 1) \\ 0, & \text{otherwise} \end{cases} \tag{318}$$

The dilations and translations of the Haar wavelet function form an orthogonal wavelet base for $L^2(\mathbb{R})$. The **mother Haar wavelets** are defined as

$$\psi_{j,k}(t) = \psi(2^j t - k), \quad j, k \in \mathbb{Z} \tag{319}$$

The Haar scaling function $\phi(t)$ is the unit-width function $\phi(t)$

$$\phi(t) = \begin{cases} 1, & \text{if} \quad 0 \le t \le 1, \\ 0, & \text{otherwise} \end{cases} \tag{320}$$

Figure 7.10 shows the Haar scaling and wavelet functions.

We can easily see that the Haar scaling function $\phi(t)$ can be constructed using $\phi(2t)$:

$$\phi(t) = \phi(2t) - \phi(2t - 1) \tag{321}$$

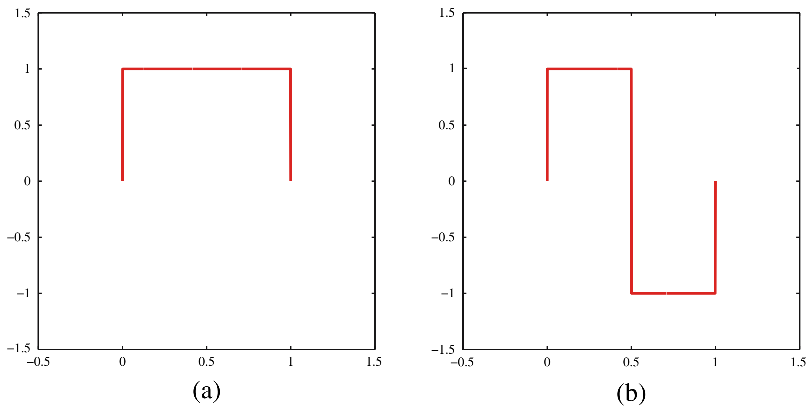

Figure 7.10. Haar scaling (a) and Haar wavelet (b) functions.

The Haar decomposition of a function $f(t) \in L^2(\mathbb{R})$ or finite-dimensional vector for a signal in the time domain can be expressed as

$$f(t) = a_{0,0}\phi_{0,0}(t) + \sum_{j=1}^{d}\sum_{k=0}^{2^{j-1}} b_{j,k}\psi_{j,k}(t) \tag{322}$$

where $f(t) \in L^2[0, 1)$, $a_{0,0}$ is the parameter, $\phi_{0,0}(t)$ is a scale function on the interval [0,1), and $\psi_{j,k}(t)$ is a set of wavelets with different resolutions. Due to the property of local analysis, the time interval of a function is not [0,1); the desired interval can be shifted to [0,1) to get the expected results. For the function $f(t) \in L^2(\mathbb{R})$, the discrete wavelet expansion of $f(t)$ is represented as

$$f(t) = a_{0,0}\phi_{0,0}(t) + \sum_{j=1}^{d}\sum_{k=0}^{2^{j-1}} b_{j,k}\psi_{j,k}(t) = a_{0,0}\phi_{0,0}(t) + \mathbf{b}\boldsymbol{\psi}_{(j)}(t) \tag{323}$$

where $\phi_{0.0}(t)$ is a scale function on interval [0,1), $\psi_{j,k}(t)$ is the set of wavelets with different resolution, $\mathbf{b}$ is the Haar wavelet coefficient vector, and $\boldsymbol{\psi}_j$ is a set of Haar wavelets, $\psi_{j,k}(t)$, which is chosen according to necessity. The two vectors $\mathbf{b}$ and $\boldsymbol{\psi}_j$ are defined by

$$\begin{aligned}\mathbf{b} &= [b_0, b_1, \cdots, b_{j-1}]^T \\ \boldsymbol{\psi}_j &= [\psi_0, \psi_1, \cdots, \psi_{j-1}]^T\end{aligned} \tag{324}$$

where ψ_i, $(i = 0, 1, \cdots, j-1)$, is some $\psi_{j,k}(t)$.

The Haar integral operational matrix $\mathbf{P}$ is given as

$$\int_{t_0}^{t} \boldsymbol{\psi}_{(j)}(t)\,dt = \mathbf{P}\boldsymbol{\psi}_{(j)}(t) \tag{325}$$

The coefficients $p(i, j)$ of $\mathbf{P}$ can be found numerically as the discrete wavelet expansion

$$\mathbf{P}(i, j) = < \int_{t_0}^{t} \psi_i dt,\ \psi_j > \tag{326}$$

2.10.8. Two-dimensional Wavelets

In general, the **two-dimensional wavelet transform** can be realized by successively applying the one-dimensional wavelet transform to data in every dimension. First, the rows are transformed by using a one-dimensional transform, and a similar transformation is provided for all columns of the intermediate results.

For a two-dimensional wavelet expansion of the two-dimensional function $f(t_1, t_2)$, we have to define a wavelet function of two variables t_1 and t_2 $\psi(t_1, t_2)$. This function should satisfy the following condition:

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\Psi(vt_1, vt_2)|^2}{|v|}\, dv < \infty \tag{327}$$

where Ψ denotes the Fourier transform of ψ.

The two-variable wavelet function can be defined as a product of two one-dimensional mother (generating) wavelets $\psi(t_1)$ and $\psi(t_2)$:

$$\psi(t_1, t_2) = \psi(t_1)\psi(t_2) \tag{328}$$

With dilation and translation parameters, the two-dimensional wavelet function is defined as

$$\psi_{(a_1,a_2),(b_1,b_2)}(t_1, t_2) = \psi_{(a_1,b_1)}(t_1)\psi_{(a_2,b_2)}(t_2) \tag{329}$$

where $\psi_{a,b}(t) = |a|^{-\frac{1}{2}}\psi(\frac{t-b}{a})$. We can also write

$$\psi_{(a_1,a_2),(b_1,b_2)}(t_1, t_2) = \frac{1}{\sqrt{|a_1 a_2|}}\psi_{(a_1,b_1)}\left(\frac{t_1 - b_1}{a_1}\right)\psi_{(a_2,b_2)}\left(\frac{t_2 - b_2}{a_2}\right) \tag{330}$$

Assuming that $a_1 = a_2$ the **two-dimensional continuous wavelet expansion** of the two-variable function $f(t_1, t_2)$ can be expressed as

$$TWT\, f((a), (b_1, b_2)) = |a|^{-1}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t_1, t_2)\psi_{(a),(b_1,b_2)}(t_1, t_2)\, dt_1\, dt_2 \tag{331}$$

The **inverse of the continuous wavelet transform** is defined as

$$f(t_1, t_2) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{a^4}\, TWT\, f((a), (b_1, b_2))\psi_{(a),(b_1,b_2)}(t_1, t_2)\, da\, db_1\, db_2 \tag{332}$$

For the discretized parameters a, b_1, and b_2,

$$a = a_0^j,\ \ b_1 = k_1 a b_{1,0},\ \ b_2 = k_2 a b_{2,0} \tag{333}$$

the two-dimensional discrete wavelet expansion can be written as

$$f(t_1, t_2) = \sum_{k_1}\sum_{k_2} a_{L,k_1,k_2}\phi_{L,k_1,k_2}(t_1, t_2) + \sum_{i=H,D,V}\sum_{j=1}^{L}\sum_{k_1}\sum_{k_2} b_{j,k_1,k_2}\psi^i_{j,k_1,k_2}(t_1, t_2) \tag{334}$$

For a two-dimensional grid of $2^n \times 2^n$ values (for example, image pixels) and for discrete parameters $a_1 = 2^{n-j_1}$, $a_2 = 2^{n-j_2}$, $b_1 = 2^{n-j_1}k_1$, $b_2 = 2^{n-j_2}k_2$, with integer values for j_1, j_2, k_1, and k_2, the 2D discrete wavelet function can be defined as

$$\psi_{j_1,j_2,k_1,k_2}(t_1, t_2) = 2^{\frac{(j_1+j_2)}{2}-n}\psi(2^{j_1-n}t_1 - k_1)\psi(2^{j_2-n}t_2 - k_2) \tag{335}$$

where j_1, j_2, k_1, and k_2 are the dilation and the translation coefficients for each variable, satisfying the conditions

$$0 \leq j_1,\ j_2 \leq n-1,\ 0 \leq k_1 \leq 2^{j_1}-1,\ 0 \leq k_2 \leq 2^{j_2}-1 \tag{336}$$

The resolution level is $j = \frac{j_1+k_1}{2}$ and corresponds to 2^{n-j}.

Additionally, defining the scaling function

$$\phi_{j_1,j_2,k_1,k_2}(t_1, t_2) = 2^{\frac{j_1+j_2}{2}-n}\phi(2^{j_1-n}t_1 - k_1)\,\phi(2^{j_2-n}t_2 - k_2) \tag{337}$$

allows us to define a complete basis to reconstruct a discrete function $f(t_1, t_2)$ (for example, a discrete image):

$$\begin{aligned}
\phi_{0,0,0,0}(t_1, t_2) &= 2^{-n}\phi(2^{-n}t_1)\phi(2^{-n}t_2) \\
\gamma^H_{0,j_2,0,k_2}(t_1, t_2) &= 2^{\frac{j_2}{2}-n}\phi(2^{-n}t_1)\psi(2^{j_2-n}t_2 - k_2) \\
\gamma^V_{j_1,0,k_1,0}(t_1, t_2) &= 2^{\frac{j_1}{2}-n}\psi(2^{j_1-n}t_1 - k_1)\phi(2^{-n}t_2)
\end{aligned} \tag{338}$$

The discrete bases satisfy orthonormality conditions

$$< \lambda_{j_1,j_2,k_1,k_2}, \lambda_{j_1',j_2',k_1',k_2'} >= \delta_{j_1,j_1'}\delta_{j_2,j_2'}\delta_{k_1,k_1'}\delta_{k_2,k_2'} \tag{339}$$

where $\lambda_{\cdot,\cdot,\cdot,\cdot}$ denotes any of the previous orthonormal bases. The 2D discrete wavelet coefficients are defined as

$$2DWT\ w_{j_1,j_2,k_1,k_2} = \int\int f(t_1, t_2)\lambda_{j_1,j_2,k_1,k_2}\,dt_1\,dt_2 \tag{340}$$

where $\lambda_{j_1,j_2,k_1,k_2}$ denotes any of the previously defined orthonormal bases. These coefficients can be formed as the Haar wavelet coefficient matrix **P**.

An **inverse discrete wavelet reconstruction** (an image reconstruction) can be described by the following expression:

$$f(t_1, t_2) = \sum_{j_1}\sum_{j_2}\sum_{k_1}\sum_{k_2} w_{j_1,j_2,k_1,k_2}\lambda_{j_1,j_2,k_1,k_2}(t_1, t_2) \tag{341}$$

Wavelet Patterns. Patterns, representing recognition objects (time-series or images), can be formed based on the coefficient matrices of certain level of wavelets transform. One can constitute a pattern through concatenation of subsequent parameter matrices rows as one pattern.

2.11. Zernike Moments

A robust pattern recognition system must be able to recognize an image (or an object within an image) regardless of its orientation, size, or position. In other words, **rotation-**, **scale-**, and **translation-invariance** are desired properties for extracted features. For example, as shown in Figure 7.11, all the images should be recognized as "8." In this section, we will introduce moments and complex Zernike moments for robust feature extraction from images. In statistics, the concept of **moments** is used extensively. Moments were first introduced for two-dimensional pattern recognition in the early 1960s. However, the recovery of an image from these moments is quite difficult and computationally expensive.

Major extension of moments has been provided through introduction of **Zernike moments** by using the idea of orthogonal moment invariants and the theory of orthogonal polynomials. This

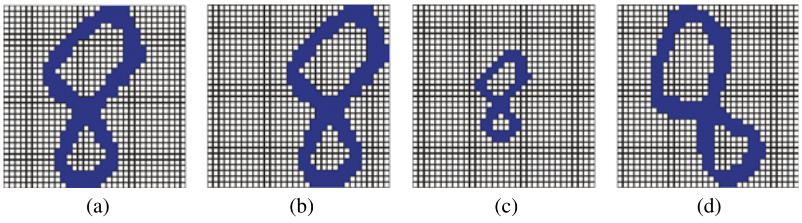

Figure 7.11. The image of character "8" (a) and its translated (b), scaled (c), and rotated (d) versions.

approach allows moment invariants to be constructed to an arbitrarily high order. In addition, Zernike moments are also rotation – invariant. Rotating the image does not change the magnitudes of the moments. Another property of Zernike moments is the simplicity of image reconstruction.

Studies have shown that orthogonal moments including Zernike moments are better than other types of moments in terms of information redundancy and image representation. Since Zernike moments are only rotation invariant, to obtain scale and translation invariance, an image must be normalized via **image normalization**. In order to understand the image normalization process, in the next section we briefly describe three basic image processing operations: **translation**, **scaling**, and **rotation**.

2.11.1. An Image Description

A computer image is a collection of **pixels** in 2D coordinate space, with the horizontal axis usually labeled x and the vertical axis usually labeled y. A gray-scale spatial domain image can be defined as

$$\{f(x, y) \in \{0, 1, \cdots, 255\} : \; x = 0, 1, \cdots, M-1; \; y = 0, 1, \cdots, N-1\} \tag{342}$$

and a binary spatial domain image as

$$\{f(x, y) \in \{0, 1\} : \; x = 0, 1, \cdots, M-1; \; y = 0, 1, \cdots, N-1\} \tag{343}$$

where x is the column index, y is the row index, M is the number of columns, N is the number of rows, and $f(x, y)$ is the pixel value at location (x, y).

2.11.2. Basic Image Transformations

Let f denote the original image and f^t the transformed image. **Translation**,

$$f^t(x + x', y + y') = f(x, y) \tag{344}$$

is a transformation that allows an image to be changed in position the along x-axis by x' and along the y-axis by y'.

The operation of **scaling**,

$$f^s(x, y) = f(x/a_x, y/a_y) \tag{345}$$

where a_x and a_y are scaling factors and a_x, $a_y > 0$, allows an image to be changed in size. If $a_x/a_y < 1$, the image is shrunk along the x-axis / y-axis. Similarly, if $a_x/a_y > 1$, the image is enlarged along x-axis / y-axis.

The **rotation** operation,

$$f^r(x, y) = f^r(\rho, \theta) = f(\rho, \theta - \alpha) \tag{346}$$

allows an image to be rotated about its center point through any arbitrarily specified angle. The angle of rotation is counterclockwise. Let f^r denote a rotated image and f the original image. The relationship between the rotated and original image can be explained as follows: ρ is the length of the vector from the origin to the (x, y) pixel, θ is the angle between ρ and the x-axis in the counterclockwise direction, and α is the angle of rotation in the counterclockwise direction.

2.11.3. Image Normalization

Zernike moments are only rotationally invariant, but the images might have scale and translation differences. Therefore, prior to the extraction of Zernike moments, the images should be normalized with respect to scaling and translation.

Translation invariance can be achieved by moving the origin to the center (centroid) of an image. To obtain the centroid location of an image, **general moments** (or **regular moments**) can be used. General moments are defined as

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y)\, dxdy \tag{347}$$

where m_{pq} is the $(p+q)^{\text{th}}$ order moment of the continuous image function $f(x, y)$. For digital images, the integrals can be replaced by summations. Given a two-dimensional $M \times N$ image, the moment m_{pq} is given by

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y) \tag{348}$$

To keep the dynamic range of m_{pq} consistent for any size of image, the $M \times N$ image plane should be first mapped onto a square defined by $x \in [-1, +1]$, $y \in [-1, +1]$. This mapping implies that grid locations will no longer be integers but will have real values in the $[-1, +1]$ range. This changes the definition of m_{pq} to

$$m_{pq} = \sum_{x=-1}^{+1} \sum_{y=-1}^{+1} x^p y^q f(x, y) \tag{349}$$

Hence, we can find the centroid location of an image by the general moment. According to Zernike, the coordinates of the image centroid $(\bar{x}, \bar{y})$ are

$$\bar{x} = \frac{m_{10}}{m_{00}}; \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{350}$$

To achieve translation invariance, one can transform the image into a new one whose first-order moments, m_{01} and m_{10}, are both equal to zero. This can be done by transforming the original image into the $f(x+\bar{x}, y+\bar{y})$ image, where $\bar{x}$ and $\bar{y}$ are centroid locations of an original image computed as in Equation (350). In other words, we need to move the origin of the coordinates to the image centroid. Let $g(x, y)$ represent the translated image; then the new image function becomes

$$g(x, y) = f(x+\bar{x}, y+\bar{y}) \tag{351}$$

Scale invariance is accomplished by enlarging or reducing each image such that its zero-order moment, m_{00}, is set equal to a predetermined value β. That is, we can achieve this outcome by transforming the original image function $f(x, y)$ into a new function $f(x/a, y/a)$, with scaling factor a, where

$$a = \sqrt{\frac{\beta}{m_{00}}} \tag{352}$$

Note that in the case of binary images, m_{00} is equal to the total number of object pixels in the image. β is chosen based on the size of the image and the object in the image. For example, one can choose $\beta = 800$ for 64×64 binary images of characters "A" to "Z"; consequently, the lower case "a" to "z" might need a lower value of β than that of upper case "A" to "Z". Another choice – for example, for 32×32 images of digits "0" to "9" – could be $\beta = 256$.

Let $g(x, y)$ be the scaled image. After scale normalization, we will obtain

$$g(x, y) = f\left(\frac{x}{a}, \frac{y}{a}\right) \tag{353}$$

2.11.4. Translation and Scale Normalization

In summary, an image function can be normalized with respect to scale and translation by transforming it into $g(x, y)$, where

$$g(x, y) = f\left(\frac{x}{a} + \bar{x}, \frac{y}{a} + \bar{y}\right) \tag{354}$$

with $(\bar{x}, \bar{y})$ being the centroid of $f(x, y)$ and $a = \sqrt{\frac{\beta}{m_{00}}}$, with β a predetermined value. However, $(x/a + \bar{x}, y/a + \bar{y})$ might not correspond to a grid location. To solve this problem, an interpolation method known as **nearest neighborhood approximation** can be used. In this technique, four nearest pixels are used and the fractional address of a pixel is truncated to the nearest integer pixel address.

2.11.5. Zernike Moments

As a result of image normalization, an image has obtained translation and scale invariance. To achieve rotational invariance, complex **Zernike moments** are used.

Zernike introduced a set of complex polynomials that form a complete orthogonal set over the interior of a unit circle, i.e., $x^2 + y^2 = 1$. Let the set of these polynomials be denoted by $V_{nl}(x, y)$. The form of these polynomials is

$$V_{nl}(x, y) = V_{nl}(\rho \sin\theta, \rho\cos\theta) = V_{nl}(\rho, \theta) = R_{nl}(\rho)\exp(il\theta), \tag{355}$$

where n is a positive integer or zero; l is a positive or negative integer, subject to the constraints $n - |l| =$ even; $|l| \leq n$, i is the complex number $i = \sqrt{-1}$; ρ is the length of the vector from the origin to the (x, y) pixel; and θ is the angle between vector ρ and the x-axis in the counterclockwise direction.

In the following, we assume that the notations $[V_{nl}(x, y)]^*$ and $V_{nl}^*(x, y)$ are equivalent where the symbol * denotes the complex conjugate. Radial polynomials $R_{nl}(\rho)$ are defined as

$$R_{nl}(\rho) = \sum_{s=0}^{\frac{n-|l|}{2}} \frac{(-1)^s[(n-s)!]\rho^{n-2s}}{s!(\frac{n+|l|}{2} - s)!(\frac{n-|l|}{2} - s)!} \tag{356}$$

We note that $R_{n,-l}(\rho) = R_{nm}(\rho)$. These polynomials are orthogonal and satisfy the equality

$$\int\int_{x^2+y^2\leq 1} [V_{nl}(x, y)]^* V_{jk}(x, y)\, dxdy = \frac{\pi}{n+1}\delta_{nj}\delta_{lk}$$

where the meaning of Kronecker delta is as follows:

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases} \tag{357}$$

where the symbol $*$ denotes the complex conjugate. Zernike moments are projection of the image function onto these orthogonal basis functions. The Zernike moments of order n, with repetition l for a continuous image function $f(x, y)$, are

$$A_{nl} = \frac{n+1}{\pi}\int\int_{x^2+y^2\leq 1} f(x, y)[V_{nl}(\rho, \theta)]^*\, dxdy = (A_{n,-l})^* \tag{358}$$

For a digital image,

$$A_{nl} = \frac{n+1}{\pi}\sum_x\sum_y f(x, y)V^*_{nl}(\rho, \theta);\;\; x^2 + y^2 \leq 1 \tag{359}$$

The real Zernike moments for $l \neq 0$ are

$$\begin{bmatrix} C_{nl} \\ S_{nl} \end{bmatrix} = \frac{2n+2}{\pi}\int\int_{x^2+y^2\leq 1} f(x, y)\, R_{nl}(\rho)\begin{bmatrix} \cos l\theta \\ -\sin l\theta \end{bmatrix} dxdy \tag{360}$$

or

$$\begin{aligned} C_{nl} &= 2\ Re\ (A_{nl}) \\ &= \frac{2n+2}{\pi}\int\int_{x^2+y^2\leq 1} f(x, y)\, R_{nl}(\rho)\, \cos l\theta\, dxdy \end{aligned} \tag{361}$$

$$\begin{aligned} S_{nl} &= -2\ Im\,(A_{nl}) \\ &= \frac{-2n-2}{\pi}\int\int_{x^2+y^2\leq 1} f(x, y)\, R_{nl}(\rho)\, \sin l\theta\, dxdy \end{aligned} \tag{362}$$

and for $l = 0$

$$\begin{aligned} C_{n0} &= A_{n0} = \frac{1}{\pi}\int\int_{x^2+y^2\leq 1} f(x, y)R_{n0}(\rho)\, dxdy \\ S_{n0} &= 0 \end{aligned} \tag{363}$$

For a digital image, when $l \neq 0$,

$$\begin{bmatrix} C_{nl} \\ S_{nl} \end{bmatrix} = \frac{2n+2}{\pi}\sum_{x=-1}^{+1}\sum_{y=-1}^{+1} f(x, y)R_{nl}(\rho)\begin{bmatrix} \cos l\theta \\ -\sin l\theta \end{bmatrix} \tag{364}$$

and when $l = 0$,

$$\begin{aligned} C_{n0} &= A_{n0} = \frac{1}{\pi}\sum_{x=-1}^{+1}\sum_{x=-1}^{+1} f(x, y)R_{n0}(\rho) \\ S_{n0} &= 0 \end{aligned} \tag{365}$$

The connection between real and complex Zernike moments is ($l > 0$)

- $C_{nl} = 2Re(A_{nl})$
- $S_{nl} = -2Im(A_{nl})$
- $A_{nl} = \frac{(C_{nl} - iS_{nl})}{2} = (A_{n,-l})^*$

To compute the Zernike moments of a given image, the center of the image is taken as the origin, and pixel coordinates are mapped to the range of the unit circle, i.e., $x^2 + y^2 \leq 1$. Those pixels that fall outside the unit circle are not used in the computation.

The image **translation** and **scale normalization** processes affect two of the Zernike features; namely, $|A_{00}|$ and $|A_{11}|$, the magnitude of Zernike moments A_{00} and A_{11}.

1. $|A_{00}|$ is going to be the same for all images.

$$C_{00} = \frac{2}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y) R_{00}(\rho)\, dxdy = \frac{2}{\pi} m_{00}; \;\; S_{00} = 0 \tag{366}$$

Since $m_{00} = \beta$,

$$|A_{00}| = \left| \left(\frac{C_{00}}{2} \right) - i \left(\frac{S_{00}}{2} \right) \right| = \frac{\beta}{\pi} \tag{367}$$

2. $|A_{11}|$ is equal to zero.

$$\begin{aligned} C_{11} &= \frac{4}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y) R_{11}(\rho) \cos\theta\, dxdy \\ &= \frac{4}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y)\ \rho\ \cos\theta\, dxdy \\ &= \frac{4}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y)\, x\, dxdy \\ &= \frac{4}{\pi} m_{10} \end{aligned} \tag{368}$$

and

$$\begin{aligned} S_{11} &= \frac{4}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y) R_{11}(\rho) \sin\theta\, dxdy \\ &= \frac{4}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y)\, \rho \sin\theta\, dxdy \\ &= \frac{4}{\pi} \int \int_{x^2+y^2 \leq 1} g(x, y)\, y\, dxdy \\ &= \frac{4}{\pi} m_{01}, \end{aligned} \tag{369}$$

Since $m_{10} = m_{01} = 0$ for all **normalized** images, then

$$|A_{11}| = \left| \left(\frac{C_{11}}{2} \right) - i \left(\frac{S_{11}}{2} \right) \right| = 0 \tag{370}$$

Therefore, $|A_{00}|$ and $|A_{11}|$ are not taken as features utilized in the classification.

Image reconstruction (**inverse transform**) from Zernike moments can be done in a simple way. Suppose we know all moments A_{nl} (C_{nl} and S_{nl}) of an image $f(x, y)$ up to a given order $n_{\max}$. We can reconstruct an image $\hat{f}$ by

$$\hat{f}(x, y) = \sum_{n=0}^{n_{\max}} \sum_{l} A_{nl} V_{nl}(\rho, \theta) \tag{371}$$

where $n - |l| =$ even and $|l| \leq n$.

Since it is easier to work with real-valued functions, we can expand Equation (371) to

$$\hat{f}(x, y) = \sum_{n=0}^{n_{\max}} \sum_{l} (C_{nl} \cos l\theta + S_{nl} \sin l\theta) R_{nl}(\rho) \tag{372}$$

where $n - |l| =$ even and $|l| \leq n$.

The reconstructed image can be generated by mapping $f(x, y)$ to the $[0, 255]$ range. To generate a binary image, we can use a threshold of 128. One can choose the values of β and order as 256 and 12, respectively. After image normalization, the input image would turn out to be close to the original.

2.11.6. Pattern Formation from Zernike Moments

Let us assume that $n_f = (p+1)(q+1)$ subsequent Zernike moments, of orders from m_{00} to m_{pq}, have been extracted from a given normalized image. One can form the Zernike moment-based pattern representing an image. Each Zernike moment of a given order is a complex number with real part C and imaginary part S. In the pattern-forming phase, we represent a given i^{th} moment by its real-valued magnitude $\sqrt{C^2 + S^2}$, and we set this value as the i^{th} element of the Zernike pattern $\mathbf{x}_{\text{zer}}$.

The **translation** and **scale normalization** processes affect two of the Zernike features, namely, $|A_{00}|$ and $|A_{11}|$. The magnitude of the Zernike moments A_{00} is the same for all images, and the moment $|A_{11}|$ is equal to zero. Therefore, $|A_{00}|$ and $|A_{11}|$ are not considered as pattern features utilized in the classification. Consequently, the length of the Zernike pattern is equal $n_z = n_f - 2$.

Values of m_{pq} (and the resulting value n_f) are found heuristically.

Example: Zernike moments have been successfully applied to handwritten character recognition. Here, Zernike moments from (2,0) through (12,12) are extracted from binarized, thinned, and normalized 32×32 pixel images of characters. From the initial number $169 - 2 = 167$ of Zernike moments, the final (reduced) 6 element patterns are selected by the rough sets methods and used in recognition. The back-propagation neural network-based classifier yielded 92% accuracy in this application.

3. Feature Selection

Pattern **dimensionality reduction** (and thus data set compression), via feature extraction and feature selection, belongs to the most fundamental steps in data processing. Feature selection can be an inherent part of feature extraction (for example, using principal component analysis) or even a processing algorithm design (as in decision tree design). However, feature selection is often isolated as a separate step in processing sequence.

We can define **feature selection** as a process of finding a subset of features, from the original set of features forming patterns in a given data set, according to the defined criterion of feature selection (a **feature goodness criterion**). Here, we consider feature selection as a process of finding the best **feature subset** X_{opt} from the original set of pattern features, according to the defined feature goodness criterion $J_{\text{feature}}(X_{\text{feature\_subset}})$, without additional feature transformation or construction. Feature selection should be stated in terms of the optimal solution of the selection problem (according to the defined goal and criterion) and with the resulting algorithm of such optimal selection.

3.1. Optimal Feature Selection

Assume that a limited-size data set T_{all} is given (consisting of N_{all} cases), constituted with n-feature patterns $\mathbf{x}$ (labeled or unlabeled by target values), sometimes accompanied by a priori knowledge about domain. Let all n features of the pattern (the pattern vector elements x_i $(i = 1, 2, \cdots, n)$) form the entire original feature set $X_{\text{all}} = \{x_1, x_2, \cdots, x_n\}$. The **optimal feature selection** is the process of finding, for a given type of predictor, a subset $X_{\text{opt}} = \{x_{1,\text{opt}}, x_{2,\text{opt}}, \cdots, x_{m,\text{opt}}\}$ containing $m \leq n$ features from the set of all original features $X_{\text{opt}} \subseteq X_{\text{all}}$ that guarantee accomplishment of a processing goal while minimizing a defined feature selection criterion (a feature goodness criterion) $J_{\text{feature}}(X_{\text{feature\_subset}})$. The optimal feature set will depend on the type of predictor designed. More precisely, optimal feature selection will depend on the overall processing goal and its performance evaluation criterion, type of predictor designed, existing data set, a priori domain knowledge, original set of pattern features, overall processing algorithm applied, and defined criterion of feature subset goodness J_{feature} (feature selection criterion). A solution for the optimal feature selection may not be unique. Different subsets of original features may result in the same performance.

The goals of data processing, roles of features, and performance evaluation criteria of processing algorithms may be different. Feature selection algorithms, based on defined feature selection (feature goodness) criteria and the resulting optimal features, will depend on these conditions. For example, for the same data set from the same domain, optimal features found for the classification task, with minimum average classification error probability criterion, might be different that those found for the data compression task with the minimum sum squares error criterion. Similarly, an optimal feature set found for a Bayesian quadratic discriminant-based classifier could be different that that found for a back-propagation neural network classifier.

Generally, the criterion J of performance evaluation for an overall processing algorithm and the criterion J_{feature} of feature goodness are different, although one can design optimal feature selection where these criteria can be the same.

3.1.1. Paradigms of Optimal Feature Selection

We will shortly discuss two paradigms in optimal feature selection: **minimal representation** and **maximal class separability**.

The general goal of feature selection typically includes the key ability of the processing algorithm (using an optimal feature subset) to best process novel instances of domain data that were not seen or used during the design (generalization problem).

Since the processing algorithm (for example, a classifier) is a data model, and since optimal feature selection influences processing algorithm complexity, we can state that optimal feature selection should have much in common with finding an optimal data model. This similarity implies that optimal feature selection might possibly be supported by some general paradigms of data model building. Even though these paradigms have mostly theoretical value, experience shows that they may have also practical implications. The most prominent paradigms in data model building, and potentially in optimal feature selection, are the so-called **minimum construction paradigms**: **Occam's razor**, **minimum description length**, and **minimum message length** (see Chapter 15).

In the light of minimum construction, a straightforward technique of best feature selection could be to choose a minimal feature subset that fully describes all concepts (for example, classes in prediction-classification) in a given data set. However, this approach, while applicable for a given (possibly limited) data set, may not be useful for processing unseen patterns, since these methods might not provide good generalization.

Methods based on minimum construction paradigms for limited-size data sets should take into account generalization problem. This kind of approach relates to the general solution of the **bias-variance dilemma** in data processing design based on limited-size data sets.

Informally, let us transform these somewhat overlapping minimum construction paradigms into the terms of optimal feature selection. The paradigms deal with generalization versus the complexity of a processing algorithm (a data model complexity) and are influenced by the size of the feature set. They indicate that in order to obtain the best generalization, we should find the processing algorithm that has minimal complexity guaranteed by the minimal feature set and that well represents the available data set. Such a paradigm sheds some light on the design of the generalizing algorithm for optimal feature selection. However, it does not provide a rigorous design procedure.

Designers of algorithms based on optimal feature selection face the **bias/variance dilemma** (see Chapter 15). This dilemma underlines the controversy related to the selection of a processing algorithm and optimal feature set complexity: namely, the need to find the best process for a given data set and simultaneously to provide the best generalization for future patterns. Given this dilemma, designers divide the generalization error criteria into the sum of two parts: squared **bias** and **variance**. If the designer too precisely fits the complex processing algorithm to given data in a large feature set, then the algorithm's ability to generalize for unseen patterns may deteriorate. By increasing the complexity of the processing algorithm and feature set, we can reduce the bias and increase the variance. On the other hand, a processing algorithm with a small feature set may not be able to process a given data set satisfactorily. A processing algorithm that is too simple and thus inflexible (with too small a number of parameters), influenced by its small feature set, may have too big a bias and too small a variance. The robust processing algorithm, with its associated set of features (reflecting complexity), implements a tradeoff between its best ability to process a given data set and its generalization capability. This problem is also called the **bias/variance tradeoff**. Despite the theoretical power of design paradigms with minimal structures (for example, with a minimal set of features), in practice, for limited size data sets, the optimal solution is not always a minimal one.

The second general paradigm of optimal feature selection, mainly used in classifier design, relates to selecting the feature subset that guarantees maximal between-class separability for a reduced data set and thus helps design a better predictor-classifier. This paradigm relates to discriminatory power of features, i.e., their ability to distinguish patterns from different classes.

Selection of the best feature subset for a given prediction task corresponds to **feature relevancy**. The relevance of a feature can be understood as its ability to contribute to improving the predictor's performance. For a predictor-classifier, relevance would mean the ability to improve classification accuracy.

A few attempts (both deterministic and probabilistic) have been made in machine learning to define feature relevancy. Let us assume a labeled data set T with N cases ($\mathbf{x}$, **target**), containing n-feature patterns $\mathbf{x}$ and associated **targets**. For classification, a **target** is a categorical class target c_{target} (a concept c) with values from the set of l discrete classes $\{c_1, c_2, \cdots, c_l\}$. For regression, a target is the desired output (scalar or vector) of a real valued predictor (see Chapter 4).

The following definition of deterministic relevancy was proposed for Boolean features in noise-free data sets for the classification task.

Definition 1. A feature x_i is **relevant to a class** c (a concept c) if x_i appears in every Boolean formula that represents c, and is **irrelevant** otherwise.

Definition 2. A feature x_i is **relevant** if there exists some value of that feature a_{x_i} and a predictor output $\mathbf{y}$ value $\mathbf{a_y}$ (generally a vector) for which $P(x_i = a_{x_i}) > 0$ such that

$$P(\mathbf{y} = \mathbf{a_y} | x_i = a_{x_i}) \neq P(\mathbf{y} = \mathbf{a_y}) \tag{373}$$

According to this definition, a feature x_i is relevant if knowledge of its value can change the estimates of $\mathbf{y}$, or in another words, if an output vector $\mathbf{y}$ is conditionally dependent on x_i. Since the above definitions do not deal with the relevance of features in the parity concept, a modification was proposed. Let us denote a vector of features $\mathbf{v}_i = (x_1, x_2, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)^T$ (with its values denoted by $\mathbf{a}_{\mathbf{v}_i}$) obtained from an original feature vector $\mathbf{x}$ by removing the x_i feature.

Definition 3. A feature x_i is **relevant** if there exists some value of that feature a_{x_i} and a predictor output $\mathbf{y}$ value $\mathbf{a_y}$ (generally a vector) for which $P(x_i = a_{x_i}) > 0$ such that

$$P(\mathbf{y} = \mathbf{a_y}, \mathbf{v}_i = \mathbf{a}_{\mathbf{v}_i} | x_i = a_{x_i}) \neq P(\mathbf{y} = \mathbf{a_y}, \mathbf{v}_i = \mathbf{a}_{\mathbf{v}_i}) \tag{374}$$

According to this definition, a feature x_i is relevant if the probability of a target (given all features) can change if we remove knowledge about a value of that feature. Since the above definitions are quite general and may provide unexpected relevancy judgments for a specific data set (for example, one with the nominal features numerically encoded by indicators), more precise definitions of so-called strong and weak relevance were introduced.

Definition 4. A feature x_i is **strongly relevant** if there exists some value of that feature a_{x_i}, a predictor output $\mathbf{y}$ value $\mathbf{a_y}$, and a value $\mathbf{a}_{\mathbf{v}_i}$ of a vector $\mathbf{v}_i$ for which $P(x_i = a_{x_i}, \mathbf{v}_i = \mathbf{a}_{\mathbf{v}_i}) > 0$ such that

$$P(\mathbf{y} = \mathbf{a_y} | \mathbf{v}_i = \mathbf{a}_{\mathbf{v}_i}, x_i = a_{x_i}) \neq P(\mathbf{y} = \mathbf{a_y} | \mathbf{v}_i = \mathbf{a}_{\mathbf{v}_i}) \tag{375}$$

Strong relevance indicates that a feature is indispensable, which means that its removal from a feature vector will decrease **prediction accuracy**.

Definition 5. A feature x_i is **weakly relevant** if it is not strongly relevant, and there exists some subset of features (forming a vector $\mathbf{z}_i$) from a set of features forming patterns $\mathbf{v}_i$ for which there exist some value of that feature a_{x_i}, a predictor output value $\mathbf{a_y}$, and a value $\mathbf{a}_{\mathbf{z}_i}$ of a vector $\mathbf{z}_i$, for which $P(x_i = a_{x_i}, \mathbf{z}_i = \mathbf{a}_{\mathbf{z}_i}) > 0$ such that

$$P(\mathbf{y} = \mathbf{a_y} | \mathbf{z}_i = \mathbf{a}_{\mathbf{z}_i}, x_i = a_{x_i}) \neq P(\mathbf{y} = \mathbf{a_y} | \mathbf{z}_i = \mathbf{a}_{\mathbf{z}_i}) \tag{376}$$

Weak relevance indicates that a feature might be dispensable but sometimes (in the company of some other features) may improve prediction accuracy.

In the light of the above definitions, a feature is **relevant** if it is either **strongly relevant** or **weakly relevant**; otherwise, it is **irrelevant**. By definition an irrelevant feature will never contribute to prediction accuracy and thus can be removed.

The theory of rough sets defines deterministic strong and weak relevance for discrete features and discrete targets. For a given data set, a set of all strongly relevant features forms a **core**. A minimal set of features satisfactory to describe concepts in a given data set, including a core and possibly some weakly relevant features, form a **reduct**. A core is an intersection of reducts.

It has been shown that, for some predictor designs, feature relevancy (even strong relevancy) does not imply that the feature must be in an optimal feature subset. Relevancy, although helpful in feature assessment, does not necessarily contribute to optimal predictor design with generalization ability.

Since an optimal feature set depends on the type of predictor used, definitions of absolute irrelevant, conditionally irrelevant, and conditionally relevant are suggested. **Absolute irrelevant** features, equivalent to irrelevant features as defined above, are those that cannot contribute to prediction performance, and thus can be removed. The remaining features are either conditionally irrelevant or relevant, depending on the designed predictor type. For a given type of predictor, **conditionally irrelevant** features are these not included in an optimal set of features (for which a predictor achieves maximal performance). **Conditionally irrelevant** features are not included in the optimal set for a given predictor and thus can be removed. However, conditionally irrelevant features for one type of a predictor could be conditionally relevant for another types. The conditional relevance depends not only on the type of predictor used but also on the applied feature optimality criterion.

3.1.2. Feature Selection Methods and Algorithms

Although optimal feature selection, related to data model discovery and processing algorithm design, is a rather general problem, so far the statistical, machine learning, and automatic control communities have developed slightly different methods of solution. The existing feature selection methods, depending on the feature selection criterion used, include two main streams:

– open loop methods (filter, preset bias, front end)
– closed loop methods (wrapper, classifier feedback)

Open loop methods, also called **filter**, **preset bias**, or the **front end** methods (Figure 7.12), are based mostly on selecting features through the use of between-class separability criteria. These methods do not consider the effect of selected features on the performance of an entire processing algorithm (for example, a classifier), since the feature selection criterion does not involve predictor evaluation for reduced data sets containing patterns with selected feature subsets only. Instead, these methods select, for example, those features for which the resulting reduced data set has maximal between-class separability, usually defined based on between-class and between-class covariances (or scatter matrices) and their combinations. The ignoring of the effect of a selected feature subset on the performance of the predictor (lack of feedback from predictor performance) is a weak side of open-loop methods. However, these methods are computationally less expensive.

Closed loop methods, also called **wrapper**, **performance bias**, or **classifier feedback** methods (Figure 7.13), are based on feature selection using predictor performance (and thus providing processing feedback) as a criterion of feature subset selection. The goodness of a selected feature subset is evaluated using as a criterion $J_{\text{feature}} = J_{\text{predictor}}$, where $J_{\text{predictor}}$ is the performance evaluation of a whole prediction algorithm for a reduced data set containing patterns with the selected features as pattern elements. Here, the selection algorithm is a “**wrapper**” around the prediction algorithm.

Closed loop methods generally provide better selection of a feature subset, since they fulfill the ultimate goal and criterion of optimal feature selection, i.e., they provide best prediction. The

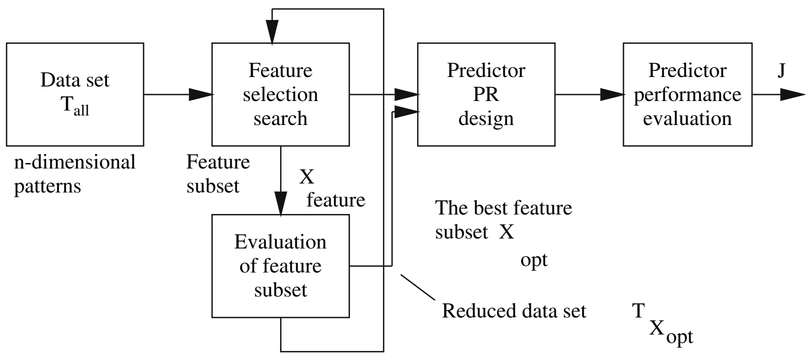
Data set T_{all}
n-dimensional patterns
Feature selection search
Feature subset
$X_{feature}$
Evaluation of feature subset
Predictor PR design
The best feature subset X_{opt}
Reduced data set $T_{X_{opt}}$
Predictor performance evaluation
J

Figure 7.12. An open loop feature selection method.

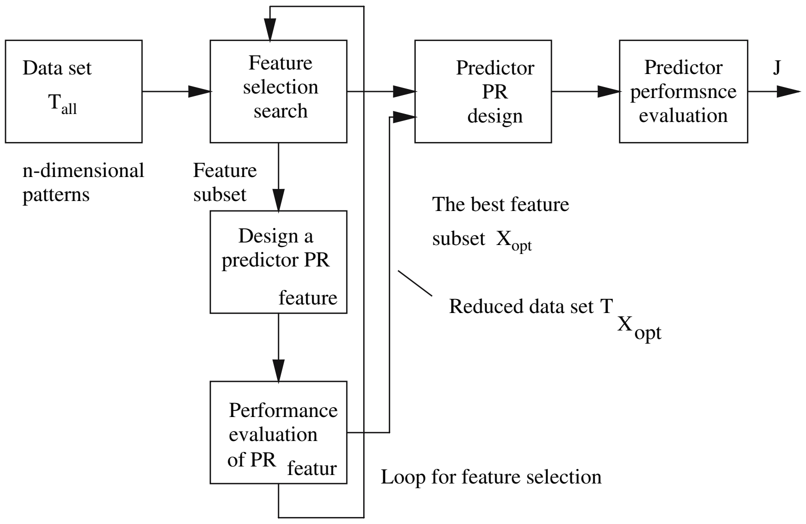
Data set
T_{all}
n-dimensional patterns
Feature selection search
Feature subset
Design a predictor PR feature
Performance evaluation of PR featur
Loop for feature selection
Predictor PR design
The best feature subset X_{opt}
Reduced data set $T_{X_{opt}}$
Predictor performsnce evaluation
J

Figure 7.13. A closed loop feature selection method.

prediction algorithm used in closed-loop feature selection may be the final predictor PR (which provides best selection, $J_{\text{feature}} = J$), or a simpler predictor PR_{feature} may be used for reasons of computational feasibility.

A procedure for optimal feature selection contains

- **feature selection criterion** J_{feature} that allows us to judge whether one subset of features is better than another (evaluation method)
- systematic **search procedure** that allows us to search through candidate subsets of features and includes the initial state of the search and stopping criteria

A search procedure selects a feature subset from among possible subsets of features, and the goodness of this subset is evaluated using the feature selection (optimality judgement) criterion.

Some feature selection criteria do not obey the monotonicity property, which limits the application of the dynamic programming type of search. Ideally, a feature selection criterion should be the same as the criterion for evaluating an entire predictor algorithm (a prediction quality evaluation). This is the ideal approach for closed-loop type (wrapper) feature selection. In practice, simplified selection criteria could be used with simpler predictors that are used exclusively for feature selection.

An ideal search procedure would implement an exhaustive search through all possible subsets of features. This approach is, in fact, the only method that ensures finding an optimal solution. In practice, for large number of features, an exhaustive search is not feasible. Thus, in order to reduce computational complexity, simplified nonexhaustive search methods are used. However, these methods usually provide only a suboptimal solution for feature subset selection.

3.1.3. Feature Selection Criteria

Depending on the criterion used, feature selection, like other optimization problems, can be described in two ways: either as the maximization of a criterion or as the minimization of the reversed criterion. Here, we will consider criteria based on maximization, where a better subset of features always gives a bigger value of a criterion, and the optimal feature subset gives the maximum value of the criterion.

A feature selection algorithm is based on defined criteria for feature selection (goodness), which ideally should be the same as the criteria for the design of a generalizing prediction.

In other words, the general goal of feature selection typically takes into account the ability of the processing algorithm to best process new data patterns. This constitutes a tradeoff between predictor generalization capability and the dimension and type of features used for pattern formation.

In the light of the generalization goal, we may expect specific behavior of the feature selection criteria during an optimal search process. Reduction of feature dimensionality initially improves the generalization ability of an entire predictor (with increasing values of the feature goodness criterion). However, when a particular reduction of pattern dimensionality is reached, the generalization ability starts to degrade. This change may correspond to the point when the best generalizing feature subset is obtained, when the selection criterion reaches its maximum. Surprisingly, however, many feature selection criteria do not behave in this way. The **monotonicity** property is defined as

$$J_{\text{feature}}(X^{+}_{\text{feature}}) \geq J_{\text{feature}}(X_{\text{feature}}) \tag{377}$$

where X_{feature} denotes a feature subset, and X^{+}_{feature} denotes a larger feature subset that contains X_{feature} as a subset. This definition means that adding a feature to a given feature set results in a criterion value that stays the same or increases:

$$\begin{aligned} J_{\text{feature}}(\{x_1\}) \leq J_{\text{feature}}(\{x_1, x_2\}) \leq J_{\text{feature}}(\{x_1, x_2, x_3\}) \leq \cdots \\ \leq J_{\text{feature}}(\{x_1, x_2, \cdots, x_n\}) \end{aligned} \tag{378}$$

On the other hand, deleting a feature does not improve performance. Several criteria, such as the class separability (based on covariance computations), the Bayes average probability of error, and some distance measures such as the Mahalanobis or Bhattacharyya distance satisfy the monotonicity condition. However, some criteria used in machine learning (such as inconsistency rate) do not obey the monotonicity condition.

Criteria that include monotonicity properties cannot be used to compare the goodness of different size feature subsets when a large subset contains a smaller one. However, such criteria can still be used to compare feature subsets of equal size. In practice, for a limited-size data set and performance estimation based on that data set, removing a feature may improve performance. Thus, by using estimates of ideal performance as criteria, we still can seek an optimal reduced subset of features.

Ideally, the feature selection criterion should be the same as the criterion for evaluating an entire predictor algorithm (a prediction quality evaluation), i.e., $J_{\text{feature}} = J$. For example, for prediction-regression the criterion could be a sum of squares error, and for prediction-classification a classification error rate. In an ideal closed-loop type (wrapper) feature selection approach, the entire predictor is used in order to evaluate the goodness of a feature subset, and this predictor's performance criterion should be the same as the feature selection criterion. In practice, simplified closed-loop feature selection criteria could be used, based on a simpler predictor PR_{feature} and only for feature evaluation, with a performance evaluation criterion equal to the feature extraction criterion, i.e., $J_{\text{feature}} = J_{PR_{\text{feature}}}$. For example, instead of evaluating feature goodness by training and testing a complex neural network-type predictor, a simpler predictor, such as k-nearest neighbors, along with its performance evaluation, for example the error rate, can be used for feature evaluation.

Open-loop feature selection criteria are frequently designed differently for real-valued features and prediction-regression and for discrete features for prediction-classification, whereas closed loop criteria are usually adequate for both real-valued and discrete features. Some criteria may include penalty terms that favor lower dimensionality of optimal feature subsets.

3.1.4. Open Loop Feature Selection Criteria

Open loop feature selection criteria are usually based on information (such as interclass separability) contained in the data set alone. They do not consider the direct influence of the selected feature subset on the performance of an entire predictor. They do not provide feedback from the predictor quality assessment to the feature selection process.

3.1.5. Criteria Based on Minimum Concept Description

Feature selection criteria based on the **minimum concept description** paradigm have been studied in machine learning and in statistics for discrete features of noise-free data sets. One technique for best feature selection is to choose a minimal feature subset that fully describes all the concepts in a given data set. Here, a criterion of feature selection could be defined as a Boolean function $J_{\text{feature}}(X_{\text{feature}})$ with value 1 if the feature subset X_{feature} is satisfactory in describing all concepts in a data set, and otherwise having a value 0. The final selection would be based on choosing a minimal subset for which the criterion gives value 1.

3.1.6. Criteria Based on Mutual Information

Based on information theory analysis, the **mutual information** (MI) measure of data sets (based on entropy) can be used as a criterion for feature selection. For two variables, mutual information can be considered to provide a reduction of uncertainty about one variable given the other one. Let us consider mutual information for classification for a given data set T containing n-dimensional pattern vectors $\mathbf{x}$ labeled by l classes $c_{\text{target}} \in \{c_1, c_2, \cdots, c_l\}$ in feature pattern vector $\mathbf{x}$. The entire set of original features is a collection of pattern vector elements: $X = \{x_1, x_2, \cdots, x_l\}$. The mutual information for the classification problem is the reduction of uncertainty about classification given a subset of features X forming a feature pattern $\mathbf{x}$. It can be understand as the suitability of the feature subset X for classification. If we consider initially only probabilistic knowledge about classes, the uncertainty is measured by entropy as

$$E(c) = -\sum_{i=1}^{l} P(c_i) \log_2 P(c_i) \tag{379}$$

where $P(c_i)$ is the a priori probability of a class c_i occurrence (which may be estimated based on the data set). **Entropy** $E(c)$ is the expected amount of information needed for class prediction. The entropy is maximal when a priori probabilities $P(c_i)$ are equal. The uncertainty about class prediction can be reduced by knowledge abut feature patterns $\mathbf{x}$ formed with features from a subset X, characterizing recognized objects and their class membership. The conditional entropy $E(c|\mathbf{x})$ (a measure of uncertainty), given pattern $\mathbf{x}$, is defined as

$$E(c|\mathbf{x}) = \sum_{i=1}^{l} P(c_i|\mathbf{x}) \ \log_2 P(c_i|\mathbf{x}) \tag{380}$$

The **conditional entropy**, given the subset of features X, is defined for discrete features as

$$E(c|X) = -\sum_{\text{all } \mathbf{x}} P(\mathbf{x}) \left(\sum_{i=1}^{l} P(c_i|\mathbf{x}) \ \log_2 P(c_i|\mathbf{x}) \right) \tag{381}$$

The outer sum considers all feature vectors $\mathbf{x}$ in a feature space. Using equality $P(c_i|\mathbf{x}) = \frac{P(c_i,\mathbf{x})}{P(\mathbf{x})}$, we can obtain

$$\begin{aligned} E(c|X) &= -\sum_{\text{all } \mathbf{x}} P(\mathbf{x}) \left(\sum_{i=1}^{l} \frac{P(c_i, \mathbf{x})}{P(\mathbf{x})} \ \log_2 \frac{P(c_i, \mathbf{x})}{P(\mathbf{x})} \right) \\ &= -\sum_{\text{all } \mathbf{x}} \sum_{i=1}^{l} P(c_i, \mathbf{x}) \ \log_2 \frac{P(c_i, \mathbf{x})}{P(\mathbf{x})} \end{aligned} \tag{382}$$

For patterns with continuous features, the outer sum should be replaced by an integral and the probabilities $P(\mathbf{x})$ by the probability density function $p(\mathbf{x})$:

$$E(c|X) = -\int_{\text{all } \mathbf{x}} p(\mathbf{x}) \left(\sum_{i=1}^{l} P(c_i|\mathbf{x}) \ \log_2 P(c_i|\mathbf{x}) \right) \tag{383}$$

Using Bayes's rule,

$$P(c_i|\mathbf{x}) = \frac{p(\mathbf{x}|c_i)P(c_i)}{p(\mathbf{x})} \tag{384}$$

The probabilities $P(c|\mathbf{x})$ that are difficult to estimate can be replaced by $p(\mathbf{x})$ and $P(\mathbf{x}|c_i)$. The initial uncertainty (based on a priori probabilities $P(c_i)$ only), might decrease given knowledge about feature pattern $\mathbf{x}$. The **mutual information** $MI(c, X)$ between the classification and the feature subset X is measured by a decrease in uncertainty about the prediction of classes, given knowledge about patterns $\mathbf{x}$ formed from features X:

$$J_{\text{feature}}(X) = MI(c, X) = E(c) - E(c|X) \tag{385}$$

Since for discrete features we can derive the equation

$$J_{\text{feature}}(X) = MI(c, X) = \sum_{\text{all } \mathbf{x}} \sum_{i=1}^{l} P(c_i, \mathbf{x}) \ \log_2 \frac{P(c_i, \mathbf{x})}{P(\mathbf{x})P(c_i)} \tag{386}$$

the mutual information is a function of c and $\mathbf{x}$; if they are independent, the mutual information is equal to zero (knowledge of $\mathbf{x}$ does not improve class prediction).

Mutual information is the unbiased information about the ability of feature subset X to predict classes. The criterion $MI(c, X)$ is a theoretical limit for feature goodness (similarly, a classifier design based on the Bayes optimal decision is a theoretical limit of accuracy for predictors). The informative power of a feature subset X is never larger than its mutual information with a predicted class. Features from a subset X are absolutely irrelevant for the classification task if their mutual information is equal to zero.

The mutual information criterion is difficult to use in practice due to the difficulties and inaccuracy of estimating conditional probabilities for limited-size data sets. These problems surface when the dimensionality of feature patterns is high and the number of cases small. For low-dimensional data patterns, application of the mutual information criterion (with probability density estimations) can be used to choose the best feature subset from all possible feature subsets. In the simplified application of the mutual information criterion for feature selection, a greedy algorithm adds one most-informative feature at a time. The added feature is chosen as that which has the maximal mutual information with a class and minimal mutual information with already selected features. This method does not solve the redundancy problem between groups of features.

3.1.7. Criteria Based on Inconsistency Count

Another criterion for feature subset evaluation for discrete feature data sets is the **inconsistency measure**. Let us consider a given feature subset X_{feature} and a reduced data set $T_{X_{\text{feature}}}$, with all N_{all} cases $(\mathbf{x}_f, c_{\text{target}})$. Each case contains a pattern $\mathbf{x}_f$ constituted with m features from a subset X_{feature} and labeled by classes c_{target}. The inconsistency criterion $J_{\text{inc}}(T_{X_{\text{feature}}})$ for a data set $T_{X_{\text{feature}}}$ can be defined as the ratio of all inconsistency counts divided by the number of cases. Two cases $(\mathbf{x}_f^j, c_{\text{target}}^i)$ and $(\mathbf{x}_f^k, c_{\text{target}}^k)$ are inconsistent if both have the same patterns $\mathbf{x}_f^j = \mathbf{x}_f^k$ but different associated classes $c_{\text{target}}^j \neq c_{\text{target}}^k$. We can find the inconsistency count of a given set of the same patterns $\mathbf{x}_f^i$ for which cases are inconsistent. Here, for the same matching patterns $\mathbf{x}_f^i$ we compute the inconsistency count as a number $n_{\text{inc,i}}$ of all inconsistent cases for the matching pattern minus the largest number of cases in one of the classes from this set of inconsistent cases. For example, let us assume that we have, in a reduced data set $T_{X_{\text{feature}}}$, v inconsistent cases for a pattern $\mathbf{x}_f^i$, with q_1 cases from a class c_1, q_2 cases from a class c_2, and q_3 cases from a class $c_3 (v = q_1 + q_2 + q_3)$. If q_2 is the largest number among the three, then the inconsistency count for matching pattern $\mathbf{x}_f^i$ is $I_i = n - q_2$. The **inconsistency rate** criterion is defined for a reduced data set $T_{X_{\text{feature}}}$ as a ratio of sum of all inconsistency counts and a number of all cases N_{all} in the data set

$$J_{\text{inc}}(T_{X_{\text{feature}}}) = \frac{\sum_{\text{all inconsistent patterns}} I_i}{N_{\text{all}}} \tag{387}$$

3.1.8. Criteria Based on Rough Sets' Quality of Classification

For prediction-classification, feature subset goodness can be measured by its ability to classify concepts (family of classes) in a given data set. This idea comes from rough sets theory. Let us consider a data set represented by the information system S with the close universe U (object sets with card $U = N$) and a full set of attributes X. Consider a subset of attributes $A \subseteq X$. Let $\Upsilon = \{Y_1,\ Y_2,\ \ldots\ ,\ Y_m\}$ for every $Y_i \subseteq U (1 \leq i \leq m)$ be a classification (a partition, a family of subsets) of U. The family of sets $\Upsilon = \{Y_1, Y_2, \cdots, Y_m\}$ is a classification in U in S, if $Y_i \cap Y_j = \emptyset$ for every $i, j \leq m, i \neq j$ and $\bigcup_{i=1}^m X_i = U$. X_i are called classes of Υ. Here, we will consider a classification based on a subset of attributes A. The **quality of classification** Υ by A, imposed by the set of attributes A, is defined as follows:

$$\rho_A(\Upsilon) = \frac{\Sigma_{i=1}^m \text{card } (\underline{A}Y_i)}{\text{card } (U)} \tag{388}$$

which represents the ratio of all A correctly classified objects to all objects in the information system S. The term $\underline{A}Y_i$ denotes the lower approximation of class Y_i by the set of attributes A.

This measure can be considered as a feature selection criterion if we assume $X_{\text{feature}} = A$:

$$J_{\text{feature}}(X_{\text{feature}}) = \rho_A(\Upsilon) = \rho_{X_{\text{feature}}} = \frac{\Sigma_{i=1}^m \text{card } (\underline{X}_{\text{feature}} Y_i)}{\text{card } (U)} \tag{389}$$

The above criterion is an open-closed loop type from the point of view of a given data set (the training set), since it considers selection of a feature subset (a reduct) that guarantees correct classification for a given data set. However, this criterion provides no assurances about how well this feature subset will perform for new unseen cases.

3.1.9. Criteria Based on Interclass Separability

Prediction-classification open loop criteria for feature selection are frequently based on **interclass separability**, computed based on covariances (or scatter matrices) estimated for given data sets for each class and between classes. They are constructed based on an evaluation paradigm such that a good feature (with high discernibility power) will cause a small within-class scatter and a large between-class scatter.

Let us first study a given original data set T_{all} containing N_{all} cases $(\mathbf{x}^i, c_{\text{target}}^i)$ with patterns $\mathbf{x}$ constituted with n features and labeled by one target class c_{target}^i from all possible l classes. For a data set T_{all}, we denote a number of cases in each class $c_i (i = 1, 2, \cdots, l)$ by $N_i (\sum_{i=1}^l N_i = N_{\text{total}})$. Recalling Fisher's analysis, one can estimate the expected value (a mean) for patterns within each class by

$$\boldsymbol{\mu}_i = \frac{1}{N_i} \sum_{j=1, \mathbf{x}^j \text{ in } c_i}^{N_i} \mathbf{x}^j, \ (i = 1, 2, \cdots, l) \tag{390}$$

In order to present a scatter of patterns around a mean and within patterns of one class c_i, one ideally would consider an unbiased estimate of a squared covariance matrix of class $n \times n$,

$$\Sigma_i = \frac{1}{N_i - 1} \sum_{j=1, \mathbf{x}^j \text{ in } c_i}^{N_i} (\mathbf{x}^j - \boldsymbol{\mu}_i)(\mathbf{x}^j - \boldsymbol{\mu}_i)^T, \quad (i = 1, 2, \cdots, l) \tag{391}$$

However, instead of a covariance matrix, a squared $n \times n$ **within-class** c_i **scatter matrix** is considered instead:

$$\mathbf{S}_i = \sum_{j=1, \mathbf{x}^j \text{ in } c_i}^{N_i} (\mathbf{x}^j - \boldsymbol{\mu}_i)(\mathbf{x}^j - \boldsymbol{\mu}_i)^T, \quad (i = 1, 2, \cdots, l) \tag{392}$$

This matrix is proportional to the covariance matrix, symmetric, and positive semidefinite (and, for $N_{\text{all}} > n$, usually nonsingular). In order to provide a summarizing measure for scatter patterns around means for all l classes, the so-called **within-class scatter matrix** is defined:

$$\mathbf{S}_w = \sum_{i=1}^{l} \mathbf{S}_i \tag{393}$$

To illustrate between-class scatter, first we define estimates of the total data mean and the total scatter matrix (for all N_{all} patterns from T_{all}) as a proportional representation of a covariance estimate. The **total data mean** can be estimated by

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}^j = \frac{1}{N} \sum_{i=1}^{l} N_i \boldsymbol{\mu}_i \tag{394}$$

and the **total scatter matrix** for all patterns as

$$\mathbf{S}_t = \sum_{j=1}^{N_{\text{all}}} (\mathbf{x}^j - \boldsymbol{\mu})(\mathbf{x}^j - \boldsymbol{\mu})^T \tag{395}$$

We find that the total scatter matrix can be decomposed into two matrices

$$\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b \tag{396}$$

where $\mathbf{S}_w$ is the within-class scatter matrix and the $n \times n$ square matrix $\mathbf{S}_b$ is the so-called **between-class scatter matrix**, defined as

$$\mathbf{S}_b = \sum_{i=1}^{l} N_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T \tag{397}$$

To form a final scalar feature selection criterion involving interclass separability, we need to define a function that gives a larger value when within-class scatter is smaller or between-class scatter is larger. Generally, this function gives a larger value when interclass separability is larger. For one feature, one can say that a feature is “good” (has a large discriminatory or predictive power) if its within-class variance is small and its between-class variance is large. For multiple classes and multifeature patterns, the following feature selection criteria, based on interclass separability, are defined:

1. Ratio of determinants for between-class and within-class scatter matrices:

$$J_{\text{feature}} = \frac{|\mathbf{S}_b|}{|\mathbf{S}_w|} = \frac{\det(\mathbf{S}_b)}{\det(\mathbf{S}_w)} \tag{398}$$

   where the determinant $|\mathbf{S}_b|$ denotes a scalar representation of the between-class scatter matrix, and similarly the determinant $|\mathbf{S}_w|$ denotes a scalar representation of the within-class scatter matrix.
2. Ratio of determinants for between-class and total scatter matrices:

$$J_{\text{feature}} = \frac{|\mathbf{S}_b|}{|\mathbf{S}_t|} = \frac{|\mathbf{S}_b|}{|\mathbf{S}_b + \mathbf{S}_w|} \tag{399}$$

   which in older literature is referred to as **Wilks' lambda**.
3. Trace of $\mathbf{S}_w^{-1}\mathbf{S}_b$:

$$J_{\text{feature}} = \text{trace}(\mathbf{S}_w^{-1}\mathbf{S}_b) \tag{400}$$

   where **trace** denotes a matrix trace.
4. Logarithm of $\mathbf{S}_w^{-1}\mathbf{S}_b$:

$$J_{\text{feature}} = \ln\left(\mathbf{S}_w^{-1}\mathbf{S}_b\right) \tag{401}$$

For single feature patterns x, and for two-class classification, the following version of the interclass separation criterion can be used:

$$J_{\text{feature}} = F_{\text{Fisher}} = \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2} \tag{402}$$

Here, data patterns mean, for each class,

$$\mu_i = \frac{1}{N_i} \sum_{j=1, x^j \text{ in } c_i}^{N_i} x^j, \quad (i = 1, 2) \tag{403}$$

and the scatter of a feature x for each class is

$$s_i^2 = \frac{1}{N_i} \sum_{j=1, x^j \text{ in } c_i}^{N_i} (\mu_i - x^j)^2, \quad (i = 1, 2) \tag{404}$$

Finally, the total scatter of feature x in an entire data set is

$$s_t^2 = s_1^2 + s_2^2 \tag{405}$$

The above single feature selection criterion, which is based on the Fisher linear discriminant analysis and on interclass separability, is called the **Fisher F-ratio**. It allows us to find a feature guaranteeing maximization of the between-class variance while simultaneously minimizing the within-class variance.

3.1.10. Closed Loop Feature Selection Criteria

The closed-loop-type feature selection criteria and their estimations are similar both for classification and for regression type predictors.

Let us consider a feature selection criterion for a prediction task based on the original data set T_{all} that includes N_{all} cases $(\mathbf{x}, \mathbf{target})$ consisting of n-dimensional input patterns $\mathbf{x}$ (whose elements represent all features X) and a **target** value. Let us assume that the m-feature subset $X_{\mathrm{feature}} \subseteq X$ ought to be evaluated based on the closed-loop-type criterion. First, a reduced data set T_{feature}, with patterns containing only m features from the subset X_{feature}, should be constructed. Then a type of predictor PR_{feature} (for example, k-nearest neighbors or a neural network), used for feature goodness evaluation, should be chosen. This predictor ideally should be the same as the final predictor PR for an entire design; however, in a simplified suboptimal solution, a computationally less expensive predictor can be used only for feature selection purposes. After a reduced data set X_{feature} has been constructed and a predictor algorithm PR_{feature} has been decided for the considered feature subset X_{feature}, then evaluation of feature goodness, equivalent to the predictor evaluation criterion, can be performed. Doing so will require defining a performance criterion $J_{PR_{\mathrm{feature}}}$, of a predictor PR_{feature}, and an error counting method showing how to estimate a performance and how to grasp its statistical character through averaging of results. Consider as an example a holdout error-counting method for predictor performance evaluation. In order to evaluate the performance of a predictor PR_{feature}, an extracted feature data set T_{feature} is split into a N_{tra}-case training set $T_{\mathrm{feature,tra}}$, and a N_{test}-case test set $T_{\mathrm{feature,test}}$ (the holdout for testing). Each case $(\mathbf{x}_f^i, \mathbf{target}^i)$ of both sets contains a feature pattern $\mathbf{x}_f$ labeled by a target.

The criteria for evaluating the performance of predictor PR_{feature} should be considered separately for regression and classification.

In prediction-regression, the predicted output variables are continuous. Prediction-regression performance criteria are based on the counting error between target and guessed real values. Let us consider defining a feature selection criterion for a prediction-regression task, with a q-dimensional output vector $\mathbf{y} \in \mathbb{R}^q$ whose elements take real values. Here, the design is based on a reduced feature data set T_{feature} (with N_{all} cases). This data set's case $(\mathbf{x}_f^i, \mathbf{y}_{\mathrm{target}}^i)$ includes m-dimensional feature input patterns $\mathbf{x}_f$ and associated real-valued $\mathbf{y}_{\mathrm{target}} \in \mathbb{R}^q$ target vectors for outputs. Elements of patterns $\mathbf{x}_f$ are features from the subset X_{feature}. One example of the predictor performance criterion $J_{PR_{\mathrm{feature}}}$, being here equivalent to the feature selection criterion $J_{\mathrm{feature}} = J_{PR_{\mathrm{feature}}}$, could be

$$\begin{aligned} J_{\mathrm{feature}} = J_{PR_{\mathrm{feature}}} = \hat{J}_{\mathrm{squared}} &= \sum_{i=1}^{N_{\mathrm{all}}} (\mathbf{y}_{\mathrm{target}}^i - \mathbf{y}^i)^T (\mathbf{y}_{\mathrm{target}}^i - \mathbf{y}^i) \\ &= \sum_{i=1}^{N_{all}} \sum_{j=1}^{m} (y_{j,\mathrm{target}}^i - y_j^i)^2 \end{aligned} \tag{406}$$

which boils down to the sum of squares errors. This criterion is evaluated based on a limited-size test set $T_{\mathrm{feature,test}}$ obtained from the example by splitting the entire set T_{feature} into subsets $T_{\mathrm{feature,tra}}$ for training (design) and $T_{\mathrm{feature,test}}$ for testing.

In prediction-classification, cases in the feature subset T_{feature} are pairs $(\mathbf{x}_f, c_{\mathrm{target}})$ that include the feature input pattern $\mathbf{x}_f$ and a categorical-type target c_{target} being one of the possible l classes c_i. The quality of classifier PR_{feature}, computed, for example (for holdout error counting), based on the limited-size test set $T_{\mathrm{feature,test}}$ with N_{test} patterns, can be measured using the performance criterion $J_{PR_{\mathrm{feature}}}$ below (here equal to the feature selection criteria J_{feature}), which estimate the probabilities of errors (expressed in percent) by relative frequencies of errors:

$$J_{PR_{\mathrm{feature}}} = \hat{J}_{\mathrm{all\ miscl}} = \frac{n_{\mathrm{all\ miscl}}}{N_{test}} \cdot 100\% \tag{407}$$

where $n_{\text{all miscl}}$ is the number of all misclassified patterns, and N_{test} is the number of all tested patterns. This measure is an estimate of the probability of error P(**assigned class different than target class**), expressed in percent (percentage of all misclassified patterns, an error rate, or the relative frequency of errors).

3.1.11. Computing Feature Selection Criteria

Let us consider a given m-feature subset X_{feature}, created from a n-feature set of original features $X = \{x_1, x_2, \cdots, x_n\}$. Computing a feature selection criterion $J_{\text{feature}}(X_{\text{feature}})$, for a given m-feature subset X_{feature}, first requires the creation of a reduced data set T_{feature} extracted from the original total data set T_{all}. This reduced data set T_{feature} contains N_{all} cases with m-dimensional feature pattern vectors $\mathbf{x}_f$, containing, as elements, features from the considered subset X_{feature}, and the same targets as in the original entire data set. Other $n - m$ “columns” of the entire data set are discarded. For this reduced data set T_{feature}, a feature selection criterion could be computed.

The computing of closed-loop-type feature selection criteria, which is based on the evaluation of feature goodness by testing the performance of an entire predictor, is computationally expensive. It involves design (training) of a predictor PR_{feature} and its performance evaluation, both based on a reduced data set T_{feature}. First, for a given m-feature subset X_{feature}, a reduced m-feature pattern data set T_{feature} is constructed (with N_{all} cases). Then, based on this reduced data set, a chosen predictor PR_{feature} is designed. Finally, the performance of this designed predictor is evaluated, according to its defined evaluation criterion $J_{PR_{\text{feature}}}$, which is equal to feature selection criterion J_{feature}. Design and performance evaluation of a predictor used for feature selection can be realized using one of the methods of predictor design-evaluation. Such a method splits the reduced data set T_{feature} into training and test sets and then uses a statistical error counting method. For example, the average holdout, leave-one-out, or leave-k-out method of predictor design/performance evaluation could be used.

For open-loop-type feature selection criteria, which are based on interclass separability, first the within-class $\mathbf{S}_w$ and between-class $\mathbf{S}_b$ scatter matrices are computed for a reduced data set X_{feature}, and then the final value of criterion J_{feature} is computed as a function of these scatter matrices. No predictor performance evaluation is considered for the tested features.

3.1.12. Search Methods

Given the large number of features constituting a pattern, the number of possible feature subsets evaluated by using exhaustive search-based feature selection could be too high to be computationally feasible. For n features, a total of 2^n subsets (including an empty subset) can be formed. For $n = 12$ features, the number of possible subsets is 4096; however, for $n = 100$, the number of possible subset is larger than 10^{30}, which makes exhaustive search unrealizable. If, for some design reason, we are searching for a feature subset containing exactly m features, then for n feature patterns, the total number of possible m-feature subsets is

$$\binom{n}{m} = \frac{n!}{(n-m)!m!} \tag{408}$$

This number (which can be much smaller than 2^n) can be, from a computational perspective, too high.

Generally, feature selection is an NP-hard problem, and for highly dimensional patterns, an exhaustive search can be impractical and suboptimal selection methods should be used. Next, we discuss an exhaustive search technique and the optimal branch and bound method. Then we consider two suboptimal greedy search methods: forward and backward search. We also discuss random search, and finally we give pointers suboptimal search methods such as simulated annealing and genetic programming.

For a small number of pattern features, the **exhaustive search** could be acceptable and could guarantee an optimal solution.

Algorithm: Feature selection based on exhaustive search

Given: A data set T_{all} with N_{all} labeled patterns constituted with n features $X = \{x_1, x_2, \cdots, x_n\}$. A feature selection criterion J_{feature} with a defined computation procedure based on a limited-size data set $T_{X_{\text{feature}}}$.

1. Set $j = 1$ (a counter of the feature subset number).
2. Select a distinct subset of features $X^j \subseteq X$ (with the number of elements $1 \leq N_{X^j} \leq n$).
3. For a selected feature subset X^j, compute a feature selection criterion $J_{\text{feature}}(X^j)$.
4. If $j \leq 2^n$, continue from step 2; otherwise, go to the next step.
5. Chose an optimal subset $\hat{X}_{\text{opt}}$ with a maximal value of the selection criterion

$$J_{\text{feature}}(\hat{X}_{\text{opt}}) \geq J_{\text{feature}}(\hat{X}^j), \quad j = 1, 2, \cdots, 2^n \tag{409}$$

A sequence of generated distinct feature subsets X^j is not important for the above algorithm. For example, for the three feature patterns $X = \{x_1, x_2, x_3\}$, one can generate the following exhaustive collection of $2^3 = 8$ feature subsets (including an empty subset):

$$\{\ \},\ \{x_1\},\ \{x_2\},\ \{x_3\},\ \{x_1, x_2\},\ \{x_1, x_3\},\ \{x_2, x_3\},\ \{x_1, x_2, x_3\} \tag{410}$$

The algorithm guarantees finding an optimal feature subset. Depending on the feature selection criterion (and the procedure for its estimation), either an open loop or a closed loop feature selection scheme is consequently employed.

3.1.13. Branch and Bound Method for Optimal Feature Selection

In combinatorial optimization, in order to avoid a costly exhaustive search, **branch and bound** methods were developed and adapted to optimal feature selection. The branch and bound search can be used for feature selection, assuming that a feature selection criterion satisfies the monotonicity relation. This method allows us to find an optimal set of features without needing to test all possible feature subsets. Using a tree representation for the exhaustive subset search, the branch and bound methods with monotonic performance criterion are based on the idea of discarding some subtrees from the exhaustive search. The removed subtrees contain subsets of feature that would not improve performance in the search procedure. To explain the method, we consider an attempt to construct a fully exhausted search tree (containing all possible subsets of features) designed in a way that allows reduction of a search by employing the idea of monotonicity of a feature selection criterion.

We start with pattern feature of all n-elements, represented in this algorithm as a sequence $x_1, x_2, \cdots, x_n$, and search for the best subset containing a known number of m features. Let us denote the indices of the $d = n - m$ discarded features (from all sets of n features X) by $z_1, z_2, \cdots, z_d$. Here, each variable z_i can take an integer number (the index number of the feature from the sequence of features taken from X) from the set of indices $\{1, 2, \cdots, n\}$. We note that each variable z_i has a distinct index number, since each feature can be discarded only one time. The order of variables z_i is not important, since any permutation of the sequence $z_1, z_2, \cdots, z_d$ gives an identical value for the feature selection criteria. In order to better design a search tree, we consider sequences of variables z_i that satisfy the relation

$$z_1 < z_2 < \cdots < z_d \tag{411}$$

and that determine a convenient way to design a search tree. By definition, a feature selection criterion $J_{\text{feature}}(X_{\text{feature}})$ is a function of the $m = n - d$ feature subset X_{feature} remaining after discarding d features from the set of all features. For convenience, we will also use the notation

$J_{\text{feature}}(z_1, z_2, \cdots, z_d)$ for the same criterion. The goal of the optimal search is to find the best m-feature subset X_{opt}, with discarded $d = n - m$ features (from the original sequence of n features) indicated by an "optimal" sequence of indices $\hat{z}_1, \hat{z}_2, \cdots, \hat{z}_d$. Searching for an optimal feature subset is then equivalent to searching for the "optimal" set of discarded features:

$$J_{\text{feature}}(\hat{z}_1, \hat{z}_2, \cdots, \hat{z}_d) = \max_{z_1, z_2, \cdots, z_d} J_{\text{feature}}(z_1, z_2, \cdots, z_d) \tag{412}$$

Using the defined convention, we can now design a specific search tree starting from all sets of n features at the root (level 0). At each subsequent level, we attempt to generate a limited number of subtrees by deleting, from the ancestor node's pool of features, one specific feature at a time (limited by convention). In the search tree, each node at the j^{th} level is labeled by a value of the variable z_j, which is equal to the index of a feature discarded from the sequence of features in the ancestor node at the previous level $j-1$. Each node at the j^{th} level can be identified by a sequence of already-discarded j features starting from the root. Here, according to defined convention, at each j^{th} level the largest value of a variable z_i must be $m-j$. This method allows the design of a search tree containing all possible $\frac{n!}{(n-m)!m!}$ subsets with m features out of all n. However, in the branch and bound search, not all subtrees need to be searched.

Assume that a feature selection criterion satisfies monotonicity:

$$J_{\text{feature}}(z_1) \geq J_{\text{feature}}(z_1, z_2) \geq J_{\text{feature}}(z_1, z_2, z_3) \geq \cdots \geq J_{\text{feature}}(z_1, z_2, \cdots, z_d) \tag{413}$$

Let as assume that at a certain level of the search, the best feature set identified so far has been found by deleting d features indexed by a sequence $z_1, z_2, \cdots, z_d$, with maximal performance criterion value $J_{\text{feature}}(z_1, z_2, \cdots, z_d) = \beta$ (set as a current threshold). Then, for a new feature subset obtained by deleting r features $(r < d)$ indexed by $z_1, z_2, \cdots, z_r$, if

$$J_{\text{feature}}(z_1, z_2, \cdots, z_r) \leq \beta \tag{414}$$

then the monotonicity property yields

$$J_{\text{feature}}(z_1, z_2, \cdots, z_r, z_{r+1}, \cdots z_d) \leq J_{\text{feature}}(z_1, z_2, \cdots, z_r) \leq \beta \tag{415}$$

for all possible sequences $z_r, z_{r+1}, \cdots z_d$. This new feature subset, obtained by deleting r features $(r < d)$, cannot be optimal, nor can its successors in the search tree. If the described technique of designing a search tree has been applied, then the above observation reveals the main idea of the branch and bound search, namely, if the value of a selection criterion evaluated at any node of a search tree (for corresponding feature subset) is smaller than the current value of a threshold β (corresponding to the best subset found so far by using best evaluation criterion value), then all nodes in the tree, including successors of that node, have a selection criterion value less than the threshold β. Consequently, this node cannot be optimal (nor can all its successors), and consequently this subtree can be removed from the search. This is why, in branch and bound feature selection, we obtain an optimal solution without needing to evaluate all possible feature subsets.

Algorithm: Feature selection by branch and bound search.

Given: A data set T_{all} with N_{all} labeled patterns constituted with n features $X = \{x_1, x_2, \cdots, x_n\}$. A number m of features in the resulting subset of best features. A feature subset selection criterion J_{feature} (satisfying the monotonicity property) with a defined procedure for its computation based on a limited-size data set $T_{X_{\text{feature}}}$.

1. Set a level number $j = 0$, $z_0 = 0$ (the notation of a node at level j), and an initial value of the threshold $\beta = -\infty$.
2. Create successors by generating a list S_j,

$$S_j = \{z_{j-1}+1, z_{j-1}+2, \cdots, m+j\}, \quad (j = 1, 2, \cdots, m) \tag{416}$$

   of all possible values that z_j at level j can take (assuming given values from previous levels $z_1, z_2, \cdots, z_{j-1}$), with a maximal index $m+j$. The successor nodes contains feature subsets with one feature deleted from the list of the previous level.
3. Select a new node. If a list S_j is empty, then go to step 5. Otherwise, find a value k (with maximal value of the criterion) for which

$$J_{\text{feature}}(z_1, z_2, \cdots, z_{j-1}, k) = \max_{i \in S_j} J_{\text{feature}}(z_1, z_2, \cdots, z_{j-1}, i) \tag{417}$$

   Set $z_j = k$, and delete k from the list S_j.
4. Test a bound. If $J_{\text{feature}}(z_1, z_2, \cdots, z_j) < \beta$, then go to the step 5. If the last level has been reached, go to step 6; otherwise, advance to a new level by setting $j = j+1$ and continuing from step 2.
5. Return (backtrack) to a lower level. For $j = 0$, terminate; otherwise, continue from step 3.
6. The last level: Set $\beta = J_{\text{feature}}(z_1, z_2, \cdots, z_d)$ and $\hat{z}_1, \hat{z}_2, \cdots, \hat{z}_d = z_1, z_2, \cdots, z_d$. Continue from step 5.

Result: An optimal subset of features with the largest value of the criterion.

Figure 7.14 shows an example of the branch and bound algorithm for selecting two features ($m = 2$) from the total number of $n = 5$ features.

The black nodes show examples of subtrees that do not need to be searched for optimal subsets because they will not provide better solutions. The number next to each node shows the index of the feature deleted from a list of ancestor features. A node at level j is labeled with the value z_j. The set of all possible subsets of two features out of five is represented by nodes of the last level (each node represents one feature subset obtained by deleting corresponding features). According to the defined convention, the maximal value of z_j is $m+j$; thus at the first level it equals 3, at the next level it equals 4, etc. To show how the algorithm work, we assume that so far the node marked by A has given the best value of the criterion set as a current threshold β. Then, according to criterion monotonicity, if at any step of the algorithm an intermediate node is considered (such as that marked by B) for which the criterion value is less

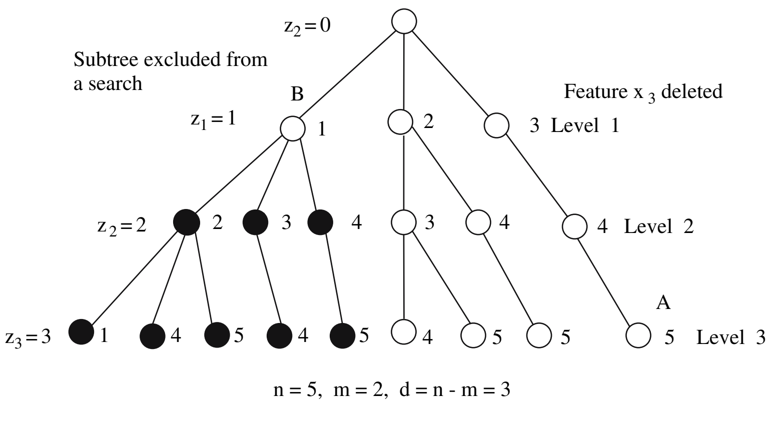

Figure 7.14. Illustration of the branch and bound algorithm.

than the current threshold, then all subtrees starting from that node can be excluded from the search. If for any node in the final level a selection criterion has a larger value than the current threshold, this criterion value becomes a new threshold. The algorithm terminates when each final level node has been evaluated or excluded based on the monotonicity property.

3.1.14. Feature Selection with Individual Feature Ranking

One of the simplest feature selection procedures is based on first evaluating the individual predictive power of each feature alone, then ranking such evaluated features, and eventually choosing the best first m features. The criterion for the individual feature could be of either open loop or closed loop type. This algorithm assumes that features are independent and that the final selection criterion can be obtained as a sum or product of criteria evaluated for each feature independently. Since these conditions are rarely satisfied, the algorithm does not guarantee an optimal selection. A single feature alone may have very low predictive power. However, this feature in combination with another feature may provide substantial predictive power. A decision concerning how many best m-ranked features should be chosen for the final feature set could be made based on experience from using another search procedure. Here, one could select the minimal number $\hat{m}$ of best-ranked features that guarantee a performance better than or equal to a predefined threshold according to a defined criterion $J_{\text{feature,ranked}}$.

Algorithm: Feature selection with individual feature ranking

Given: A data set T_{all} with N_{all} labeled patterns consisting of n features $X = \{x_1, x_2, \cdots, x_n\}$; a feature evaluation criterion $J_{\text{feature,single}}$ with a defined procedure for its computation based on a limited-size data set $T_{X_{\text{feature}}}$; and an evaluation criterion $J_{\text{feature,ranked}}$ for a final collection of m ranked features.

1. Set $j = 1$, and choose a feature x_j.
2. Evaluate the predictive power of a single feature x_j alone by computing the criterion $J_{\text{feature,single}}(x_j)$.
3. If $j \leq n$, continue from step 1; otherwise, go to the next step.
4. Rank all n features according to the value of the computed criterion $J_{\text{feature,single}}$:

$$x_a, x_b, \cdots, x_m, \cdots, x_r, \quad J_{\text{feature,single}}(x_a) \geq J_{\text{feature,single}}(x_b), \text{ etc.} \tag{418}$$

5. Find the minimal number of first-ranked $\hat{m}$ features according to the criterion $J_{\text{feature,ranked}}$.
6. Select the first $\hat{m}$ best-ranked features as a final subset of selected features.

Result: An optimal subset of features.

3.1.15. Sequential Suboptimal Forward and Backward Feature Selection

In order to reduce the computational burden associated with an exhaustive search, several suboptimal feature selection methods have been proposed. Let us present two suboptimal sequential (stepwise) methods of feature selection: **forward** and **backward feature selection**.

Let us consider selecting the best m-feature subset from $n (m < n)$ features constituting an original pattern. Figure 7.15 shows an example of finding an $m = 3$ feature subset from an $n = 4$ feature pattern.

A **forward selection** search starts with individual evaluation of each feature. For each feature, a feature selection criterion, $J_{feature}$, is evaluated, and the feature with the best score (maximal value of the performance criterion) is selected for the next step of the search (a "winner" – an ancestor of the subtree). Then, in the second step, one additional feature is added to the selected "winner" feature (having the best value of the criterion) from previous step, forming all possible

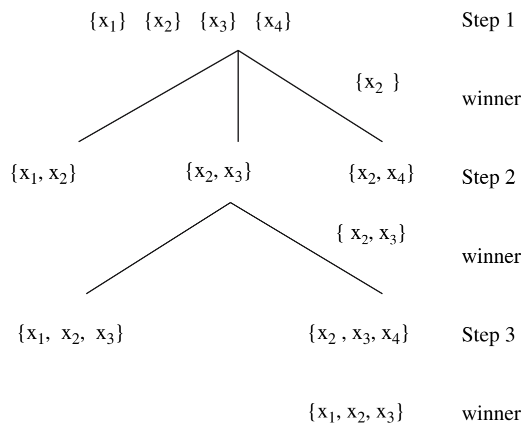

Figure 7.15. Sequential forward feature selection search.

two-feature subsets containing a "winner." Each subset with its pair of features is evaluated, and those presenting the maximal increase of the performance criterion are selected as a winner and successor of the next step. The procedure continues until the best m-feature subset (the "winner" of the m^{th} step) has been processed.

Algorithm: Feature selection by stepwise forward search.

Given: A data set T_{all} with N_{all} labeled patterns consisting of n features $X = \{x_1, x_2, \cdots, x_n\}$; a number m of features in the resulting subset of best features; and a feature subset evaluation criterion J_{feature} with a defined procedure for its computation based on a limited-size data set $T_{X_{\text{feature}}}$.

1. Set an initial "winner" feature subset as an empty set $X_{\text{winner},0} = \{\ \}$.
2. Set a step number $j = 1$.
3. Form all possible $n - j + 1$ subsets, with a total of j features, that contain a winning $j - 1$ feature subset $X_{\text{winner},j-1}$ from the previous step, with one new feature added.
4. Evaluate the feature selection criterion for each feature subset formed in step j. Select as a winner a subset $X_{\text{winner},j}$ with a larger increase Δ of the performance criterion J_{feature} as compared with the maximal criterion value (for the winner subset $X_{\text{winner},j-1}$) from the previous step.
5. If $j = m$, then stop. The winner $X_{\text{winner},j}$ subset in step j is the final selected subset of m features. Otherwise, set $j = j + 1$ and continue from step 3.

The forward selection algorithm provides a suboptimal solution, since it does not examine all possible subsets of features.

The basic forward selection procedure assumes that the number of features m in a resulting subset is known. This procedure will require exactly m steps. In some cases, the proper number of features m has to be found. This situation defines another search process with stopping criterion $J_{\text{feature,length}}$. Here, a possible stopping criterion for finding the proper number m of features in a final selected feature subset could be, for example, a defined threshold ϵ_{length} of maximal performance increase for two consecutive steps. In other words, the stopping point is reached when the increase in the feature selection criterion for the j^{th}-step winning feature subset $X_{\text{winner},j}$, as compared with the corresponding performance for a winner feature subset from the previous step $j - 1$, is less than the defined threshold ϵ_{length}:

$$J_{\text{feature,length}} = J_{\text{feature}}(X_{\text{winner},j}) - J_{\text{feature}}(X_{\text{winner},j-1}) < \epsilon_{\text{length}} \tag{419}$$

Backward selection is similar to forward selection, but it applies a reversed procedure of feature selection, starting with the entire feature set and eliminating features one at a time. In backward selection, assuming a known number m of final features, the search starts with the evaluation of the entire set of n features. For the entire feature set, a selection criterion J_{feature} is evaluated. Then, in the next step, all possible subsets containing features from the previous step with one feature discarded are formed and their performance criteria are evaluated. At each step, one feature, which gives the smallest decrease in the value the feature selection criterion included in the previous step, is discarded. The procedure continues until the best m-feature subset is found. Figure 7.16 depicts an example of finding an $m = 2$ optimal feature subset from an $n = 4$ feature pattern.

Algorithm: Feature selection by stepwise backward search.

Given: A data set T_{all} with N_{all} labeled patterns consisting of n features $X = \{x_1, x_2, \cdots, x_n\}$; a number m of features in the resulting subset of best features; and a feature subset evaluation criterion J_{feature} with a defined procedure for its computation based on a limited-size data set $T_{X_{\text{feature}}}$.

1. Evaluate a feature selection criterion $J_{\text{feature}}(X)$ for a set X of all n features.
2. Set a step number $j = 1$ with a list X of all n features.
3. Form all $n - j + 1$ possible subsets with $n - j$ features by discarding one feature at a time from the list of features of the previous step.
4. Evaluate a feature selection criterion for each feature subset formed in step j. Select as a "winner" a subset $X_{\text{winner},j}$ with the smallest decrease of a performance criterion $J_{\text{feature}}(X_{\text{winner},j})$ as compared with the criterion value from the previous step (which corresponds to its biggest value for this step from a pool of all subsets). The discarded feature from the previous step, which caused the creation of the winning subset $X_{\text{winner},j}$, is then discarded from a pool of features used in the next step, and winning subset becomes an ancestor of a deeper subtree.
5. If $j = m$, then stop: the winner subset in step j is the final selected subset of m features. Otherwise, set $j = j + 1$ and continue from step 3.

The forward selection algorithm provides a suboptimal solution, since it does not examine all possible subsets of features. The backward selection algorithm requires more intensive computations than the forward selection. Despite of similarities, both algorithms may provide different results for the same conditions.

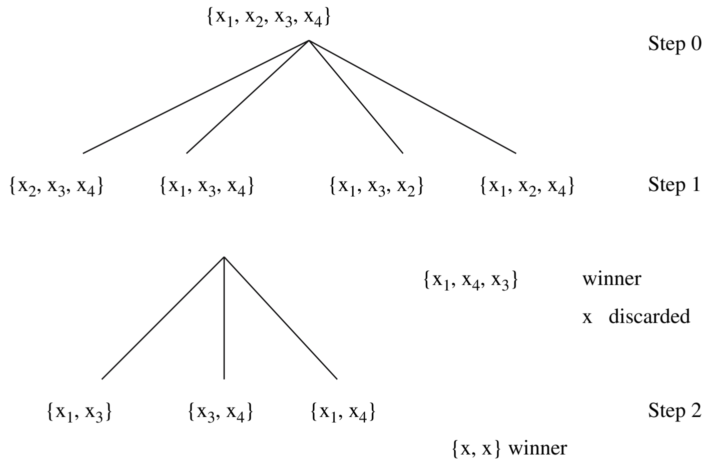

Figure 7.16. A sequential backward search.

If the number m of final features is a unknown a priori, then another optimal search should be employed. Finding the proper number m of features in the final selected feature subset could be realized in a manner similar to the method described earlier for forward selection.

The forward and backward search methods can be combined in several ways, allowing them to cover more feature subsets through increased computations, and thereby to find better suboptimal feature sets. For example, in the so-called **full stepwise search**, operations at each step start as in the backward search. All subsets created by removing one variable from the previous-step pool are evaluated. If the feature selection criterion decrease is below a defined threshold, then a variable is removed. If none of the variables provide a decrease below the threshold, then a variable is added, as in the forward search method.

Based on the concept of **Monte Carlo techniques**, several feature selection methods have been developed using probabilistic search. These methods make probabilistic choices of feature subsets in search of the best subset in a feature space. These random search methods can be used both for open loop and closed loop feature selection algorithms. In general, they do not require that a feature selection criterion must obey the monotonicity condition. Probabilistic algorithms use the inconsistency rate in open loop selection, as a feature goodness criterion and select the feature subset with the minimal number of features that has an inconsistency rate smaller than the predefined threshold. Allowing for features to have some degree of inconsistency opens a way to define the robust feature subset that is not only the best for a given data set but also potentially good for unseen cases (i.e., with generalization ability). The closed loop feature selection scheme has also been proposed for random searches with the ID3 algorithm as classifier.

These probabilistic methods are simple to implement and guarantee finding the best subset of features, if a required number of random trials for subset selection will be performed. These algorithms provide satisfactory results for highly correlated features.

Algorithm: Probabilistic (Monte Carlo) method of feature selection

Given: A data set T_{all} with N_{all} patterns labeled by classes and consisting of n features $X = \{x_1, x_2, \cdots, x_n\}$; a feature subset selection criterion J_{feature} with a defined procedure for its computation based on a limited-size data set $T_{X_{\text{feature}}}$; and a maximum number of random subset search trials *max_runs*.

1. Set initially the best-feature subset as equal to an original n-feature set $X_{\text{opt}} = X$. Compute the value of the criterion $J_{\text{feature}}(X_{\text{feature},0}) = J_{\text{feature}}(T_{\text{all}})$ for a data set T_{all}.
2. Set $j = 1$ (a search trial number).
3. From all possible 2^n feature subsets, select randomly a distinct subset of features $X_{\text{feature},j}$ (with number of features $1 \leq m_j \leq n$).
4. Create a reduced data set $T_{X_{\text{feature}},j}$ with all N_{all} cases with patterns constituted with m_j features from a subset $X_{\text{feature},j}$.
5. Compute the value of the criterion $J_{\text{feature}}(T_{X_{\text{feature},j}})$ for the data set $T_{X_{\text{feature},j}}$.
6. If $J_{\text{feature}}(X_{\text{feature},j}) > J_{\text{feature}}(X_{\text{feature},j-1})$, then set $X_{\text{best}} = X_{\text{feature},j}$ and continue from step 7. Otherwise, continue from step 7.
7. Set $j = j + 1$. If $j \leq max\_runs$, then stop; otherwise, continue from step 3.

The version of the probabilistic algorithm with the open loop inconsistency criterion defined earlier may be formed as follows.

Algorithm: Probabilistic (Monte Carlo) method of open loop feature selection with inconsistency criterion

Given: A data set T_{all} with N_{all} patterns labeled by classes and consisting of n features $X = \{x_1, x_2, \cdots, x_n\}$; a feature subset selection criterion $J_{\text{feature}} = J_{\text{inc}}$ with a defined procedure for its computation based on a limited-size data set $T_{X_{\text{feature}}}$; a rejection threshold β of feature subset inconsistency; and a maximum number of random subset search trials max_runs.

1. Set an initial value for the best (minimal) feature number as equal to the number n of all original features $m_{\text{best}} = n$. Set initially the best feature subset as equal to an original n-feature set $X_{\text{opt}} = X$.
2. Set $j = 1$ (a search trial number).
3. From all possible 2^n feature subsets, select randomly a distinct subset of features $X_{\text{feature},j}$ (with the number of features $1 \leq m_j \leq n$).
4. Compute the number of features in the subset

$$X_{feature,j}; \quad m_j = cardinality(X_{feature,j}) \tag{420}$$

5. Create a reduced data set $T_{X_{\text{feature},j}}$ with all N_{all} cases with patterns consisting of m_j features from a subset $X_{\text{feature},j}$.
6. Compute a value of the inconsistency criterion $J_{\text{inc}}(T_{X_{\text{feature},j}})$ for a data set $T_{X_{\text{feature},j}}$
7. If $m_j < m_{\text{best}}$ and $J_{\text{inc}}((X_{\text{feature},j}) < \beta$, then set $X_{\text{best}} = X_{\text{feature},j}$ and $m_{\text{best}} = m_j$, and continue from step 8. Otherwise, continue from step 8.
8. Set $j = j + 1$. If $j \leq max\_runs$, then stop; otherwise, continue from step 3.

Random selection of the feature subset X_{feature} from an original set of all ordered features $x_1, x_2, \cdots, x_n$ can be realized, for example, as follows. First, a random number generator uniformly distributed in [0,1] is executed n times. Each generated number r_i corresponds to one feature x_i. Then, if $r_i > 0.5$, a feature x_i is selected for a subset $X_{\text{feature},j}$; otherwise, it is not selected.

3.1.16. Feature Scaling

Feature scaling (weighting) is used in feature selection. If we consider different discriminatory power of pattern features, feature scaling can be described as assigning continuous weight values from the range [0, 1] to each feature, depending on its impact on prediction. Given an original set of features X forming an n-dimensional pattern $\mathbf{x}$, feature scaling is the transformation of $\mathbf{x}$ into $\mathbf{x}_s$ using the n-dimensional weights vector $\mathbf{w}$ ($w_i \in [0, 1]$):

$$x_{s,i} = w_i x_i, \quad (i = 1, 2, \cdots, n) \tag{421}$$

In the extreme situation of weight taken only to binary values $w_i \in \{0, 1\}$, feature scaling becomes feature selection. For $w_i = 1$, feature x_i is selected for the final pattern; otherwise, for $w_i = 0$, the feature is removed. Choosing the best feature weights is an optimization problem, which can be as difficult as optimal feature selection. Feature scaling also faces a feature dimensionality problem.

4. Summary and Bibliographical Notes

In this Chapter we have introduced feature selection and feature extraction methods. The most important topics discussed were the unsupervised **Principal Component Analysis** (PCA) and supervised **Fisher's linear discriminant** technique. These can be used for pattern projection, **feature extraction**, and dimensionality reduction. The other unsupervised method covered was the **Independent component analysis** (ICA) used for discovering unknown intrinsic independent variables in the data. ICA estimates unknown **mixing matrix** and **independent components** representing given data. It can be used for **blind source separation** and linear feature extraction and reduction. The **Singular Value Decomposition** method is used often used for extracting features from images and image compression. Feature extraction and compression can be also achieved by using **vector quantization**, **Fourier transform** and **wavelets**. Vector quantization is the technique of representing (encoding) input data patterns by a smaller finite set of code vectors that approximate the input pattern space. **Learning vector quantization** technique combines supervised learning and vector quantization. **Fourier analysis** can be used to preprocess time-series data and images. **Fourier transform** (both one- and two-dimensional) converts input data into **frequency domain** that provides better understanding and representation of the data. A **fast Fourier transform** is a computationally efficient algorithm for computing a **discrete Fourier transform**. **One-dimensional Fourier transform** is a fundamental technique in processing feature extraction for time-series and speech data, whereas **two-dimensional Fourier transform** is widely used in image processing, including feature extraction. **Wavelets analysis** provides multi-resolution approximation of a time-series signal and images using a fast-decaying oscillating waveform. Wavelets provide localization in time and in a space and a powerful technique for feature extraction from data. **Zernike moments** are used to extract **rotation-invariant** robust features from images using orthogonal moment invariants and orthogonal polynomials.

In **feature selection**, features are divided into **strongly relevant**, **weakly relevant**, and **irrelevant**. Feature selection methods are divided into **open loop** and **closed loop** (**wrapper**) methods. All feature selection methods depend on the selection criteria and search method. The **selection criteria** include minimum concept description, mutual information, inconsistency count, rough sets, and interclass separability. In the Chapter we have described **exhaustive search**, **branch and bound**, **feature ranking**, and **forward** and **backward** feature selection methods.

Feature selection and extraction methods, including PCA and Fisher's transformations are well presented in [6, 7, 12, 14, 16, 17, 26]. Independent component analysis is nicely described in [11], while the blind source separation problem is described in [4], and applications of ICA are presented in [24, 25]. A description of vector quantization can be found in [9, 15, 18]. Fourier transform and Wavelets are covered in [1, 5, 8, 21]. The Zernike moments were introduced in [28] and later extended in [13, 23]. The open loop feature selection methods include Focus [2] and Relief algorithms [16] and their extensions [3, 6, 12, 17, 22, 26]. The closed loop (wrapper) feature selection methods are described in [3, 6, 10, 12, 22, 23]. The sequential backward selection was introduced in [19], and the forward selection and stepwise methods were presented in [14]. The branch and bound search for feature selection (based on the idea of dynamic programming) is covered in [20, 27].

References

1. Ainsworth, W.A. 1988. *Speech Recognition by Machine*, Peter Peregrinus Ltd., London, UK
2. Almuallim, H., and Dietterich, T.G. 1992. Efficient algorithms for identifying relevant features. *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, 38–45. Vancouver, Canada
3. Bazan, J.G., Skowron, A., and Swiniarski, R. 2006. Rough sets and vague concept approximation: From sample approximation to adaptive learning, Transactions on Rough Sets V; Journal Subline, Lecture Notes in Computer Science 4100, Springer, Heidelberg, 39–62
4. Bell, A.J., and Sejnowski, T.J. 1995. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7:1129–1159
5. Burrus, C., Gopinath, R., and Guo, H. 1998. *Introduction to Wavelets and Wavelet Transformations: A Primer*, Prentice Hall
6. Cios, K.J., Pedrycz, W., and Swiniarski, R. 1998. *Data Mining Methods for Knowledge Discovery*, Kluwer
7. Duda, R.O., and Hart, P.E. 2001. *Pattern Recognition and Scene Analysis*, Wiley
8. Fant, C.G. 1973. *Speech Sounds and Features*, MIT Press
9. Gersho, A., and Gray, R. 1992. *Vector Quantization and Signal Compression*, Boston, Kluwer

10. Grzymala-Busse, J.W, Kostek, B., Swiniarski, R., and Szczuka, M. 2004. (Editors-in Chief of a special I volume) Transaction on Rough Sets I. In (Editors-in-Chief Peters, J., and Skowron, A.), Lecture Notes in Computer Sciences on Rough Sets, 3100, Springer, Berlin, New York, pp. 1–404
11. Hyvarinen, A., Karhunen, J., and Oja, E. 2001. *Independent Component Analysis*, John Wiley, New York
12. John, G., Kohavi, R., and Pfleger, K. 1994. Irrelevant features and the subset selection problem. *Proceedings of the Eleventh International Conference on Machine Learning* (ICML-94), 121–129, New Brunswick, NJ
13. Khotanzad, A., Hong, Y.H. 1990. Invariant image recognition by Zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):489–497
14. Kittler, J. 1986. Feature selection and extraction. In Young, T.Y., and Fu, K.S. (Eds.), *Handbook of Pattern Recognition and Image Processing*, Academic Press, 59–83
15. Kohonen, T. 1997. *Self-Organizing Maps*, Springer
16. Kononenko, I. 1994. Estimating attributes: Analysis and extension of Relief. *Proceedings of European Conference on Machine Learning*, 171–182, Catania, Italy
17. Langley, P. 1994. Selection of relevant features in machine learning. *Proceedings of the AAAI Fall Symposium on Relevance*, 140–144, Orlando, FL
18. Linde, Y., Buzo, A., and Gray, R. 1980. An algorithm for vector quantizer design. *IEEE Transaction on Communications*, 28(1):84–94
19. Marill, T., and Green, D. 1963. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9:11–17
20. Narendra, P.M., and Fukunaga, K. 1977. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C–26:917–922
21. Rabiner, L.R., and Juang, B.H. 1993. *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, N.J.
22. Skowron, A., Swiniarski, R., Synak, P., and Peters, J.F. 2004. Approximation Spaces and Information Granulation. Tsumoto, S., Slowinski, R., and Komorowski, J. (Eds.) *Rough Sets and Current Trends in Computing*, Proceedings of 4th International Conference, RSCTC 2004, Uppsala, Sweden, Springer, pp. 116–126
23. Swiniarski, R. 2004. Application of Zernike Moments, Independent Component Analysis, and Rough and Fuzzy Classifier for Hand-Written Character Recognition. In Klopotek, M.K., Wierzchon, S., and Trojanowski, K. (Eds.), *Intelligent Information Processing and Web Mining*. Proceedings of the International IIS:IIPWM'04 Conference. Zakopane, Poland, May 17–20, Springer, pp. 623–632
24. Swiniarski, R., Lim Hun Ki, Shin Joo Heon and Skowron, A. 2006. Independent Component Analysis, Principal Component Analysis and Rough Sets in Hybrid Mammogram Classification. *Proceedings of the 2006 International Conference on Image Processing, Computer Vision, & Pattern Recognition*, volume II, 640–645, Las Vegas
25. Swiniarski, R., and Skowron, A. 2004. Independent Component Analysis and Rough Sets in Face Recognition. In Grzymala-Busse, J., Kostek, B., Swiniarski, R., and Szczuka, M. (Editors-in Chief of a special I volume) Transaction on Rough Sets I. In (Editors-in-Chief Peters, J., and Skowron, A.), Lecture Notes in Computer Sciences on Rough Sets, 3100, Springer, Berlin, New York, pp. 392–404
26. Swiniarski, R. and Skowron, A. 2003. Rough sets methods in feature selection and recognition. *Pattern Recognition Letters*, 24(6):883–849
27. Yu, B., and Yuan, B. 1993. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition*, 26(6):883–889
28. Zernike, F. 1934. Beugungstheorie des schneidenverfahrens und seimer verbesserten form, der phasenkontrastmethode, *Physica*, 1:689–706

5. Exercises

1. Let us consider the data set containing patterns with three nominal attributes {MONITOR, OS, CPU} labeled by two categorical classes $c_1 = \text{Poor}$, $c_2 = \text{Good}$ (Table 7.4). For this data set (decision table):

(a) Compute a set of strongly relevant attributes (for the entire pattern **x**) with respect to a class attribute d, using the idea of core defined by rough sets theory.
(b) Compute sets of weakly relevant attributes (for a whole pattern **x**) with respect to a class attribute d using the idea of reduct defined by rough sets theory.
(c) For selected minimal reduct, design decision rules using the rough sets method.
(d) Find a minimal set of attributes describing all concepts in the data set using exhaustive search.

2. For a data set from Table 7.4, find an optimal set of attributes, using the open loop scheme with individual feature ranking, using the criteria of

(a) mutual information
(b) inconsistency count

3. Let us consider a data set containing 20 cases (Table 7.5). Each case is composed of the four feature patterns $\mathbf{x} \in \mathbb{R}^4$ labeled by categorical classes $c_1 = 1$ and $c_2 = 2$ (10 patterns in each class). A pattern's attributes take on real values.
For the considered data set, find the best feature subset, using the open loop feature selection method, with the following feature selection criteria:

(a) mutual information
(b) inconsistency count
(c) interclass separability

while applying

(a) exhaustive search
(b) branch and bound search
(c) sequential forward search

4. For a data set from Table 7.5, find the best feature subset, using the closed loop feature selection method, with the relative frequency of classification errors as a feature selection criterion. Use the k-nearest neighbors classifier for feature evaluation and also as a final classifier. For performance evaluation of classifiers, use the holdout error counting method (with partion of the data set used for design and the other portion held out for testing).
5. For a data set from Table 7.5, the provide principal component analysis (PCA) (global for all patterns for all classes):

Table 7.4. Example of the decision table PC.

Object	**Condition attributes**			**Decision attributes**
U	**C**			**D**
PC	**MONITOR**	**OS**	**CPU**	**d**
x_1	Color	DOS	486	Good
x_2	Color	Windows	Pentium	Good
x_3	Monochrome	DOS	Pentium	Poor
x_4	Color	Windows	486	Good
x_5	Color	DOS	386	Poor
x_6	Monochrome	Windows	486	Good
x_7	Monochrome	Windows	Pentium	Good
x_8	Color	Windows	386	Good

Table 7.5. A data set with real-valued attributes.

x_1	x_2	x_3	x_4	Class
0.8	0.7	1.1	0.5	1
1.3	0.6	0.9	1.3	1
1.9	1.7	0.3	0.6	1
2.0	0.1	0.3	1.8	1
1.1	1.6	0.1	1.9	1
0.1	0.2	2.1	2.3	1
2.2	2.4	0.3	1.3	1
2.1	1.9	0.5	2.9	1
1.3	2.7	0.3	2.2	1
2.9	3.2	0.8	0.1	1
5.2	2.5	10.6	5.5	2
7.8	9.5	12.2	6.5	2
4.5	7.6	3.9	2.3	2
8.9	6.2	2.9	8.3	2
4.2	5.4	11.3	9.3	2
3.2	8.7	2.5	15.9	2
6.6	6.3	10.3	12.2	2
9.9	2.2	6.8	15.1	2
12.8	4.2	9.8	4.1	2
4.9	9.2	4.8	7.9	2

(a) Compute the covariance matrix
(b) Find an optimal Karhunen-Loéve transform
(c) Transform the original patterns to full-size principal component space. Compute the inverse transform
(d) Transform the original patterns to the reduced principal component space (consider $m = 1, 2, 3$). Compute the inverse transform. Compute the reconstruction error: a) numerically (as a sum of squared errors); b) as a sum of trailing eigenvalues (remove the least significant principal components).

6. For a data set from Table 7.5,

(a) Provide principal component analysis (PCA) with transformation of data into the full-size principal component space. For the full-size PCA feature vector, compute and evaluate the k-nearest neighbors classifier.
(b) Select the first m principal components ($m = 4, 3, 2, 1$) as a reduced feature vector. For this feature pattern, design and evaluate the k-nearest neighbors classifier.

7. Generate samples for two time signals $\sin(0.2 * x)$ and $\sin(2 * x)$, for x from 0 to 10π with the step $\pi/100$. Mix the signals linearly using the mixing matrix

$$x = 0 : \pi/100 : 10 * \pi; \qquad H = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix}$$

Separate sources using ICA method. Solve the problems

(a) Without whitening.
(b) With whitening and with reduction of the dimension to 1.
(c) With whitening implemented as PCA, and with reduction of the dimension to 1.

8. Design a Matlab program to perform vector quantization on a gray-scale image using a 4×4 pixel block as a quantization unit. Unfold the pixel block as a 16-element block vector $\mathbf{x}$ by concatenating the subsequent rows of the block. Form the training set T_{tra} containing subsequent block vectors for all blocks of an image (for example, considering the sequence of blocks for an image from the left to the right, etc.), design your optimal codebook using all block vectors from the training data set. Use the generalized Lloyd algorithm to find the optimal codebook. Select the size of codebook (say, $M = 128$). Then quantize the image represented by the training set T_{tra} using your codebook. Find the quantization quality using a PSNR measure. Provide quantization of the image for codebook size equal to 32, 64, and 256. Reconstruct an image from the compressed representation. Compare the results. Provide quantization experiments and compare the resulting quality of quantization.
   As a result of quantization of an image, we obtain the codebook with M codevectors labeled by M indices. The quantization result of the considered image is the vector of encoding indices T_{ind}, which entry contains an index of encoding codevector representing the corresponding block vector in the set T_{tra}. In order to find the reconstructed image from T_{ind} each encoding codevector for a given block is folded into an image block (through row-by-row folding).
9. Synthesize (or get from the Internet) a data set of gray-scale images of handwritten digits $(0, 1, \cdots, 9)$ (at least 10 instances per digit).

   (a) Write in MatLab language (or in another language) a program implementing the computation of complex Zernike moments.
   (b) Extract from each image the Zernike moments with maximal order (6,6) and (12, 12). Design Zernike patterns and construct supervised training and testing sets (with 10 classes).
   (c) Design and test the following classifiers: k-nearest neighbors, error back-propagation neural networks, and learning vector quantization.
   (d) Apply principal component analysis (PCA) to obtained supervised data sets. Project all patterns into principal component space, and reduce the projected patterns to a heuristically selected number. Design and test the classifiers listed in the previous question.
   (e) Form the training and test sets containing raw patterns composed from digit images as concatenated columns. Apply principal component analysis (PCA) to such raw data sets. Project all patterns into the principal component space, and reduce the length of the projected patterns to a heuristically selected number. Design and test the handwritten digit recognitions using classifiers listed in the previous question.

10. Get (from the Internet) a data set of gray-scale images of human faces (with at least 10 classes, and at least 10 instances of face for a class). Extract the face image features and form a pattern. Compare different methods of feature extraction and pattern forming and their impact on classifier accuracy. Write the required operations in Matlab language. Consider the following methods of feature extraction:

    (a) Principal Component Analysis (PCA)
    (b) Independent Component Analysis (ICA)
    (c) Singular Values Decomposition (SVD)
    (d) Zernike Moments
    (e) Two-dimensional Fourier transform (power spectrum features)
    (f) Synthetic features (see previous Chapters) derived from the normalized power spectrum map of two-dimensional Fourier transform (power spectrum features)
    (g) Haar wavelets
    (h) Morlet wavelets