4

LEARNING FROM DATA

Chapter Objectives

- Analyze the general model of inductive learning in observational environments.
- Explain how the learning machine selects an approximating function from the set of functions it supports.
- Introduce the concepts of risk functional for regression and classification problems.
- Identify the basic concepts in statistical learning theory (SLT) and discuss the differences between inductive principles, empirical risk minimization (ERM), and structural risk minimization (SRM).
- Discuss the practical aspects of the Vapnik-Chervonenkis (VC) dimension concept as an optimal structure for inductive-learning tasks.
- Compare different inductive-learning tasks using graphical interpretation of approximating functions in a two-dimensional (2-D) space.
- Explain the basic principles of support vector machines (SVMs).

Data Mining: Concepts, Models, Methods, and Algorithms, Second Edition. Mehmed Kantardzic.

- Specify K nearest neighbor classifier: algorithm and applications.
- Introduce methods for validation of inductive-learning results.

Many recent approaches to developing models from data have been inspired by the learning capabilities of biological systems and, in particular, those of humans. In fact, biological systems learn to cope with the unknown, statistical nature of the environment in a data-driven fashion. Babies are not aware of the laws of mechanics when they learn how to walk, and most adults drive a car without knowledge of the underlying laws of physics. Humans as well as animals also have superior pattern-recognition capabilities for such tasks as identifying faces, voices, or smells. People are not born with such capabilities, but learn them through data-driven interaction with the environment.

It is possible to relate the problem of learning from data samples to the general notion of inference in classical philosophy. Every predictive-learning process consists of two main phases:

1. learning or estimating unknown dependencies in the system from a given set of samples, and
2. using estimated dependencies to predict new outputs for future input values of the system.

These two steps correspond to the two classical types of inference known as *induction* (progressing from particular cases—training data—to a general dependency or model) and *deduction* (progressing from a general model and given input values to particular cases of output values). These two phases are shown graphically in Figure 4.1.

A unique estimated model implies that a learned function can be applied everywhere, that is, for all possible input values. Such global-function estimation can be overkill, because many practical problems require one to deduce estimated outputs only for a few given input values. In that case, a better approach may be to estimate the outputs of the unknown function for several points of interest directly from the training data without building a global model. Such an approach is called *transductive* inference in which a local estimation is more important than a global one. An important applica-

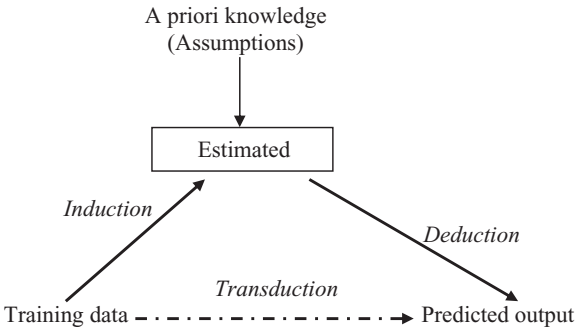

Figure 4.1. Types of inference: induction, deduction, and transduction.

tion of the *transductive* approach is a process of mining association rules, which is described in detail in Chapter 8. It is very important to mention that the standard formalization of machine learning does not apply to this type of inference.

The process of inductive learning and estimating the model may be described, formalized, and implemented using different learning methods. A learning method is an algorithm (usually implemented in software) that estimates an unknown mapping (dependency) between a system's inputs and outputs from the available data set, namely, from known samples. Once such a dependency has been accurately estimated, it can be used to predict the future outputs of the system from the known input values. Learning from data has been traditionally explored in such diverse fields as statistics, engineering, and computer science. Formalization of the learning process and a precise, mathematically correct description of different inductive-learning methods were the primary tasks of disciplines such as SLT and artificial intelligence. In this chapter, we will introduce the basics of these theoretical fundamentals for inductive learning.

4.1 LEARNING MACHINE

Machine learning, as a combination of artificial intelligence and statistics, has proven to be a fruitful area of research, spawning a number of different problems and algorithms for their solution. These algorithms vary in their goals, in the available training data sets, and in the learning strategies and representation of data. All of these algorithms, however, learn by searching through an n-dimensional space of a given data set to find an acceptable generalization. One of the most fundamental machine-learning tasks is *inductive machine learning* where a generalization is obtained from a set of samples, and it is formalized using different techniques and models.

We can define inductive learning as the process of estimating an unknown input–output dependency or structure of a system, using limited number of observations or measurements of inputs and outputs of the system. In the theory of inductive learning, all data in a learning process are organized, and each instance of input–output pairs we use a simple term known as a sample. The general learning scenario involves three components, represented in Figure 4.2.

1. a generator of random input vectors X,
2. a system that returns an output Y for a given input vector X, and

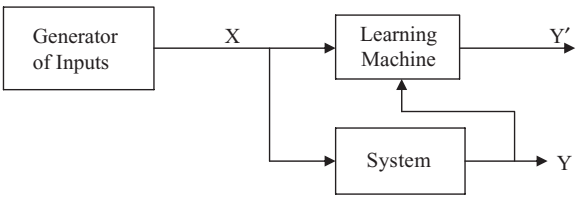

Figure 4.2. A learning machine uses observations of the system to form an approximation of its output.

3. a learning machine that estimates an unknown (input X, output Y´) mapping of the system from the observed (input X, output Y) samples.

This formulation is very general and describes many practical, inductive-learning problems such as interpolation, regression, classification, clustering, and density estimation. The generator produces a random vector X, which is drawn independently from any distribution. In statistical terminology, this situation is called an observational setting. It differs from the designed-experiment setting, which involves creating a deterministic sampling scheme, optimal for a specific analysis according to experimental design theory. The learning machine has no control over which input values were supplied to the system, and therefore, we are talking about an observational approach in inductive machine-learning systems.

The second component of the inductive-learning model is the system that produces an output value Y for every input vector X according to the conditional probability p(Y/X), which is unknown. Note that this description includes the specific case of a deterministic system where Y = f(X). Real-world systems rarely have truly random outputs; however, they often have unmeasured inputs. Statistically, the effects of these unobserved inputs on the output of the system can be characterized as random and represented with a probability distribution.

An inductive-learning machine tries to form generalizations from particular, true facts, which we call the training data set. These generalizations are formalized as a set of functions that approximate a system's behavior. This is an inherently difficult problem, and its solution requires a priori knowledge in addition to data. All inductive-learning methods use a priori knowledge in the form of the selected class of approximating functions of a learning machine. In the most general case, the learning machine is capable of implementing a set of functions f(X, w), w $\in$ W, where X is an input, w is a parameter of the function, and W is a set of abstract parameters used only to index the set of functions. In this formulation, the set of functions implemented by the learning machine can be any set of functions. Ideally, the choice of a set of approximating functions reflects a priori knowledge about the system and its unknown dependencies. However, in practice, because of the complex and often informal nature of a priori knowledge, specifying such approximating functions may be, in many cases, difficult or impossible.

To explain the selection of approximating functions, we can use a graphical interpretation of the inductive-learning process. The task of inductive inference is this: Given a collection of samples (xi, f[xi]), return a function h(x) that approximates f(x). The function h(x) is often called a hypothesis. Figure 4.3 shows a simple example of

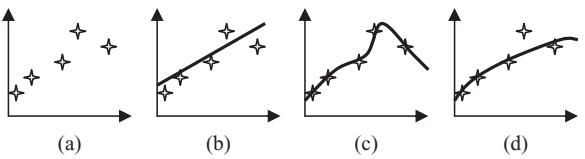

Figure 4.3. Three hypotheses for a given data set.

this task, where the points in 2-D are given in Figure 4.3a, and it is necessary to find "the best" function through these points. The true f(x) is unknown, so there are many choices for h(x). Without more knowledge, we have no way of knowing which one to prefer among three suggested solutions (Fig. 4.3b,c,d). Because there are almost always a large number of possible, consistent hypotheses, all learning algorithms search through the solution space based on given criteria. For example, the criterion may be a linear approximating function that has a minimum distance from all given data points. This a priori knowledge will restrict the search space to the functions in the form given in Figure 4.3b.

There is also an important distinction between the two types of approximating functions we usually use in an inductive-learning process. Their parameters could be *linear or nonlinear.* Note that the notion of linearity is with respect to parameters rather than input variables. For example, polynomial regression in the form

$$Y = w_1 x^n + w_2 x^{n-1} + \ldots + w_0$$

is a linear method, because the parameters w_i in the function are linear (even if the function by itself is nonlinear). We will see later that some learning methods such as multilayer, artificial, neural networks provide an example of nonlinear parametrization, since the output of an approximating function depends nonlinearly on parameters. A typical factor in these functions is e^{-ax}, where a is a parameter and x is the input value. Selecting the approximating functions f(X, w) and estimating the values of parameters w are typical steps for every inductive-learning method.

Before further formalization of a learning process, it is necessary to make a clear distinction between two concepts that are highly connected with a learning process. Let us discuss the differences between *statistical dependency and causality.* The statistical dependency between input X and output Y is expressed with the approximating functions of the learning method. The main point is that causality cannot be inferred from data analysis alone and concluded with some inductive, learned model using input–output approximating functions, Y = f(X, w); instead, it must be assumed or demonstrated by arguments outside the results of inductive-learning analysis. For example, it is well known that people in Florida are on average older than in the rest of the United States. This observation may be supported by inductive-learning dependencies, but it does not imply, however, that the climate in Florida causes people to live longer. The cause is totally different; people just move there when they retire and that is possibly the cause, and maybe not the only one, of people being older in Florida than elsewhere. Similar misinterpretation could be based on the data analysis of life expectancy for a married versus a single man. Statistics show that the married man lives longer than the single man. But do not hurry with sensational causality and conclusions: that marriage is good for one's health and increases life expectancy. It can be argued that males with physical problems and/or socially deviant patterns of behavior are less likely to get married, and this could be one of possible explanations why married men live longer. Unobservable factors such as a person's health and social behavior are more likely the cause of changed life expectancy, and not the observed variable, marriage status. These illustrations should lead us to understand that inductive-learning processes build the

model of dependencies but they should not automatically be interpreted as causality relations. Only experts in the domain where the data are collected may suggest additional, deeper semantics of discovered dependencies.

Let us return again to the learning machine and its task of system modeling. The problem encountered by the learning machine is to select a function from the set of functions this machine supports, which best approximates the system's responses. The learning machine is limited to observing a finite number of samples *n* in order to make this selection. The finite number of samples, which we call a training data set, is denoted by (X_i, y_i), where i = 1, . . . , n. The quality of an approximation produced by the learning machine is measured by the *loss function* L(y, f[X, w]), where

- y is the output produced by the system,
- X is a set of inputs,
- f(X, w) is the output produced by the learning machine for a selected approximating function, and
- w is the set of parameters in the approximating functions.

L measures the difference between the outputs produced by the system y_i and that produced by the learning machine $f(X_i,w)$ for every input point X_i. By convention, the loss function is nonnegative, so that large positive values correspond to poor approximation, and small positive values close to 0 show a good approximation. The expected value of the loss is called the *risk functional* R(w)

$$R(w) = \iint L(y, f[X, w])\, p(X, y)\, dX\, dy$$

where L(y, f[X, w])is a loss function and p(X, y) is a probability distribution of samples. The R(w) value, for a selected approximating functions, is dependent only on a set of parameters w. Inductive learning can be now defined as the process of estimating the function $f(X,w_{opt})$, which minimizes the risk functional R(w) over the set of functions supported by the learning machine, using only the training data set, and not knowing the probability distribution p(X, y). With finite data, we cannot expect to find $f(X, w_{opt})$ exactly, so we denote $f(X, w^*_{opt})$ as the estimate of parameters w^*_{opt} of the optimal solution w_{opt} obtained with finite training data using some learning procedure.

For common learning problems such as classification or regression, the nature of the loss function and the interpretation of risk functional are different. In a two-class classification problem, where the output of the system takes on only two symbolic values, y = {0, 1}, corresponding to the two classes, a commonly used loss function measures the classification error.

$$L(y, f(X, w)) = \begin{cases} 0 & \text{if } y = f(X, w) \\ 1 & \text{if } y \neq f(X, w) \end{cases}$$

Using this loss function, the risk functional quantifies the *probability of misclassification*. Inductive learning becomes a problem of finding the classifier function

f(X, w), which minimizes the probability of misclassification using only the training data set.

Regression is a process of estimating a real-value function based on a finite data set of noisy samples. A common loss function for regression is the squared error measure

$$L(y, f[X, w]) = (y - f[X, w])^2$$

The corresponding risk functional measures the *accuracy* of the learning machine's predictions of the system output. Maximum accuracy will be obtained by minimizing the risk functional because, in that case, the approximating function will describe the best set of given samples. Classification and regression are only two of many typical learning tasks. For the other data-mining tasks, different loss functions may be selected and they are supported with different interpretations of a risk functional.

What is a learning procedure? Or how should a learning machine use training data? The answer is given by the concept known as *inductive principle*. An inductive principle is a general prescription for obtaining an estimate $f(X,w_{opt}*)$ in the class of approximating functions from the available finite training data. An inductive principle tells us *what* to do with the data, whereas the learning method specifies *how* to obtain an estimate. Hence a learning method or learning algorithm is a constructive implementation of an inductive principle. For a given inductive principle, there are many learning methods corresponding to a different set of functions of a learning machine. The important issue here is to choose the candidate models (approximating functions of a learning machine) of the right complexity to describe the training data.

The mathematical formulation and formalization of the learning problem explained in this section may give the unintended impression that learning algorithms do not require human intervention, but this is clearly not the case. Even though available literature is concerned with the formal description of learning methods, there is an equally important, informal part of any practical learning system. This part involves such practical and human-oriented issues as selection of the input and output variables, data encoding and representation, and incorporating a priori domain knowledge into the design of a learning system. In many cases, the user also has some influence over the generator in terms of the sampling rate or distribution. The user very often selects the most suitable set of functions for the learning machine based on his/her knowledge of the system. This part is often more critical for an overall success than the design of the learning machine itself. Therefore, all formalizations in a learning theory are useful only if we keep in mind that inductive learning is a process in which there is some overlap between activities that can be formalized and others that are not a part of formalization.

4.2 SLT

SLT is relatively new, but it is perhaps one of the best currently available formalized theories for finite-sample inductive learning. It is also known as the Vapnik-Chervonenkis (VC) theory. It rigorously defines all the relevant concepts for inductive learning and

provides mathematical proofs for most inductive-learning results. In contrast, other approaches such as neural networks, Bayesian inference, and decision rules are more engineering-oriented, with an emphasis on practical implementation without needing strong theoretical proofs and formalizations.

SLT effectively describes statistical estimation with small samples. It explicitly takes into account the sample size and provides quantitative description of the trade-off between the complexity of the model and the available information. The theory includes, as a special case, classical statistical methods developed for large samples. Understanding SLT is necessary for designing sound, constructive methods of inductive learning. Many nonlinear learning procedures recently developed in neural networks, artificial intelligence, data mining, and statistics can be understood and interpreted in terms of general SLT principles. Even though SLT is quite general, it was originally developed for pattern recognition or classification problems. Therefore, the widely known, practical applications of the theory are mainly for classification tasks. There is growing empirical evidence, however, of successful application of the theory to other types of learning problems.

The goal of inductive learning is to estimate unknown dependencies in a class of approximating functions using available data. The optimal estimate corresponds to the minimum expected risk functional that includes general distribution of data. This distribution is unknown, and the only available information about distribution is the finite training sample. Therefore, the only possibility is to substitute an unknown *true risk functional* with its approximation given as *empirical risk*, which is computable based on the available data set. This approach is called ERM and it represents the basic inductive principle. Using the ERM inductive principle, one seeks to find a solution f(X, w*) that minimizes the empirical risk expressed through the training error as a substitute for the unknown true risk, which is a measure of the true error on the entire population. Depending on the chosen loss function and the chosen class of approximating functions, the ERM inductive principle can be implemented by a variety of methods defined in statistics, neural networks, automatic learning, and so on. The ERM inductive principle is typically used in a learning setting where the model is given or approximated first and then its parameters are estimated from the data. This approach works well only when the number of training samples is large relative to the prespecified model complexity, expressed through the number of free parameters.

A general property necessary for any inductive principle including ERM is asymptotic *consistency*, which is a requirement that the estimated model converge to the true model or the best possible estimation, as the number of training samples grows large. An important objective of the SLT is to formulate the conditions under which the ERM principle is consistent. The notion of consistency is illustrated in Figure 4.4. When the number of samples increases, empirical risk also increases while true, expected risk decreases. Both risks approach the common minimum value of the risk functional: min R(w) over the set of approximating functions, and for an extra large number of samples. If we take the classification problem as an example of inductive learning, the empirical risk corresponds to the probability of misclassification for the training data, and the expected risk is the probability of misclassification averaged over a large amount of data not included into a training set, and with unknown distribution.

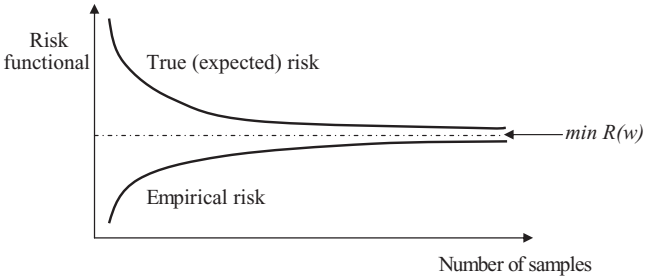

Figure 4.4. Asymptotic consistency of the ERM.

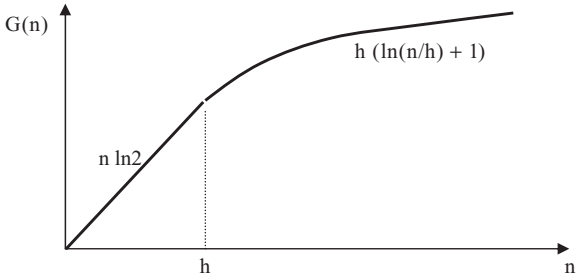

Figure 4.5. Behavior of the growth function G(n).

Even though it can be intuitively expected that for $n \to \infty$ the empirical risk converges to the true risk, this by itself does not imply the consistency property, which states that minimizing one risk for a given data set will also minimize the other risk. To ensure that the consistency of the ERM method is always valid and does not depend on the properties of the approximating functions, it is necessary that consistency requirement should hold for all approximating functions. This requirement is known as nontrivial consistency. From a practical point of view, conditions for consistency are at the same time prerequisites for a good generalization obtained with the realized model. Therefore, it is desirable to formulate conditions for convergence of risk functions in terms of the general properties of a set of the approximating functions.

Let us define the concept of a *growth function* G(n) as a function that is either linear or bounded by a logarithmic function of the number of samples n. Typical behavior of the growth function G(n) is given in Figure 4.5. Every approximating function that is in the form of the growth function G(n) will have a consistency property and potential for a good generalization under inductive learning, because empirical and true risk functions converge. The most important characteristic of the growth function G(n) is the concept of *VC dimension*. At a point n = h where the growth starts to slow down, it is a characteristic of a set of functions. If h is finite, then the G(n) function does not grow linearly for enough large training data sets, and it is bounded by a logarithmic function. If G(n) is only linear, then $h \to \infty$, and no valid generalization through selected approximating functions is possible. The finiteness of h provides necessary and sufficient conditions for the quick convergence of risk functions, consis-

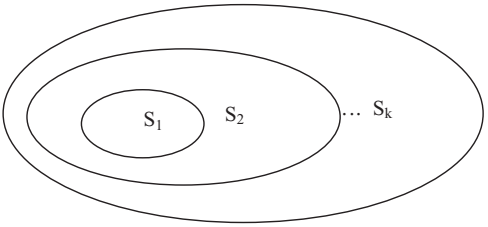

Figure 4.6. Structure of a set of approximating functions.

tency of ERM, and potentially good generalization in the inductive-learning process. These requirements place analytic constraints on the ability of the learned model to generalize, expressed through the empirical risk. All theoretical results in the SLT use the VC dimension defined on the set of loss functions. But, it has also been proven that the VC dimension for theoretical loss functions is equal to the VC dimension for approximating functions in typical, inductive-learning tasks such as classification or regression.

The ERM inductive principle is intended for relatively large data sets, namely, when the ratio n/h is large and the empirical risk converges close to the true risk. However, if n/h is small, namely, when the ratio n/h is less than 20, then a modification of the ERM principle is necessary. The inductive principle called SRM provides a formal mechanism for choosing a model with optimal complexity in finite and small data sets. According to SRM, solving a learning problem with a finite data set requires a priori specification of a structure on a set of approximating functions. For example, a set of functions S_1 is a subset of S_2, and S_2 is a subset of S_3. The set of approximating functions S_1 has the lowest complexity, but the complexity increases with each new superset S_2, S_3, . . . , S_k. A simplified graphical representation of the structure is given in Figure 4.6.

For a given data set, the optimal model estimation is performed following two steps:

1. selecting an element of a structure having optimal complexity, and
2. estimating the model based on the set of approximating functions defined in a selected element of the structure.

Through these two steps the SRM provides a quantitative characterization of the trade-off between the complexity of approximating functions and the quality of fitting the training data. As the complexity increases (increase of the index k for S_k), the minimum empirical risk decreases, and the quality of fitting the data improves. But estimated true risk, measured through the additional testing data set, has a convex form, and in one moment it moves in a direction opposite that of the empirical risk, as shown in Figure 4.7. The SRM chooses an optimal element of the structure that yields the minimal guaranteed bound on the true risk.

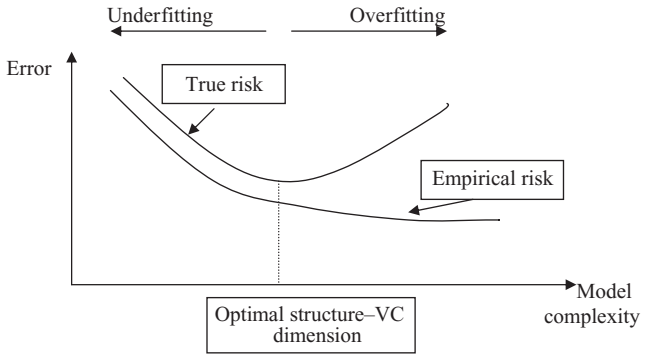

Figure 4.7. Empirical and true risk as a function of h (model complexity).

In practice, to implement the SRM approach, it is necessary to be able to

1. calculate or estimate the VC dimension for any element S_k of the structure, and then
2. minimize the empirical risk for each element of the structure.

For most practical inductive-learning methods that use nonlinear approximating functions, finding the VC dimension analytically is difficult, as is the nonlinear optimization of empirical risk. Therefore, rigorous application of the SRM principle cannot only be difficult but, in many cases, impossible with nonlinear approximations. This does not, however, imply that the SLT is impractical. There are various heuristic procedures that are often used to implement SRM implicitly. Examples of such heuristics include early stopping rules and weight initialization, which are often used in artificial neural networks. These heuristics will be explained together with different learning methods in the following chapters. The choice of an SRM-optimization strategy suitable for a given learning problem depends on the type of approximating functions supported by the learning machine. There are three commonly used optimization approaches:

1. *Stochastic Approximation (or Gradient Descent).* Given an initial estimate of the values for the approximating functions of parameter w, the optimal parameter values are found by repeatedly updating them. In each step, while computing the gradient of the risk function, the updated values of the parameters cause a small movement in the direction of the steepest descent along the risk (error) function.
2. *Iterative Methods.* Parameter values w are estimated iteratively so that at each iteration the value of the empirical risk is decreased. In contrast to stochastic approximation, iterative methods do not use gradient estimates; instead, they rely on a particular form of approximating functions with a special iterative parameter.
3. *Greedy Optimization.* The greedy method is used when the set of approximating functions is a linear combination of some basic functions. Initially, only the first term of the approximating functions is used and the corresponding

parameters optimized. Optimization corresponds to minimizing the differences between the training data set and the estimated model. This term is then held fixed, and the next term is optimized. The optimization process is repeated until values are found for all parameters w and for all terms in the approximating functions.

These typical optimization approaches and also other more specific techniques have one or more of the following problems:

1. *Sensitivity to Initial Conditions.* The final solution is very sensitive to the initial values of the approximation function parameters.
2. *Sensitivity to Stopping Rules.* Nonlinear approximating functions often have regions that are very flat, where some optimization algorithms can become "stuck" for a long time (for a large number of iterations). With poorly designed stopping rules these regions can be interpreted falsely as local minima by the optimization algorithm.
3. *Sensitivity to Multiple Local Minima.* Nonlinear functions may have many local minima, and optimization methods can find, at best, one of them without trying to reach global minimum. Various heuristics can be used to explore the solution space and move from a local solution toward a globally optimal solution.

Working with finite data sets, SLT reaches several conclusions that are important guidelines in a practical implementation of data-mining techniques. Let us briefly explain two of these useful principles. First, when solving a problem of inductive learning based on finite information, one should keep in mind the following general commonsense principle: Do not attempt to solve a specified problem by indirectly solving a harder general problem as an intermediate step. We are interested in solving a specific task, and we should solve it directly. Following SLT results, we stress that for estimation with finite samples, it is always better to solve a specific learning problem rather than attempt a general one. Conceptually, this means that posing the problem directly will then require fewer samples for a specified level of accuracy in the solution. This point, while obvious, has not been clearly stated in most of the classical textbooks on data analysis.

Second, there is a general belief that for inductive-learning methods with finite data sets, the best performance is provided by a model of optimal complexity, where the optimization is based on the general philosophical principle known as Occam's razor. According to this principle, limiting the model complexity is more important than using true assumptions with all details. We should seek simpler models over complex ones and optimize the trade-off between model complexity and the accuracy of the model's description and fit to the training data set. Models that are too complex and fit the training data very well or too simple and fit the data poorly are both not good models because they often do not predict future data very well. Model complexity is usually controlled in accordance with Occam's razor principle by a priori knowledge.

Summarizing SLT, in order to form a unique model of a system from finite data, any inductive-learning process requires the following:

1. A wide, flexible set of *approximating functions* f(X, w), w ∈ W, that can be linear or nonlinear in parameters w.
2. A priori knowledge (or assumptions) used to impose constraints on a potential solution. Usually such a priori knowledge orders the functions, explicitly or implicitly, according to some measure of their flexibility to fit the data. Ideally, the choice of a set of approximating functions reflects a priori knowledge about a system and its unknown dependencies.
3. An inductive principle, or method of inference, specifying what has to be done. It is a general prescription for combining a priori knowledge with available training data in order to produce an estimate of an unknown dependency.
4. A learning method, namely, a constructive, computational implementation of an inductive principle for a given class of approximating functions. There is a general belief that for learning methods with finite samples, the best performance is provided by a model of optimum complexity, which is selected based on the general principle known as Occam's razor. According to this principle, we should seek simpler models over complex ones and optimize the model that is the trade-off between model complexity and the accuracy of fit to the training data.

4.3 TYPES OF LEARNING METHODS

There are two common types of the inductive-learning methods. They are known as

1. supervised learning (or learning with a teacher), and
2. unsupervised learning (or learning without a teacher).

Supervised learning is used to estimate an unknown dependency from known input–output samples. Classification and regression are common tasks supported by this type of inductive learning. Supervised learning assumes the existence of a teacher—fitness function or some other external method of estimating the proposed model. The term "supervised" denotes that the output values for training samples are known (i.e., provided by a "teacher").

Figure 4.8a shows a block diagram that illustrates this form of learning. In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of input–output examples. The environment with its characteristics and model is, however, unknown to the learning system. The parameters of the learning system are adjusted under the combined influence of the training samples and the error signal. The error signal is defined as the difference between the desired response and the actual response of the learning system. Knowledge of the environment available to the teacher is transferred to the learning system through the training samples, which adjust the parameters of the learning system. It is a closed-loop feedback system, but the unknown environment is not in the loop. As a performance measure for the system, we may think in terms of the mean-square error or the

sum of squared errors over the training samples. This function may be visualized as a multidimensional error surface, with the free parameters of the learning system as coordinates. Any learning operation under supervision is represented as a movement of a point on the error surface. For the system to improve the performance over time and therefore learn from the teacher, the operating point on an error surface has to move down successively toward a minimum of the surface. The minimum point may be a local minimum or a global minimum. The basic characteristics of optimization methods such as stochastic approximation, iterative approach, and greedy optimization have been given in the previous section. An adequate set of input–output samples will move the operating point toward the minimum, and a supervised learning system will be able to perform such tasks as pattern classification and function approximation. Different techniques support this kind of learning, and some of them such as logistic regression, multilayered perceptron, and decision rules and trees will be explained in more detail in Chapters 5, 6, and 7.

Under the unsupervised learning scheme, only samples with input values are given to a learning system, and there is no notion of the output during the learning process. Unsupervised learning eliminates the teacher and requires that the learner form and evaluate the model on its own. The goal of unsupervised learning is to discover "natural" structure in the input data. In biological systems, perception is a task learned via unsupervised techniques.

The simplified schema of unsupervised or self-organized learning, without an external teacher to oversee the learning process, is indicated in Figure 4.8b. The emphasis in this learning process is on a task-independent measure of the quality of representation that is learned by the system. The free parameters w of the learning system are optimized with respect to that measure. Once the system has become tuned to the regularities of the input data, it develops the ability to form internal representations for encoding features of the input examples. This representation can be *global*, applicable to the entire input data set. These results are obtained with methodologies such as cluster analysis or some artificial neural networks, explained in Chapters 6 and 9. On the other hand, learned representation for some learning tasks can only be *local*, applicable to the specific subsets of data from the environment; association rules are a typical example of an appropriate methodology. It has been explained in more detail in Chapter 8.

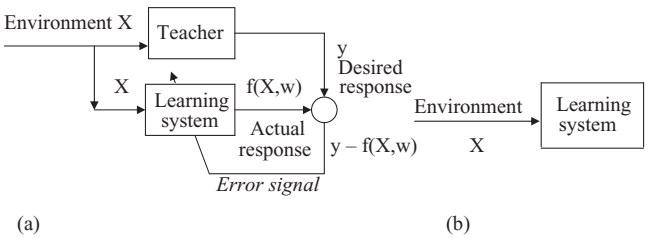

Figure 4.8. Two main types of inductive learning. (a) Supervised learning; (b) unsupervised learning.

4.4 COMMON LEARNING TASKS

The generic inductive-learning problem can be subdivided into several common learning tasks. The fundamentals of inductive learning, along with the classification of common learning tasks, have already been given in the introductory chapter of this book. Here, we would like to analyze these tasks in detail, keeping in mind that for each of these tasks, the nature of the loss function and the output differ. However, the goal of minimizing the risk based on training data is common to all tasks. We believe that visualization of these tasks will give the reader the best feeling about the complexity of the learning problem and the techniques required for its solution.

To obtain a graphical interpretation of the learning tasks, we start with the formalization and representation of data samples that are the "infrastructure" of the learning process. Every sample used in data mining represents one entity described with several attribute–value pairs. That is, one row in a tabular representation of a training data set, and it can be visualized as a point in an n-dimensional space, where n is the number of attributes (dimensions) for a given sample. This graphical interpretation of samples is illustrated in Figure 4.9, where a student with the name John represents a point in a 4-D space that has four additional attributes.

When we have a basic idea of the representation of each sample, the training data set can be interpreted as a set of points in the n-dimensional space. Visualization of data and a learning process is difficult for large number of dimensions. Therefore, we will explain and illustrate the common learning tasks in a 2-D space, supposing that the basic principles are the same for a higher number of dimensions. Of course, this approach is an important simplification that we have to take care of, especially keeping in mind all the characteristics of large, multidimensional data sets, explained earlier under the topic "the curse of dimensionality."

Let us start with the first and most common task in inductive learning: *classification*. This is a learning function that classifies a data item into one of several predefined classes. The initial training data set is given in Figure 4.10a. Samples belong to different

STUDENT NAME	SEX	YEAR OF BIRTH	MAJOR	CREDITS
John	M	1982	CS	64

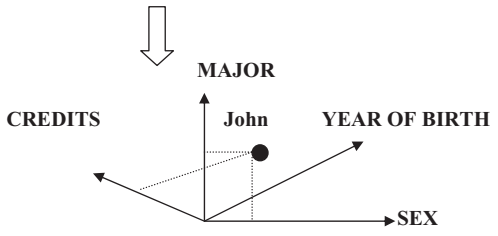

Figure 4.9. Data samples = points in an n-dimensional space.

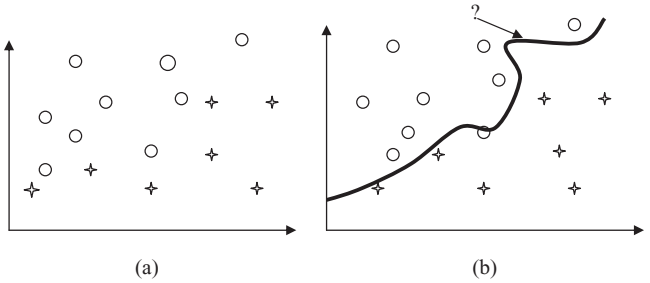

Figure 4.10. Graphical interpretation of classification. (a) Training data set; (b) classification function.

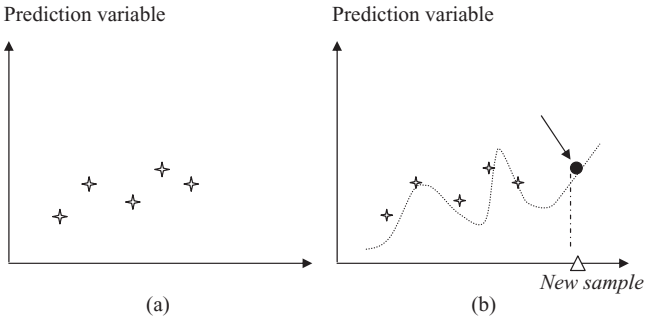

Figure 4.11. Graphical interpretation of regression. (a) Training data set; (b) regression function.

classes and therefore we use different graphical symbols to visualize each class. The final result of classification in a 2-D space is the curve shown in Figure 4.10b, which best separates samples into two classes. Using this function, every new sample, even without a known output (the class to which it belongs), may be classified correctly. Similarly, when the problem is specified with more than two classes, more complex functions are a result of a classification process. For an n-dimensional space of samples the complexity of the solution increases exponentially, and the classification function is represented in the form of hypersurfaces in the given space.

The second learning task is *regression* .The result of the learning process in this case is a learning function, which maps a data item to a real-value prediction variable. The initial training data set is given in Figure 4.11a. The regression function in Figure 4.11b was generated based on some predefined criteria built inside a data-mining technique. Based on this function, it is possible to estimate the value of a prediction variable for each new sample. If the regression process is performed in the time domain, specific subtypes of data and inductive-learning techniques can be defined.

Clustering is the most common unsupervised learning task. It is a descriptive task in which one seeks to identify a finite set of categories or clusters to describe the data. Figure 4.12a shows the initial data, and they are grouped into clusters, as shown in Figure 4.12b, using one of the standard distance measures for samples as points in an

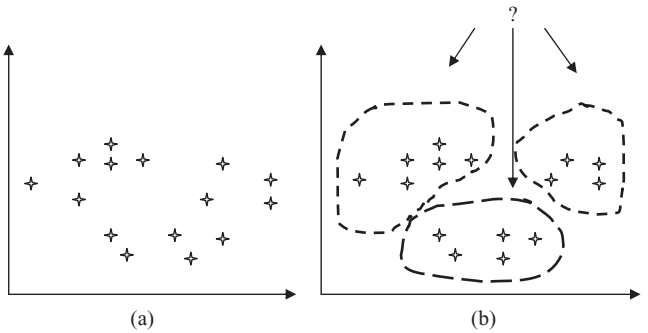

Figure 4.12. Graphical interpretation of clustering. (a) Training data set; (b) description of clusters.

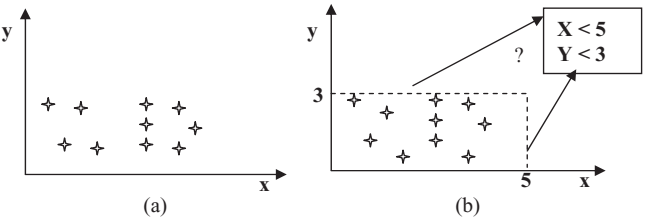

Figure 4.13. Graphical interpretation of summarization. (a) Training data set; (b) formalized description.

n-dimensional space. All clusters are described with some general characteristics, and the final solutions differ for different clustering techniques. Based on results of the clustering process, each new sample may be assigned to one of the previously found clusters, using its similarity with the cluster characteristics of the sample as a criterion.

Summarization is also a typical descriptive task, where the inductive-learning process is without a teacher. It involves methods for finding a *compact description* for a set (or subset) of data. If a description is formalized, as given in Figure 4.13b, that information may simplify and therefore improve the decision-making process in a given domain.

Dependency modeling is a learning task that discovers local models based on a training data set. The task consists of finding a model that describes significant dependency between features or between values in a data set covering not the entire data set, but only some specific subsets. An illustrative example is given in Figure 4.14b, where the ellipsoidal relation is found for one subset and a linear relation for the other subset of the training data. These types of modeling are especially useful in large data sets that describe very complex systems. Discovering general models based on the entire data set is, in many cases, almost impossible, because of the computational complexity of the problem at hand.

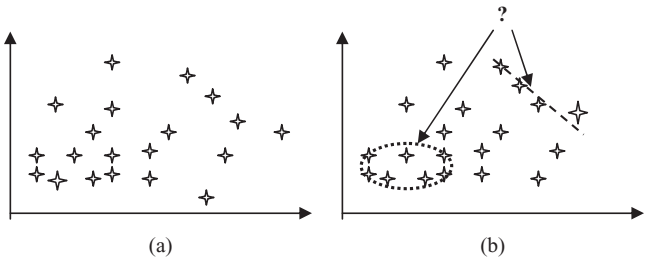

Figure 4.14. Graphical interpretation of dependency-modeling task. (a) Training data set; (b) discovered local dependencies.

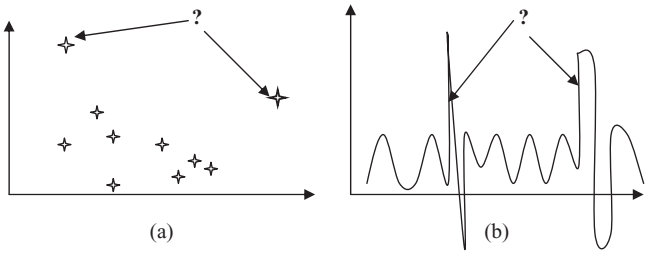

Figure 4.15. Graphical interpretation of change and detection of deviation (a) Outliers; (b) changes in time.

Change and deviation detection is a learning task, and we have been introduced already to some of its techniques in Chapter 2. These are the algorithms that detect outliers. In general, this task focuses on discovering the most significant changes in a large data set. Graphical illustrations of the task are given in Figure 4.15. In Figure 4.15a the task is to discover outliers in a given data set with discrete values of features. The task in Figure 4.15b is detection of time-dependent deviations for the variable in a continuous form.

The list of inductive-learning tasks is not exhausted with these six classes that are common specifications for data-mining problems. With wider and more intensive applications of the data-mining technology, new specific tasks are being developed, together with the corresponding techniques for inductive learning.

Whatever the learning task and whatever the available data-mining techniques are, we have to accept that the foundation for successful data-mining processes is data-preprocessing and data-reduction methods. They transform raw and usually messy data into valuable data sets for mining, using methodologies explained in Chapters 2 and 3. As a review, we will enumerate some of these techniques just to show how many alternatives the data-mining designer has in the beginning phases of the process: scaling and normalization, encoding, outliers detection and removal, feature selection and composition, data cleansing and scrubbing, data smoothing, missing-data elimination, and cases reduction by sampling.

When the data are preprocessed and when we know what kind of learning task is defined for our application, a list of data-mining methodologies and corresponding computer-based tools are available. Depending on the characteristics of the problem at hand and the available data set, we have to make a decision about the application of one or more of the data-mining and knowledge-discovery techniques, which can be classified as follows:

1. *Statistical Methods.* The typical techniques are Bayesian inference, logistic regression, analysis of variance (ANOVA), and log-linear models.
2. *Cluster Analysis.* The common techniques of which are divisible algorithms, agglomerative algorithms, partitional clustering, and incremental clustering.
3. *Decision Trees and Decision Rules.* The set of methods of inductive learning developed mainly in artificial intelligence. Typical techniques include the Concept Learning System (CLS) method, the ID3 algorithm, the C4.5 algorithm, and the corresponding pruning algorithms.
4 *Association Rules.* They represent a set of relatively new methodologies that include algorithms such as market basket analysis, a priori algorithm, and WWW path-traversal patterns.
5. *Artificial Neural Networks.* The common examples of which are multilayer perceptrons with backpropagation learning and Kohonen networks.
6. *Genetic Algorithms.* They are very useful as a methodology for solving hard-optimization problems, and they are often a part of a data-mining algorithm.
7. *Fuzzy Inference Systems.* These are based on the theory of fuzzy sets and fuzzy logic. Fuzzy modeling and fuzzy decision making are steps very often included in a data-mining process.
8. *N-dimensional Visualization Methods.* These are usually skipped in the literature as a standard data-mining methodology, although useful information may be discovered using these techniques and tools. Typical data-mining visualization techniques are geometric, icon-based, pixel-oriented, and hierarchical.

This list of data-mining and knowledge-discovery techniques is not exhaustive, and the order does not suggest any priority in the application of these methods. Iterations and interactivity are basic characteristics of these data-mining techniques. Also, with more experience in data-mining applications, the reader will understand the importance of not relying on a single methodology. Parallel application of several techniques that cover the same inductive-learning task is a standard approach in this phase of data mining. In that case, for each iteration in a data-mining process, the results of the different techniques must additionally be evaluated and compared.

4.5 SVMs

The foundations of SVMs have been developed by Vladimir Vapnik and are gaining popularity due to many attractive features, and promising empirical performance. The

formulation embodies the SRM principle. SVMs were developed to solve the classification problem, but recently they have been extended to the domain of regression problems (for prediction of continuous variables). SVMs can be applied to regression problems by the introduction of an alternative loss function that is modified to include a distance measure. The term SVM is referring to both classification and regression methods, and the terms Support Vector Classification (SVC) and Support Vector Regression (SVR) may be used for more precise specification.

An SVM is a supervised learning algorithm creating learning functions from a set of labeled training data. It has a sound theoretical foundation and requires relatively small number of samples for training; experiments showed that it is insensitive to the number of samples' dimensions. Initially, the algorithm addresses the general problem of learning to discriminate between members of two classes represented as n-dimensional vectors. The function can be a classification function (the output is binary) or the function can be a general regression function.

SVM's classification function is based on the concept of decision planes that define decision boundaries between classes of samples. A simple example is shown in Figure 4.16a where the samples belong either to class gray or black. The separating line defines a boundary on the right side of which all samples are gray, and to the left of which all samples are black. Any new unclassified sample falling to the right will be classified as gray (or classified as black should it fall to the left of the separating line).

The classification problem can be restricted to consideration of the two-class problem without loss of generality. Before considering n-dimensional analysis, let us look at a simple 2-D example. Assume that we wish to perform a classification, and our data has a categorical target variable with two categories. Also assume that there are two input attributes with continuous values. If we plot the data points using the value of one attribute on the x-axis and the other on the y-axis, we might end up with an image such as shown in the Figure 4.16b. In this problem the goal is to separate the two classes by a function that is induced from available examples. The goal is to produce a classifier that will work well on unseen examples, that is, it generalizes well. Consider the data in Figure 4.16b. Here there are many possible linear classifiers that

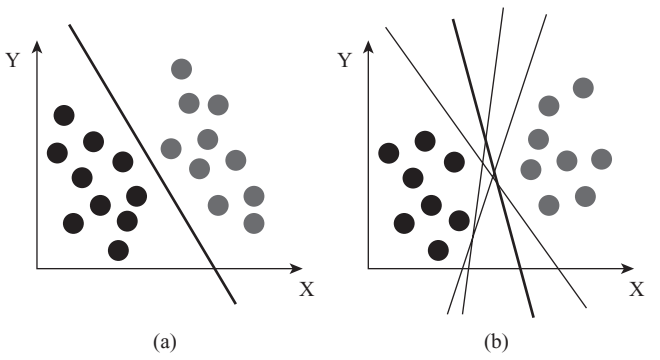

Figure 4.16. Linear separation in a 2-D space. (a) Decision plane in 2-D space is a line. (b) How to select optimal separating line?

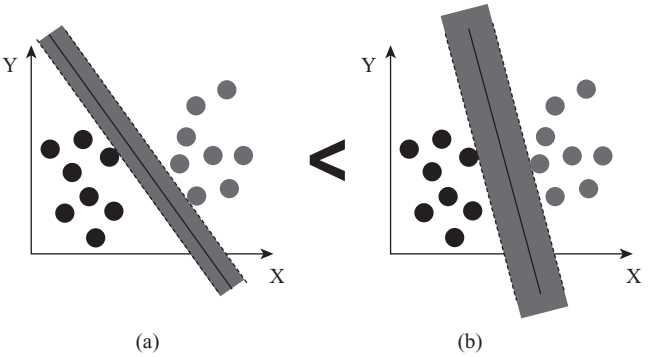

Figure 4.17. Comparison between sizes of margin of different decision boundaries. (a) Margin decision boundary 1; (b) margin decision boundary 2.

can separate the two classes of samples. Are all decision boundaries equally good? How do we prove that the selected one is the best?

The main idea is: The decision boundary should be as far away as possible from the data points of both classes. There is only one that maximizes the margin (maximizes the distance between it and the nearest data point of each class). Intuitively, the margin is defined as the amount of space, or separation between the two classes as defined by the hyperplane. Geometrically, the margin corresponds to the shortest distance between the closest data points to a point on the hyperplane. SLT suggests that the choice of the maximum margin hyperplane will lead to maximal generalization when predicting the classification of previously unseen examples.

Therefore, a linear SVM classifier is termed the optimal separating hyperplane with the maximum margin such as the margin in Figure 4.17b. The goal of SVM modeling in n-dimensional spaces is to find the optimal hyperplane that separates classes of n-dimensional vectors. The split will be chosen again to have the largest distance from the hypersurface to the nearest of the positive and negative samples. Intuitively, this makes the classification correct for testing data that is near, but not identical to the training data.

Why should we maximize the margin? Skinny *margin* is more flexible, thus more complex, and the complexity is not the goal. Fat *margin* is less complex. SRM principle expresses a trade-off between training error and model complexity. It recommends maximum margin, such as the one in Figure 4.18, as optimal separation criteria ensuring that SVM worst case generalization errors are minimized.

Based on the vector equation of the line in 2-D we can define function $f(x) = w\ x + b$ as a separation model. For all points above line $f(x) > 0$, and for the points below line $f(x) < 0$. The sign of this function $h(x) = \text{sign}(f[x])$ we define as a classification function because it has the value 1 for all points above the line, and the value -1 for all points below line. An example is given in Figure 4.19.

Before we continue, it is important to note that while the examples mentioned show a 2-D data set, which can be conveniently represented by points in a plane, in fact we will typically be dealing with higher dimensional data. The question is how to determine

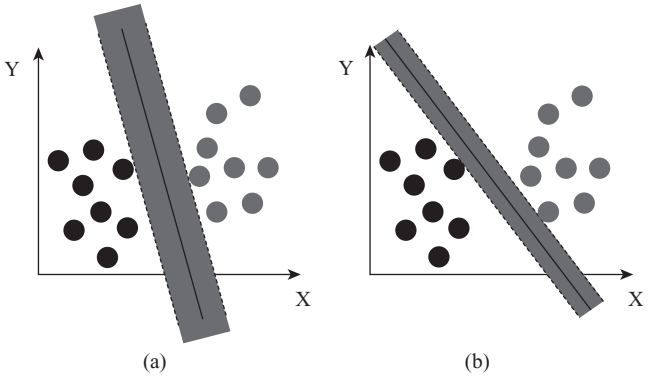

Figure 4.18. SRM principle expresses a trade-off between training error and model complexity. (a) "Fat" margin; (b) "skinny" margin.

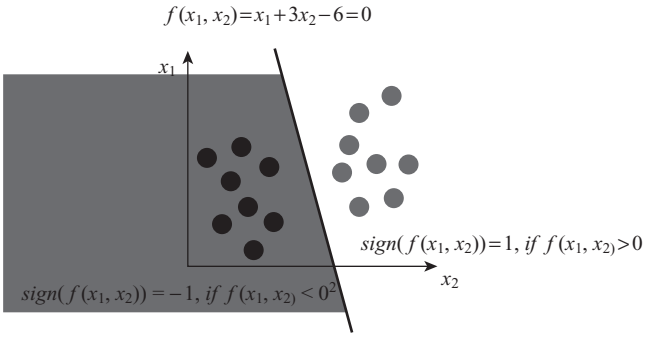

Figure 4.19. Classification function, sign(), on a 2-D space.

an optimal hyperplane in n-dimensional spaces based on a given set of training samples. Consider the problem of separating the set of training vectors D belonging to two classes (coded binary with −1 and 1).

$$D=\{(\mathbf{x}^1,\mathbf{y}^1),\ldots,(\mathbf{x}^l\ \mathbf{y}^l)\},\mathbf{x}\in\Re^n,\mathbf{y}\in\{-1,1\},$$

with a hyperplane

$$\langle\mathbf{w},\mathbf{x}\rangle+b=0.$$

The set of vectors is said to be optimally separated by the hyperplane if it is separated without error and the distance between the closest vectors to the hyperplane is maximal. An illustration of the separation with a graphical interpretation of main parameters w and b is given in Figure 4.20a. In this way we have parameterized the function by the weight vector w and the scalar *b*. The notation <w,x> denotes the inner or scalar product of vectors w and x, defined by

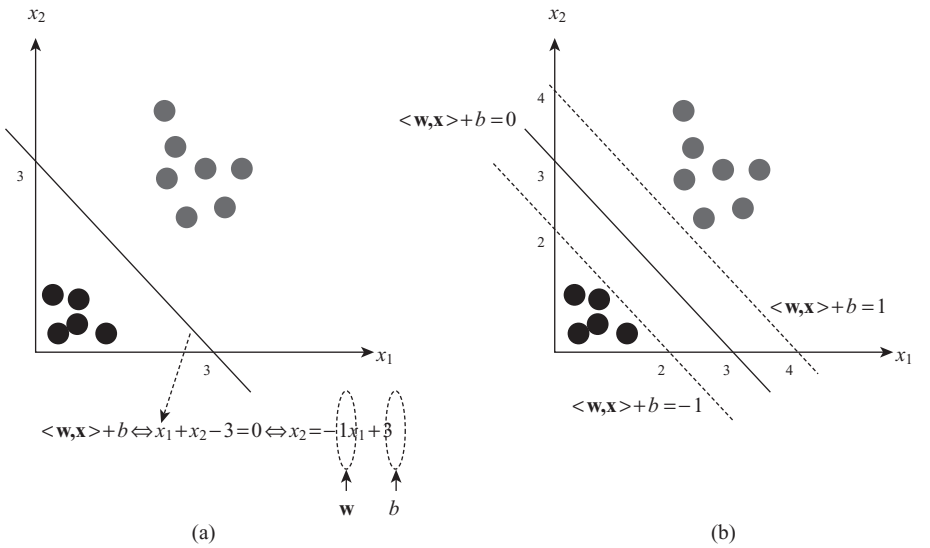

Figure 4.20. A separating hyperplane (w,b) for 2-D data. (a) Parameters **w** and *b*; (b) two parallel hyperplanes define margin.

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{n} w_i x_i$$

In order for our hyperplane to correctly separate the two classes, we need to satisfy the following constraints:

$$\langle \mathbf{w}, \mathbf{x}^i \rangle + b > 0, \quad \text{for all} \quad y^i = 1$$
$$\langle \mathbf{w}, \mathbf{x}^i \rangle + b < 0, \quad \text{for all} \quad y^i = -1$$

The set of constraints that we have so far is equivalent to saying that these data must lie on the correct side (according to class label) of this decision surface. Next notice that we have also plotted as dotted lines two other hyperplanes represented in Figure 4.20b, which are the hyperplanes where the function <w,x> + b is equal to −1 (on the lower left) and +1 (on the upper right). In order to find the maximum margin hyperplane, we can see intuitively that we should keep the dotted lines parallel and equidistant to the decision surface, and maximize their distance from one another, while satisfying the constraint that the data lie on the correct side of the dotted lines associated with that class. In mathematical form, the final clause of this sentence (the constraints) can be written as

$$y^i \left(\langle \mathbf{w}, \mathbf{x}^i \rangle + b \right) \geq 1, \quad i = 1, \ldots, l.$$

The distance between these two margin hyperplanes may be formalized, because it is the parameter we want to maximize. We may obtain the distance between hyperplanes in an n-dimensional space using equations

$$\langle \mathbf{w}, \mathbf{x}^1 \rangle + b = 1$$
$$\langle \mathbf{w}, \mathbf{x}^2 \rangle + b = -1$$

where x^1 and x^2 are any points on corresponding hyperplanes. If we subtract these equations

$$\langle \mathbf{w}, (\mathbf{x}^1 - \mathbf{x}^2) \rangle = 2$$

and representing scalar product of vectors by definition,

$$\mathrm{w}\ (\mathrm{x}_1 - \mathrm{x}_2)\ cos\ \gamma = 2$$

we obtain

$$\|\mathbf{w}\| \times d = 2$$

where || || represents Euclidean normor

$$d = \frac{2}{\|\mathbf{w}\|}$$

Therefore, the problem of "maximum margin" is represented as a maximum of a distance parameter d, which is a function of parameters w. Maximizing d means maximizing 1/|w| or minimizing |w|. The learning problem may be reformulated as:

$$\arg\min_{w} \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$$

subject to the constraints of linear separability. So, the final problem for optimization is

$$\arg\min_{\mathbf{w},b} \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) \quad \text{such that} \quad y^i \left(\langle \mathbf{w}, \mathbf{x}^i \rangle + b \right) \geq 1 \quad \text{for all} \quad i = 1, 2, \ldots, l$$

The problem of optimization under constraints may be transformed using the Lagrangian $L(\mathbf{w},b)$

$$L(\mathbf{w}, b, \alpha) = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{l} \alpha_i \left\{ \left(\langle \mathbf{w}, \mathbf{x}^i \rangle + b \right) y^i - 1 \right\}$$

where α_i are the Lagrange multipliers, one for each data point. The first term is the same as the original objective function, and the second term captures the inequality constraints. The Lagrangian has to be minimized with respect to **w** and b:

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=0}^{l} \alpha_i y^i = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w}_0 = \sum_{i=0}^{l} y^i \alpha_i \mathbf{x}^i = 0$$

Substituting results of partial derivatives into L lead to the dual formulation of the optimization problem that has to be maximized with respect to the constraints $\alpha_i \geq 0$.

$$D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y^i y^j \left(\mathbf{x}^i \cdot \mathbf{x}^j \right)$$

The dual Lagrangian D(α) involves only the Lagrangian multipliers α_i and the training data (there is no more parameters w and b). Finding the solution for real-world problems will usually require application of quadratic programming (QP) optimization techniques. This problem has a global optimum. The optimization approach for SVM provides an accurate implementation of the SRM inductive principle. When α_i parameters are determined, it is necessary to determine the values for w and b and they determine final classification hyperplane. It is important to note that dual function D is a function of only scalar products of sample vectors, not of vectors alone. Once the solution has been found in the form of a vector $\alpha^{\tilde{0}}$ the optimal separating hyperplane is given by

$$\mathbf{w}_0 = \sum_{i \in SVs} y^i \alpha_i^0 \mathbf{x}^i$$

$$b_0 = -\frac{1}{2} \mathbf{w}_0 \cdot \left[\mathbf{x}^r + \mathbf{x}^s \right]$$

where x^r and x^s are any support vectors (SVs) from each class. The classifier can then be constructed as:

$$f(x) = sign(<\mathbf{w}_0, \mathbf{x}> + b_0) = sign\left(\sum_{i \in SVs} y^i \alpha_i^0 \left(\mathbf{x}^i \cdot \mathbf{x} \right) + b_0 \right)$$

Only the points x^i, which will have nonzero Lagrangian multipliers α^0, are termed SVs. If the data are linearly separable, all the SVs will lie on the margin and hence the number of SVs can be very small as it is represented in Figure 4.21. This "sparse" representation can be viewed as data compression in the construction of the classifier. The SVs are the "hard" cases; these are the training samples that are most difficult to classify correctly and that lie closest to the decision boundary.

The SVM learning algorithm is defined so that, in a typical case, the number of SVs is small compared with the total number of training examples. This property allows the SVM to classify new examples efficiently, since the majority of the training examples can be safely ignored. SVMs effectively remove the uninformative patterns from

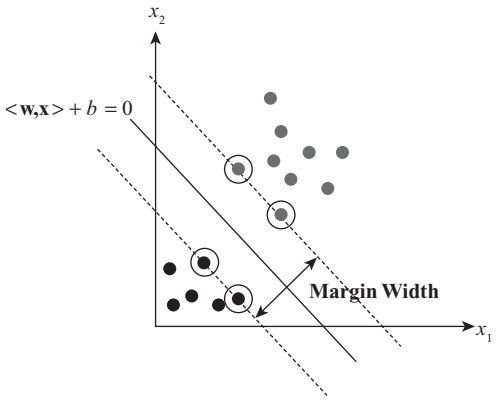

Figure 4.21. A maximal margin hyperplane with its support vectors encircled.

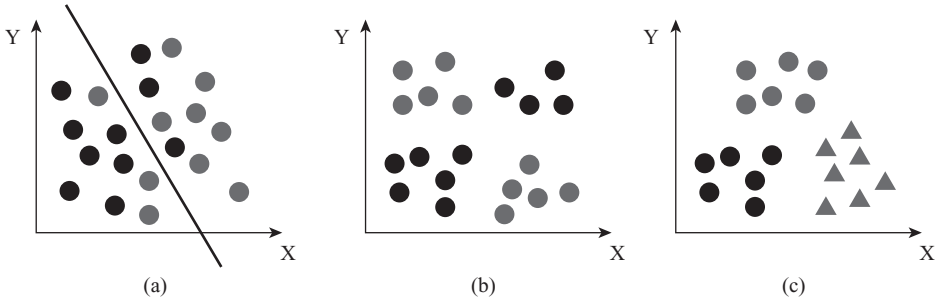

Figure 4.22. Issues for an SVM in real-world applications. (a) Subset cannot be completely separated; (b) nonlinear separation; (c) three categories.

the data set by assigning them α_i weights of 0. So, if internal points that are not SVs are changed, no effect will be made on the decision boundary. The hyperplane is represented sparsely as a linear combination of "SV" points. The SVM automatically identifies a subset of these informative points and uses them to represent the solution.

In real-world applications, SVMs must deal with (a) handling the cases where subsets cannot be completely separated, (b) separating the points with nonlinear surfaces, and (c) handling classifications with more than two categories. Illustrative examples are given in Figure 4.22. What are solutions in these cases? We will start with the problem of data that are not linearly separable. The points such as shown on the Figure 4.23a could be separated only by a nonlinear region. Is it possible to define a linear margin where some points may be on opposite sides of hyperplanes?

Obviously, there is no hyperplane that separates all of the samples in one class from all of the samples in the other class. In this case there would be no combination of w and b that could ever satisfy the set of constraints. This situation is depicted in Figure 4.23b, where it becomes apparent that we need to *soften* the constraint that these

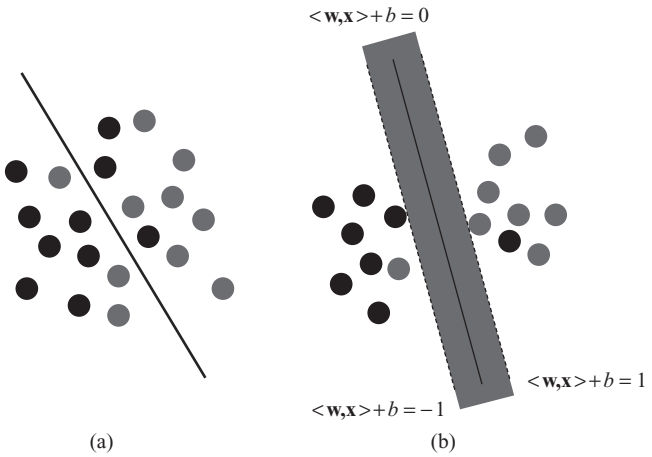

Figure 4.23. Soft margin SVM. (a) Soft separating hyperplane; (b) error points with their distances.

data lay on the correct side of the +1 and −1 hyperplanes. We need to allow some, but not too many data points to violate these constraints by a preferably small amount. This alternative approach turns out to be very useful not only for datasets that are not linearly separable, but also, and perhaps more importantly, in allowing improvements in generalization. We modify the optimization problem including cost of violation factor for samples that violate constraints:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i$$

under new constraints:

$$\left(\left\langle \mathbf{w}, \mathbf{x}^i \right\rangle + b\right) y^i \geq 1 - \xi_i$$

where C is a parameter representing the cost of violating the constraints, and ξ_i are distances of samples that violate constraints. To allow some flexibility in separating the categories, SVM models introduce a cost parameter, C, that controls the trade-off between allowing training errors and forcing rigid margins. It creates a *soft margin* (as the one in Figure 4.24) that permits some misclassifications. If C is too small, then insufficient stress will be placed on fitting the training data. Increasing the value of C increases the cost of misclassifying points and forces the creation of a more accurate model that may not generalize well.

This SVM model is a very similar case to the previous optimization problem for the linear separable data, except that there is an upper bound C on all α_i parameters. The value of C trades between how large of a margin we would prefer, as opposed to how many of the training set examples violate this margin (and by how much). The process of optimization is going through the same steps: Lagrangian, optimization of

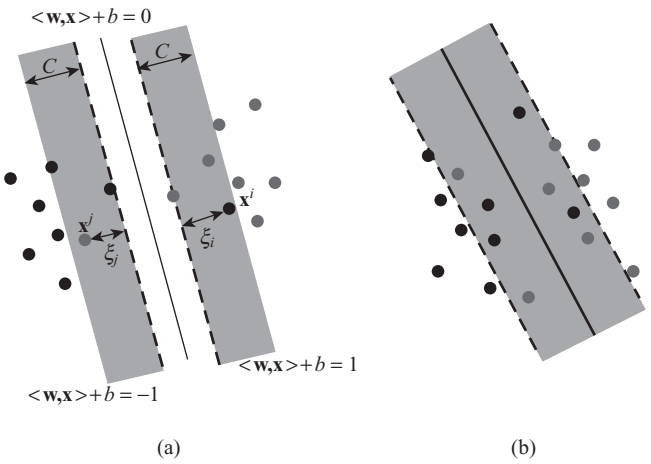

Figure 4.24. Trade-off between allowing training errors and forcing rigid margins. (a) Parameters *C* and X for a soft margin; (b) soft classifier with a fat margin ($C > 0$).

α_i parameters, and determining **w** and b values for classification hyperplane. The dual stay the same, but with additional constraints on α parameters: $0 \leq \alpha_i \leq C$.

Most classification tasks require more complex models in order to make an optimal separation, that is, correctly classify new test samples on the basis of the trained SVM. The reason is that the given data set requires nonlinear separation of classes. One solution to the inseparability problem is to map the data into a higher dimensional space and define a separating hyperplane there. This higher dimensional space is called the *feature space*, as opposed to the *input space* occupied by the training samples. With an appropriately chosen feature space of sufficient dimensionality, any consistent training set can be made linearly separable. However, translating the training set into a higher dimensional space incurs both computational and learning costs. Representing the feature vectors corresponding to the training set can be extremely expensive in terms of memory and time. Computation in the feature space can be costly because it is high-dimensional. Also, in general, there is the question of which function is appropriate for transformation. Do we have to select from infinite number of potential functions?

There is one characteristic of the SVM optimization process that helps in determining the steps in the methodology. The SVM decision function for classifying points with respect to the hyperplane only involves dot products between points. Furthermore, the algorithm that finds a separating hyperplane in the feature space can be stated entirely in terms of vectors in the input space and dot products in the feature space. We are transforming training samples from one space into the other. But we are making computation only with scalar products of points in this new space. This product is computationally inexpensive because only a small subset of points is SVs involved in product computation. Thus, an SVM can locate a separating hyperplane in the feature space and classify points in that space without ever representing the space explicitly, simply by defining a function, called a *kernel function*. Kernel function K always plays the role of the dot product in the feature space:

$$K(x,y) = \langle \Phi(x), \Phi(y) \rangle$$

This approach avoids the computational burden of explicitly representing all transformed source data and high-dimensional feature vectors. The two most widely used kernel functions are Polynomial Kernel

$$K(x,y) = (\langle x,y \rangle + 1)^d$$

and Gaussian Kernel

$$K(x,y) = \exp\left(\frac{-\|x-y\|^2}{\sigma^2}\right)$$

The polynomial kernel is valid for all positive integers d ≥ 1. The Gaussian kernel is one of a group of kernel functions known as radial basis functions (RBFs). RBFs are kernel functions that depend only on the geometric distance between x and y, and the kernel is valid for all nonzero values of the kernel width σ. It is probably the most useful and commonly applied kernel function. The concept of a kernel mapping function is very powerful, such as in the example given in Figure 4.25. It allows SVM models to perform separations even with very complex boundaries. The relation between a kernel function and a feature space can analyze for a simplified version of quadratic kernel k(x, y) = < x, y >^{2} where x, y ∈ R^2:

$$\begin{aligned}
\langle \mathbf{x}, \mathbf{y} \rangle^2 &= (x_1 y_1 + x_2 y_2)^2 \\
&= (x_1 y_1 + x_2 y_2)(x_1 y_1 + x_2 y_2) \\
&= (x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 \boldsymbol{x}_2 y_1 y_2) \\
&= (x_1^2, x_2^2, \sqrt{2} x_1 x_2)(y_1^2, y_2^2, \sqrt{2} y_1 y_2) \\
&= \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle
\end{aligned}$$

defining a 3-D feature space $\Phi(\mathrm{x}) = (\mathrm{x}_1^2, \mathrm{x}_2^2, \sqrt{2}\ \mathrm{x}_1\mathrm{x}_2)$. Similar analysis may be performed for other kernel function. For example, through the similar process verify that for the "full" quadratic kernel (<x, y > +1)^2 the feature space is 6-D.

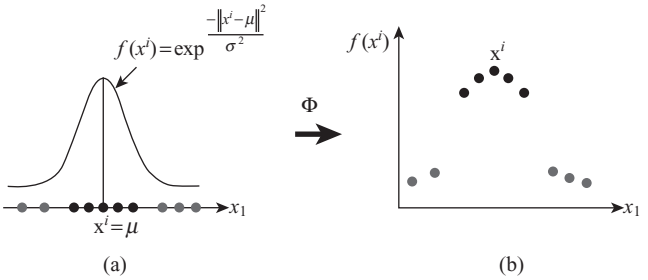

Figure 4.25. An example of a mapping Φ to a feature space in which the data become linearly separable. (a) One-dimensional input space; (b) two-dimensional feature space.

In practical use of SVM, only the kernel function k (and not transformation function Φ) is specified. The selection of an appropriate kernel function is important, since the kernel function defines the feature space in which the training set examples will be classified. As long as the kernel function is legitimate, an SVM will operate correctly even if the designer does not know exactly what features of the training data are being used in the kernel-induced feature space. The definition of a legitimate kernel function is given by Mercer's theorem: The function must be continuous and positive-definite.

Modified and enhanced SVM constructs an optimal separating hyperplane in the higher dimensional space. In this case, the optimization problem becomes

$$D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y^i y^j K\left(\mathbf{x}^i \cdot \mathbf{x}^j\right)$$

where $K(x,y)$ is the kernel function performing the nonlinear mapping into the feature space, and the constraints are unchanged. Using kernel function we will perform minimization of dual Lagrangian in the feature space, and determine all margin parameter, without representing points in this new space. Consequently, everything that has been derived concerning the linear case is also applicable for a nonlinear case by using a suitable kernel K instead of the dot product.

The approach with kernel functions gives a modular SVM methodology. One module is always the same: Linear Learning Module. It will find margin for linear separation of samples. If the problem of classification is more complex, requiring nonlinear separation, then we include a new preparatory module. This module is based on kernel function, and it transforms input space into higher, feature space where the same Linear Learning Module may be applied, and the final solution is nonlinear classification model. Illustrative example is given in Figure 4.26. This combination of different kernel functions with standard SVM learning algorithm for linear separation gives the flexibility to the SVM methodology for efficient application in nonlinear cases.

The idea of using a hyperplane to separate the feature vectors into two groups works well when there are only two target categories, but how does SVM handle the case where the target variable has more than two categories? Several approaches have been suggested, but two are the most popular: (a) "one against many" where each category is split out and all of the other categories are merged; and (b) "one against one" where $k\ (k-1)/2$ models are constructed and k is the number of categories.

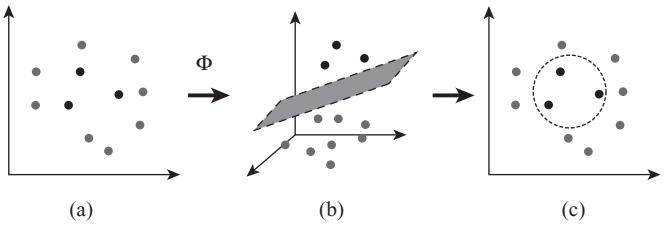

Figure 4.26. SVM performs nonlinear classification by kernel-based transformations. (a) 2-D input space; (b) 3-D feature space; (c) 2-D input space.

A preparation process for SVM applications is enormously important for the final results, and it includes preprocessing of raw data and setting model parameters. SVM requires that each data sample is represented as a vector of real numbers. If there are categorical attributes, we first have to convert them into numeric data. Multi-attribute coding is recommended in this case. For example, a three-category attribute such as red, green, and blue can be represented with three separate attributes and corresponding codes such as (0,0,1), (0,1,0), and (1,0,0). This approach is appropriate only if the number of values in an attribute is not too large. Second, scaling values of all numerical attributes before applying SVM is very important in successful application of the technology. The main advantage is to avoid attributes with greater numeric ranges to dominate those in smaller ranges. Normalization for each attribute may be applied to the range [−1; +1] or [0; 1].

Selection of parameters for SVM is very important and the quality of results depends on these parameters. Two most important parameters are cost C and parameter γ for Gaussian kernel. It is not known beforehand which C and σ are the best for one problem; consequently, some kind of parameter search must be done. The goal is to identify good (C; σ) so that the classifier can accurately predict unknown data (i.e., testing data). Note that it may not be required to achieve high-training accuracy. Small cost C is appropriate for close to linear separable samples. If we select small C for nonlinear classification problem, it will cause under-fitted learning. Large C for nonlinear problems is appropriate, but not too much because the classification margin will become very thin resulting in over-fitted learning. Similar analysis is for Gaussian kernel σ parameter. Small σ will cause close to linear kernel with no significant transformation in future space and less flexible solutions. Large σ generates extremely complex nonlinear classification solution.

The experience in many real-world SVM applications suggests that, in general, RBF model is a reasonable first choice. The RBF kernel nonlinearly maps samples into a higher dimensional space, so, unlike the linear kernel, it can handle the case when the relation between classes is highly nonlinear. The linear kernel should be treated a special case of RBF. The second reason for RBF selection is the number of hyperparameters that influences the complexity of model selection. For example, polynomial kernels have more parameters than the RBF kernel and the tuning proves is much more complex and time-consuming. However, there are some situations where the RBF kernel is not suitable, and one may just use the linear kernel with extremely good results. The question is when to use the linear kernel as a first choice. If the number of features is large, one may not need to map data to a higher dimensional space. Experiments showed that the nonlinear mapping does not improve the SVM performance. Using the linear kernel is good enough, and C is the only tuning parameter. Many microarray data in bioinformatics and collection of electronic documents for classification are examples of this data set type. As the number of features is smaller, and the number of samples increases, SVM successfully maps data to higher dimensional spaces using nonlinear kernels.

One of the methods for finding optimal parameter values for an SVM is a grid search. The algorithm tries values of each parameter across the specified search range using geometric steps. Grid searches are computationally expensive because the model

must be evaluated at many points within the grid for each parameter. For example, if a grid search is used with 10 search intervals and an RBF kernel function is used with two parameters (C and σ), then the model must be evaluated at 10*10 = 100 grid points, that is, 100 iterations in a parameter-selection process.

At the end we should highlight main strengths of the SVM methodology. First, a training process is relatively easy with a small number of parameters and the final model is never presented with local optima, unlike some other techniques. Also, SVM methodology scales relatively well to high-dimensional data and it represents a trade-off between a classifier's complexity and accuracy. Nontraditional data structures like strings and trees can be used as input samples to SVM, and the technique is not only applicable to classification problems, it is also accommodated for prediction. Weaknesses of SVMs include computational inefficiency and the need to experimentally choose a "good" kernel function.

The SVM methodology is becoming increasingly popular in the data-mining community. Software tools that include SVM are becoming more professional, user-friendly, and applicable to many real-world problems where data sets are extremely large. It has been shown that SVM outperforms other techniques such as logistic regression or artificial neural networks on a wide variety of real-world problems. Some of the most successful applications of the SVM have been in image processing, in particular handwritten digit recognition and face recognition. Other interesting application areas for SVMs are in text mining and categorization of large collection of documents, and in the analysis of genome sequences in bioinformatics. Furthermore, the SVM has been successfully used in a study of text and data for marketing applications. As kernel methods and maximum margin methods including SVM are further improved and taken up by the data-mining community, they are becoming an essential tool in any data miner's tool kit.

4.6 KNN: NEAREST NEIGHBOR CLASSIFIER

Unlike SVM's global classification model, k nearest neighbor or kNN classifier determines the decision boundary locally. For 1NN we assign each new sample to the class of its closest neighbor as shown in Figure 4.27a. Initially we have samples belonging to two classes (+ and −). The new sample "?" should be labeled with the class of its

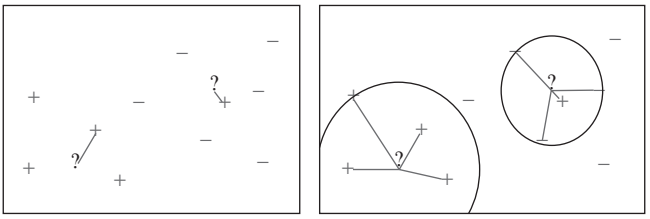

Figure 4.27. k nearest neighbor classifier. (a) k = 1; (b) k = 4.

closest neighbor. 1NN classifier is not a very robust methodology. The classification decision of each test sample relies on the class of a single training sample, which may be incorrectly labeled or atypical. For larger k, kNN will assign new sample to the majority class of its k closest neighbors where k is a parameter of the methodology. An example for k = 4 is given in Figure 4.27b. kNN classifier for $k > 1$ is more robust. Larger k values help reduce the effects of noisy points within the training data set.

The rationale of kNN classification is that we expect a test sample X to have the same label as the training sample located in the local region surrounding X. kNN classifiers are lazy learners, that is, models are not built explicitly unlike SVM and the other classification models given in the following chapters. Training a kNN classifier simply consists of determining k. In fact, if we preselect a value for k and do not preprocess given samples, then kNN approach requires no training at all. kNN simply memorizes all samples in the training set and then compares the test sample with them. For this reason, kNN is also called *memory-based learning* or *instance-based learning*. It is usually desirable to have as much training data as possible in machine learning. But in kNN, large training sets come with a severe efficiency penalty in classification of testing samples.

Building the kNN model is cheap (just store the training data), but classifying unknown sample is relatively expensive since it requires the computation of the kNN of the testing sample to be labeled. This, in general, requires computing the distance of the unlabeled object to all the objects in the labeled set, which can be expensive particularly for large training sets. Among the various methods of supervised learning, the nearest neighbor classifier achieves consistently high performance, without a priori assumptions about the distributions from which the training examples are drawn. The reader may have noticed the similarity between the problem of finding nearest neighbors for a test sample and ad hoc retrieval methodologies. In standard information retrieval systems such as digital libraries or web search, we search for the documents (samples) with the highest similarity to the query document represented by a set of key words. Problems are similar, and often the proposed solutions are applicable in both disciplines.

Decision boundaries in 1NN are concatenated segments of the *Voronoi diagram* as shown in Figure 4.28. The Voronoi diagram decomposes space into Voronoi cells, where each cell consists of all points that are closer to the sample than to other samples. Assume that we have X training samples in a 2-D space. The diagram then partitions

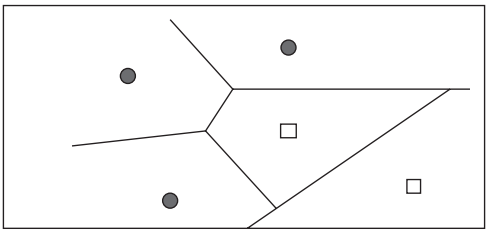

Figure 4.28. Voronoi diagram in a 2-D space.

the 2-D plane into |X| convex polygons, each containing its corresponding sample (and no other), where a convex polygon is a convex region in a 2-D space bounded by lines. For general $k > 1$ case, consider the region in the space for which the set of kNN is the same. This again is a convex polygon and the space is partitioned into convex polygons, within each of which the set of kNN is invariant.

The parameter k in kNN is often chosen based on experience or knowledge about the classification problem at hand. It is desirable for k to be odd to make ties less likely. $k = 3$ and $k = 5$ are common choices, but much larger values up to 100 are also used. An alternative way of setting the parameter is to select k through the iterations of testing process, and select k that gives best results on testing set.

Time complexity of the algorithm is linear in the size of the training set as we need to compute the distance of each training sample from the new test sample. Of course, the computing time goes up as k goes up, but the advantage is that higher values of k provide smoothing of the classification surface that reduces vulnerability to noise in the training data. At the same time high value for k may destroy the locality of the estimation since farther samples are taken into account, and large k increases the computational burden. In practical applications, typically, k is in units or tens rather than in hundreds or thousands. The nearest neighbor classifier is quite simple algorithm, but very computationally intensive especially in the testing phase. The choice of the distance measure is another important consideration. It is well known that the Euclidean distance measure becomes less discriminating as the number of attributes increases, and in some cases it may be better to use cosine or other measures rather than Euclidean distance.

Testing time of the algorithm is independent of the number of classes, and kNN therefore has a potential advantage for classification problems with multiple classes. For the example in Figure 4.29 we have three classes (ω_1, ω_2, ω_3) represented by a set of training samples, and the goal is to find a class label for the testing sample x_u. In this case, we use the Euclidean distance and a value of k = 5 neighbors as the threshold.

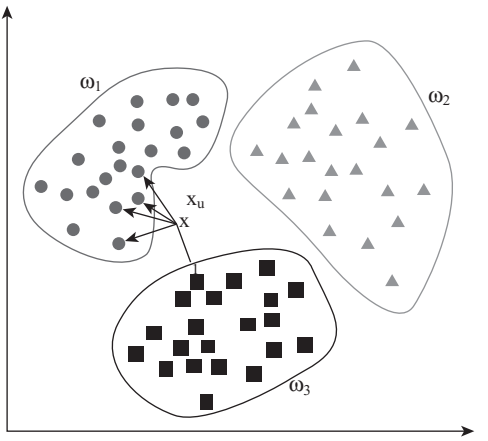

Figure 4.29. Nearest neighbor classifier for k = 5.

Of the five closest neighbors, four belong to ω_1 class and 1 belongs to ω_3 class, so x_u is assigned to ω_1 as the predominant class in the neighborhood.

In summary, kNN classifier only requires a parameter *k*, a set of labeled training samples, and a metric measure for determining distances in an n-dimensional space. kNN classification process is usually based on the following steps:

- Determine parameter k—number of nearest neighbors.
- Calculate the distance between each testing sample and all the training samples.
- Sort the distance and determine nearest neighbors based on the k-th threshold.
- Determine the category (class) for each of the nearest neighbors.
- Use simple majority of the category of nearest neighbors as the prediction value of the testing sample classification.

There are many techniques available for improving the performance and speed of a nearest neighbor classification. One solution is to choose a subset of the training data for classification. The idea of the *Condensed Nearest Neighbor (CNN)* is to select the smallest subset Z of training data X such that when Z is used instead of X, error in classification of new testing samples does not increase. 1NN is used as the nonparametric estimator for classification. It approximates the classification function in a piecewise linear manner. Only the samples that define the classifier need to be kept. Other samples, inside regions, need not be stored because they belong to the same class. An example of CNN classifier in a 2-D space is given in Figure 4.30. Greedy CNN algorithm is defined with the following steps:

1. Start wit empty set Z.
2. Pass samples from X one by one in a random order, and check whether they can be classified correctly by instances in Z.

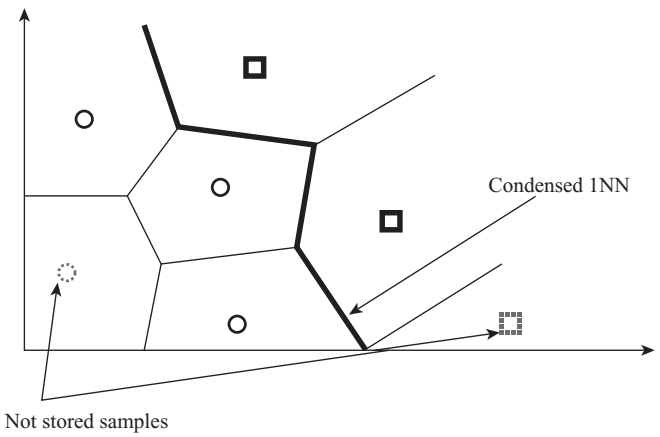

Figure 4.30. CNN classifier in a 2-D space.

3. If a sample is misclassified it is added to Z; if it is correctly classified Z is unchanged.
4. Repeat a few times over a training data set until Z is unchanged. Algorithm does not guarantee a minimum subset for Z.

kNN methodology is relatively simple and could be applicable in many real-world problems. Still, there are some methodological problems such as scalability, "course of dimensionality," influence of irrelevant attributes, weight factors in the distance measure, and weight factors for votes of k neighbors.

4.7 MODEL SELECTION VERSUS GENERALIZATION

We assume that the empirical data are given according to an unknown probability distribution. The question arises as to whether a finite set of empirical data includes sufficient information such that the underlying regularities can be learned and represented with a corresponding model. A positive answer to this question is a necessary condition for the success of any learning algorithm. A negative answer would yield the consequence that a learning system may remember the empirical data perfectly, and the system may have an unpredictable behavior with unseen data.

We may start discussion about appropriate model selection with an easy problem of learning a Boolean function from examples. Assume that both inputs and outputs are binary. For d inputs, there are 2^d different samples, and there are 2^{2d} possible Boolean functions as outputs. Each function is a potential hypothesis h_i. In the case of two inputs x_1 and x_2, there are $2^4 = 16$ different hypotheses as shown in Table 4.1.

Each training (learning) sample with two inputs and one output value (x_1, x_2, o) removes half the hypotheses (h_i). For example, sample (0, 0, 0) removes h_9 to h_{16} because these hypotheses have output value 1 for the input pair (0, 0). This is one way of interpreting learning: We start with all possible hypotheses, and as we see more training samples we remove non-consistent hypotheses. After seeing N samples, there remain 2^{2d-N} possible hypotheses, that is, Boolean functions as a model for a given data set. In reality, where all inputs are usually not binary but with k different values (k>>), and also data are high-dimensional (d>>), then $k^d >> N$. The number of samples for real-world data is significantly lower than the number of hypotheses (or the number of potential models). Therefore, data set by itself is not sufficient to find a unique solution—

TABLE 4.1. Boolean Functions as Hypotheses

Inputs		Hypotheses			
x_1	x_2	h_1	h_2	. . .	h_{16}
0	0	0	0		1
0	1	0	0		1
1	0	0	0		1
1	1	0	1		1

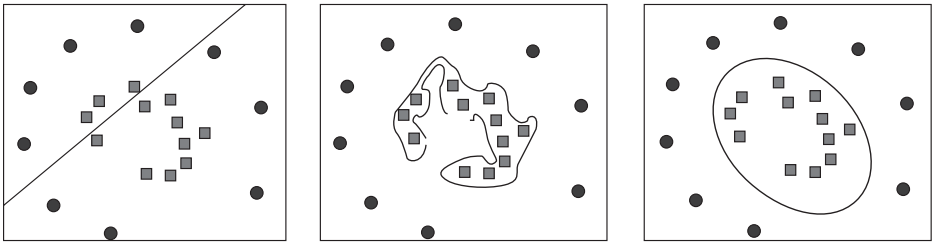

Figure 4.31. Trade-off between model complexity and the amount of data. (a) Too simple model; (b) too complex model; (c) appropriate model.

model. There is still huge number of hypotheses. We have to make some extra assumptions to reach a unique solution with the given data (N samples). These assumptions we call inductive bias (principle) of the learning algorithm. It influences a model selection. The main question is: How well does a model trained on training data set predict the right output for new samples (not available in training data). This represents an essential requirement for model generalization. For best generalization, we should match the complexity of the hypothesis class with the complexity of the function underlying training data. We made in the learning process a trade-off between the complexity of the hypothesis, the amount of data, and the generalization error of new samples (Fig. 4.31). Therefore, building a data-mining model is not a straightforward procedure, but a very sensitive process requiring, in many cases, feedback information for multiple mining iterations.

In the final phase of the data-mining process, when the model is obtained using one or more inductive-learning techniques, one important question still exists. How does one verify and validate the model? At the outset, let us differentiate between *validation* and *verification.*

Model *validation* is substantiating that the model, within its domain of applicability, behaves with satisfactory accuracy consistent with the objectives defined by the users. In other words, in model validation, we substantiate that the data have transformed into the model and that they have sufficient accuracy in representing the observed system. Model validation deals with building the *right* model, the model that corresponds to the system. Model *verification* is substantiating that the model is transformed from the data as intended into new representations with sufficient accuracy. Model verification deals with building the model *right,* the model that corresponds correctly to the data.

Model validity is a necessary but insufficient condition for the credibility and acceptability of data-mining results. If, for example, the initial objectives are incorrectly identified or the data set is improperly specified, the data-mining results expressed through the model will not be useful; however, we may still find the model valid. We can claim that we conducted an "excellent" data-mining process, but the decision makers will not accept our results and we cannot do anything about it. Therefore, we

always have to keep in mind, as it has been said, that a problem correctly formulated is a problem half-solved. Albert Einstein once indicated that the correct formulation and preparation of a problem was even more crucial than its solution. The ultimate goal of a data-mining process should not be just to produce a model for a problem at hand, but to provide one that is sufficiently credible and accepted and implemented by the decision makers.

The data-mining results are validated and verified by the testing process. Model testing is demonstrating that inaccuracies exist or revealing the existence of errors in the model. We subject the model to test data or test cases to see if it functions properly. "Test failed" implies the failure of the model, not of the test. Some tests are devised to evaluate the behavioral accuracy of the model (i.e., validity), and some tests are intended to judge the accuracy of data transformation into the model (i.e., verification).

The objective of a model obtained through the data-mining process is to classify/predict new instances correctly. The commonly used measure of a model's quality is predictive accuracy. Since new instances are not supposed to be seen by the model in its learning phase, we need to estimate its predictive accuracy using the true error rate. The true error rate is statistically defined as the error rate of the model on an asymptotically large number of new cases that converge to the actual population distribution. In practice, the true error rate of a data-mining model must be estimated from all the available samples, which are usually split into training and testing sets. The model is first designed using training samples, and then it is evaluated based on its performance on the test samples. In order for this error estimate to be reliable in predicting future model performance, not only should the training and the testing sets be sufficiently large, they must also be independent. This requirement of independent training and test samples is still often overlooked in practice.

How should the available samples be split to form training and test sets? If the training set is small, then the resulting model will not be very robust and will have low generalization ability. On the other hand, if the test set is small, then the confidence in the estimated error rate will be low. Various methods are used to estimate the error rate. They differ in how they utilize the available samples as training and test sets. If the number of available samples is extremely large (say, 1 million), then all these methods are likely to lead to the same estimate of the error rate. If the number of samples is smaller, then the designer of the data-mining experiments has to be very careful in splitting the data. There are no good guidelines available on how to divide the samples into subsets. No matter how the data are split, it should be clear that different random splits, even with the specified size of training and testing sets, would result in different error estimates.

Let us discuss different techniques, usually called *resampling methods*, for splitting data sets into training and test samples. The main advantage of using the resampling approach over the analytical approach for estimating and selecting models is that the former does not depend on assumptions about the statistical distribution of the data or specific properties of approximating functions. The main disadvantages of resampling techniques are their high computational effort and the variation in estimates depending on the resampling strategy.

The basic approach in model estimation is first to prepare or to learn a model using a portion of the training data set and then to use the remaining samples to estimate the prediction risk for this model. The first portion of the data is called a learning set, and the second portion is a validation set, also called a testing set. This *naïve strategy* is based on the assumption that the learning set and the validation set are chosen as representatives of the same, unknown distribution of data. This is usually true for large data sets, but the strategy has an obvious disadvantage for smaller data sets. With a smaller number of samples, the specific method of splitting the data starts to have an impact on the accuracy of the model. The various methods of resampling are used for smaller data sets, and they differ according to the strategies used to divide the initial data set. We will give a brief description of the resampling methods that are common in today's data-mining practice, and a designer of a data-mining system will have to make a selection based on the characteristics of the data and the problem.

1. *Resubstitution Method.* This is the simplest method. All the available data are used for training as well as for testing. In other words, the training and testing sets are the same. Estimation of the error rate for this "data distribution" is optimistically biased (estimated error is often smaller than could be expected in real applications of the model), and therefore the method is very seldom used in real-world data-mining applications. This is especially the case when the ratio of sample size to dimensionality is small.
2. *Holdout Method.* Half the data, or sometimes two-thirds of the data, is used for training and the remaining data are used for testing. Training and testing sets are independent and the error estimation is pessimistic. Different partitioning will give different estimates. A repetition of the process, with different training and testing sets randomly selected, and integration of the error results into one standard parameter, will improve the estimate of the model.
3. *Leave-One-Out Method.* A model is designed using $(n - 1)$ samples for training and evaluated on the one remaining sample. This is repeated n times with different training sets of size $(n - 1)$. This approach has large computational requirements because n different models have to be designed and compared.
4. *Rotation Method (n-Fold Cross-Validation).* This approach is a compromise between holdout and leave-one-out methods. It divides the available samples into P disjoint subsets, where $1 \leq P \leq n$. $(P - 1)$ subsets are used for training and the remaining subset for testing. This is the most popular method in practice, especially for problems where the number of samples is relatively small.
5. *Bootstrap Method.* This method resamples the available data with replacements to generate a number of "fake" data sets of the same size as the given data set. The number of these new sets is typically several hundreds. These new training sets can be used to define the so-called bootstrap estimates of the error rate. Experimental results have shown that the bootstrap estimates can outperform the cross-validation estimates. This method is especially useful in small data set situations.

4.8 MODEL ESTIMATION

A model realized through the data-mining process using different inductive-learning techniques might be estimated using the standard error rate parameter as a measure of its performance. This value expresses an approximation of the true error rate, a parameter defined in SLT. The error rate is computed using a testing data set obtained through one of applied resampling techniques. In addition to the accuracy measured by the error rate, data-mining models can be compared with respect to their speed, robustness, scalability, and interpretability; all these parameters may have an influence on the final verification and validation of the model. In the short overview that follows, we will illustrate the characteristics of the error-rate parameter for classification tasks; similar approaches and analyses are possible for other common data-mining tasks.

The computation of error rate is based on counting of errors in a testing process. These errors are, for a classification problem, simply defined as misclassification (wrongly classified samples). If all errors are of equal importance, an error rate R is the number of errors E divided by the number of samples S in the testing set:

$$R = E/S$$

The accuracy AC of a model is a part of the testing data set that is classified correctly, and it is computed as one minus the error rate:

$$AC = 1 - R = (S - E)/S$$

For standard classification problems, there can be as many as $m^2 - m$ types of errors, where m is the number of classes.

Two tools commonly used to assess the performance of different classification models are the *confusion matrix* and the *lift chart.* A confusion matrix, sometimes called a classification matrix, is used to assess the prediction accuracy of a model. It measures whether a model is confused or not, that is, whether the model is making mistakes in its predictions. The format of a confusion matrix for a two-class case with classes yes and no is shown in Table 4.2.

If there are only two classes (positive and negative samples, symbolically represented with T and F or with 1 and 0), we can have only two types of errors:

1. It is expected to be T, but it is classified as F: These are false negative errors (C: False-), and

TABLE 4.2. Confusion Matrix for Two-Class Classification Model

Actual Class / Predicted Class	Class 1	Class 2
Class 1	**A**: True +	**B**: False +
Class 2	**C**: False –	**D**: True –

TABLE 4.3. Confusion Matrix for Three Classes

	True Class			
Classification Model	0	1	2	Total
0	28	**1**	**4**	33
1	**2**	28	**2**	32
2	**0**	**1**	24	25
Total	30	30	30	90

2. It is expected to be F, but it is classified as T: These are false positive errors (B: False+).

If there are more than two classes, the types of errors can be summarized in a confusion matrix, as shown in Table 4.3. For the number of classes m = 3, there are six types of errors ($m^2 - m = 3^2 - 3 = 6$), and they are represented in bold type in Table 4.3. Every class contains 30 samples in this example, and the total is 90 testing samples.

The error rate for this example is

$$R = E/S = 10/90 = 0.11$$

and the corresponding accuracy is

$$AC = 1 - R = 1 - 0.11 = 0.89 \text{ (or as a percentage: A} = 89\%).$$

Accuracy is not always the best measure of the quality of the classification model. It is especially true for the real-world problems where the distribution of classes is unbalanced. For example, if the problem is classification of healthy persons from those with the disease. In many cases the medical database for training and testing will contain mostly healthy persons (99%), and only small percentage of people with disease (about 1%). In that case, no matter how good the accuracy of a model is estimated to be, there is no guarantee that it reflects the real world. Therefore, we need other measures for model quality. In practice, several measures are developed, and some of the best known are presented in Table 4.4. Computation of these measures is based on parameters A, B, C, and D for the confusion matrix in Table 4.2. Selection of the appropriate measure depends on the application domain, and for example in medical field the most often used are measures: sensitivity and specificity.

So far we have considered that every error is equally bad. In many data-mining applications, the assumption that all errors have the same weight is unacceptable. So, the differences between various errors should be recorded, and the final measure of the error rate will take into account these differences. When different types of errors are associated with different weights, we need to multiply every error type with the given weight factor c_{ij}. If the error elements in the confusion matrix are e_{ij}, then the total cost function C (which replaces the number of errors in the accuracy computation) can be calculated as

TABLE 4.4. Evaluation Metrics for Confusion Matrix 2 × 2

Evaluation Metrics	Computation Using Confusion Matrix
True positive rate (TP)	TP = A/(A + C)
False positive rate (FP)	FP = B/(B + D)
Sensitivity (SE)	SE = TP
Sensitivity (SP)	SP = 1 − FP
Accuracy (AC)	AC = (A + D)/(A + B + C + D)
Recall (R)	R = A/(A + B)
Precision (P)	P = A/(A + C)
F measure (F)	F = 2PR/(P + R)

$$C = \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij}\, e_{ij}$$

In many data-mining applications, it is not adequate to characterize the performance of a model by a single number that measures the overall error rate. More complex and global measures are necessary to describe the quality of the model. A lift chart, sometimes called a cumulative gains chart, is an additional measure of classification model performance. It shows how classification results are changed by applying the model to different segments of a testing data set. This change ratio, which is hopefully the increase in response rate, is called the "lift." A lift chart indicates which subset of the dataset contains the greatest possible proportion of positive responses or accurate classification. The higher the lift curve is from the baseline, the better the performance of the model since the baseline represents the null model, which is no model at all. To explain a lift chart, suppose a two-class prediction where the outcomes were yes (a positive response) or no (a negative response). To create a lift chart, instances in the testing dataset are sorted in descending probability order according to the predicted probability of a positive response. When the data are plotted, we can see a graphical depiction of the various probabilities as it is represented with the black histogram in Figure 4.32a. The baseline, represented as the white histogram on the same figure, indicates the expected result if no model was used at all. Note that the best model is not the one with the highest lift when it is being built with the training data. It is the model that performs the best on unseen, future data.

The lift chart is also a big help in evaluating the usefulness of a model. It shows how responses are changed in percentiles of testing samples population, by applying the data mining model. For example, in Figure 4.32a, instead of a 10% response rate when a random 10% of the population is treated, the response rate of the top selected 10% of the population is over 35%. The lift is 3.5 in this case.

Another important component of interpretation is to assess the financial benefits of the model. Again, a discovered model may be interesting and relatively accurate, but acting on it may cost more than the revenue or savings it generates. The Return on

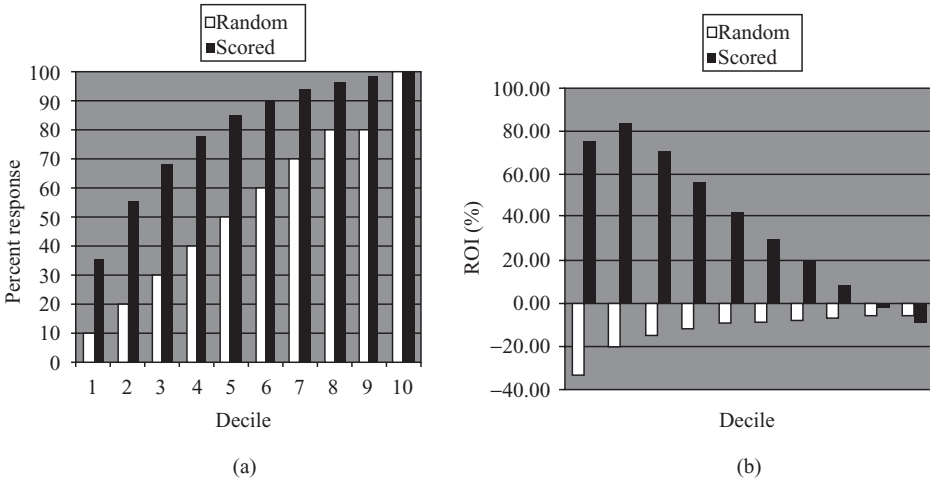

Figure 4.32. Assessing the performances of data-mining model. (a) Lift chart; (b) ROI chart.

Investment (ROI) chart, given in Figure 4.32b, is a good example of how attaching values to a response and costs to a program can provide additional guidance to decision making. Here, ROI is defined as ratio of profit to cost. Note that beyond the eighth decile (80%), or 80% of testing population, the ROI of the scored model becomes negative. It is at a maximum for this example at the second decile (20% of population).

We can explain the interpretation and practical use of lift and ROI charts on a simple example of a company who wants to advertise their products. Suppose they have a large database of addresses for sending advertising materials. The question is: Will they send these materials to everyone in the database? What are the alternatives? How do they obtain the maximum profit from this advertising campaign? If the company has additional data about "potential" costumers in their database, they may build the predictive (classification) model about the behavior of customers and their responses to the advertisement. In estimation of the classification model, lift chart is telling the company what the potential improvements in advertising results are. What are benefits if they use the model and based on the model select only the most promising (responsive) subset of database instead of sending ads to everyone? If the results of the campaign are presented in Figure 4.32a the interpretation may be the following. If the company is sending the advertising materials to the top 10% of customers selected by the model, the expected response will be 3.5 times greater than sending the same ads to randomly selected 10% of customers. On the other hand, sending ads involves some cost and receiving response and buying the product results in additional profit for the company. If the expenses and profits are included in the model, ROI chart shows the level of profit obtained with the predictive model. From Figure 4.32b it is obvious that the profit will be negative if ads are sent to all customers in the database. If it is sent only to 10%

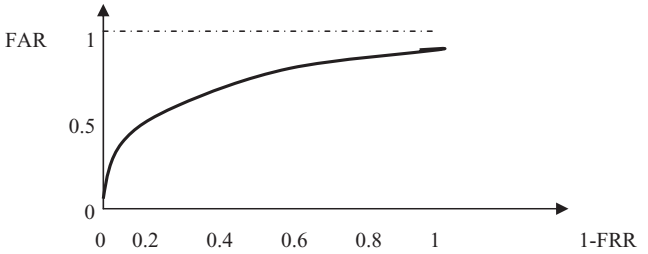

Figure 4.33. The ROC curve shows the trade-off between sensitivity and 1-specificity values.

of the top customers selected by the model, ROI will be about 70%. This simple example may be translated to large number of different data-mining applications.

While lift and ROI charts are popular in the business community for evaluation of data-mining models, the scientific community "likes" the Receiver Operating Characteristic (ROC) curve better. What is the basic idea behind an ROC curve construction? Consider a classification problem where all samples have to be labeled with one of two possible classes. A typical example is a diagnostic process in medicine, where it is necessary to classify the patient as being with or without disease. For these types of problems, two different yet related error rates are of interest. The False Acceptance Rate (FAR) is the ratio of the number of test cases that are incorrectly "accepted" by a given model to the total number of cases. For example, in medical diagnostics, these are the cases in which the patient is wrongly predicted as having a disease. On the other hand, the False Reject Rate (FRR) is the ratio of the number of test cases that are incorrectly "rejected" by a given model to the total number of cases. In the previous medical example, these are the cases of test patients who are wrongly classified as healthy.

For the most of the available data-mining methodologies, a classification model can be tuned by setting an appropriate threshold value to operate at a desired value of FAR. If we try to decrease the FAR parameter of the model, however, it would increase the FRR and vice versa. To analyze both characteristics at the same time, a new parameter was developed, the ROC curve. It is a plot of FAR versus 1 − FRR for different threshold values in the model. This curve permits one to assess the performance of the model at various operating points (thresholds in a decision process using the available model) and the performance of the model as a whole (using as a parameter the area below the ROC curve). The ROC curve is especially useful for a comparison of the performances of two models obtained by using different data-mining methodologies. The typical shape of an ROC curve is given in Figure 4.33 where the axes are *sensitivity* (FAR) and *1-specificity* (1 − FRR).

How does one construct an ROC curve in practical data-mining applications? Of course, many data-mining tools have a module for automatic computation and visual representation of an ROC curve. What if this tool is not available? At the beginning we are assuming that we have a table with actual classes and predicted classes for all training samples. Usually, predicted values as an output from the model are not computed as 0 or 1 (for two class problem) but as real values on interval [0, 1]. When we

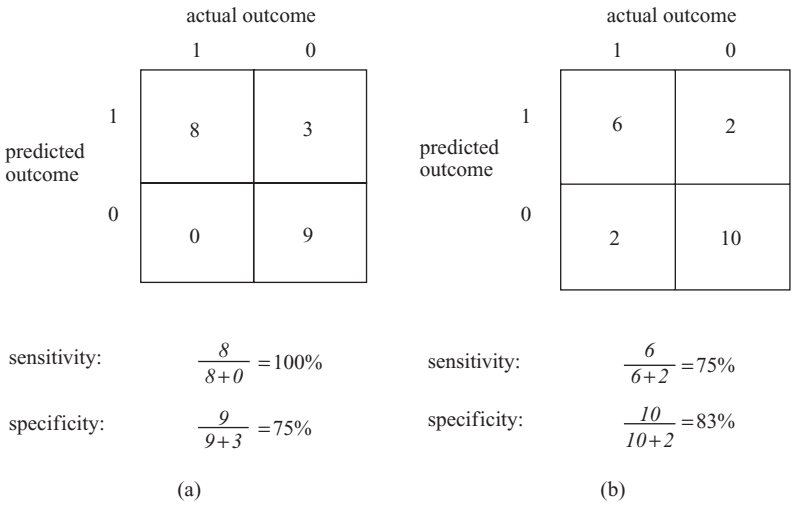

Figure 4.34. Computing points on an ROC curve. (a) Threshold = 0.5; (b) threshold = 0.8.

select a threshold value, we may assume that all predicted values above the threshold are 1, and all values below the threshold are 0. Based on this approximation we may compare actual class with predicted class by constructing a confusion matrix, and compute sensitivity and specificity of the model for the given threshold value. In general, we can compute sensitivity and specificity for large numbers of different threshold values. Every pair of values (sensitivity, 1-specificity) represents one discrete point on a final ROC curve. Examples are given in Figure 4.34. Typically, for graphical presentation, we are systematically selecting threshold values. For example, starting from 0 and increasing by 0.05 until 1, we have 21 different threshold values. That will generate enough points to reconstruct an ROC curve.

When we are comparing two classification algorithms we may compare the measures as accuracy or F measure, and conclude that one model is giving better results than the other. Also, we may compare lift charts, ROI charts, or ROC curves, and if one curve is above the other we may conclude that a corresponding model is more appropriate. But in both cases we may not conclude that there are significant differences between models, or more important, that one model shows better performances than the other with statistical significance. There are some simple tests that could verify these differences. The first one is McNemar's test. After testing models of both classifiers, we are creating a specific contingency table based on classification results on testing data for both models. Components of the contingency table are explained in Table 4.5.

After computing the components of the contingency table, we may apply the χ^2 statistic with one degree of freedom for the following expression:

$$[(|e_{01} - e_{10}| - 1)^2]/(e_{01} + e_{10}) \sim \chi^2$$

TABLE 4.5. Contingency Table for McNemar's Test

e_{00}: Number of samples misclassified by both classifiers	e_{01}: Number of samples misclassified by classifier 1, but not classifier 2
e_{10}: Number of samples misclassified by classifier 2, but not classifier 1	e_{11}: Number of samples correctly classified by both classifier s

McNemar's test rejects the hypothesis that the two algorithms have the same error at the significance level α, if previous value is greater than $\chi^2_{\alpha, 1}$. For example, for $\alpha = 0.05$, $\chi^2_{0.05, 1} = 3.84$.

The other test is applied if we compare two classification models that are tested with the K-fold cross-validation process. The test starts with the results of K-fold cross-validation obtained from K training/validation set pairs. We compare the error percentages in two classification algorithms based on errors in K validation sets that are recorded for two models as: p_i^1 and p_i^2, i = 1, . . . , K.

The difference in error rates on fold i is $P_i = p_i^1 - p_i^2$. Then, we can compute:

$$m = \left[\sum_{i=1}^{k} P_i\right] / K \quad \text{and} \quad S^2 = \left[\sum_{i=1}^{k} (P_i - m)^2\right] / K - 1$$

We have a statistic that is t distributed with K-1 degrees of freedom, and the following test:

$$(\sqrt{K} * m) / S \sim t_{K-1}$$

Thus, the K-fold cross-validation paired *t*-test rejects the hypothesis that two algorithms have the same error rate at significance level α, if previous value is outside interval $(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1})$. For example, the threshold values could be for $\alpha = 0.05$ and K = 10 or 30: $t_{0.025, 9} = 2.26$, and $t_{0.025, 29} = 2.05$.

Over time, all systems evolve. Thus, from time to time the model will have to be retested, retrained, and possibly completely rebuilt. Charts of the residual differences between forecasted and observed values are an excellent way to monitor model results.

4.9 90% ACCURACY: NOW WHAT?

Often forgotten in texts on data mining is a discussion of the deployment process. Any data-mining student may produce a model with relatively high accuracy over some small data set using the tools available. However, an experienced data miner sees beyond the creation of a model during the planning stages. There needs to be a plan created to evaluate how useful a data-mining model is to a business, and how the model will be rolled out. In a business setting the value of a data-mining model is not simply the accuracy, but how that model can impact the bottom line of a company. For example, in fraud detection, algorithm A may achieve an accuracy of 90% while algorithm B

achieves 85% on training data. However, an evaluation of the business impact of each may reveal that algorithm A would likely underperform algorithm B because of larger number of very expensive false negative cases. Additional financial evaluation may recommend algorithm B for the final deployment because with this solutions company saves more money. A careful analysis of the business impacts of data-mining decisions gives much greater insight of a data-mining model.

In this section two case studies are summarized. The first case study details the deployment of a data-mining model that improved the efficiency of employees in finding fraudulent claims at an insurance company in Chile. The second case study involves a system deployed in hospitals to aid in counting compliance with industry standards in caring for individuals with cardiovascular disease (CVD).

4.9.1 Insurance Fraud Detection

In 2005, the insurance company Banmedica S.A. of Chile received 800 digital medical claims per day. The process of identifying fraud was entirely manual. Those responsible for identifying fraud had to look one-by-one at medical claims to find fraudulent cases. Instead it was hoped that data-mining techniques would aid in a more efficient discovery of fraudulent claims.

The first step in the data-mining process required that the data-mining experts gain a better understanding of the processing of medical claims. After several meetings with medical experts, the data-mining experts were able to better understand the business process as it related to fraud detection. They were able to determine the current criteria used in manually discriminating between claims that were approved, rejected, and modified. A number of known fraud cases were discussed and the behavioral patterns that revealed these documented fraud cases.

Next, two data sets were supplied. The first data set contained 169 documented cases of fraud. Each fraudulent case took place over an extended period of time, showing that time was an important factor in these decisions as cases developed. The second data set contained 500,000 medical claims with labels supplied by the business of “approved,” “rejected,” or “reduced.”

Both data sets were analyzed in detail. The smaller data set of known fraud cases revealed that these fraudulent cases all involved a small number of medical professionals, affiliates, and employers. From the original paper, “19 employers and 6 doctors were implicated with 152 medical claims.” The labels of the larger data set were revealed to be not sufficiently accurate for data mining. Contradictory data points were found. A lack of standards in recording these medical claims, with a large number of missing values contributed to the poorly labeled data set. Instead of the larger 500,000 point data set, the authors were “forced” to rebuild a subset of thesee data. This required manual labeling of the subset.

The manual labeling would require a much smaller set of data points to be used from the original 500,000. To cope with a smaller set of data points, the problem was split into four smaller problems, namely identifying fraudulent medical claims, affiliates, medical professionals, and employers. Individual data sets were constructed for each of these four subtasks ranging in size from 2838 samples in the medical claims

task to 394 samples in the employer subtask. For each subtask a manual selection of features was performed. This involved selecting only one feature from highly correlated features, replacing categorical features with numerical features, and design new features that "summarize temporal behavior over an extended time span." The original 125 features were paired down to between 12 and 25 features depending on the subtask. Additionally, the output of all other subtasks became inputs to each subtask, thus providing feedback to each subtask. Last, 2% of outliers were removed and features were normalized.

When modeling the data it was found that initially the accuracy of a single neural network on these data sets could vary by as much as 8.4%. Instead of a single neural network for a particular data set, a committee of neural networks was used. Each data set was also divided into a training set, a validation set, and a testing set to avoid overfitting the data. At this point it was also decided that each of the four models would be retrained monthly to keep up with the ever evolving process of fraud.

Neural networks and committees of neural networks output scores rather than an absolute fraud classification. It was necessary that a threshold be set for the output. The threshold was decided after accounting for personnel costs, false alarm costs, and the cost of not detecting a particular instance of fraud. All of these factors figured into an ROC curve to decide upon acceptable false and true positive rates. When the medical claims model using the input of the other three subtasks scored a medical claim above the chosen threshold, then a classification of fraud is given to that claim. The system was tested on a historical data set of 8819 employers that contains 418 instances of fraud. After this historical data set was split into training, validation, and test set, the results showed that the system identified 73.4% of the true fraudsters and had a false positive rate of 6.9%.

The completed system was then run each night giving each new medical claim a fraud probability. The claims are then reviewed being sorted by the given probabilities. There were previously very few documented cases of fraud. After implementation there were approximately 75 rejected claims per month. These newly found cases of fraud accounted for nearly 10% of the raw overall costs to the company. Additionally, the culture of fraud detection changed. A taxonomy of the types of fraud was created and further improvements were made on the manual revision process. The savings covered the operational costs and increased the quality of health coverage.

Overall this project was a big success. The authors spent a lot of time first understanding the problem and second analyzing the data in detail, before the data was modeled. The final models produced were analyzed in terms of real business costs. In the end the results showed that the costs of the project were justified and Banmedica S.A. greatly benefited from the final system.

4.9.2 Improving Cardiac Care

CVD leads to nearly 1 million deaths (or 38% of all deaths) in the United States per year. Additionally, in 2005 the estimated cost of CVD was $394 billion compared with an estimated $190 billion on all cancers combined. CVD is a real problem that appears to be growing in the number of lives claimed and the percent of the population that

will be directly affected by this disease. Certainly we can gain a better understanding of this disease. There already exist guidelines for the care of patients with CVD that were created by panels of experts. With the current load on the medical system, doctors are able to only spend a short amount of time with each patient. With the large number of guidelines that exists, it is not reasonable to expect that doctors will follow every guideline on every patient. Ideally a system would aid a doctor in following the given guidelines without adding additional overheads.

This case study outlines the use and deployment of a system called REMIND, which is meant both to find patients at need within the system, and to enable a better tracking of when patients are being cared for according to guidelines. Currently two main types of records are kept for each patient, financial and clinical. The financial records are used for billing. These records use standardized codes (e.g., ICD-9) for doctor assessments and drugs prescribed. This standardization makes it straightforward for computer systems to extract information from these records and used by data-mining processes. However, it has been found that these codes are accurate only 60–80% of the time for various reasons. One reason is that when these codes are used for billing, although two conditions are nearly identical in symptoms and prescriptions, the amount of money that will be paid out by an insurer may be very different. The other form of records kept is clinical records. Clinical records are made up of unstructured text, and allow for the transfer of knowledge about a patient's condition and treatments from one doctor to another. These records are much more accurate, but are not in a form that is easily used by automated computer systems.

It is not possible that with great demands on the time of doctors and nurses that additional data may be recorded specifically for this system. Instead, the REMIND system combines data from all available systems. This includes extracting knowledge from the unstructured clinical records. The REMIND system combines all available sources of data present, then using redundancy in data the most likely state of the patient is found. For example to determine a patient is diabetic one may use any of the following pieces of data: billing code 250.xx for diabetes, a free text dictation identifying a diabetic diagnosis, a blood sugar value >300, a treatment of insulin or oral antidiabetic, or a common diabetic complication. The likelihood of the patient having diabetes increases as more relevant information is found. The REMIND system uses extracted information from all possible data sources, combined in a Bayesian network. The various outputs of the network are used along with temporal information to find the most probable sequence of states with a predefined disease progression modeled as a Markov model. The probabilities and structure of the Bayesian network are provided as domain knowledge provided beforehand by experts and tunable per deployment. The domain knowledge in the REMIND system is fairly simple as stated by the author of the system. Additionally, by using a large amount of redundancy the system performs well for a variety of probability settings and temporal settings for the disease progression. However, before a wide distribution of the REMIND system a careful tuning of all the parameters must take place.

One example deployment was to the South Carolina Heart Center where the goal was to identify among 61,027 patients those at risk of Sudden Cardiac Death (SCD). Patients who have previously suffered a myocardial infarction (MI; heart attack) are at

the highest risk of SCD. In 1997 a study was performed on the efficacy of implantable cardioverter defribillators (ICDs). It was found that patients, with a prior MI and low ventricular function, had their 20-month mortality rate drop from 19.8% to 14.2%. This implantation is now a standard recommendation. Previous to the REMIND system one had two options to find who would require the implantation of an ICD. The first option is to manually review the records of all patients to identify those who were eligible for an ICD. This would be extremely time-consuming considering the large number of records. The other approach would be to evaluate the need for an ICD during regular checkups. However, not all patients come in for regular checkups and there would be a high chance that not every patient would be carefully considered for the need of an ICD. The REMIND system was given access to billing and demographics databases and transcribed free text including histories, physical reports, physician progress notes, and lab reports. From these data the REMIND system processed all records on a single laptop in 5 h and found 383 patients who qualified for an ICD.

To check the validity of the 383 found patients, 383 randomly chosen patients were mixed with the 383 found previously. Then 150 patients were chosen from the 766 patient samples. An electrophysiologist manually reviewed the 150 patients being blinded to the selection made by the REMIND system. The REMIND system concurred with the manual analysis in 94% (141/150) of the patients. The sensitivity was 99% (69/70) and the specificity was 90% (72/80). Thus it was shown that the REMIND system could fairly accurately identify at-risk patients in a large database. An expert was required to verify the results of the system. Additionally, all of the patients found would be reviewed by a physician before implantation would occur.

From the previous cases we see that a great deal of time was required from experts to prepare data for mining, and careful analysis of a model application needed to take place after deployment. Although applied data-mining techniques (neural and Bayesian networks) will be explained in the following chapters, the emphasis of these stories is on complexity of a data-mining process, and especially deployment phase, in real-world applications. The system developed for Banmedica was measured after analysis in terms of fraudulent cases found and the amount of money saved. If these numbers were not in favor of the system, then it would have been rolled back. In the case of the REMIND system, the results of the system wide search had to be manually analyzed for accuracy. It was not enough that the rules were good, but the actual patients found needed to be reviewed.

4.10 REVIEW QUESTIONS AND PROBLEMS

1. Explain the differences between the basic types of inferences: induction, deduction, and transduction.
2. Why do we use the observational approach in most data-mining tasks?
3. Discuss situations in which we would use the interpolated functions given in Figure 4.3b,c,d as “the best” data-mining model.

4. Which of the functions have linear parameters and which have nonlinear? Explain why.

(a) $y = a\ x^5 + b$
(b) $y = a/x$
(c) $y = a\ e^x$
(d) $y = e^{a\ x}$

5. Explain the difference between interpolation of loss function for classification problems and for regression problems.

6. Is it possible that empirical risk becomes higher than expected risk? Explain.

7. Why is it so difficult to estimate the VC dimension for real-world data-mining applications?

8. What will be the practical benefit of determining the VC dimension in real-world data-mining applications?

9. Classify the common learning tasks explained in Section 4.4 as supervised or unsupervised learning tasks. Explain your classification.

10. Analyze the differences between validation and verification of inductive-based models.

11. In which situations would you recommend the leave-one-out method for validation of data-mining results?

12. Develop a program for generating "fake" data sets using the bootstrap method.

13. Develop a program for plotting an ROC curve based on a table of FAR–FRR results.

14. Develop an algorithm for computing the area below the ROC curve (which is a very important parameter in the evaluation of inductive-learning results for classification problems).

15. The testing data set (inputs: A, B, and C, output: Class) is given together with testing results of the classification (predicted output). Find and plot two points on the ROC curve for the threshold values of 0.5 and 0.8.

A	B	C	Class	Predicted
			(Output)	Output
10	2	A	1	*0.97*
20	1	B	0	*0.61*
30	3	A	1	*0.77*
40	2	B	1	*0.91*
15	1	B	0	*0.12*

16. Machine-learning techniques differ from statistical techniques in that machine learning methods

(a) typically assume an underlying distribution for the data,
(b) are better able to deal with missing and noisy data,
(c) are not able to explain their behavior, and
(d) have trouble with large-sized data sets.

17. Explain the difference between sensitivity and specificity.

18. When do you need to use a separate validation set, in addition to train and test sets?

19. In this question we will consider learning problems where each instance x is some integer in the set $X = \{1, 2, \ldots, 127\}$, and where each hypothesis $h \in H$ is an interval of the form $a \leq x \leq b$, where a and b can be any integers between 1 and 127 (inclusive), so long as $a \leq b$. A hypothesis $a \leq x \leq b$ labels instance x positive if x falls into the interval defined by a and b, and labels the instance negative otherwise. Assume throughout this question that the teacher is only interested in teaching concepts that can be represented by some hypothesis in H.
(a) How many distinct hypotheses are there in H?
(b) Suppose the teacher is trying to teach the specific target concept $32 \leq x \leq 84$. What is the minimum number of training examples the teacher must present to guarantee that any consistent learner will learn this concept exactly?

20. Is it true that the SVM learning algorithm is guaranteed to find the globally optimal hypothesis with respect to its object function? Discuss your answer.

4.11 REFERENCES FOR FURTHER STUDY

Alpaydin, A, *Introduction to Machine Learning*, 2nd edition, The MIT Press, Boston, 2010.

The goal of machine learning is to program computers to use example data or past experience to solve a given problem. Many successful applications of machine learning exist already, including systems that analyze past sales data to predict customer behavior, optimize robot behavior so that a task can be completed using minimum resources, and extract knowledge from bioinformatics data. *Introduction to Machine Learning* is a comprehensive textbook on the subject, covering a broad array of topics not usually included in introductory machine-learning texts. In order to present a unified treatment of machine-learning problems and solutions, it discusses many methods from different fields, including statistics, pattern recognition, neural networks, artificial intelligence, signal processing, control, and data mining. All learning algorithms are explained so that the student can easily move from the equations in the book to a computer program.

Berthold, M., D. J. Hand, eds., *Intelligent Data Analysis—An Introduction*, Springer, Berlin, Germany, 1999.

The book is a detailed, introductory presentation of the key classes of intelligent data-analysis methods including all common data-mining techniques. The first half of the book is devoted to the discussion of classical statistical issues, ranging from basic concepts of probability and inference to advanced multivariate analyses and Bayesian methods. The second part of the book covers theoretical explanations of data-mining techniques that have their roots in disciplines other than statistics. Numerous illustrations and examples enhance the readers' knowledge about theory and practical evaluations of data-mining techniques.

Cherkassky, V., F. Mulier, *Learning from Data: Concepts, Theory and Methods*, 2nd edition, John Wiley, New York, 2007.

The book provides a unified treatment of the principles and methods for learning dependencies from data. It establishes a general conceptual framework in which various learning methods from statistics, machine learning, and other disciplines can be applied—showing that a few fundamental principles underlie most new methods being proposed today. An additional strength of this primarily theoretical book is the large number of case studies and examples that simplify and make understandable concepts in SLT.

Engel, A., C. Van den Broeck, *Statistical Mechanics of Learning*, Cambridge University Press, Cambridge, UK, 2001.

The subject of this book is the contribution of machine learning over the last decade by researchers applying the techniques of statistical mechanics. The authors provide a coherent account of various important concepts and techniques that are currently only found scattered in papers. They include many examples and exercises, making this a book that can be used with courses, or for self-teaching, or as a handy reference.

Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, 1999.

The book provides a comprehensive foundation for the study of artificial neural networks, recognizing the multidisciplinary nature of the subject. The introductory part explains the basic principles of SLT and the concept of VC dimension. The main part of the book classifies and explains artificial neural networks as learning machines with and without a teacher. The material presented in the book is supported with a large number of examples, problems, and computer-oriented experiments.