CHAPTER

CLASSIFICATION: A TOUR OF THE CLASSICS

7

CHAPTER OUTLINE

7.1 INTRODUCTION

The classification task was introduced in Chapter 3. There, it was pointed out that, in principle, one could employ the same loss functions as those used for regression in order to optimize the design of a classifier; however, for most cases in practice, this is not the most reasonable way to attack such problems. This

Machine Learning. http://dx.doi.org/10.1016/B978-0-12-801522-3.00007-0

is because in classification the output variable, y, is of a *discrete nature*; hence, different measures than those used for the regression task are more appropriate for quantifying performance quality.

The goal of this chapter is to present a number of widely used loss functions and methods. Most of the techniques covered are conceptually simple and constitute the basic pillars on which classification is built. Besides their pedagogical importance, these techniques are still in use in a number of practical applications and often form the basis for the development of more advanced methods, to be covered later in the book.

The classical Bayesian classification rule; the notion of minimum distance classifiers; the logistic regression loss function; classification trees; and the method of combining classifiers, including the powerful technique of boosting, will be discussed. The perceptron rule, although it boasts to be among the most basic classification rules, will be treated in Chapter 18 and it will be used as the starting point for introducing neural networks and deep learning techniques. Support vector machines are treated in the framework of reproducing Kernel Hilbert spaces, in Chapter 11.

In a nutshell, this chapter can be considered a beginner's tour of the task of designing classifiers.

7.2 BAYESIAN CLASSIFICATION

In Chapter 3, a linear classifier was designed via the least-squares (LS) cost function. However, the LS criterion cannot serve well the needs of the classification task. In Chapters 3 and 6, we have proved that the LS estimator is an efficient one only if the conditional distribution of the output variable, y, given the feature values, $\boldsymbol{x}$, follows a Gaussian distribution of a special type. However, in classification, the dependent variable is discrete, hence it is not Gaussian; thus, the use of the LS criterion cannot be justified, in general. We will return to this issue in Section 7.10 (Remarks 7.7), when the LS criterion is discussed against other loss functions used in classification.

In this section, the classification task will be approached via a different path, inspired by the *Bayesian decision theory*. In spite of its conceptual simplicity, which ties very well with common sense, Bayesian classification possesses a strong optimality flavor with respect to the probability of error; that is, the probability of wrong decisions/class predictions that a classifier commits.

Bayesian classification rule: Given a set of M classes, $\omega_i,\ i = 1, 2, \ldots, M$, and the respective *posterior probabilities* $P(\omega_i|\boldsymbol{x})$, classify an unknown feature vector, $\boldsymbol{x}$, according to the rule:

$$\boxed{\text{Assign } \boldsymbol{x} \text{ to } \omega_i = \arg\max_{\omega_j} P(\omega_j|\boldsymbol{x}), \quad j = 1, 2, \ldots, M.} \tag{7.1}$$

In words, the unknown pattern, represented by $\boldsymbol{x}$, is assigned to the class for which the posterior probability becomes maximum.

Note that prior to receiving any observation, our uncertainty concerning the classes is expressed via the prior probabilities, denoted by $P(\omega_i),\ i = 1, 2, \ldots, M$. Once the observation $\boldsymbol{x}$ has been obtained, this extra information removes part of our original uncertainty, and the related statistical information is now provided by the posterior probabilities, which are then used for the classification.

Employing in (7.1) Bayes theorem,

$$P(\omega_j|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_j)P(\omega_j)}{p(\boldsymbol{x})}, \quad j = 1, 2, \ldots, M, \tag{7.2}$$

where $p(\boldsymbol{x}|\omega_j)$ are the respective conditional probability distribution densities (pdf), the Bayesian classification rule becomes

$$\text{Assign } \boldsymbol{x} \text{ to } \omega_i = \arg\max_{\omega_j} p(\boldsymbol{x}|\omega_j)P(\omega_j), \quad j = 1, 2 \ldots, M. \tag{7.3}$$

Note that the data pdf, $p(\boldsymbol{x})$, in the denominator of (7.2) does not enter in the maximization task, because it is a positive quantity independent of the classes ω_j; hence, it does not affect the maximization. In other words, the classifier depends on the a priori class probabilities and the respective conditional pdfs. Also, note that

$$p(\boldsymbol{x}|\omega_j)P(\omega_j) = p(\omega_j, \boldsymbol{x}) := p(y, \boldsymbol{x}).$$

The last equation verifies what said in Chapter 3: the Bayesian classifier is a generative modeling technique.

We now turn our attention to how one can obtain estimates of the involved quantities. Recall that in practice, all one has at one's disposal is a set of training data, from which estimates of the prior probabilities as well as the conditional pdfs must be obtained. Let us assume that we are given a set of training points, $(y_n, \boldsymbol{x}_n) \in D \times \mathbb{R}^l$, $n = 1, 2, \ldots, N$, where D is the set of class labels, and consider the general task comprising M classes. Assume that each class, ω_i, $i = 1, 2, \ldots, M$, is represented by N_i points in the training set, with $\sum_{i=1}^{M} N_i = N$. Then, the a priori probabilities can be approximated by

$$P(\omega_i) \approx \frac{N_i}{N}, \quad i = 1, 2, \ldots, M. \tag{7.4}$$

For the conditional pdfs, $p(\boldsymbol{x}|\omega_i)$, $i = 1, 2 \ldots, M$, any method for estimating pdfs can be mobilized. For example, one can assume a known parametric form for each one of the conditionals and adopt the maximum likelihood (ML) method, discussed in Section 3.10, or the maximum a posteriori (MAP) estimator, discussed in Section 3.11.1, in order to obtain estimates of the parameters using the training data from each one of the classes. Another alternative is to resort to nonparametric histogram-like techniques, such as Parzen windows and the k-nearest neighbor density estimation techniques, as discussed in Section 3.15. Other methods for pdf estimation can also be employed, such as mixture modeling, to be discussed later in Chapter 12. The interested reader may also consult [52, 53].

The Bayesian classifier minimizes the misclassification error

In Section 3.4, it was pointed out that the goal of designing a classifier is to partition the space in which the feature vectors lie into regions, and associate each one of the regions to one and only one class. For a two-class task (the generalization to more classes is straightforward), let $\mathcal{R}_1, \mathcal{R}_2$ be the two regions in $\mathbb{R}^l$, where we decide in favor of class ω_1 and ω_2, respectively. The probability of classification error is given by

$$P_e = P(\boldsymbol{x} \in \mathcal{R}_1, \boldsymbol{x} \in \omega_2) + P(\boldsymbol{x} \in \mathcal{R}_2, \boldsymbol{x} \in \omega_1). \tag{7.5}$$

That is, it is equal to the probability of the feature vector to belong to class ω_1 (ω_2) *and* to lie in the "wrong" region $\mathcal{R}_2$ ($\mathcal{R}_1$) in the feature space.

Equation (7.5) can be written as

$$\boxed{P_e = P(\omega_2) \int_{\mathcal{R}_1} p(\boldsymbol{x}|\omega_2)d\boldsymbol{x} + P(\omega_1) \int_{\mathcal{R}_2} p(\boldsymbol{x}|\omega_1)d\boldsymbol{x} : \quad \text{Probability of Error.}} \tag{7.6}$$

It turns out that the Bayesian classifier, as defined in (7.3), minimizes P_e with respect to $\mathcal{R}_1$ and $\mathcal{R}_2$ [17, 52]. This is also true for the general case of M classes (Problem 7.1).

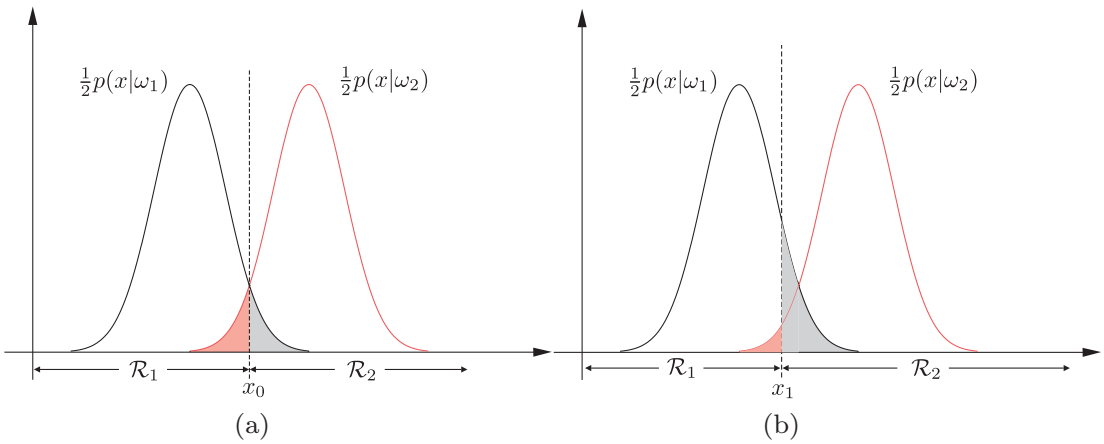

FIGURE 7.1

(a) The classification error probability for dividing the feature space, according to the Bayesian optimal classifier, is equal to the area of the shaded region. (b) Moving the threshold value away from the value corresponding to the optimal Bayes rule increases the probability of error, as is indicated by the increase of the area of the corresponding shaded region.

Figure 7.1a demonstrates geometrically the optimality of the Bayesian classifier for the two-class one-dimensional case and assuming equiprobable classes ($P(\omega_1) = P(\omega_2) = 1/2$). The region $\mathcal{R}_1$, to the left of the threshold value, x_0, corresponds to $p(x|\omega_1) > p(x|\omega_2)$, and the opposite is true for region $\mathcal{R}_2$. The probability of error is equal to the area of the shaded region, which is equal to the sum of the two integrals in (7.6). In Figure 7.1b, the threshold has been moved away from the optimal Bayesian value, and as a result the probability of error, given by the total area of the corresponding shaded region, increases.

7.2.1 AVERAGE RISK

Because in classification the dependent variable (label), y, is of a discrete nature, the classification error probability may seem like the most natural cost function to be optimized. However, this is not always true. In certain applications, not all errors are of the same importance. For example, in a medical diagnosis system, committing an error by predicting the class of a finding in an X-ray image as being "malignant" while its true class is "normal" is less significant than an error the other way around. In the former case, the wrong diagnosis will be revealed in the next set of medical tests. However, the opposite may have unwanted consequences. For such cases, one uses an alternative to the probability of error cost function that puts relative weights on the errors according to their importance. This cost function is known as the *average risk* and it results in a rule that resembles that of the Bayesian classifier, yet it is slightly modified due to the presence of the weights.

For the M-class problem, the *risk* or *loss* associated with class ω_k is defined as

$$r_k = \sum_{i=1}^{M} \lambda_{ki} \int_{\mathcal{R}_i} p(\boldsymbol{x}|\omega_k)\, d\boldsymbol{x}, \tag{7.7}$$

where, $\lambda_{kk} = 0$ and λ_{ki} is the weight that controls the significance of committing an error by assigning a pattern from class ω_k to class ω_i. The *average risk* is given by

$$r = \sum_{k=1}^{M} P(\omega_k) r_k = \sum_{i=1}^{M} \int_{\mathcal{R}_i} \left(\sum_{k=1}^{M} \lambda_{ki} P(\omega_k) p(\boldsymbol{x}|\omega_k) \right) \mathrm{d}\boldsymbol{x}. \tag{7.8}$$

The average risk is minimized if we partition the input space by selecting each $\mathcal{R}_i$ (where we decide in favor of class ω_i) so that each one of the M integrals in the summation becomes minimum; this is achieved if we adopt the rule

$$\text{Assign } \boldsymbol{x} \text{ to } \omega_i : \sum_{k=1}^{M} \lambda_{ki} P(\omega_k) p(\boldsymbol{x}|\omega_k) < \sum_{k=1}^{M} \lambda_{kj} P(\omega_k) p(\boldsymbol{x}|\omega_k), \quad \forall j \neq i,$$

or equivalently,

$$\boxed{\text{Assign } \boldsymbol{x} \text{ to } \omega_i : \sum_{k=1}^{M} \lambda_{ki} P(\omega_k|\boldsymbol{x}) < \sum_{k=1}^{M} \lambda_{kj} P(\omega_k|\boldsymbol{x}), \quad \forall j \neq i.} \tag{7.9}$$

For the two-class case, it is readily seen that the rule becomes

$$\text{Assign } \boldsymbol{x} \text{ to } \omega_1 \ (\omega_2) \text{ if } : \lambda_{12} P(\omega_1|\boldsymbol{x}) > (<) \ \lambda_{21} P(\omega_2|\boldsymbol{x}),$$

or equivalently

$$\text{Assign } \boldsymbol{x} \text{ to } \omega_1 \ (\omega_2) \text{ if } : \lambda_{12} P(\omega_1) p(\boldsymbol{x}|\omega_1) > (<) \ \lambda_{21} P(\omega_2) p(\boldsymbol{x}|\omega_2). \tag{7.10}$$

If one sets $P'(\omega_1) := \lambda_{12} P(\omega_1)$ $(P'(\omega_2) := \lambda_{21} P(\omega_2))$, one may view the effect of the weights as modifying the respective prior class probabilities by relatively increasing the value of the more important class.

It is common to consider the weights λ_{ij} as defining an $M \times M$ matrix

$$L := [\lambda_{ij}], \quad i,j = 1, 2, \ldots, M, \tag{7.11}$$

which is known as the *loss* matrix. Note that if we set $\lambda_{ki} = 1,\ k = 1, 2, \ldots, M, i = 1, 2, \ldots, M, k \neq i$, then we obtain the Bayes rule (verify it).

Remarks 7.1.

- *The reject option*: Bayesian classification relies on the maximum value of the posterior probabilities, $P(\omega_i|\boldsymbol{x}),\ i = 1, 2, \ldots, M$. However, often in practice, it may happen that for some value, $\boldsymbol{x}$, the maximum value is comparable to the values the posterior obtains for other classes. For example, in a two-class task, it may turn out that $P(\omega_1|\boldsymbol{x}) = 0.51$ and $P(\omega_2|\boldsymbol{x}) = 0.49$. If this happens, it may be more sensible not to make a decision for this particular pattern, $\boldsymbol{x}$. This is known as the reject option. If such a decision scenario is adopted, a user-defined threshold value, θ, is chosen and classification is carried out only if the maximum posterior is larger than this threshold value, so that, $P(\omega_i|\boldsymbol{x}) > \theta$. Otherwise, no decision is taken. Similar arguments can be adopted for the average risk classification.

Example 7.1. In a two-class, one-dimensional classification task, the data in the two classes are distributed according to the following two Gaussians:

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right),$$

and

$$p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-1)^2}{2}\right).$$

The problem is more sensitive with respect to errors committed on patterns form class ω_1, which is expressed via the following loss matrix:

$$L = \begin{bmatrix} 0 & 1 \\ 0.5 & 0 \end{bmatrix}.$$

In other words, $\lambda_{12} = 1$ and $\lambda_{21} = 0.5$. The two classes are considered equiprobable. Derive the threshold value, x_r, which partitions the feature space, $\mathbb{R}$, into the two regions, $\mathcal{R}_1$, $\mathcal{R}_2$, in which we decide in favor of class ω_1 and ω_2, respectively. What is the value of the threshold when the Bayesian classifier is used instead?

Solution: According to the average risk rule, the region for which we decide in favor of class ω_1 is given by

$$\mathcal{R}_1 : \lambda_{12}\frac{1}{2}p(x|\omega_1) > \lambda_{21}\frac{1}{2}p(x|\omega_2),$$

and the respective threshold value, x_r, is computed by the equation

$$\exp\left(-\frac{x_r^2}{2}\right) = 0.5\exp\left(-\frac{(x_r-1)^2}{2}\right),$$

which, after taking the logarithm and solving the respective equation, trivially results in

$$x_r = \frac{1}{2}(1 - 2\ln 0.5).$$

The threshold for the Bayesian classifier results if we set $\lambda_{21} = 1$, which gives

$$x_B = \frac{1}{2}.$$

The geometry is shown in Figure 7.2. In other words, the use of the average risk moves the threshold to the right of the value corresponding to the Bayesian classifier; that is, it enlarges the region in which we decide in favor of the more significant class, ω_1. Note that this would also be the case if the two classes were not equiprobable, as shown by $P(\omega_1) > P(\omega_2)$ (for our example, $P(\omega_1) = 2P(\omega_2)$).

7.3 DECISION (HYPER)SURFACES

The goal of any classifier is to partition the feature space into regions. The partition is achieved via points in ($\mathbb{R}$), curves in ($\mathbb{R}^2$), surfaces in ($\mathbb{R}^3$), and hypersurfaces in ($\mathbb{R}^l$). Any hypersurface, S, is expressed in terms of a function

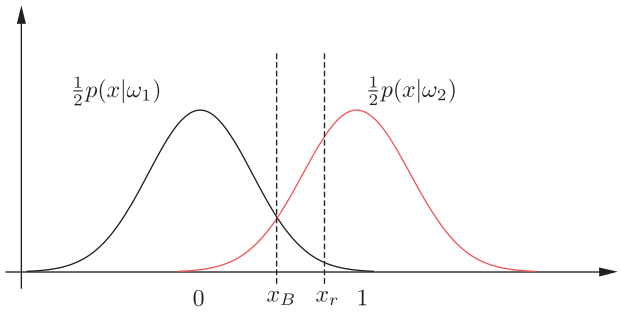

FIGURE 7.2

The class distributions and the resulting threshold values for the two cases of Example 7.1. Note that minimizing the average risk enlarges the region in which we decide in favor of the most sensitive class, ω_1.

$$g : \mathbb{R}^l \longmapsto \mathbb{R},$$

and it comprises all the points such that

$$S = \left\{ \boldsymbol{x} \in \mathbb{R}^l : \quad g(\boldsymbol{x}) = 0 \right\}.$$

Recall that all points lying on one side of this hypersurface score $g(\boldsymbol{x}) > 0$ and all the points on the other side score $g(\boldsymbol{x}) < 0$. The resulting (hyper)surfaces are knows as *decision (hyper)surfaces*, for obvious reasons. Take as an example the case of the two-class Bayesian classifier. The respective decision hypersurface is (implicitly) formed by

$$g(\boldsymbol{x}) := P(\omega_1|\boldsymbol{x}) - P(\omega_2|\boldsymbol{x}) = 0. \tag{7.12}$$

Indeed, we decide in favor of class ω_1 (region $\mathcal{R}_1$) if $\boldsymbol{x}$ falls on the positive side of the hypersurface defined in (7.12), and in favor of ω_2 for the points falling on the negative side (region $\mathcal{R}_2$). This is illustrated in Figure 7.3. At this point, recall the reject option from Remarks 7.1. Points where no decision is taken are those that lie close to the decision hypersurface.

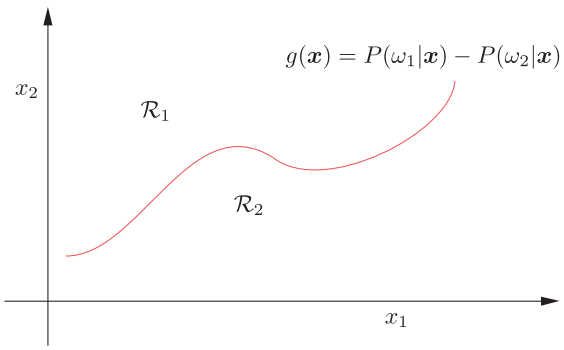

FIGURE 7.3

The Bayesian classifier implicitly forms hypersurfaces defined by $g(\boldsymbol{x}) = P(\omega_1|\boldsymbol{x}) - P(\omega_2|\boldsymbol{x}) = 0$.

Once we move away from the Bayesian concept of designing classifiers (as we will soon see, and this will be done for a number of reasons), different families of functions for selecting $g(\boldsymbol{x})$ can be adopted and the specific form will be obtained via different optimization criteria, which are not necessarily related to the probability of error/average risk.

In the sequel, we focus on investigating the form that the decision hypersurfaces take for the special case of the Bayesian classifier and where the data in the classes are distributed according to the Gaussian pdf. This can provide further insight into the way a classifier partitions the feature space and it will also lead to some useful implementations of the Bayesian classifier, under certain scenarios. For simplicity, the focus will be on two-class classification tasks, but the results are trivially generalized to the more general M-class case.

7.3.1 THE GAUSSIAN DISTRIBUTION CASE

Assume that the data in each class are distributed according to the Gaussian pdf, so that,

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{(2\pi)^{l/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^{\mathrm{T}}\Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right), \quad i = 1, 2, \ldots, M.$$

Because the logarithmic function is a monotonically increasing one, it does not affect the maximum of a function. Thus, taking into account the exponential form of the Gaussian, the computations can be facilitated if the Bayesian rule is expressed in terms of the following functions:

$$g_i(\boldsymbol{x}) := \ln P(\omega_i|\boldsymbol{x}) = \ln p(\boldsymbol{x}|\omega_i) + \ln P(\omega_i), \quad i = 1, 2, \ldots, M, \tag{7.13}$$

and search for the class for which the respective function scores the maximum value. Such functions are also known as *discriminant functions*.

Let us now focus on the two-class classification task. The decision hypersurface, associated with the Bayesian classifier, is expressed as

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x}) = 0, \tag{7.14}$$

which, after plugging into (7.13) the specific forms of the Gaussian conditionals, and after a bit of trivial algebra, becomes

$$\begin{aligned} g(\boldsymbol{x}) = & \underbrace{\frac{1}{2}\left(\boldsymbol{x}^{\mathrm{T}}\Sigma_2^{-1}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\Sigma_1^{-1}\boldsymbol{x}\right)}_{\text{quadratic terms}} \\ & \underbrace{+\boldsymbol{\mu}_1^{\mathrm{T}}\Sigma_1^{-1}\boldsymbol{x} - \boldsymbol{\mu}_2^{\mathrm{T}}\Sigma_2^{-1}\boldsymbol{x}}_{\text{linear terms}} \\ & \underbrace{-\frac{1}{2}\boldsymbol{\mu}_1^{\mathrm{T}}\Sigma_1^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^{\mathrm{T}}\Sigma_2^{-1}\boldsymbol{\mu}_2 + \ln\frac{P(\omega_1)}{P(\omega_2)} + \frac{1}{2}\ln\frac{|\Sigma_2|}{|\Sigma_1|}}_{\text{constant terms}} = 0. \end{aligned} \tag{7.15}$$

This is of a quadratic nature, hence the corresponding (hyper)surfaces are *(hyper)quadrics*, including (hyper)ellipsoids, (hyper)parabolas, hyperbolas. Figure 7.4 shows two examples, in the two-dimensional space, corresponding to $P(\omega_1) = P(\omega_2)$, and

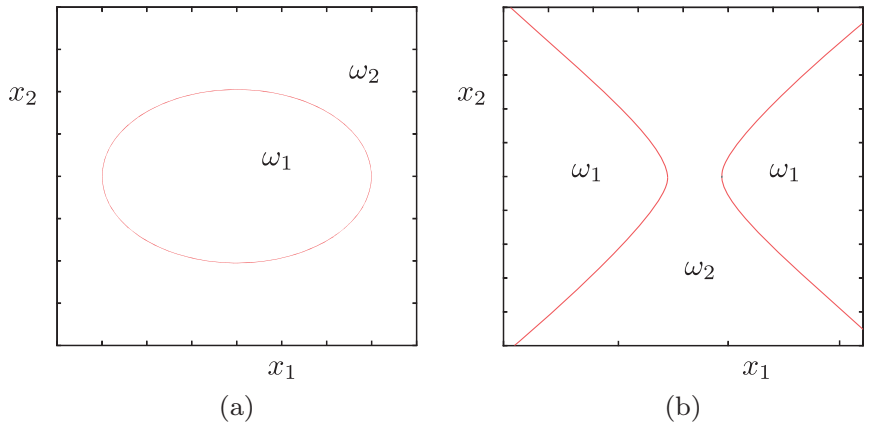

FIGURE 7.4

The Bayesian classifier for the case of Gaussian distributed classes partitions the feature space via quadrics. (a) The case of an ellipse and (b) the case of a hyperbola.

$$\text{(a)} \quad \boldsymbol{\mu}_1 = [0,0]^{\mathrm{T}},\ \boldsymbol{\mu}_2 = [4,0]^{\mathrm{T}},\ \Sigma_1 = \begin{bmatrix} 0.3 & 0.0 \\ 0.0 & 0.35 \end{bmatrix},\ \Sigma_2 = \begin{bmatrix} 1.2 & 0.0 \\ 0.0 & 1.85 \end{bmatrix},$$

and

$$\text{(b)} \quad \boldsymbol{\mu}_1 = [0,0]^{\mathrm{T}},\ \boldsymbol{\mu}_2 = [3.2,0]^{\mathrm{T}},\ \Sigma_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.75 \end{bmatrix},\ \Sigma_2 = \begin{bmatrix} 0.75 & 0.0 \\ 0.0 & 0,1 \end{bmatrix},$$

respectively. In Figure 7.4a, the resulting curve for scenario (a) is an ellipse, and in Figure 7.4b, the corresponding curve for scenario (b) is a hyperbola.

Looking carefully at (7.15), it is readily noticed that once the covariance matrices for the two classes become equal, then the quadratic terms cancel out and the discriminant function becomes linear; thus, the corresponding hypersurface is a hyperplane. That is, under the previous assumptions, the optimal Bayesian classifier becomes a *linear* classifier, which after some straightforward algebraic manipulations (try it) can be written as

$$g(\boldsymbol{x}) = \boldsymbol{\theta}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0) = 0, \tag{7.16}$$

$$\boldsymbol{\theta} := \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \tag{7.17}$$

$$\boldsymbol{x}_0 := \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) - \ln\frac{P(\omega_1)}{P(\omega_2)}\frac{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2}{||\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2||_{\Sigma^{-1}}}, \tag{7.18}$$

where Σ is common to the two-class covariance matrix and

$$||\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2||_{\Sigma^{-1}} := \sqrt{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)},$$

is the Σ^{-1}-norm of the vector $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$; alternatively, this is also known as the *Mahalanobis distance* between $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$. For $\Sigma = I$ this becomes the Euclidean distance.

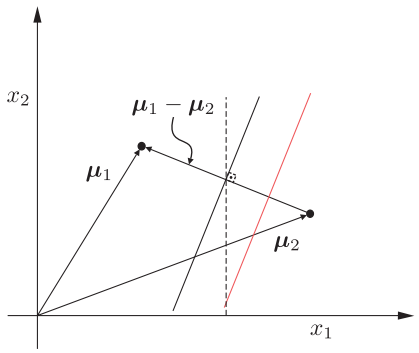

FIGURE 7.5

The full gray line corresponds to the Bayesian classifier for two equiprobable Gaussian classes that share a common covariance matrix of the specific form, $\Sigma = \sigma^2 I$; the line bisects the segment joining the two mean values (minimum Euclidean distance classifier). The red one is for the same case but for $P(\omega_1) > P(\omega_2)$. The dotted line is the optimal classifier for equiprobable classes, and a common covariance of a more general form, different than $\sigma^2 I$ (minimum Mahalanobis distance classifier).

Figure 7.5 shows three cases for the two-dimensional space. The full black line corresponds to the case of equiprobable classes with a covariance matrix of the special form, $\Sigma = \sigma^2 I$. The corresponding decision hyperplane is given by

$$g(\boldsymbol{x}) = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0) = 0. \tag{7.19}$$

The separating line (hyperplane) crosses the middle point of the line segment joining the mean value points, $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ ($\boldsymbol{x}_0 = \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)$). Also, it is perpendicular to this segment, defined by the vector $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$, as is readily verified by the above hyperplane definition. The red line corresponds to the case where $P(\omega_1) > P(\omega_2)$. It gets closer to the mean value point of class ω_2, thus enlarging the region where one decides in favor of the more probable class. Finally, the dotted line corresponds to the equiprobable case with the common covariance matrix being of a more general form, $\Sigma \neq \sigma^2 I$. The separating hyperplane crosses $\boldsymbol{x}_0$ but it is rotated in order to be perpendicular to the vector $\Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, according to (7.16)–(7.17). An unknown point is classified according to the side of the respective hyperplane on which it lies.

What was said before for the two-class task is generalized to the more general M-class problem; the separating hypersurfaces of two contiguous regions, $\mathcal{R}_i$, $\mathcal{R}_j$, associated with two classes, ω_i and ω_j, obey the same arguments as the ones adopted before. For example, assuming that all covariance matrices are the same, then the regions are partitioned via hyperplanes, as illustrated in Figure 7.6. Moreover, each region $\mathcal{R}_i$, $i = 1, 2, \ldots, M$, is convex (Problem 7.2); in other words, joining any two points within $\mathcal{R}_i$, all the points lying on the respective segment lie in $\mathcal{R}_i$, too.

Two special cases are of particular interest, leading to a simple classification rule. The rule will be expressed for the general M-class problem.

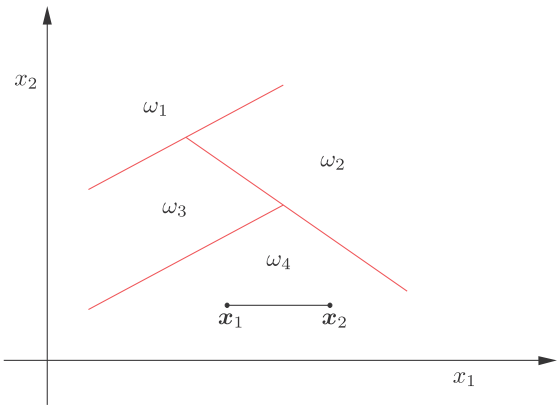

FIGURE 7.6

When data are distributed according to the Gaussian distribution and they share the same covariance matrix in all classes, the feature space is partitioned via hyperplanes, which form polyhedral regions. Note that each region is associated with one class and it is convex.

Minimum distance classifiers

- *Minimum Euclidean distance classifier*: Under the assumptions of (a) Gaussian distributed data in each one of the classes, (b) equiprobable classes, and (c) common covariance matrix in all classes of the special form $\Sigma = \sigma^2 I$ (individual features are independent and share a common variance), the Bayesian classification rule is equivalent with

$$\text{Assign } \boldsymbol{x} \text{ to class } \omega_i : \ i = \arg\min_j (\boldsymbol{x} - \boldsymbol{\mu}_j)^{\mathrm{T}} (\boldsymbol{x} - \boldsymbol{\mu}_j), \quad j = 1, 2, \ldots M. \tag{7.20}$$

  This is a direct consequence of the Bayesian rule under the adopted assumptions. In other words, the Euclidean distance of $\boldsymbol{x}$ is computed from the mean values of all classes and it is assigned to the class for which this distance becomes smaller.

  For the case of the two classes, this classification rule corresponds to the full black line of Figure 7.5. Indeed, recalling our geometry basics, any point that lies on the left of this hyperplane is closer to $\boldsymbol{\mu}_1$ than to $\boldsymbol{\mu}_2$. The opposite is true for any point lying on the right of the hyperplane.
- *Minimum Mahalanobis distance classifier*: Under the previously adopted assumptions, but with the covariance matrix being of the more general form, $\Sigma \neq \sigma^2 I$, the rule becomes

$$\text{Assign } \boldsymbol{x} \text{ to class } \omega_i : \ i = \arg\min_j (\boldsymbol{x} - \boldsymbol{\mu}_j)^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_j), \quad j = 1, 2, \ldots M. \tag{7.21}$$

  Thus, instead of looking for the minimum Euclidean distance, one searches for the minimum Mahalanobis distance; the latter is a weighted form of the Euclidean distance, in order to account for the shape of the underlying Gaussian distributions [52]. For the two-class case, this rule corresponds to the dotted line of Figure 7.5.

Remarks 7.2.

- In Statistics, adopting the Gaussian assumption for the data distribution is sometimes called *linear discriminant analysis* (LDA) or *quadratic discriminant analysis* (QDA), depending on the adopted assumptions with respect to the underlying covariance matrices, which will lead to either linear or quadratic discriminant functions, respectively. In practice, the ML method is usually employed in order to obtain estimates of the unknown parameters, namely, the mean values and the covariance matrices. Recall from Example 3.5 of Chapter 3 that the ML estimate of the mean value of a Gaussian pdf, obtained via N observations, $\boldsymbol{x}_n$, $n = 1, 2, \ldots, N$, is equal to

$$\hat{\boldsymbol{\mu}}_{ML} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n.$$

  Moreover, the ML estimate of the covariance matrix of a Gaussian distribution, using N observations, is given by (Problem 7.4),

$$\hat{\Sigma}_{ML} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ML})^{\mathrm{T}}. \tag{7.22}$$

  This corresponds to a biased estimator of the covariance matrix. An unbiased estimator results if (Problem 7.5),

$$\hat{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ML})^{\mathrm{T}}.$$

  Note that the number of parameters to be estimated in the covariance matrix is $O(l^2/2)$, taking into account its symmetry.

Example 7.2.

Consider a two-class classification task in the two-dimensional space with, $P(\omega_1) = P(\omega_2) = 1/2$. Generate 100 points, 50 from each class. The data from each class, ω_i, $i = 1, 2$, stem from a corresponding Gaussian, $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, where

$$\boldsymbol{\mu}_1 = [0, -2]^{\mathrm{T}}, \quad \boldsymbol{\mu}_2 = [0, 2]^{\mathrm{T}},$$

and (a)

$$\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1.2 & 0.4 \\ 0.4 & 1.2 \end{bmatrix},$$

or (b)

$$\Sigma_1 = \begin{bmatrix} 1.2 & 0.4 \\ 0.4 & 1.2 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & -0.4 \\ -0.4 & 1 \end{bmatrix}.$$

Figure 7.7 shows the decision curves formed by the Bayesian classifier. Observe that in the case of Figure 7.7a, the classifier turns out to be a linear one, while for the case of Figure 7.7b, it is nonlinear of a parabola shape.

Example 7.3. In a two-class classification task, the data in each one of the classes are distributed according to the Gaussian distribution, with mean values, $\boldsymbol{\mu}_1 = [0, 0]^{\mathrm{T}}$ and $\boldsymbol{\mu}_2 = [3, 3]^{\mathrm{T}}$, respectively, sharing a common covariance matrix

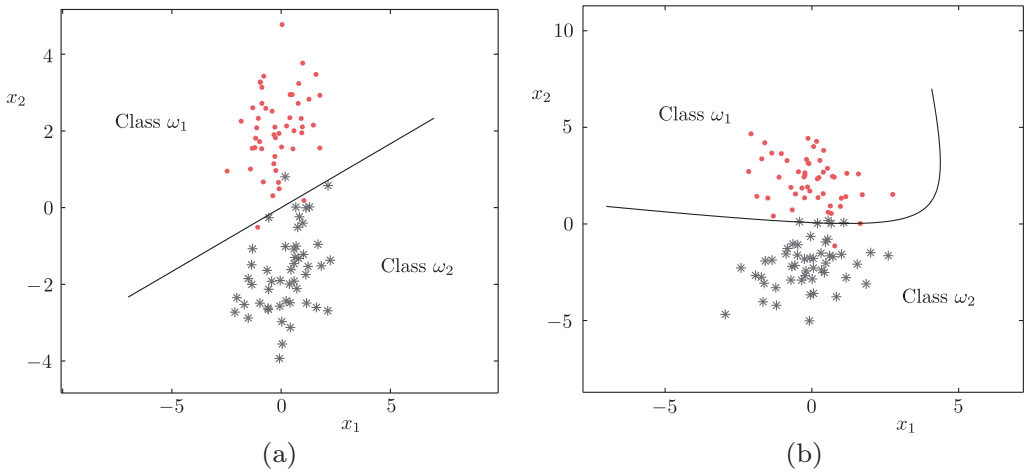

FIGURE 7.7

If the data in the feature space follow a Gaussian distribution in each one of the classes, then the Bayesian classifier is (a) a hyperplane, if all the covariance matrices are equal; (b) otherwise, it is a quadric hypersurface.

$$\Sigma = \begin{bmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{bmatrix}.$$

Use the Bayesian classifier to classify the point $\boldsymbol{x} = [1.0, 2.2]^{\mathrm{T}}$ into one of the two classes.

Because the classes are distributed according to the Gaussian distribution and share the same covariance matrix, the Bayesian classifier is equivalent with the minimum Mahalanobis distance classifier. The (square) Mahalanobis distance of the point $\boldsymbol{x}$ from the mean value of class ω_1 is

$$d_1^2 = [1.0, 2.2] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.2 \end{bmatrix} = 2.95,$$

where the matrix in the middle on the left-hand side is the inverse of the covariance matrix. Similarly for class ω_2, we obtain that

$$d_2^2 = [-2.0, -0.8] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} -2.0 \\ -0.8 \end{bmatrix} = 3.67.$$

Hence, the pattern is assigned to class ω_1, because its distance from $\boldsymbol{\mu}_1$ is smaller compared to that from $\boldsymbol{\mu}_2$. Verify that if the Euclidean distance were used instead, the pattern would be assigned to class ω_2.

7.4 THE NAIVE BAYES CLASSIFIER

We have already seen that in case the covariance matrix is to be estimated, the number of unknown parameters is of the order of $\mathcal{O}(l^2/2)$. For high dimensional spaces, besides the fact that this estimation task is a formidable one, it also requires a large number of data points, in order to obtain statistically

good estimates and avoid overfitting, as discussed in Chapter 3. In such cases, one has to be content with suboptimal solutions. Indeed, adopting an optimal method, while using bad estimates of the involved parameters can lead to a bad overall performance.

The *naive Bayes classifier* is a typical and popular example of a suboptimal classifier. The basic assumption is that the components (features) in the feature vector are statistically independent; hence, the joint pdf can be written as a product of l marginals,

$$p(\boldsymbol{x}|\omega_i) = \prod_{k=1}^{l} p(x_k|\omega_i), \quad i = 1, 2, \ldots, M.$$

Having adopted the Gaussian assumption, each one of the marginals is described by two parameters, the mean and the variance; this leads to a total of $2l$ per class, unknown parameters to be estimated. This is a substantial saving compared to the $O(l^2/2)$ number of parameters. It turns out that this simplistic assumption can end up with better results compared to the optimal Bayes classifier, when the size of the data samples is limited.

Although the naive Bayes classifier was introduced in the context of Gaussian distributed data, its use is also justified for the more general case. In Chapter 3, we discussed the curse of dimensionality issue and it was stressed that high dimensional spaces are sparsely populated. In other words, for a fixed finite number of data points, N, within a cube of fixed size for each dimension, the larger the dimension of the space, the larger the average distance between any two points becomes. Hence, in order to get good estimates of a set of parameters in large spaces, increased number of data is required. Roughly speaking, if N data points are needed in order to get a good enough estimate of a pdf in the real axis, as happens when using the histogram method, N^l data points would be needed for similar accuracy in an l-dimensional space. Thus, by assuming the features to be mutually independent, one will end up estimating l one-dimensional pdfs, hence substantially reducing the need for data.

The independence assumption is a common one in a number of machine learning and statistics tasks. As we will see in Chapter 15, one can adopt more "mild" independence assumptions that lie in between the two extremes, which are full independence and full dependence.

7.5 THE NEAREST NEIGHBOR RULE

Although the Bayesian rule provides the optimal solution with respect to the classification error probability, its application requires the estimation of the respective conditional pdfs; this is not an easy task, once the dimensionality of the feature space assumes relatively large values. This paves the way for considering alternative classification rules, which becomes our focus from now on.

The *k-nearest neighbor* (k-NN) rule is a typical nonparametric classifier and it is one among the most popular and well-known classifiers. In spite of its simplicity, it is still in use and stands next to more elaborate schemes.

Consider N training points, $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$, for an M-class classification task. At the heart of the method lies a parameter k, which is a user-defined parameter. Once k is selected, then given a pattern, $\boldsymbol{x}$, assign it to the class in which the majority of its k nearest (according to a metric, e.g., Euclidean or Mahalanobis distance) neighbors, among the training points, belong. The parameter k should not be a multiple of M, in order to avoid ties. The simplest form of this rule is to assign the pattern to the class in which its nearest neighbor belongs, meaning $k = 1$.

It turns out that this conceptually simple rule tends to the Bayesian classifier if (a) $N \longrightarrow \infty$, (b) $k \longrightarrow \infty$, and (c) $k/N \longrightarrow 0$. More specifically, it can be shown that the classification errors P_{NN} and $P_{k\text{NN}}$ satisfy, asymptotically, the following bounds [14],

$$P_{\text{B}} \leq P_{\text{NN}} \leq 2P_{\text{B}}, \tag{7.23}$$

for the $k = 1$ NN rule and,

$$P_{\text{B}} \leq P_{k\text{NN}} \leq P_{\text{B}} + \sqrt{\frac{2P_{\text{NN}}}{k}}, \tag{7.24}$$

for the more general k-NN version. P_{B} is the error corresponding to the optimal Bayesian classifier. The previous two formulae are quite interesting. Take for example (7.23). It says that the simple NN rule will never give an error larger than twice the optimal one. If, for example, $P_{\text{B}} = 0.01$, then $P_{\text{NN}} \leq 0.02$. This is not bad for such a simple classifier. All this says is that if one has an easy task (as indicated by the very low value of P_{B}), the NN rule can also do a good job. This, of course, is not the case if the problem is not an easy one and larger error values are involved. The bound in (7.24) says that for large values of k (provided, of course, N is large enough), the performance of the k-NN tends to that of the optimal classifier. In practice, one has to make sure that k does not get values close to N, but remains a relatively small fraction of it.

One may wonder how a performance close to the optimal classifier can be obtained, even in theory and asymptotically, because the Bayesian classifier exploits the statistical information for the data distribution while the k-NN does not take into account such information. The reason is that if N is a very large value (hence the space is densely populated) and k is a relatively small number, with respect to N, then the nearest neighbors will be located very close to $\boldsymbol{x}$. Then, due to the *continuity* of the involved pdfs, the values of their posterior probabilities will be close to $P(\omega_i|\boldsymbol{x})$, $i = 1, 2, \ldots, M$. Furthermore, for large enough k, the majority of the neighbors must come from the class that scores the maximum value of the posterior probability given $\boldsymbol{x}$.

A major drawback of the k-NN rule is that every time a new pattern is considered, its distance from all the training points has to be computed, then selecting the k closest to it points. To this end, various searching techniques have been suggested over the years. The interested reader may consult [52] for a related discussion.

Remarks 7.3.

- The use of the k-nearest rule concept can also be adopted in the context of the regression task. Given an observation, $\boldsymbol{x}$, one searches for its k closer input vectors in the training set, denoted as $\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(k)}$, and computes an estimate of the output value, $\hat{y}$, as an average of the respective outputs in the training set, represented by

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_{(i)}.$$

Example 7.4. An example that illustrates the decision curves for a two-class classification task in the two-dimensional space, obtained by the Bayesian, the 1-NN and the 13-NN classifier, is given in Figure 7.8. A number of $N = 100$ data are generated for each class by Gaussian distributions. The decision curve of the Bayes classifier has the form of a parabola, while the 1-NN classifier exhibits a highly nonlinear nature. The 13-NN rule forms a decision line close to the Bayesian one.

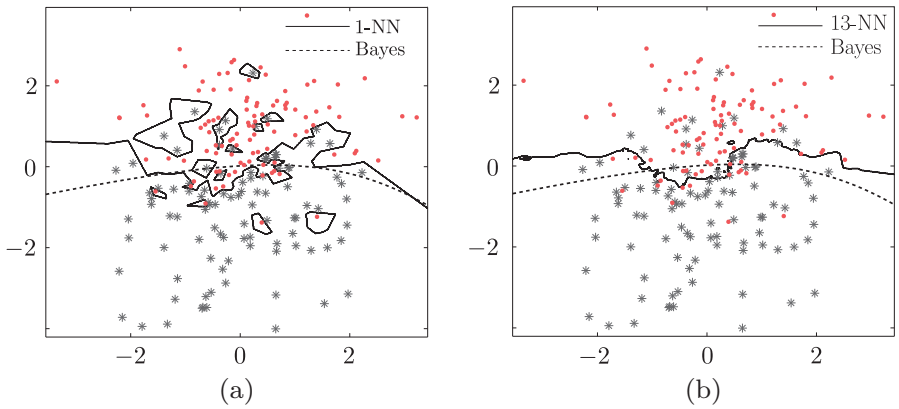

FIGURE 7.8

A two-class classification task. The dotted curve corresponds to the optimal Bayesian classifier. The full line curves correspond to (a) the 1-NN and (b) the 13-NN classifiers. Observe that the 13-NN is closer to the Bayesian one.

7.6 LOGISTIC REGRESSION

In Bayesian classification, the assignment of a pattern in a class is performed based on the posterior probabilities, $P(\omega_i|\boldsymbol{x})$. The posteriors are estimated via the respective conditional pdfs, which is not, in general, an easy task. The goal in this section is to model the posterior probabilities directly, via the *logistic regression* method. This name has been established in the statistics community, although the model refers to classification and not to regression. This is a typical example of the discriminative modeling approach, where the distribution of data is of no interest.

The two-class case: The starting point is to model the ratio of the posteriors as

$$\boxed{\ln\frac{P(\omega_1|\boldsymbol{x})}{P(\omega_2|\boldsymbol{x})} = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}: \quad \text{Two-class Logistic Regression,}} \tag{7.25}$$

where the constant term, θ_0, has been absorbed in $\boldsymbol{\theta}$. Taking into account that

$$P(\omega_1|\boldsymbol{x}) + P(\omega_2|\boldsymbol{x}) = 1,$$

and defining

$$t := \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x},$$

it is readily seen that the model in (7.25) is equivalent to

$$P(\omega_1|\boldsymbol{x}) = \sigma(t) \tag{7.26}$$

$$\sigma(t) := \frac{1}{1+\exp(-t)}, \tag{7.27}$$

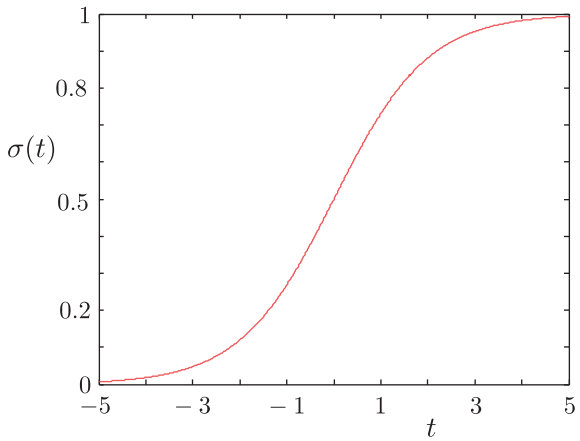

FIGURE 7.9

The sigmoid link function.

and

$$P(\omega_2|\boldsymbol{x}) = 1 - P(\omega_1|\boldsymbol{x}) = \frac{\exp(-t)}{1+\exp(-t)}. \tag{7.28}$$

The function $\sigma(t)$ is known as the *logistic sigmoid* or *sigmoid link* function and it is shown in Figure 7.9.

Although it may sound a bit mystical as to how one thought of such a model, it suffices to look more carefully at (7.13)–(7.15) to demystify it. Assuming the data in the classes follow Gaussian distributions with $\Sigma_1 = \Sigma_2 \equiv \Sigma$ and for simplicity that $P(\omega_1) = P(\omega_2)$, the latter of the previously stated equations is written as

$$\ln \frac{P(\omega_1|\boldsymbol{x})}{P(\omega_2|\boldsymbol{x})} = (\boldsymbol{\mu_1} - \boldsymbol{\mu}_2)^{\mathrm{T}} \Sigma^{-1} \boldsymbol{x} + \text{constants}. \tag{7.29}$$

In other words, when the distributions underlying the data are Gaussians with a common covariance matrix, then the log ratio of the posteriors is a linear function. Thus, in logistic regression, all we do is adopt such a model, irrespective of the data distribution. Moreover, even if the data are distributed according to Gaussians, it may still be preferable to adopt the logistic regression formulation instead of that in (7.29). In the latter formulation, the covariance matrix has to be estimated, amounting to $\mathcal{O}(l^2/2)$ parameters. The logistic regression formulation only involves $l+1$ parameters. That is, once we know about the linear dependence of the log ratio on $\boldsymbol{x}$, we can use this a priori information to simplify the model. Of course, assuming that the Gaussian assumption is valid, if one can obtain good estimates of the covariance matrix, employing this extra information can lead to more efficient estimates, in the sense of lower variance. The issue is treated in Ref. [18]. This is natural, because more information concerning the distribution of the data is exploited. In practice, it turns out that using the logistic regression is, in general, a safer bet compared to the linear discriminant analysis (LDA).

The parameter vector, $\boldsymbol{\theta}$, is estimated via the ML method applied on the set of training samples, $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$, $y_n \in \{0, 1\}$. The likelihood function can be written as

$$P(y_1, \ldots, y_N; \boldsymbol{\theta}) = \prod_{n=1}^{N} \left(\sigma(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n)\right)^{y_n} \left(1 - \sigma(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n)\right)^{1-y_n}. \tag{7.30}$$

Usually, we consider the negative log-likelihood given by

$$L(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \big(y_n \ln s_n + (1 - y_n) \ln(1 - s_n)\big), \tag{7.31}$$

where

$$s_n := \sigma(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n). \tag{7.32}$$

The log-likelihood cost function in (7.31) is also known as the *cross-entropy* error. Minimization of $L(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is carried out iteratively by any iterative minimization scheme, such as the steepest descent or Newton's method. Both schemes need the computation of the respective gradient, which in turn is based on the derivative of the sigmoid link function (Problem 7.6)

$$\frac{d\sigma(t)}{dt} = \sigma(t)\big(1 - \sigma(t)\big). \tag{7.33}$$

The gradient is given by (Problem 7.7)

$$\begin{aligned} \nabla L(\boldsymbol{\theta}) &= \sum_{n=1}^{N} (s_n - y_n) \boldsymbol{x}_n \\ &= X^{\mathrm{T}} (\boldsymbol{s} - \boldsymbol{y}), \end{aligned} \tag{7.34}$$

where

$$X^{\mathrm{T}} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N], \quad \boldsymbol{s} := [s_1, \ldots, s_N]^{\mathrm{T}}, \quad \boldsymbol{y} = [y_1, \ldots, y_N]^{\mathrm{T}}.$$

The Hessian matrix is given by (Problem 7.8)

$$\begin{aligned} \nabla^2 L(\boldsymbol{\theta}) &= \sum_{n=1}^{N} s_n (1 - s_n) \boldsymbol{x}_n \boldsymbol{x}_n^{\mathrm{T}} \\ &= X^{\mathrm{T}} R X, \end{aligned} \tag{7.35}$$

where

$$R := \mathrm{diag}\big\{s_1(1 - s_1), \ldots, s_N(1 - s_N)\big\}. \tag{7.36}$$

Note that because $0 < s_n < 1$, by definition of the sigmoid link function, matrix R is positive definite; hence, the Hessian matrix is also positive definite (Problem 7.9). This is a necessary and sufficient condition for convexity.[1] Thus, the negative log-likelihood function is convex, which guarantees the existence of a unique minimum (e.g., [3]).

[1] Convexity is discussed in more detail in Chapter 8.

Two of the possible iterative minimization schemes to be used are

- Steepest descent

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i X^{\mathrm{T}}(\boldsymbol{s}^{(i-1)} - \boldsymbol{y}). \tag{7.37}$$

- Newton's scheme

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu_i \left(X^{\mathrm{T}} R^{(i-1)} X\right)^{-1} X^{\mathrm{T}}(\boldsymbol{s}^{(i-1)} - \boldsymbol{y}) \\ &= \left(X^{\mathrm{T}} R^{(i-1)} X\right)^{-1} X^{\mathrm{T}} R^{(i-1)} \boldsymbol{z}^{(i-1)}\end{aligned} \tag{7.38}$$

where

$$\boldsymbol{z}^{(i-1)} := X\boldsymbol{\theta}^{(i-1)} - \left(R^{(i-1)}\right)^{-1} (\boldsymbol{s}^{(i-1)} - \boldsymbol{y}). \tag{7.39}$$

Equation (7.38) is a weighted version of the LS solution (Chapters 3 and 6); however, the involved quantities are iteration-dependent and the resulting scheme is known as *iterative reweighted least squares* scheme (IRLS) [50].

Maximizing the likelihood may run into problems if the training data set is linearly separable. In this case, any point on a hyperplane, $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x} = 0$, that solves the classification task and separates the samples from each class (note that there are infinite many such hyperplanes), results in $\sigma(\boldsymbol{x}) = 0.5$, and every training point from each class is assigned a posterior probability equal to one. Thus, ML forces the logistic sigmoid to become a step function in the feature space and equivalently $||\boldsymbol{\theta}|| \to \infty$. This can lead to overfitting and it is remedied by including a regularization term, $||\boldsymbol{\theta}||^2$, in the respective cost function.

The M-class case: For the more general M-class classification task, the logistic regression is defined for $m = 1, 2, \ldots, M$, as

$$\boxed{P(\omega_m|\boldsymbol{x}) = \frac{\exp(\boldsymbol{\theta}_m^{\mathrm{T}}\boldsymbol{x})}{\sum_{j=1}^{M} \exp(\boldsymbol{\theta}_j^{\mathrm{T}}\boldsymbol{x})} : \quad \text{Multiclass Logistic Regression.}} \tag{7.40}$$

The previous definition is easily brought into the form of a linear model for the log ratio of the posteriors. Divide, for example, by $P(\omega_M|\boldsymbol{x})$ to obtain

$$\ln \frac{P(\omega_m|\boldsymbol{x})}{P(\omega_M|\boldsymbol{x})} = (\boldsymbol{\theta}_m - \boldsymbol{\theta}_M)^T \boldsymbol{x} = \hat{\boldsymbol{\theta}}_m^T \boldsymbol{x}.$$

Let us define, for notational convenience,

$$\phi_{nm} := P(\omega_m|\boldsymbol{x}_n), \quad n = 1, 2, \ldots, N, \; m = 1, 2, \ldots, M,$$

and

$$t_m := \boldsymbol{\theta}_m^{\mathrm{T}}\boldsymbol{x}, \quad m = 1, 2, \ldots, M.$$

The likelihood function is now written as

$$P(\mathbf{y}; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M) = \prod_{n=1}^{N} \prod_{m=1}^{M} (\phi_{nm})^{y_{nm}}, \tag{7.41}$$

where $y_{nm} = 1$ if $\boldsymbol{x}_n \in \omega_m$ and zero otherwise. The respective negative log-likelihood function becomes

$$L(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M) = -\sum_{n=1}^{N}\sum_{m=1}^{M} y_{nm} \ln \phi_{nm}. \tag{7.42}$$

Minimization with respect to $\boldsymbol{\theta}_m$, $m = 1, \ldots, M$, takes place iteratively. To this end, the following gradients are used (Problems 7.10-7.12):

$$\frac{\partial \phi_{nm}}{\partial t_j} = \phi_{nm}(\delta_{mj} - \phi_{nj}), \tag{7.43}$$

where δ_{mj} is one if $m = j$ and zero otherwise. Also,

$$\nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M) = \sum_{n=1}^{N} (\phi_{nj} - y_{nj})\boldsymbol{x}_n. \tag{7.44}$$

The respective Hessian matrix is an $(lM) \times (lM)$ matrix, comprising $l \times l$ blocks. Its k, j block element is given by

$$\nabla_{\boldsymbol{\theta}_k} \nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M) = \sum_{n=1}^{N} \phi_{nj}(\delta_{kj} - \phi_{nk})\boldsymbol{x}_n \boldsymbol{x}_n^T. \tag{7.45}$$

The Hessian matrix is also positive definite, which guarantees uniqueness of the minimum as in the two-class case.

Remarks 7.4.

- *Probit regression*: Instead of using the logistic sigmoid function in (7.26) (for the two-class case), other functions can also be adopted. A popular function in the statistical community is the *probit* function, which is defined as

$$\begin{aligned} \Phi(t) &:= \int_{-\infty}^{t} \mathcal{N}(z|0, 1)dz \\ &= \frac{1}{2}\left(1 + \frac{1}{\sqrt{2}}\text{erf}(t)\right), \end{aligned} \tag{7.46}$$

  where erf is the *error* function defined as

$$\text{erf}(t) = \frac{2}{\sqrt{\pi}} \int_0^t \exp\left(-\frac{z^2}{2}\right) dz.$$

  In other words, $P(\omega_1|t)$ is modeled to be equal to the probability of a normalized Gaussian variable to lie in the interval $(-\infty, t]$. The graph of the probit function is very similar to that of the logistic one.

7.7 FISHER'S LINEAR DISCRIMINANT

We now turn our focus to designing linear classifiers. In other words, irrespective of the data distribution in each class, we decide to partition the space in terms of hyperplanes, so that

$$g(\boldsymbol{x}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x} + \theta_0 = 0. \tag{7.47}$$

We have dealt with the task of designing linear classifiers in the framework of the LS cost in Chapter 3. In this section, the unknown parameter vector will be estimated via a path that exploits a number of important notions relevant to classification. The method is known as *Fisher's discriminant* and it can be dressed up with different interpretations. Thus, its significance lies not only in its practical use but also in its pedagogical value.

Two of the major phases in designing a pattern recognition system are the *feature generation* and *feature selection* phases. Selecting information-rich features is of paramount importance. If "bad" features are selected, whatever smart classifier one adopts, the performance is bound to be poor. Feature generaton/selection techniques are treated in detail in Refs. [52, 53], to which the interested reader may refer to for further information. At this point, we only touch on a few notions that are relevant to our current design of a linear classifier. Let us first quantify what a "bad" and a "good" feature is. The main goal in selecting features, and, thus, in selecting the feature space in which one is going to work, can be summarized this way: Select the features to create a feature space in which the points, which represent the training patterns, are distributed such as to have

Large Between-Classes Distance
and
Small Within-Class Variance.

Figure 7.10 illustrates three different choices for the case of two-dimensional feature spaces. Common sense dictates that the choice in Figure 7.10c is the best one; the points in the three classes form groups that lie relatively far away from each other, and at the same time the data in each class are compactly clustered together. The worst of the three choices is that of Figure 7.10b, where data in each class are spread around their mean value and the three groups are relatively close to each other. The goal in feature selection is to develop measures that quantify the "slogan" given in the box above. The notion

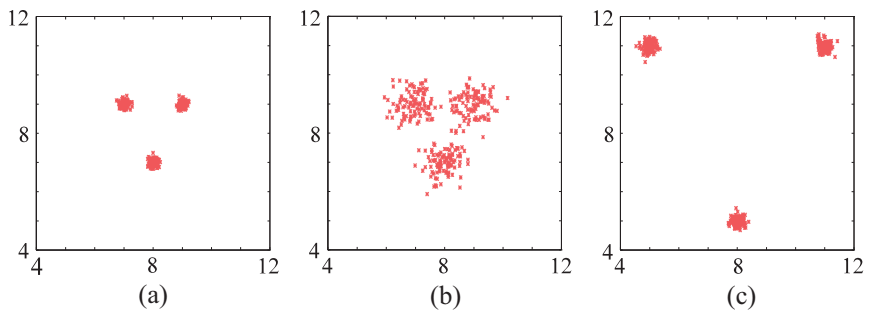

FIGURE 7.10

Three different choices of two-dimensional feature spaces: (a) small within-class variance and small between-classes distance; (b) large within class variance and small between classes distance; and (c) small within class variance and large between-classes distance. The last one is the best choice out of the three.

of *scatter matrices* is of relevance to us here. Although we could live without these definitions, it is a good opportunity to present this important notion and put our discussion in a more general context.

- *Within-class scatter matrix*

$$\Sigma_w = \sum_{k=1}^{M} P(\omega_k)\Sigma_k, \tag{7.48}$$

  where Σ_k is the covariance matrix of the points in the kth among M classes. In words, Σ_w is the average covariance matrix of the data in the specific l-dimensional feature space.
- *Between-classes scatter matrix*

$$\Sigma_b = \sum_{m=1}^{M} P(\omega_m)(\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)^{\mathrm{T}}, \tag{7.49}$$

  where $\boldsymbol{\mu}_0$ is the overall mean value defined by

$$\boldsymbol{\mu}_0 = \sum_{m=1}^{M} P(\omega_m)\boldsymbol{\mu}_m. \tag{7.50}$$

  Another commonly used related matrix is the following:
- *Mixture scatter matrix*

$$\Sigma_m = \Sigma_w + \Sigma_b. \tag{7.51}$$

A number of criteria that measure the "goodness" of the selected feature space are built around these scatter matrices; three typical examples are: [23, 52]:

$$J_1 := \frac{\mathrm{trace}\{\Sigma_m\}}{\mathrm{trace}\{\Sigma_w\}}, \quad J_2 = \frac{|\Sigma_m|}{|\Sigma_w|}, \quad J_3 = \mathrm{trace}\{\Sigma_w^{-1}\Sigma_b\}, \tag{7.52}$$

where $|\cdot|$ denotes the determinant of a matrix.

The two-class case: In Fisher's linear discriminant analysis, the emphasis in Eq. (7.47) is only on $\boldsymbol{\theta}$; the bias term, θ_0, is left out of the discussion. The inner product $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$ can be viewed as the projection of $\boldsymbol{x}$ along the vector $\boldsymbol{\theta}$. From geometry, we know that the respective projection is also a vector, $\boldsymbol{y}$, given by (e.g., Section 5.6)

$$\boldsymbol{y} = \frac{\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}}{||\boldsymbol{\theta}||}\frac{\boldsymbol{\theta}}{||\boldsymbol{\theta}||}.$$

From now on, we will focus on the scalar value of the projection, $y := \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$, and ignore the scaling factor in the denominator, because scaling all features by the same value has no effect on our discussion. The goal, now, is to select that direction, $\boldsymbol{\theta}$, so that after projecting along this direction, (a) the data in the two classes are as far away as possible from each other, and (b) the respective variances of the points around their means, in each one of the classes, are as small as possible. A criterion that quantifies the aforementioned goal is *Fisher's discriminant ratio* (FDR), defined as

$$\boxed{\mathrm{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}: \quad \text{Fisher's Discriminant Ratio,}} \tag{7.53}$$

where, μ_1 and μ_2 are the (scalar) mean values of the two classes, after the projection along $\boldsymbol{\theta}$, meaning

$$\mu_i = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\mu}_i, \quad i = 1, 2.$$

However, we have that

$$\begin{aligned}(\mu_1 - \mu_2)^2 &= \boldsymbol{\theta}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{\theta} = \boldsymbol{\theta}^{\mathrm{T}} S_b \boldsymbol{\theta}, \\ S_b &:= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}.\end{aligned}$$

Note that if the classes are equiprobable, S_b is a scaled version of the between-classes scatter matrix in (7.49) (under this assumption, $\boldsymbol{\mu}_0 = 1/2(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)$), and we have

$$(\mu_1 - \mu_2)^2 \propto \boldsymbol{\theta}^{\mathrm{T}} \Sigma_b \boldsymbol{\theta}. \tag{7.54}$$

Moreover,

$$\sigma_i^2 = \mathbb{E}\left[(\mathrm{y} - \mu_i)^2\right] = \mathbb{E}\left[\boldsymbol{\theta}^T(\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T\boldsymbol{\theta}\right] = \boldsymbol{\theta}^T \Sigma_i \boldsymbol{\theta}, \quad i = 1, 2, \tag{7.55}$$

which leads to

$$\sigma_1^2 + \sigma_2^2 = \boldsymbol{\theta}^{\mathrm{T}} S_w \boldsymbol{\theta},$$

where $S_w = \Sigma_1 + \Sigma_2$. Note that if the classes are equiprobable, S_w becomes a scaled version of the within-class scatter matrix defined in (7.48), and we have that

$$\sigma_1^2 + \sigma_2^2 \propto \boldsymbol{\theta}^{\mathrm{T}} \Sigma_w \boldsymbol{\theta}. \tag{7.56}$$

Combining (7.53), (7.54), and (7.56) and neglecting the proportionality constants, we end up with

$$\boxed{\mathrm{FDR} = \frac{\boldsymbol{\theta}^{\mathrm{T}} \Sigma_b \boldsymbol{\theta}}{\boldsymbol{\theta}^{\mathrm{T}} \Sigma_w \boldsymbol{\theta}} : \quad \text{Generalized Rayleigh Quotient.}} \tag{7.57}$$

Our goal now becomes that of maximizing the FDR with respect to $\boldsymbol{\theta}$. This is a case of the *generalized Rayleigh ratio*, and it is known from linear algebra that it is maximized if $\boldsymbol{\theta}$ satisfies

$$\Sigma_b \boldsymbol{\theta} = \lambda \Sigma_w \boldsymbol{\theta},$$

where λ is the maximum eigenvalue of the matrix $\Sigma_w^{-1}\Sigma_b$ (Problem 7.14). However, for our specific case[2] here, we can bypass the need for solving an eigenvalue-eigenvector problem. Observe that the last equation can be rewritten as

$$\lambda \Sigma_w \boldsymbol{\theta} \propto (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{\theta} \propto (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2).$$

In other words, $\Sigma_w \boldsymbol{\theta}$ lies in the direction of $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, and because we are only interested in the direction, we can finally write that

$$\boxed{\boldsymbol{\theta} = \Sigma_w^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2),} \tag{7.58}$$

assuming of course that Σ_w is invertible. In practice, Σ_w is obtained as the respective sample mean using the available observations.

Figure 7.11a shows the resulting direction for two spherically distributed (isotropic) classes in the two-dimensional space. In this case, the direction for projecting the data is parallel to $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$.

[2] Σ_b is a rank-one matrix and there is only one nonzero eigenvalue.

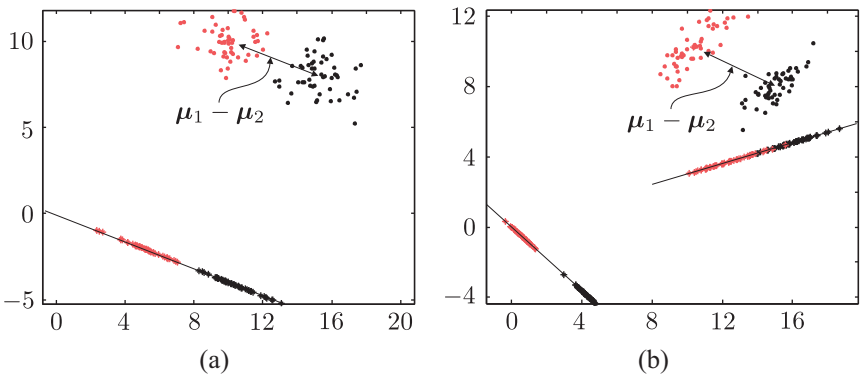

FIGURE 7.11

(a) The optimal direction resulting from Fisher's discriminant for two spherically distributed classes. The direction on which projection takes place is parallel to the segment joining the mean values of the data in the two classes. (b) The line on the bottom left of the figure corresponds to the direction that results from Fisher's discriminant; observe that it is no longer parallel to $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$. For the sake of comparison, observe that projecting on the other line on the right results in class overlap.

In Figure 7.11b, the distribution of the data in the two classes is not spherical, and the direction of projection (the line to the bottom left of the figure) is not parallel to the segment joining the two mean points. Observe that if the line to the right is selected, then after projection the classes do overlap.

In order for the Fisher's discriminant method to be used as a classifier, a threshold θ_0 must be adopted, and decision in favor of a class is performed according to the rule

$$y = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}} \Sigma_w^{-1} \boldsymbol{x} + \theta_0 \begin{cases} > 0, & \text{class } \omega_1, \\ < 0, & \text{class } \omega_2. \end{cases} \tag{7.59}$$

Compare, now, (7.59) with (7.16)–(7.18); the latter were obtained via the Bayes rule for the Gaussian case, when both classes share the same covariance matrix. Observe that for this case, the resulting hyperplanes are parallel and the only difference is in the threshold value. Note, however, that the Gaussian assumption was not needed for Fisher's discriminant. This justifies the use of (7.16)–(7.18), even when the data are not normally distributed. In practice, depending on the data, different threshold values may be used.

Finally, because the world is often small, it can be shown that Fisher's discriminant can also be seen as a special case of the LS solution, if the target class labels, instead of ± 1, are chosen as $\frac{N_1}{N}$ and $\frac{-N_2}{N}$, respectively, where N is the total number of training samples, N_1 is the number of samples in class ω_1, and N_2 is the corresponding number in class ω_2 [55].

Another point of view for Fisher's discriminant method is that it performs dimensionality reduction by projecting the data from the original l-dimensional space to a lower one-dimensional space. This reduction in dimensionality is performed in a *supervised* way, by exploiting the class labels of the training data. As we will see in Chapter 19, there are other techniques during which the dimensionality reduction takes place in an unsupervised way. The obvious question now is whether it is possible to use Fisher's idea in order to reduce the dimensionality not to one but to another intermediate value

between one and l. It turns out that this is possible, but it also depends on the number of classes. More on dimensionality reduction techniques can be found in Chapter 19.

Multiclass Fisher's discriminant: Our starting point is the J_3 criterion defined in (7.52). It can be readily shown that the FDR criterion, used in the two-class case, is directly related to the J_3 one, once the latter is considered for the one-dimensional case and for equiprobable classes. For the more general multiclass formulation, the task becomes that of estimating an $l \times m,\ m < l$ matrix, A, such that the linear transformation from the original $\mathbb{R}^l$ to the new $\mathbb{R}^m$ space, or

$$\boldsymbol{y} = A^{\mathrm{T}}\boldsymbol{x}, \tag{7.60}$$

to retain as much classification-related information as possible. Note that in any dimensionality reduction technique, some of the original information is, in general, bound to be lost. Our goal is for the loss to be as small as possible. Because we chose to measure classification-related information by the J_3 criterion, the goal is to compute A in order to maximize

$$J_3(A) = \mathrm{trace}\{\Sigma_{wy}^{-1}\Sigma_{by}\}, \tag{7.61}$$

where Σ_{wy} and Σ_{by} are the within-class and between-classes scatter matrices measured in the transformed lower dimensional space. Maximization follows standard arguments of optimization with respect to matrices. The algebra gets a bit involved and we will state the final result. Details of the proof can be found in Refs. [23, 52]. Matrix A is given by the following equation:

$$(\Sigma_{wx}^{-1}\Sigma_{bx})A = A\Lambda. \tag{7.62}$$

Matrix Λ is a diagonal matrix having as elements m of the eigenvalues of the $l \times l$ matrix, $\Sigma_{wx}^{-1}\Sigma_{bx}$, where Σ_{wx} and Σ_{bx} are the within-class and between-classes scatter matrices, respectively, in the original $\mathbb{R}^l$ space. The matrix of interest, A, comprises columns that are the respective eigenvectors. The problem, now, becomes to select the m eigenvalues/eigenvectors. Note that by its definition, Σ_b, being the sum of M related (via $\boldsymbol{\mu}_0$) rank one matrices, is of rank $M-1$ (Problem 7.15). Thus, the product $\Sigma_{wx}^{-1}\Sigma_{bx}$ has only $M-1$ nonzero eigenvalues. This imposes a stringent constraint on the dimensionality reduction. The maximum dimension, m, that one can obtain is $m = M-1$ (for the two-class task, $m = 1$), irrespective of the original dimension l. There are two cases, that are worth focusing on:

- $m = M-1$. In this case, it is shown that if A is formed having as columns all the eigenvectors corresponding to the nonzero eigenvalues, then

  $$J_{3y} = J_{3x}.$$

  In other words, there is no loss of information (as measured via the J_3 criterion) by reducing the dimension from l to $M-1$! Note that in this case, Fisher's method produces $m = M-1$ discriminant (linear) functions. This complies with a general result in classification stating that the minimum number of discriminant functions needed for an M-classification problem is $M-1$ [52]. Recall that in Bayesian classification, we need M functions, $P(\omega_i|\boldsymbol{x}),\ i = 1, 2, \ldots, M$; however, only $M-1$ of those are independent, because they must all add to one. Hence, Fisher's method provides the minimum number of linear discriminants required.
- $m < M-1$. If A is built having as columns the eigenvectors corresponding to the maximum m eigenvalues, then

  $$J_{3y} < J_{3x}.$$

  However, the resulting value J_{3y} is the maximum possible one.

Remarks 7.5.

- If J_3 is used with other matrix combinations, as might be achieved by using Σ_m in place of Σ_b, the constraint of the rank being equal to $M-1$ is removed, and larger values for m can be obtained.
- In a number of practical cases, Σ_w may not be invertible. This is, for example, the case in the *small sample size* problems, where the dimensionality of the feature space, l, may be larger than the number of the training data, N. Such problems may be encountered in applications such as web-document classification, gene expression profiling, and face recognition. There are different escape routes in this problem; see [52] for a discussion and related references.

7.8 CLASSIFICATION TREES

Classification trees are based on a simple, yet powerful, idea, and they are among the most popular techniques for classification. They are *multistage* systems, and classification of a pattern into a class is achieved *sequentially*. Through a series of tests, classes are *rejected* in a sequential fashion until a decision is finally reached in favor of one remaining class. Each one of the tests, whose outcome decides which classes are rejected, is of a *binary* "Yes" or "No" type and is applied to a *single* feature. Our goal is to present the main philosophy around a special type of trees known as *ordinary binary classification trees* (OBCT). They belong to a more general class of methods that construct trees, both for classification as well as regression, known as *classification and regression trees* (CART) [4, 45]. Variants of the method have also been proposed [49].

The basic idea around OBCTs is to partition the feature space into (*hyper*) *rectangles*; that is, the space is partitioned via hyperplanes, which are parallel to the axes. This is illustrated in Figure 7.12.

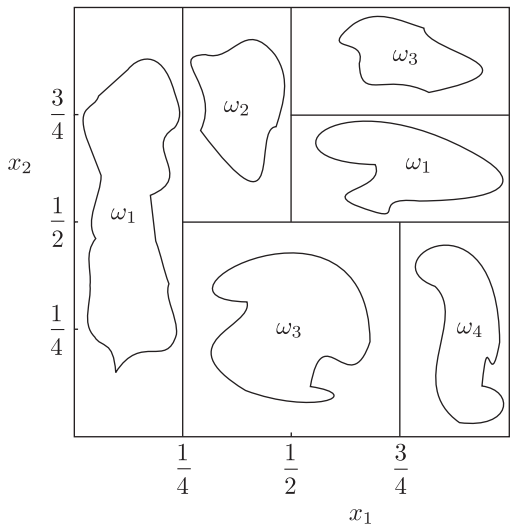

FIGURE 7.12

Partition of the two-dimensional features space, corresponding to three classes, via a classification (OBCT) tree.

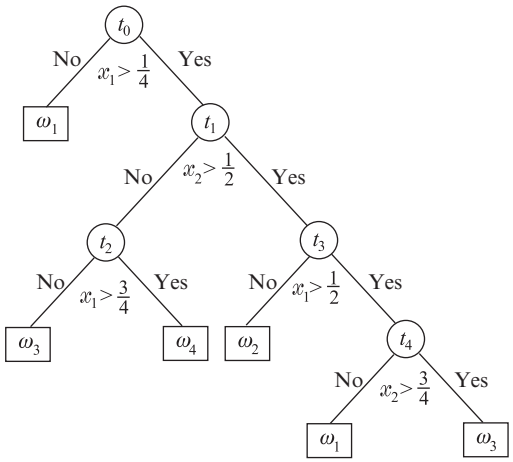

FIGURE 7.13

The classification tree that performs the space partitioning for the task indicated in Figure 7.12.

The partition of the space in (hyper)rectangles is performed via a series of "questions" of this form: is the value of the feature $x_i < a$? This is also known as the *splitting criterion*. The sequence of questions can nicely be realized via the use of a tree. Figure 7.13 shows the tree corresponding to the case illustrated in Figure 7.12. Each node of the tree performs a test against an *individual* feature and, if it is not a leaf node, it is connected to two *descendant* nodes: one is associated with the answer "Yes" and the other with the answer "No."

Starting from the root node, a path of successive decisions is realized until a leaf node is reached. Each leaf node is associated with a *single* class. The assignment of a point to a class is done according to the label of the respective leaf node. This type of classification is conceptually simple and easily interpretable. For example, in a medical diagnosis system, one may start with a question: is the temperature high? If yes, a second question can be: is the nose runny? The process carries on until a final decision concerning the disease has been reached. Also, trees are useful in building up reasoning systems in artificial intelligence [51]. For example, the existence of specific objects, which is deduced via a series of related questions based on the values of certain (high-level) features, can lead to the recognition of a scene or of an object depicted in an image.

Once a tree has been developed, classification is straightforward. The major challenge lies in constructing the tree, by exploiting the information that resides in the training data set. The main questions one is confronted with while designing a tree are:

- Which splitting criterion should be adopted?
- When should one stop growing a tree and declare a node as final?
- How is a leaf node associated with a specific class?

Besides the above issues, there are more that will be discussed later on.

Splitting criterion: We have already stated that the questions asked at each node are of the type

$$\text{is } x_i < a?$$

The goal is to select an appropriate value for the threshold value a. Assume that starting from the root node, the tree has grown up to the current node, t. Each node, t, is associated with a subset $X_t \subseteq X$ of the training data set, X. This is the set of the training points that have survived to this node, after the tests that have taken place at the previous nodes in the tree. For example, in Figure 7.13, a number of points, which belong to, say, class ω_1, will not be involved in node t_1 because they have already been assigned in a previously labeled leaf node. The purpose of a splitting criterion is to split X_t into two *disjoint* subsets, namely X_{tY}, and X_{tN}, depending on the answer to the specific question at node t. For every split, the following is true:

$$X_{tY} \cap X_{tN} = \emptyset,$$
$$X_{tY} \cup X_{tN} = X_t.$$

The goal in each node is to select which feature is to be tested and also what is the best value of the corresponding threshold value a. The adopted philosophy is to make the choice so that every split generates sets, X_{tY}, X_{tN}, which are more *class-homogeneous* compared to X_t. In other words, the data in each one of the two descendant sets must show a higher preference to specific classes, compared to the ancestor set. For example, assume that the data in X_t consist of points that belong to four classes, ω_1, ω_2, ω_3, ω_4. The idea is to perform the splitting so that most of the data in X_{tY} belong to, say, ω_1, ω_2 and most of the data in X_{tN} to ω_3, ω_4. In the adopted terminology, the sets X_{tY} and X_{tN} should be *purer* compared to X_t. Thus, we must first select a criterion that measures *impurity* and then compute the threshold value and choose the specific feature (to be tested) to maximize the decrease in node impurity. For example, a common measure to quantify impurity of node, t, is the *entropy*, defined as

$$I(t) = -\sum_{m=1}^{M} P(\omega_m|t) \log_2 P(\omega_m|t), \tag{7.63}$$

where $\log_2(\cdot)$ is the base-two logarithm. The maximum value of $I(t)$ occurs if all probabilities are equal (maximum impurity), and the smallest value, which is equal to zero, when only one of the probability values is one and the rest equal zero. Probabilities are approximated as

$$P(\omega_m|t) = \frac{N_t^m}{N_t}, \quad m = 1, 2, \ldots, M,$$

where, N_t^m is the number of the points from class m in X_t, and N_t the total number of points in X_t. The decrease in node impurity, after splitting the data into two sets, is defined as

$$I(t) = \sum_{m=1}^{M} P(\omega_m|t)\big(1 - P(\omega_m|t)\big). \tag{7.64}$$

where $I(t_Y)$ and $I(t_N)$ are the impurities associated with the two new sets, respectively. The goal now becomes to select the specific feature, x_i, and the threshold a_t so that $\Delta I(t)$ becomes maximum. This will now define two new descendant nodes of t, namely, t_N and t_Y; thus, the tree grows with two new nodes. A way to search for different threshold values is the following: For each one of the features, x_i, $i = 1, 2, \ldots, l$, rank the values, x_{in}, $n = 1, 2, \ldots, N_t$, which this feature takes among the training points in X_t. Then define a sequence of corresponding threshold values, a_{in}, to be halfway between consecutive distinct values of x_{in}. Then test the impurity change that occurs for each one of these

threshold values and keep the one that achieves the maximum decrease. Repeat the process for all features, and finally, keep the combination that results in the best maximum decrease.

Besides entropy, other impurity measuring indices can be used. A popular alternative, which results in a slightly sharper maximum compared to the entropy one, is the so-called *Gini index*, defined as

$$I(t) = \sum_{m=1}^{M} P(\omega_m|t)\big(1 - P(\omega_m|t)\big). \tag{7.65}$$

This index is also zero if one of the probability values is equal to 1 and the rest are zero, and it takes its maximum value when all classes are equiprobable.

Stop-splitting rule: The obvious question when growing a tree is when to stop growing it. One possible way is to adopt a threshold value, T, and stop splitting a node once the maximum value $\Delta I(t)$, for all possible splits, is smaller than T. Another possibility is to stop when the cardinality of X_t becomes smaller than a certain number or if the node is pure, in the sense that all points in it belong to a single class.

Class assignment rule: Once a node, t, is declared to be a leaf node, it is assigned a class label; usually this is done on a majority voting rationale. That is, it is assigned the label of the class where most of the data in X_t belong.

Pruning a tree: Experience has shown that growing a tree and using a stopping rule does not always work well in practice; growing may either stop early or may result in trees of very large size. A common practice is to first grow a tree up to a large size and then adopt a pruning technique to eliminate nodes. Different pruning criteria can be used; a popular one is to combine an estimate of the error probability with a complexity measuring index; see [4, 45].

Remarks 7.6.

- Among the notable advantages of decision trees is the fact that they can naturally treat mixtures of numeric and categorical variables. Moreover, they scale well with large data sets. Also, they can treat missing variables in an effective way. In many domains, not all the values of the features are known for every pattern. The values may have gone unrecorded, or they may be too expensive to obtain. Finally, due to their structural simplicity, they are easily interpretable; in other words, it is possible for a human to understand the reason for the output of the learning algorithm. In some applications, such as in financial decisions, this is a legal requirement.

  On the other hand, the prediction performance of the tree classifiers is not as good as other methods, such as support vector machines and neural networks, to be treated in Chapters 11 and 18, respectively.
- A major drawback associated with the tree classifiers is that they are *unstable*. That is, a small change in the training data set can result in a very different tree. The reason for this lies in the hierarchical nature of the tree classifiers. An error that occurs in a node at a high level of the tree propagates all the way down to the leaves below it.

  Bagging (Bootstrap Aggregating) [5] is a technique that can reduce the variance and improve the generalization error performance. The basic idea is to create a number of B variants, $X_1, X_2, \ldots, X_B$, of the training set, X, using *bootstrap* techniques, by uniformly sampling from X with replacement. For each of the training set variants, X_i, a tree, T_i, is constructed. The final decision for the classification of a given point is in favor of the class predicted by the majority of the subclassifiers, T_i, $i = 1, 2, \ldots, B$.

Random forests use the idea of bagging in tandem with random feature selection [7]. The difference with bagging lies in the way the decision trees are constructed. The feature to split in each node is selected as the best among a set of F *randomly* chosen features, where F is a user-defined parameter. This extra introduced randomness is reported to have a substantial effect in performance improvement.

Random forests often have very good predictive accuracy and have been used in a number of applications, including for body pose recognition in terms of Microsoft's popular Kinect sensor [48].

Besides the previous methods, more recently, Bayesian techniques have also been suggested and used to stabilize the performance of trees; see [11, 58]. Of course, the effect of using multiple trees is losing a main advantage of the trees, that is, their fairly easy interpretability.

- Besides the OBCT rationale, a more general partition of the feature space has also been proposed via hyperplanes that are not parallel to the axes. This is possible via questions of the type: *Is* $\sum_{i=1}^{l} c_i x_i < a$*?* This can lead to a better partition of the space. However, the training now becomes more involved; see [49].
- Decision trees have also been proposed for regression tasks, albeit with less success. The idea is to split the space into regions, and prediction is performed based on the average of the output values in the region where the observed input vector lies; such an averaging approach has as a consequence the lack of smoothness, as one moves from one region to another, which is a major drawback of regression trees. The splitting into regions is performed based on the LS criterion [26].

7.9 COMBINING CLASSIFIERS

So far, we have discussed a number of classifiers, and more methods will be presented in Chapters 11, 13 and 18 concerning support vector machines, Bayesian methods, and neural/deep networks. The obvious question an inexperienced practitioner/researcher is confronted with is, which method then? Unfortunately, there is no definitive answer.

No free lunch theorem: The goal of the design of any classifier, and in general of any learning scheme, is to provide a good generalization performance. However, there are no context-independent or usage-independent reasons to support one learning technique over another. Each learning task, represented by the available data set, will show a preference for a specific learning scheme that fits the specificities of the particular problem at hand. An algorithm that scores tops in one problem can score low for another. This is sometimes summarized as the no free lunch theorem [57].

In practice, one should try different learning methods from the available palette, each optimized to the specific task, and test its generalization performance against an independent data set different from the one used for training, using, for example, the leave-one-out-method or any of its variants (Chapter 3). Then, keep and use the method that scored best for the specific task.

To this end, there are a number of major efforts to compare different classifiers against different data sets and measure the "average" performance, via the use of different statistical indices in order to quantify the overall performance of each classifier against the data sets.

Experimental comparisons

One of the very first efforts, to compare the performance of different classifiers, was the Statlog project, [36]. Two subsequent efforts are summarized in Refs. [9, 35]. In the former, 17 popular classifiers were

tested against 21 data sets. In the latter, 10 classifiers and 11 data sets were employed. The results verify what has already been said: different classifiers perform better for different sets. However, it is reported that boosted trees (Section 7.11), random forests, bagged decision trees, and support vector machines were ranked among the top ones for most of the data sets.

Neural Information Processing Systems Workshop (NIPS-2003) organized a classification competition based on five data sets. The results of the competition are summarized in Ref. [25]. The competition was focused on feature selection [38]. In a follow-up study, [29], more classifiers were added. Among the considered classifiers, a Bayesian-type neural network scheme (Chapter 18) scored at the top, albeit at significantly higher run time requirements. The other classifiers considered were random forests and boosting, where trees and neural networks were used as base classifiers (Section 7.10). Random forests also performed well, at much lower computational times compared to the Bayesian-type classifier.

Schemes for combining classifiers

A trend to improve performance is to combine different classifiers together and exploit their individual advantages. An observation that justifies such an approach is that during testing, there are patterns on which even the best classifier for a particular task fails to predict their true class. In contrast, the same patterns can be classified correctly by other classifiers, with an inferior overall performance. This shows that there may be some complementarity among different classifiers, and combination can lead to boosted performance compared to that obtained from the best (single) classifier. Recall that bagging, mentioned in Section 7.8, is a type of classifier combination.

The issue that arises now is to select a combination scheme. There are different schemes, and the results they provide can be different. Below, we summarize the more popular combination schemes.

- *Arithmetic averaging rule*: Assuming that we use L classifiers, where each one outputs a value of the posterior probability, $P_j(\omega_i|\boldsymbol{x})$, $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, L$, a decision concerning the class assignment is based on the following rule:

$$\boxed{\text{Assign } \boldsymbol{x} \text{ to class } \omega_i = \arg\max_k \frac{1}{L}\sum_{j=1}^{L} P_j(\omega_k|\boldsymbol{x}),\ k = 1, 2, \ldots, M.} \tag{7.66}$$

  This rule is equivalent with computing the "final" posterior probability, $P(\omega_i|\boldsymbol{x})$, in order to minimize the average Kullback-Leibler distance (Problem 7.16),

$$D_{av} = \frac{1}{L}\sum_{j=1}^{L} D_j,$$

  where

$$D_j = \sum_{i=1}^{M} P_j(\omega_i|\boldsymbol{x}) \ln \frac{P_j(\omega_i|\boldsymbol{x})}{P(\omega_i|\boldsymbol{x})}.$$

- *Geometric averaging rule*: This rule is the outcome of minimizing the alternative formulation of Kullback-Leibler distance (note that this distance is not symmetric); in other words,

$$D_j = \sum_{i=1}^{M} P(\omega_i|\boldsymbol{x}) \ln \frac{P(\omega_i|\boldsymbol{x})}{P_j(\omega_i|\boldsymbol{x})},$$

which results in (Problem 7.17),

$$\boxed{\text{Assign } \boldsymbol{x} \text{ to class } \omega_i = \arg\max_k \prod_{j=1}^{L} P_j(\omega_k|\boldsymbol{x}), \quad k = 1, 2, \ldots, M.} \tag{7.67}$$

- *Stacking*: An alternative way is to use a weighted average of the outputs of the individual classifiers, where the combination weights are obtained optimally using the training data. Assume that the output of each individual classifier, $f_j(\boldsymbol{x})$, is of a soft-type; for example, an estimate of the posterior probability, as before. Then, the combined output is given by

$$f(\boldsymbol{x}) = \sum_{j=1}^{L} w_j f_j(\boldsymbol{x}), \tag{7.68}$$

where the weights are estimated via the following optimization task:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} \mathcal{L}\left(y_n, f(\boldsymbol{x}_n)\right) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} \mathcal{L}\left(y_n, \sum_{j=1}^{L} w_j f_j(\boldsymbol{x}_n)\right), \tag{7.69}$$

where, $\mathcal{L}(\cdot, \cdot)$ is a loss function; for example, the squared error one. However, adopting the previous optimization, based on the training data set, can lead to overfitting. According to stacking [56], a cross-validation rationale is adopted and instead of $f_j(\boldsymbol{x}_n)$, we employ the $f_j^{(-n)}(\boldsymbol{x}_n)$, where the latter is the output of the jth classifier trained on the data after *excluding* the pair $(y_n, \boldsymbol{x}_n)$. In other words, the weights are estimated by

$$\boxed{\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} \mathcal{L}\left(y_n, \sum_{j=1}^{L} w_j f_j^{(-n)}(\boldsymbol{x}_n)\right).} \tag{7.70}$$

Sometimes, the weights are constrained to be positive and add to one, giving rise to a constrained optimization task.

- *Majority voting rule*: The previous methods belong to the family of soft-type rules. A popular alternative is a hard-type rule, which is based on a voting scheme. One decides in favor of the class for which either there is a consensus or when at least l_c of the classifiers agree on the class label, where

$$l_c = \begin{cases} \frac{L}{2} + 1, & L \text{ is even}, \\ \frac{L+1}{2}, & L \text{ is odd}. \end{cases}$$

Otherwise, the decision is rejection (i.e., no decision is taken).

In addition to the sum, product, and majority voting, other combinations rules have also been suggested, which are inspired by the following inequalities [32]:

$$\prod_{j=1}^{L} P_j(\omega_i|\boldsymbol{x}) \leq \min_{j=1}^{L} P_j(\omega_i|\boldsymbol{x}) \leq \frac{1}{L} \sum_{j=1}^{L} P_j(\omega_i|\boldsymbol{x}) \leq \max_{j=1}^{L} P_j(\omega_i|\boldsymbol{x}), \tag{7.71}$$

and classification is achieved by using the max or min bounds instead of the sum and product. When outliers are present, one can instead use the *median* value:

$$\boxed{\text{Assign } \boldsymbol{x} \text{ to class } \omega_i = \arg\max_k \text{median}\left\{P_j(\omega_k|\boldsymbol{x})\right\}, \quad k = 1, 2, \ldots, M.} \tag{7.72}$$

It turns out that a no free lunch theorem is also valid for the combination rules; there is not a universally optimal rule. It all depends on the data at hand; see [28].

There are a number of other issues related to the theory of combining classifiers; for example, how one chooses the classifiers to be combined. Should the classifiers be dependent or independent? Furthermore, combination does not necessarily imply improved performance; in some cases, one may experience a performance loss (higher error rate) compared to that of the best (single) classifier [27, 28]. Thus, combining has to take place with care. More on these issues can be found in Refs. [33, 52] and the references therein.

7.10 THE BOOSTING APPROACH

The origins of the *boosting* method for designing learning machines is traced back to the work of Valiant and Kearns [30, 54], who posed the question of whether a weak learning algorithm, meaning one that does slightly better than random guessing, can be *boosted* into a strong one with a good performance index. At the heart of such techniques lies the *base learner*, which is a weak one. Boosting consists of an iterative scheme, where at each step the base learner is optimally computed using a different training set; the set at the current iteration is generated either according to an iteratively obtained data distribution or, usually, via a weighting of the training samples, each time using a different set of weights. The latter are computed in order to take into account the achieved performance up to the current iteration step. The final learner is obtained via a *weighted average* of all the *hierarchically* designed base learners. Thus, boosting can also be considered a scheme for combining different learners.

It turns out that, given a sufficient number of iterations, one can significantly improve the (poor) performance of the weak learner. For example, in some cases in classification, the training error may tend to zero as the number of iterations increases. This is very interesting indeed. Training a weak classifier, by appropriate manipulation of the training data (as a matter of fact, the weighting mechanism identifies hard samples, the ones that keep failing, and places more emphasis on them) one can obtain a strong classifier. Of course, as we will discuss, the fact that the training error may tend to zero does *not* necessarily mean the test error goes to zero, too.

The AdaBoost algorithm

We now focus on the two-class classification task and assume that we are given a set of N training observations, $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$, with $y_n \in \{-1, 1\}$. Our goal is to design a binary classifier,

$$f(\boldsymbol{x}) = \text{sgn}\left\{F(\boldsymbol{x})\right\}, \tag{7.73}$$

where

$$\boxed{F(\boldsymbol{x}) := \sum_{k=1}^{K} a_k \phi(\boldsymbol{x}; \boldsymbol{\theta}_k),} \tag{7.74}$$

where $\phi(\boldsymbol{x};\boldsymbol{\theta}_k) \in \{-1, 1\}$ is the base classifier at iteration k, defined in terms of a set of parameters, $\boldsymbol{\theta}_k,\ k = 1, 2, \ldots, K$, to be estimated. The base classifier is selected to be a binary one. The set of unknown parameters is obtained in a *step wise* approach and in a *greedy* way; that is, at each iteration step, i, we only optimize with respect to a single pair, $(a_i, \boldsymbol{\theta}_i)$, by keeping the parameters, $a_k, \boldsymbol{\theta}_k,\ k = 1, 2, \ldots, i-1$, obtained from the previous steps, fixed. Note that ideally, one should optimize with respect to all the unknown parameters, $a_k, \boldsymbol{\theta}_k,\ k = 1, 2, \ldots, K$, simultaneously; however, this would lead to a very computationally demanding optimization task. Greedy algorithms are very popular, due to their computational simplicity, and lead to a very good performance in a wide range of learning tasks. Greedy algorithms will also be discussed in the context of sparsity-aware learning in Chapter 10.

Assume that we are currently at the ith iteration step; consider the partial sum of terms

$$F_i(\cdot) = \sum_{k=1}^{i} a_k \phi(\cdot;\boldsymbol{\theta}_k). \tag{7.75}$$

Then we can write the following recursion:

$$F_i(\cdot) = F_{i-1}(\cdot) + a_i \phi(\cdot;\boldsymbol{\theta}_i), \quad i = 1, 2, \ldots, K, \tag{7.76}$$

starting from an initial condition. According to the greedy rationale, $F_{i-1}(\cdot)$ is assumed known and the goal is to optimize with respect to the set of parameters, $a_i, \boldsymbol{\theta}_i$. For optimization, a loss function has to be adopted. No doubt different options are available, giving different names to the derived algorithm. A popular loss function used for classification is the exponential loss, defined as

$$\boxed{\mathcal{L}\big(y, F(\boldsymbol{x})\big) = \exp\big(-yF(\boldsymbol{x})\big): \quad \text{Exponential Loss Function,}} \tag{7.77}$$

and it gives rise to the *Adaptive Boosting* (AdaBoost) algorithm. The exponential loss function is shown in Figure 7.14, together with the 0-1 loss function. The former can be considered a (differentiable) upper bound of the (nondifferentiable) 0-1 loss function. Note that the exponential loss weighs misclassified ($yF(\boldsymbol{x}) < 0$) points more heavily compared to the correctly identified ones ($yF(\boldsymbol{x}) > 0$). Employing the

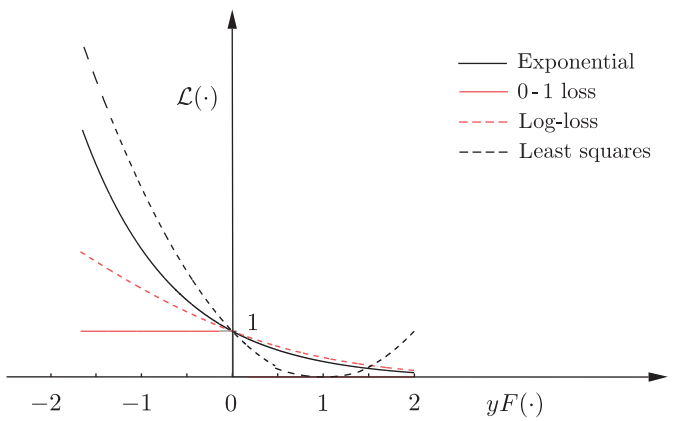

FIGURE 7.14

The 0-1, the exponential, the log-loss and the LS loss functions. They have all been normalized to cross the point (0, 1). The horizontal axis for the squared error (LS) corresponds to $y - F(\boldsymbol{x})$.

exponential loss function, the set $a_i, \boldsymbol{\theta}_i$ is obtained via the respective empirical cost function, in the following manner

$$(a_i, \boldsymbol{\theta}_i) = \arg\min_{a,\boldsymbol{\theta}} \sum_{n=1}^{N} \exp\Big(-y_n\big(F_{i-1}(\boldsymbol{x}_n) + a\phi(\boldsymbol{x}_n;\boldsymbol{\theta})\big)\Big). \tag{7.78}$$

This optimization is also performed in two steps. First, a is treated fixed and we optimize with respect to $\boldsymbol{\theta}$,

$$\boldsymbol{\theta_i} = \arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} w_n^{(i)} \exp\big(-y_n a\phi(\boldsymbol{x}_n;\boldsymbol{\theta})\big), \tag{7.79}$$

where

$$w_n^{(i)} := \exp\big(-y_n F_{i-1}(\boldsymbol{x}_n)\big),\ n = 1, 2, \ldots, N. \tag{7.80}$$

Observe that $w_n^{(i)}$ depends neither on a nor on $\phi(\boldsymbol{x}_n;\boldsymbol{\theta})$, hence it can be considered a weight associated with sample n. Moreover, its value depends entirely on the results obtained from the previous recursions.

We now turn our focus on the cost in (7.79). The optimization depends on the specific form of the base classifier. However, due to the exponential form of the loss, and the fact that the base classifier is a binary one, so that $\phi(\boldsymbol{x};\boldsymbol{\theta}) \in \{-1, 1\}$, optimizing (7.79) is readily seen to be equivalent with optimizing the following cost:

$$\boldsymbol{\theta}_i = \arg\min_{\boldsymbol{\theta}} P_i, \tag{7.81}$$

where

$$P_i := \sum_{n=1}^{N} w_n^{(i)} \chi_{(-\infty,0]}\big(y_n\phi(\boldsymbol{x}_n;\boldsymbol{\theta})\big), \tag{7.82}$$

and $\chi_{[-\infty,0]}(\cdot)$ is the 0-1 loss function.[3] Note that P_i is the weighted empirical classification error. Obviously, when the misclassification error is minimized, the cost in (7.79) is also minimized, because the exponential loss weighs the misclassified points heavier. To guarantee that P_i remains in the $[0, 1]$ interval, the weights are normalized to unity by dividing by the respective sum; note that this does not affect the optimization process. In other words, $\boldsymbol{\theta}_i$ can be computed in order to minimize the empirical misclassification error committed by the base classifier. For base classifiers of very simple structure, such a minimization is computationally feasible.

Having computed the optimal $\boldsymbol{\theta}_i$, the following are easily established from the respective definitions,

$$\sum_{y_n\phi(\boldsymbol{x}_n;\boldsymbol{\theta}_i)<0} w_n^{(i)} = P_i, \tag{7.83}$$

and

$$\sum_{y_n\phi(\boldsymbol{x}_n;\boldsymbol{\theta}_i)>0} w_n^{(i)} = 1 - P_i. \tag{7.84}$$

[3] The characteristic function $\chi_A(x)$ is equal to one if $x \in A$ and zero otherwise.

Combining (7.83) and (7.84) with (7.78) and (7.80), it is readily shown that

$$a_i = \arg\min_a \left\{ \exp(-a)(1 - P_i) + \exp(a) P_i \right\}. \tag{7.85}$$

Taking the derivative with respect to a and equating to zero results in

$$a_i = \frac{1}{2} \ln \frac{1 - P_i}{P_i}. \tag{7.86}$$

Once a_i and $\boldsymbol{\theta}_i$ have been estimated, the weights for the next iteration are readily given by

$$w_n^{(i+1)} = \frac{\exp\left(-y_n F_i(\boldsymbol{x}_n)\right)}{Z_i} = \frac{w_n^{(i)} \exp\left(-y_n a_i \phi(\boldsymbol{x}_n; \boldsymbol{\theta}_i)\right)}{Z_i}, \tag{7.87}$$

where Z_i is the normalizing factor

$$Z_i := \sum_{n=1}^{N} w_n^{(i)} \exp\left(-y_n a_i \phi(\boldsymbol{x}_n; \boldsymbol{\theta}_i)\right). \tag{7.88}$$

Looking at the way the weights are formed, one can grasp one of the major secrets underlying the AdaBoost algorithm: The weight associated with a training sample, $\boldsymbol{x}_n$, is increased (decreased) with respect to its value at the previous iteration, depending on whether the pattern has failed (succeeded) in being classified correctly. Moreover, the percentage of the decrease (increase) depends on the value of a_i, which controls the relative importance in the buildup of the final classifier. Hard samples, which keep failing over successive iterations, gain importance in their participation in the weighted empirical error value. For the case of the AdaBoost, it can be shown that the training error tends to zero exponentially fast (Problem 7.18). The scheme is summarized in Algorithm 7.1.

Algorithm 7.1 (The AdaBoost algorithm).

- Initialize: $w_n^{(1)} = \frac{1}{N}$, $i = 1, 2, \ldots, N$
- Initialize: $i = 1$
- Repeat
  - Compute the optimum $\boldsymbol{\theta}_i$ in $\phi(\cdot; \boldsymbol{\theta}_i)$ by minimizing P_i; (7.81)
  - Compute the optimum P_i; (7.82)
  - $a_i = \frac{1}{2} \ln \frac{1 - P_i}{P_i}$
  - $Z_i = 0$
  - **For** $n = 1$ to N **Do**
    - $w_n^{(i+1)} = w_n^{(i)} \exp\left(-y_n a_i \phi(\boldsymbol{x}_n; \boldsymbol{\theta}_i)\right)$
    - $Z_i = Z_i + w_n^{(i+1)}$
  - **End For**
  - **For** $n = 1$ to N **Do**
    - $w_n^{(i+1)} = w_n^{(i+1)} / Z_i$
  - **End For**
  - $K = i$
  - $i = i + 1$
- Until a termination criterion is met.
- $f(\cdot) = \text{sgn}\left(\sum_{k=1}^{K} a_k \phi(\cdot, \boldsymbol{\theta}_k)\right)$

The AdaBoost was first derived in Ref. [20] in a different way. Our formulation follows that given in Ref. [21]. Yoav Freund and Robert Schapire received the prestigious Gödel award for this algorithm in 2003.

The log-loss function

In AdaBoost, the exponential loss function was employed. From a theoretical point of view, this can be justified by the following argument: Consider the mean value with respect to the binary label, y, of the exponential loss function,

$$\mathbb{E}\left[\exp\left(-yF(\boldsymbol{x})\right)\right] = P(y=1)\exp\left(-F(\boldsymbol{x})\right) + P(y=-1)\exp\left(F(\boldsymbol{x})\right). \tag{7.89}$$

Taking the derivative with respect to $F(\boldsymbol{x})$ and equating to zero, we readily obtain that the minimum of (7.89) occurs at

$$F_*(\boldsymbol{x}) = \arg\min_f \mathbb{E}\left[\exp\left(-yf\right)\right] = \frac{1}{2}\ln\frac{P(y=1|\boldsymbol{x})}{P(y=-1|\boldsymbol{x})}. \tag{7.90}$$

The logarithm of the ratio on the right-hand side is known as the *log-odds* ratio. Hence, if one views the minimizing function in (7.78) as the empirical approximation of the mean value in (7.89), it fully justifies considering the sign of the function in (7.73) as the classification rule.

A major problem associated with the exponential loss function, as is readily seen in Figure (7.14), is that it weights heavily wrongly classified samples, depending on the value of the respective margin, defined as

$$m_x := |yF(\boldsymbol{x})|. \tag{7.91}$$

Note that the farther the point is from the decision surface ($F(\boldsymbol{x}) = 0$), the larger the value of $|F(\boldsymbol{x})|$. Thus, points that are located at the wrong side of the decision surface ($yF(\boldsymbol{x}) < 0$) and far away are weighted with (exponentially) large values, and their influence in the optimization process is large compared to the other points. Thus in the presence of outliers, the exponential loss is not the most appropriate one. As a matter of fact, in such environments, the performance of the AdaBoost can degrade dramatically.

An alternative loss function is the *log-loss* or *binomial deviance*, defined as

$$\boxed{\mathcal{L}\big(y, F(\boldsymbol{x})\big) := \ln\big(1+\exp\big(-yF(\boldsymbol{x})\big)\big): \quad \text{Log-loss Function,}} \tag{7.92}$$

which is also shown in Figure 7.14. Observe that its increase is almost linear for large negative values. Such a function leads to a more balanced influence of the loss among all the points. We will return to the issue of *robust loss functions*, that is, loss functions that are more immune to the presence of outliers, in Chapter 11. Note that the function that minimizes the mean of the log-loss, with respect to y, is the same as the one given in (7.90) (try it). However, if one employs the log-loss instead of the exponential, the optimization task gets more involved, and one has to resort to gradient descent or Newton-type schemes for optimization; see [22].

Remarks 7.7.

- For comparison reasons, in Figure 7.14, the LS loss is shown. The LS depends on the value $(y - F(\boldsymbol{x}))$, which is the equivalent of the margin defined above, $yF(\boldsymbol{x})$. Observe that, besides the relatively large influence that large values of error have, the error is also penalized for patterns whose label has been predicted correctly. This is one more justification that the LS criterion is not, in general, a good choice for classification.

- Multiclass generalizations of the Boosting scheme have been proposed in Refs. [19, 21]. In Ref. [16], regularized versions of the AdaBoost scheme have been derived in order to impose sparsity. Different regularization schemes are considered, including ℓ_1, ℓ_2, and ℓ_∞. The end result is a family of coordinate-descent algorithms that integrate forward feature induction and back-pruning. In Ref. [47], a version is presented where a priori knowledge is brought into the scene. The so-called AdaBoost$^*_\nu$ is introduced in Ref. [43], where the margin is explicitly taken into account.
- Note that the boosting rationale can be applied equally well to regression tasks involving respective loss functions, such as the LS. A robust alternative to the LS is the absolute error, instead of the square, loss function [22].
- The boosting technique has attracted a lot of attention among researchers in the field in order to justify its good performance in practice and its relative immunity to overfitting. While the training error may become zero, still this does not necessarily imply overfitting. A first explanation was attempted in terms of bounds, concerning the respective generalization performance. The derived bounds are independent of the number of iterations K, and they are expressed in terms of the margin Ref. [46]. However, these bounds tend to be very loose. Another explanation may lie in the fact that each time, optimization is carried out with only a single set of parameters. The interested reader may find the discussions following the papers [6, 8, 21] very enlightening on this issue.

Example 7.5. Consider a 20-dimensional two-class classification task. The data points from the first class (ω_1) stem from either of the two Gaussian distributions with means $\boldsymbol{\mu}_{11} = [0,\ 0, \ldots, 0]^{\mathrm{T}}$, $\boldsymbol{\mu}_{12} = [1,\ 1, \ldots, 1]^{\mathrm{T}}$, while the points of the second class (ω_2) stem from the Gaussian distribution

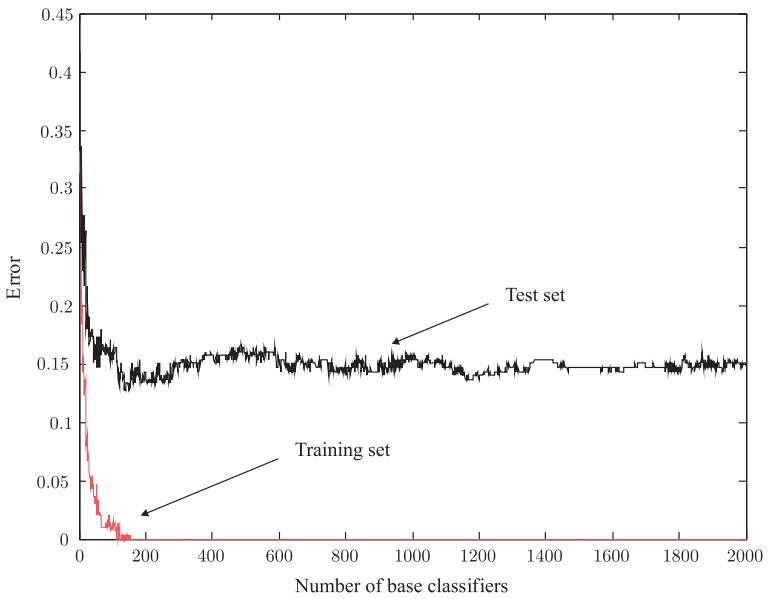

FIGURE 7.15

Training and test error rate curves as a function of the number of iterations, for the case of Example 7.5.

with mean $\boldsymbol{\mu}_2 = [\overbrace{0,\ldots,0}^{10}, \overbrace{1,\ldots,1}^{10}]^{\mathrm{T}}$. The covariance matrices of all distributions are equal to the 20-dimensional identity matrix. Each one of the training and the test sets consists of 300 points, 200 from ω_1 (100 from each distribution) and 100 from ω_2.

For the AdaBoost, the base classifier was selected to be a *stump*. This is a very naive type of tree, consisting of a single node, and classification of a feature vector $\boldsymbol{x}$ is achieved on the basis of the value of only one of its features, say, x_i. Thus, if $x_i < a$, where a is an appropriate threshold, $\boldsymbol{x}$ is assigned to class ω_1. If $x_i > a$, it is assigned to class ω_2. The decision about the choice of the specific feature, x_i, to be used in the classifier was randomly made. Such a classifier results in a training error rate slightly better than 0.5. The AdaBoost algorithm was run on the training data for 2000 iteration steps. Figure 7.15 verifies the fact that the training error rate converges to zero very fast. The test error rate keeps decreasing even after the training error rate becomes zero and then levels off at around 0.15.

7.11 BOOSTING TREES

In the discussion on experimental comparison of various methods in Section 7.9, it was stated the boosted trees are among the most powerful learning schemes for classification and data mining. Thus, it is worth spending some more time on this special type of boosting techniques.

Trees were introduced in Section 7.8. From the knowledge we have now acquired, it is not difficult to see that the output of a tree can be compactly written as

$$T(\boldsymbol{x};\boldsymbol{\Theta}) = \sum_{j=1}^{J} \hat{y}_j \chi_{R_j}(\boldsymbol{x}), \tag{7.93}$$

where J is the number of leaf nodes, R_j is the region associated with the jth leaf, after the space partition imposed by the tree, $\hat{y}_j$ is the respective label associated with R_j (output/prediction value for regression) and χ is our familiar characteristic function. The set of parameters, $\boldsymbol{\Theta}$, consists of $(\hat{y}_j, R_j)$, $j = 1, 2, \ldots, J$, which are estimated during training. These can be obtained by selecting an appropriate cost function. Also, suboptimal techniques are usually employed, in order to build up a tree, as the ones discussed in Section 7.8.

In a boosted tree model, the base classifier comprises a tree. For example, the stump used in Example 7.5 is a very special case of a boosted tree. In practice, one can employ trees of larger size. Of course, the size must not be very large, in order to be closer to a weak classifier. In practice, values of J between three and eight are advisable.

The boosted tree model can be written as

$$F(\boldsymbol{x}) = \sum_{k=1}^{K} T(\boldsymbol{x};\boldsymbol{\Theta}_k), \tag{7.94}$$

where

$$T(\boldsymbol{x};\boldsymbol{\Theta}_k) = \sum_{j=1}^{J} \hat{y}_{kj} \chi_{R_{kj}}(\boldsymbol{x}).$$

Equation (7.94) is basically the same as (7.74), with the a's being equal to one. We have assumed the size of all the trees to be the same, although this may not be necessarily the case. Adopting a loss

function, $\mathcal{L}$, and the greedy rationale, used for the more general boosting approach, we arrive at the following recursive scheme of optimization:

$$\boldsymbol{\Theta}_i = \arg\min_{\boldsymbol{\Theta}} \sum_{n=1}^{N} \mathcal{L}\big(y_n, F_{i-1}(\boldsymbol{x}_n) + T(\boldsymbol{x}_n; \boldsymbol{\Theta})\big). \tag{7.95}$$

Optimization with respect to $\boldsymbol{\Theta}$ takes place into two steps: one with respect to $\hat{y}_{ij}$, $j = 1, 2, \ldots, J$, given R_{ij}, and then one with respect to the regions R_{ij}. The latter is a difficult task and only simplifies in very special cases. In practice, a number of approximations can be employed. Note that in the case of the exponential loss and the two-class classification task, the above is directly linked to the AdaBoost scheme.

For more general cases, numeric optimization schemes are mobilized; see [22]. The same rationale applies for regression trees, where now loss functions for regression, such as LS or the absolute error value, are used. Such schemes are also known as *multiple additive regression trees* (MARTs). A related implementation code for boosted trees is freely available in the R gbm package, [44].

There are two critical factors concerning boosted trees. One is the size of the trees, J, and the other is the choice of K. Concerning the size of the trees, usually one tries different sizes, $4 \leq J \leq 8$, and selects the best one. Concerning the number of iterations, for large values, the training error may get close to zero, but the test error can increase due to overfitting. Thus, one has to stop early enough, usually by monitoring the performance.

Another way to cope with overfitting is to employ *shrinkage* methods, which tend to be equivalent to regularization. For example, in the stage-wise expansion of $F_i(\boldsymbol{x})$ used in the optimization step (7.95), one can instead adopt the following:

$$F_i(\cdot) = F_{i-1}(\cdot) + \nu T(\cdot; \boldsymbol{\Theta}_i).$$

The parameter ν takes small values and it can be considered as controlling the learning rate of the boosting procedure. Values smaller than $\nu < 0.1$ are advised. However, the smaller the value of ν, the larger the value K should be to guarantee good performance. For more on MARTs, the interested reader can peruse [26].

7.12 A CASE STUDY: PROTEIN FOLDING PREDICTION

One of the most challenging modalities in bioinformatics is that of genetic data, that is, DNA sequences that can be stored and used either as raw input for models (e.g., sequence alignment, statistical analysis) or as the basis for the discovery of higher-level intrinsic attributes (e.g., the 3-D structure of the protein they produce). Since the discovery of the DNA structure in 1953 by James Watson and Francis Crick, but especially after the completion of the Human Genome Project in 2003, the basic building blocks of all life can be traced down to simple chemical components. In terms of data storage and analysis, these components are no more than a sequence of symbols in a biometric signal: the DNA strand.

The most basic building element in a DNA sequence is the set of four *nucleotides* or *nucleobases*: adenine (A), cytosine (C), guanine (G), and thymine (T). These four bases are complementary in pairs, such that stable chemical bonds can be formed between guanine with cytosine (G-T) and adenine with thymine (A-T). These bonds produce the celebrated double helix in the DNA macromolecule and provide a redundant encoding mechanism for the genetic information. Each nonoverlapping triplet of such subsequent nucleotides in a DNA sequence is called a *codon* and corresponds to one of the

Table 7.1 The genetic code for the 20 standard proteinogenic amino acids. They are the building blocks of protein-coding strands of every DNA and they are formed as triples of the four nucleobases (T, C, A, G). Each triplet (*'codon'*) is read left-up-right; for example, Alanine ("*Ala*") is encoded by the triplets "*GCx*" (*x*=any). The "*(st)*" codon signals the end-of-translation in a coding sequence [1] .

	T	*C*	*A*	*G*	
T	Phe	Ser	Tyr	Cys	*T*
	Phe	Ser	Tyr	Cys	*C*
	Leu	Ser	(st)	(st)	*A*
	Leu	Ser	(st)	Trp	*G*
C	Leu	Pro	His	Arg	*T*
	Leu	Pro	His	Arg	*C*
	Leu	Pro	Gln	Arg	*A*
	Leu	Pro	Gln	Arg	*G*
A	Ile	Thr	Asn	Ser	*T*
	Ile	Thr	Asn	Ser	*C*
	Ile	Thr	Lys	Arg	*A*
	Met	Thr	Lys	Arg	*G*
G	Val	Ala	Asp	Gly	*T*
	Val	Ala	Asp	Gly	*C*
	Val	Ala	Glu	Gly	*A*
	Val	Ala	Glu	Gly	*G*

20 basic or *standard proteinogenic* amino-acids that are coded directly in DNA. Amino acids are the building blocks of proteins, the most important elements in the functionality of living cells. In fact, four different symbols combined in triplets produce 64 possible combinations; however, each of the 20 standard amino acids is encoded redundantly with a variable number of alternatives, and there are also three "stop" codons to signify the end-of-translation in a sequence; Table 7.1. There is also a "start" codon to signify the beginning; this is the "ATG" triplet, which coincides with the "Met." If this is met for a first time, signifies the beginning of a sequence, and if it is in the middle it corresponds to an amino acid.

Each possible protein is encoded by a variable number of amino acids, ranging typically from 80 to 170 codons; this corresponds to sequences of roughly 240-510 nucleotides, which essentially form the low-level encoding of the most useful information content of DNA (Figure 7.16).

The human DNA sequence comprises approximately 3.2×10^6 base pairs of nucleotides [1]. Regions in the sequence that have been identified as protein-encoding regions are limited to roughly 1-2% of the total length of the DNA sequence. Such regions are also known as *genes*. Early estimates of the number of human genes was around 5,0000-10,0000, however the analysis of the human genome in

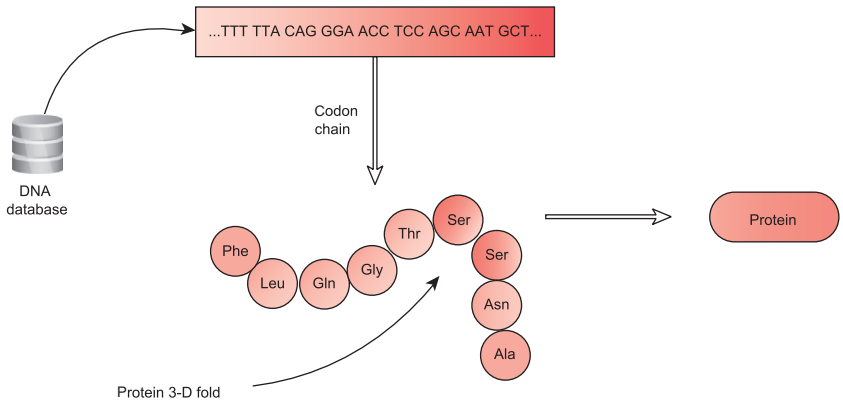

FIGURE 7.16

The DNA genetic code translation process: from the sequence of nucleobases into a valid codon sequence of standard amino acids that can synthesize a specific protein.

recent years suggests that this number is limited to about 20,500 genes. The rest of the DNA sequence is known as noncoding. Some of the remaining (over 98% in humans) areas of the DNA sequence serve different functions, from sequence alignment to regulatory mechanisms (approximately 8.2%) and are known as the *functional* part; the rest of the DNA sequence (roughly 91.8%) is known as "junk" DNA; see [42]. Sometimes an organism's DNA encodes different proteins with overlapping reading frames, producing different (valid) results for various start/end codon offsets. This problem is not much different than decoding a serial bit stream into bytes in an asynchronous mode of operation. In short, the DNA strand is an encoding scheme that employs (a) two complementary streams (double helix) and (b) redundancy in symbols and overlapping reading frames, with (c) additional information outside the actual "useful" coding blocks ("*exons*").

Protein folding prediction as a classification task

Each standard amino acid sequence is an encoding scheme for a specific protein. The corresponding molecular 3-D structure of each protein is one of the most important aspects of proteomics. One of the main structural properties of a 3-D molecular structure is known as *fold*, and it is associated with the way that a protein is deployed in space. Each fold affects various chemical properties and inherent functionality of the corresponding gene in a DNA strand, which are of paramount importance to biologists. The prediction of the protein folding from the amino acid sequence has been one of the most challenging tasks since the creation of detailed genome databases, including the human genome project. Biologists have identified a number of possible folds that a protein can acquire, and each one of them is associated with certain properties. It is thus very important, once an unknown protein is identified (equivalently, once an encoding protein sequence is detected), to be able to find the folds associated with the 3-D structural form that the corresponding molecule acquires in space. This is achieved by trying to classify the new finding into one of the previously known classes, where each class is associated with a specific fold and its respective properties. This now becomes a typical classification task.

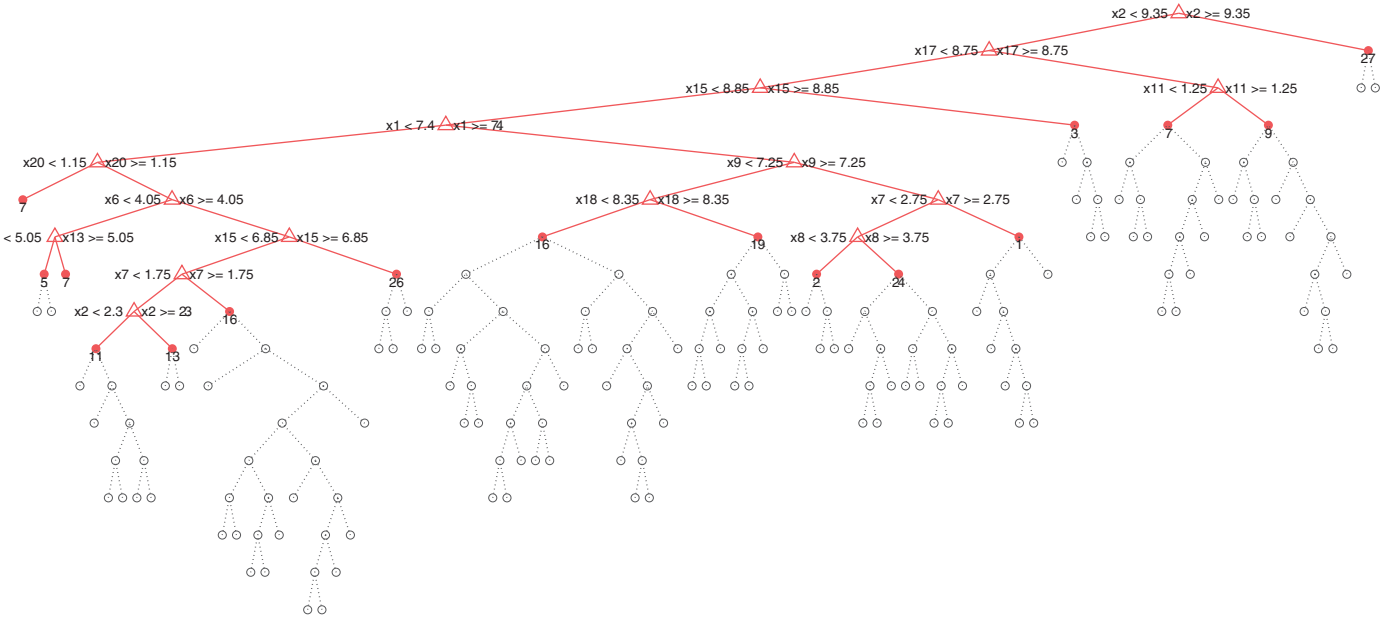

FIGURE 7.17

CART decision tree for the benchmark data set – Node details and thresholds for the first few levels.

Classification of folding prediction via decision trees

The data set used is available at UCSD-MKL repository [37] for protein fold prediction, based on a subset of the PDB-40D SCOP collection [41]. This data set contains two splits: a training subset with 311 samples and a testing subset with 383 samples. In the original data set, for every sample there are 12 different sets of feature vectors, each one giving rise to a different feature space. In this example, the 20-value "composition" vector was used: Once a DNA encoding sequence has been identified, the corresponding feature vector is constructed according to the relative frequencies (%) of the appearance of each one of the 20 standard amino acids in the sequence, which corresponds to the sample [39, 40]. Indeed, the composition vector has been established as a valid metric correlated with the 3-D properties of the corresponding protein molecule [2, 12]. Thus, the dimensionality of the feature space is equal to 20. The number of classes we are going to consider is 27, which is a selected subset comprising the most characteristic protein folds in the original PDB-40B database (from a total of more than 600 classes-folds).

The classifier model used in this example was a typical classification and regression tree - CART (Section 7.8) and it was carried out utilizing the Statistics toolbox of MATLAB (ver 8.0+).

Figure 7.17 illustrates the first few levels of the trained CART decision tree. Internal nodes include the (binary) decision threshold associated with it; for example, $x1 < 7.4$ tests whether the relative frequency of the alphabetically first amino acid (Ala) in the considered sample protein sequence is lower than 7.4%. The overall classification accuracy in the testing subset is 31.85%, which is close to the performance of other more advanced classifiers, including multi-layered perceptrons (Chapter 18) and support vector machines (Chapter 11) applied on the same task, i.e., using only the composition vector as input; see [15].

Note that even though the classification accuracy is low for a fully automated procedure, these predictive models can be used as valuable tools for limiting the search scope and prioritizing the relevant experiments for the characterization of new proteins [34].

Our focus was to present an application area of immense interest, while the approach we followed was rather on the pedagogic side, by employing simple features and a single classifier. More state-of-the-art approaches employ much more descriptive statistics as feature vectors, instead of the composition vector as described above. For example, the 20×20 correlation matrix of subsequent amino acids can be calculated from the coding sequence and used as input in classification schemes [10, 34]. A current trend is to combine classifiers as well as feature spaces with additional information content (i.e., not only the coding sequence itself), and classification accuracy rates up to 70% have been reported [13, 24, 31].

PROBLEMS

7.1 Show that the Bayesian classifier is optimal, in the sense that it minimizes the probability of error.

Hint. Consider a classification task of M classes and start with the probability of correct label prediction, $P(C)$. Then the probability of error will be $P(e) = 1 - P(C)$.

7.2 Show that if the data follow the Gaussian distribution in an M class task, with equal covariance matrices in all classes, the regions formed by the Bayesian classifier are convex.

7.3 Derive the form of the Bayesian classifier for the case of two equiprobable classes, when the data follow the Gaussian distribution of the same covariance matrix. Furthermore, derive the equation that describes the LS linear classifier. Compare and comment on the results.

7.4 Show that the ML estimate of the covariance matrix of a Gaussian distribution, based on N i.i.d. observations, $\boldsymbol{x}_n,\ n = 1, 2, \ldots, N$, is given by

$$\hat{\Sigma}_{ML} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ML})^{\mathrm{T}},$$

where

$$\hat{\boldsymbol{\mu}}_{ML} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n.$$

7.5 Prove that the covariance estimate

$$\hat{\Sigma} = \frac{1}{N-1}\sum_{k=1}^{N}(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})^T$$

defines an unbiased estimator, where

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{k=1}^{N}\boldsymbol{x}_k.$$

7.6 Show that the derivative of the logistic link function is given by

$$\frac{d\sigma(t)}{dt} = \sigma(t)\left(1 - \sigma(t)\right).$$

7.7 Derive the gradient of the negative log-likelihood function associated with the two-class logistic regression.

7.8 Derive the Hessian matrix of the negative log-likelihood function associated with the two-class logistic regression.

7.9 Show that the Hessian matrix of the negative log-likelihood function of the two-class logistic regression is a positive definite matrix.

7.10 Show that if

$$\phi_m = \frac{\exp(t_m)}{\sum_{j=1}^{M}\exp(t_j)},$$

the derivative with respect to $t_j,\ j = 1, 2, \ldots, M$, is given by

$$\frac{\partial\phi_m}{\partial t_j} = \phi_m(\delta_{mj} - \phi_j).$$

7.11 Derive the gradient of the negative log-likelihood for the multiclass logistic regression case.

7.12 Derive the j, k block element of the Hessian matrix of the negative log-likelihood function for the multiclass logistic regression.

7.13 Consider the Rayleigh ratio,

$$R = \frac{\boldsymbol{\theta}^{\mathrm{T}}A\boldsymbol{\theta}}{||\boldsymbol{\theta}||^2},$$

where A is a symmetric positive definite matrix. Show that R is maximized, with respect to $\boldsymbol{\theta}$, if $\boldsymbol{\theta}$ is the eigenvector corresponding to the maximum eigenvalue of A.

7.14 Consider the generalized Rayleigh quotient,

$$R_g = \frac{\boldsymbol{\theta}^{\mathrm{T}} B \boldsymbol{\theta}}{\boldsymbol{\theta}^{\mathrm{T}} A \boldsymbol{\theta}}.$$

where A and B are a symmetric positive definite matrices. Show that R_g is maximized with respect to $\boldsymbol{\theta}$, if $\boldsymbol{\theta}$ is the eigenvector that corresponds to the maximum eigenvalue of $A^{-1}B$, assuming that the inversion is possible.

7.15 Show that the between-class scatter matrix Σ_b for an M class problem is of rank $M-1$.

7.16 Derive the arithmetic rule for combination, by minimizing the average KL distance.

7.17 Derive the product rule via the minimization of the Kullback-Leibler distance, as pointed out in the text.

7.18 Show that the error rate on the training set of the final classifier, obtained by boosting, tends to zero exponentially fast.

MATLAB Exercises

7.19 Consider a two-dimensional class problem that involves two classes, ω_1 and ω_2, which are modeled by Gaussian distributions with means $\boldsymbol{\mu}_1 = [0,\ 0]^{\mathrm{T}}$ and $\boldsymbol{\mu}_2 = [2,\ 2]^{\mathrm{T}}$, respectively, and common covariance matrix $\Sigma = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix}$.

- **(i)** Form and plot a data set $\mathcal{X}$ consisting from 500 points from ω_1 and another 500 points from ω_2.
- **(ii)** Assign each one of the points of $\mathcal{X}$ to either ω_1 or ω_2, according to the Bayes decision rule, and plot the points with different colors, depending on the class they are assigned to. Plot the corresponding classifier.
- **(iii)** Based on (ii), estimate the error probability.
- **(iv)** Let $L = \begin{bmatrix} 0 & 1 \\ 0.005 & 0 \end{bmatrix}$ be a loss matrix. Assign each one of the points of $\mathcal{X}$ to either ω_1 or ω_2, according to the average risk minimization rule (Eq. (7.9)), and plot the points with different colors, depending on the class they are assigned to.
- **(v)** Based on (iv), estimate the average risk for the above loss matrix.
- **(vi)** Comment on the results obtained by (ii)-(iii) and (iv)-(v) scenarios.

7.20 Consider a two-dimensional class problem that involves two classes, ω_1 and ω_2, which are modeled by Gaussian distributions with means $\boldsymbol{\mu}_1 = [0,\ 2]^{\mathrm{T}}$ and $\boldsymbol{\mu}_2 = [0,\ 0]^{\mathrm{T}}$ and covariance matrices $\Sigma_1 = \begin{bmatrix} 4 & 1.8 \\ 1.8 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 4 & 1.2 \\ 1.2 & 1 \end{bmatrix}$, respectively.

- **(i)** Form and plot a data set $\mathcal{X}$ consisting from 5000 points from ω_1 and another 500 points from ω_2.
- **(ii)** Assign each one of the points of $\mathcal{X}$ to either ω_1 or ω_2, according to the Bayes decision rule, and plot the points with different colors, according to the class they are assigned to.
- **(iii)** Compute the error classification probability.
- **(iv)** Assign each one of the points of $\mathcal{X}$ to either ω_1 or ω_2, according to the naive Bayes decision rule, and plot the points with different colors, according to the class they are assigned to.
- **(v)** Compute the error classification probability, for the naive Bayes classifier.

(vii) Repeat steps (i)-(v) for the case where $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$.

(viii) Comment on the results.

Hint. Use the fact that the marginal distributions of $P(\omega_1|\boldsymbol{x})$, $P(\omega_1|x_1)$, and $P(\omega_1|x_2)$ are also Gaussians with means 0 and 2 and variances 4 and 1, respectively. Similarly, the marginal distributions of $P(\omega_2|\boldsymbol{x})$, $P(\omega_2|x_1)$, and $P(\omega_2|x_2)$ are also Gaussians with means 0 and 0 and variances 4 and 1, respectively.

7.21 Consider a two-class, two-dimensional classification problem, where the first class (ω_1) is modeled by a Gaussian distribution with mean $\boldsymbol{\mu}_1 = [0,\ 2]^{\mathrm{T}}$ and covariance matrix $\Sigma_1 = \begin{bmatrix} 4 & 1.8 \\ 1.8 & 1 \end{bmatrix}$, while the second class (ω_2) is modeled by a Gaussian distribution with mean $\boldsymbol{\mu}_2 = [0,\ 0]^{\mathrm{T}}$ and covariance matrix $\Sigma_2 = \begin{bmatrix} 4 & 1.8 \\ 1.8 & 1 \end{bmatrix}$.

(i) Generate and plot a training set $\mathcal{X}$ and a test set $\mathcal{X}_{test}$, each one consisting of 1500 points from each distribution.

(ii) Classify the data vectors of $\mathcal{X}_{test}$ using the Bayesian classification rule.

(iii) Perform logistic regression and use the data set $\mathcal{X}$ to estimate the involved parameter vector $\boldsymbol{\theta}$. Evaluate the classification error of the resulting classifier based on $\mathcal{X}_{test}$.

(iv) Comment on the results obtained by (ii) and (iii).

(v) Repeat the previous steps (i)-(iv), for the case where $\Sigma_2 = \begin{bmatrix} 4 & -1.8 \\ -1.8 & 1 \end{bmatrix}$ and compare the obtained results with those produced by the previous setting. Draw your conclusions.

Hint. For the estimation of $\boldsymbol{\theta}$ in (iii), perform steepest descent (Eq. (7.37)) and set the learning parameter μ_i equal to 0.001.

7.22 Consider a two-dimensional classification problem involving three classes ω_1, ω_2, and ω_3. The data vectors from ω_1 stem from either of the two Gaussian with means $\boldsymbol{\mu}_{11} = [0,\ 3]^{\mathrm{T}}$, $\boldsymbol{\mu}_{12} = [11,\ -2]^{\mathrm{T}}$ and covariance matrices $\Sigma_{11} = \begin{bmatrix} 0.2 & 0 \\ 0 & 2 \end{bmatrix}$ and $\Sigma_{12} = \begin{bmatrix} 3 & 0 \\ 0 & 0.5 \end{bmatrix}$, respectively. Similarly, the data vectors from ω_2 stem from either of the two Gaussians distributions with means $\boldsymbol{\mu}_{21} = [3,\ -2]^{\mathrm{T}}$, $\boldsymbol{\mu}_{22} = [7.5,\ 4]^{\mathrm{T}}$ and covariance matrix $\Sigma_{21} = \begin{bmatrix} 5 & 0 \\ 0 & 0.5 \end{bmatrix}$ and $\Sigma_{22} = \begin{bmatrix} 7 & 0 \\ 0 & 0.5 \end{bmatrix}$, respectively. Finally, ω_3 is modeled by a single Gaussian distribution with mean $\boldsymbol{\mu}_3 = [7,\ 2]^{\mathrm{T}}$ and covariance matrix $\Sigma_3 = \begin{bmatrix} 8 & 0 \\ 0 & 0.5 \end{bmatrix}$.

(i) Generate and plot a training data set $\mathcal{X}$ consisting of 1000 data points from ω_1 (500 from each distribution), 1000 data points from ω_2 (again 500 from each distribution), and 500 points from ω_3 (use 0 as the seed for the initialization of the Gaussian random number generator). In a similar manner, generate a test data set $\mathcal{X}_{test}$ (use 100 as the seed for the initialization of the Gaussian random number generator).

(ii) Generate and view a decision tree based on using $\mathcal{X}$ as the training set.

(iii) Compute the classification error on both the training and the test sets. Comment briefly on the results.

(iv) Prune the produced tree at levels 0 (no actual pruning), $1, \ldots, 11$ (In MATLAB, trees are pruned based on an optimal pruning scheme that first prunes branches giving less improvement in error cost). For each pruned tree compute the classification error based on the test set.

(v) Plot the classification error versus the pruned levels and locate the pruned level that gives the minimum test classification error. What conclusions can be drawn by the inspection of this plot?

(vi) View the original decision tree as well as the best pruned one.

Hint. The MATLAB functions that generate a decision tree (DT), display a DT, prune a DT, evaluate the performance of a DT on a given data set, are *classregtree*, *view*, *prune*, and *eval*, respectively.

7.23 Consider a two-class, two-dimensional classification problem where the classes are modeled as the first two classes in the previous exercise.

(i) Generate and plot a training set $\mathcal{X}$, consisting of 100 data points from each distribution of each class (that is, $\mathcal{X}$ contains 400 points in total, 200 points from each class). In a similar manner, generate a test set.

(ii) Use the training set to built a boosting classifier, utilizing as weak classifier a single-node decision tree. Perform $12,000$ iterations.

(iii) Plot the training and the test error versus the number of iterations and comment on the results.

Hint.
- For (i) use $randn('seed', 0)$ and $randn('seed', 100)$ to initialize the random number generator for the training and the test set, respectively.
- For (ii) use
  $ens = fitensemble(X', y, 'AdaBoostM1', no\_of\_base\_classifiers, 'Tree')$,
  where X' has in its rows the data vectors, y is an ordinal vector containing the class where each row vector of X' belongs, $AdaBoostM1$ is the boosting method used, $no\_of\_base\_classifiers$ is the number of base classifiers that will be used, and *Tree* denotes the weak classifier.
- For (iii) use $L = loss(ens, X', y, 'mode', 'cumulative')$, which for a given boosting classifier *ens*, returns the vector L of errors performed on X', such that $L(i)$ being the error committed when only the first i weak classifiers are taken into account.

7.24 Consider the classification task for protein folding prediction as described in Section 7.12. Using the same subset of the PDB-40D SCOP collection [41] from the UCSD-MKL repository [37], write a MATLAB program to reproduce these results.

(i) Read the training subset $\mathcal{X}$, consisting of 311 data points, and the testing subset $\mathcal{Y}$, consisting of 383 data points. Each data point represents a sample "fold" described by the 20-value "composition" feature vector of amino acids and assigned to one of the 27 classes.

(ii) Using the Statistics toolbox of MATLAB, create and train a standard CART in classification mode for this task. Evaluate the classifier using the testing subset and report the overall accuracy rate.

Hint. – For (ii) use the 'ClassificationTree' object from the Statistics toolbox in MATLAB (v8.1+).
– The tree object provides a "fit" method for training and a "predict" method for testing the constructed model.

REFERENCES

[1] J.M. Berg, J.L. Tymoczko, L. Stryer, Biochemistry, fifth ed., Freedman, New York, 2002.
[2] H. Bohr et al., A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks, FEBS Lett. 261 (1990) 43-46.
[3] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
[4] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth, 1984.
[5] L. Breiman, Bagging predictors, Machine Learn. 24 (1996) 123-140.
[6] L. Breiman, Arcing classifiers, Ann. Stat. 26(3) (1998) 801-849.
[7] L. Breiman, Random forests, Machine Learn. 45 (2001) 5-32.
[8] P. Bühlman, T. Hothorn, Boosting algorithms: regularization, prediction and model fitting (with discussion), Stat. Sci. 22(4) (2007) 477-505.
[9] A. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in International Conference on Machine Learning, 2006.
[10] Y. Chen, F. Ding, H. Nie, et al., Protein folding: Then and now, Arch. Biochem. Biophys. 469(1) (2008) 4-19.
[11] H. Chipman, E. George, R. McCulloch, BART: Bayesian additive regression trees, Ann. Appl. Stat. 4(1) (2010) 266-298.
[12] F. Crick, The recent excitement about neural networks, Nature 337 (1989) 129-132.
[13] A. Dehzangi, K. Paliwal, A. Sharma, O. Dehzangi, A. Sattar, A combination of feature extraction methods with an ensemble of different classifiers for protein structural class prediction problem, IEEE/ACM Trans. Comput. Biol. Bioinform. 10(3) (2013) 564-575.
[14] L. Devroye, L. Gyorfi, G.A. Lugosi, A Probabilistic Theory of Pattern Recognition, Springer Verlag city, 1996.
[15] C. Ding, I. Dubchak, Multi-class protein fold recognition using support vector machines and neural networks, Bioinformatics 17 (4) (2001) 349-358.
[16] J. Duchi, Y. Singer, Boosting with structural sparsity, in: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada, 2009.
[17] R. Duda, P. Hart, D. Stork, Pattern Classification, second ed., Wiley, New York, 2000.
[18] B. Efron, The efficiency of logistic regression compared to normal discriminant analysis, J. Amer. Stat. Assoc. 70 (1975) 892-898.
[19] G. Eibl, K.P. Pfeifer, Multiclass boosting for weak classifiers, J. Machine Learn. Res. 6 (2006) 189-210.
[20] Y. Freund, R.E. Schapire, A decision theoretic generalization of on-line learning and an applications to boosting, J. Comput. Syst. Sci. 55(1) (1997) 119-139.
[21] J. Friedman, T. Hastie, R. Tibshirani, Additive logisitc regression: a statistical view of boosting, Ann. Stat. 28(2) (2000) 337-407.
[22] J. Freidman, Greedy function approxiamtion: a gradient boosting machine, Ann. Stat. 29(5) (2001) 1189-1232.
[23] K. Fukunaga, Introduction to Statistical Pattern Recognition, second ed., Academic Press, 1990.

[24] P. Ghanty, N.R. Pal, Prediction of protein folds: extraction of new features, dimensionality reduction, and fusion of heterogeneous classifiers, IEEE Trans. NanoBiosci. 8(1) (2009) 100-110.
[25] I. Guyon, S. Gunn, M. Nikravesh, L. Zadeh (Eds.), Feature Extraction, Foundations and Applications, Springer Verlag, New York, 2006.
[26] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, second ed., Springer Verlag, 2009.
[27] R. Hu, R.I. Damper, A no panacea theorem for classifier combination, Pattern Recogn. 41 (2008) 2665-2673.
[28] A.K. Jain, P.W. Duin, J. Mao, Statistical pattern recognition: a review, IEEE Trans. Pattern Anal. Machine Intell. 22(1) (2000) 4-37.
[29] N. Johnson, A study of the NIPS feature selection challenge, Technical Report, Stanford University, http://statweb.stanford.edu/~tibs/ElemStatLearn/comp.pdf, 2009.
[30] M. Kearns, L.G. Valiant, Cryptographic limitations of learning Boolean formulae and finite automata, J. ACM 41(1) (1994) 67-95.
[31] K.-L. Lin, C.-Y. Lin, C.-D. Huang, et al., Feature selection and combination criteria for improving accuracy in protein structure prediction, IEEE Trans. NanoBiosci. 6(2) (2007) 186-196.
[32] J. Kittler, M. Hatef, R. Duin, J. Matas, On combining classifiers, IEEE Trans. Pattern Anal. Machine Intell. 20(3) (1998) 228-234.
[33] I.L. Kuncheva, Pattern Classifiers: Methods and Algorithms, John Wiley, 2004.
[34] L. Hunter (Ed.), Artificial Intelligence in Molecular Biology, AAAI/MIT Press, CA, 1993.
[35] D. Meyer, F. Leisch, K. Hornik, The support vector machine under test, Neurocomputing 55 (2003) 169-186.
[36] D. Michie, D.J. Spiegelhalter, C.C. Taylor (Eds.), Machine Learning, Neural, and Statistical Classification, Ellis Horwood, London, 1994.
[37] https://mldata.org/repository/data/viewslug/protein-fold-prediction-ucsd-mkl.
[38] R. Neal, J. Zhang, High dimensional classification with Bayesian neural networks and Dirichlet diffusion trees, in: I. Guyon, S. Gunn, M. Nikravesh, L. Zadeh (Eds.), Feature Extraction, Foundations and Applications, Springer Verlag, New York, 2006, pp. 265-296.
[39] K. Nishikawa, T. Ooi, Correlation of the amino acid composition of a protein to its structural and biological characteristics, J. Biochem. 91 (1982) 1821-1824.
[40] K. Nishikawa, Y. Kubota, T. Ooi, Classification of proteins into groups based on amino acid composition and other characters, J. Biochem. 94 (1983) 981-995.
[41] http://scop.berkeley.edu.
[42] C.M. Rands, S. Meader, C.P. Ponting, G. Lunter, 8.2% of the human genome is constrained: Variation in rates of turnover across functional element classes in the human lineage, PLOS Genet. 10(7) (2014) 1-12.
[43] G. Ratsch, M.K. Warmuth, Efficient margin maximizing with boosting, J. Machine Learn. Res. 6 (2005) 2131-2152.
[44] G. Ridgeway, The state of boosting, Comput. Sci. Stat. 31 (1999) 172-181.
[45] B.D. Ripley, Pattern Recognition and Neural Netowrks, Cambridge University Press, 1996.
[46] R.E. Schapire, V. Freund, P. Bartlett, W.S. Lee, Boosting the margin: a new explanation for the effectiveness of voting methods, Ann. Stat. 26(5) (1998) 1651-1686.
[47] R.E. Schapire, M. Rochery, M. Rahim, N. Gupta, Boosting with prior knowledge for call classification, IEEE Trans. Speech Audio Process. 13(2) (2005) 174-181.
[48] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A.A. Blake, Real-time human pose recognition in parts from single depth images, in: Proceedings of the Conference on Computer Vision and Pattern Recogntion, CVPR, 2011.
[49] R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, 1993.
[50] D.B. Rubin, Iterative reweighted least squares, in: Encyclopedia of Statistical Sciences, vol. 4, John Wiley, city 1983, pp. 272-275.
[51] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, third ed., Pearson, 2010.

[52] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, 2009.
[53] S. Theodoridis, A. Pikrakis, K. Koutroumbas, D. Cavouras, An Introduction to Pattern Recognition: A MATLAB Approach, Academic Press, 2010.
[54] L.G. Valiant, A theory of the learnable, Commun. ACM 27(11) (1984) 1134-1142.
[55] A. Webb, Statistical Pattern Recognition, second ed., John Wiley, 2002.
[56] D. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241-259.
[57] D. Wolpert, The lack of a priori distinctions between learning algorithms, Neural Comput. 8(7) (1996) 1341-1390.
[58] Y. Wu, H. Tjelmeland, M. West, Bayesian CART: Prior structure and MCMC computations, J. Comput. Graph. Stat. 16(1) (2007) 44-66.