1 Learning and Soft Computing: Rationale, Motivations, Needs, Basics

Since the late 1980s there has been an explosion in research activity in neural networks (NNs), support vector machines (SVMs), and fuzzy logic (FL) systems. Together with new algorithms and statements of fundamental principles there has been an increase in real-world applications. Today, these areas are mature to the point where successful applications are reported across a range of fields. Examples of applications in diverse fields are given in section 1.1.

These three modeling tools complement one other. NNs and SVMs are paradigms of learning tools. They recover underlying dependencies between the given inputs and outputs by using training data sets. After training, both NNs and SVMs represent high-dimensional nonlinear functions. They are mathematical models obtained in an experimental way. If there are no data (examples, patterns, observations, or measurements), there will be no learning, and consequently no modeling by NNs and SVMs can take place.

However, one can still model the causal relations (also known as functions) between some variables provided one has an understanding about the system or process under investigation. This is the purpose of fuzzy logic. It is a tool for embedding existing structured human knowledge into mathematical models. If one has neither prior knowledge nor measurements, it may be difficult to believe that the problem at hand may be solved easily. This is by all accounts a very hopeless situation indeed. This book does not cover cases where both measurements and prior knowledge are lacking. However, even when faced with a modeling problem without either experimental data or knowledge, one is not entirely lost because there is an old scientific solution: if one cannot solve the problem posed, one poses another problem. In this book, problems are not reformulated. Rather, the text demonstrates how various real-world (conceptual or practical) tasks can be solved by learning from experimental data or by embedding structured human knowledge into mathematical models.

This chapter describes some typical nonlinear and high-dimensional problems from various fields in which soft models have been successfully applied. The range of problems that can be solved by using soft modeling approaches is wide, but all the problems belong to two major groups: pattern recognition (classification) tasks or functional approximation (regression) tasks. In this way, soft models can be looked at as being nonlinear extensions to classic linear regression and classification. This is how these standard statistical problems are introduced in section 1.4. Having a sound understanding of the concepts, performance, and limitations of linear statistical models is good preparation for understanding nonlinear modeling tools.

NNs and SVMs solve regression and classification problems by changing parameters that control how they learn as they cycle through training data. These parameters, usually called weights, influence how well the trained model performs. In order to measure the model's performance one must define some measure of goodness of the model. In mathematical terms, one should define some suitable norm. Here the cost or error (or risk) functional E is used, which expresses a dependency between an error measure and the weights, $E = E(\mathbf{w})$.

Unfortunately, as mentioned in section 1.3, genuine soft models are nonlinear approximators in the sense that an error functional $E = E(\mathbf{w})$ (the norm or measure of model goodness) depends nonlinearly upon weights that are the very subjects of learning. This means that the error hypersurface is generally not a convex function with guaranteed minimum. Therefore, a search after the best set of parameters (weights) that will ensure the best performance of the model falls into the category of nonlinear optimization problems.

As is well known, there is no general optimization method for nonlinear learning tasks. Section 1.3 introduces possibly the simplest to understand and the easiest to use: the gradient method. Despite being simple, the first-order gradient learning algorithm (also known as the error backpropagation algorithm) was the first learning procedure that made a key breakthrough in training multilayer neural networks. But this simplicity has a price: the learning procedure is too long and does not guarantee finding the best set of weights for a given NN structure. (Some improvements are discussed in chapter 8). It should be stressed that the only difference between NNs and SVMs is in how these models learn. After the learning phase, NNs and SVMs are essentially the same.

1.1 Examples of Applications in Diverse Fields

Soft computing, which comprises fuzzy logic modeling and the theory and application of the (statistical) learning techniques embedded in SVMs and NNs, is still in an early stage of development. Nevertheless, many research institutions, industries, and commercial firms have already started to apply these novel tools successfully to many diverse types of real-world problems. Practically no area of human activity is left untouched by NNs, SVMs, or fuzzy logic models. The most important applications include

- Pattern (visual, sound, olfactory, tactile) recognition (i.e., classification)
- Time series forecasting (financial, weather, engineering time series)
- Diagnostics (e.g., in medicine or engineering)
- Robotics (control, navigation, coordination, object recognition)
- Process control (nonlinear and multivariable control of chemical plants, power stations, vehicles or missiles)

- Optimization (combinatorial problems like resource scheduling, routing)
- Signal processing, speech and word recognition
- Machine vision (inspection in manufacturing, check reader, face recognition, target recognition)
- Financial forecasting (interest rates, stock indices, currencies)
- Financial services (credit worthiness, forecasting, data mining, data segmentation), services for trade (segmentation of customer data)

In certain application areas, such as speech and word recognition, NNs, FL models, or SVMs outperform conventional statistical methods. In other fields, such as specific areas in robotics and financial services, they show promising application in real-world situations.

Because of various shortcomings of both neural networks and fuzzy logic models and the advantages of combining them with other technologies, hybrid and modular solutions are becoming popular. In addition, complex real-world problems require more complex solutions than a single network (or a one-sided approach) can provide. The generic soft computing approach also supports the design of solutions to a wide range of complex problems. They include satellite image classification, advanced data analysis, optical character recognition, sales forecasting, traffic forecasting, and credit approval prediction.

The theoretical foundations, mathematics, and software techniques applied are common for all these different areas. This book describes the common fundamental principles and underlying concepts of statistical learning, neural networks, and fuzzy logic modeling as well as some of the differences between them.

A natural and direct connection exists between soft computing models (NNs, SVMs, and FL models) and classical statistics. The models presented here can be viewed as nonlinear extensions of linear regression and classification methods, tools, and approaches. However, introducing nonlinearities (i.e., nonlinear dependence of the approximating models upon model parameters) increases the complexity of the learning tools dramatically. Learning usually means nonlinear optimization, which becomes the most important task to solve in machine learning theory. This book deals with the various nonlinear optimization techniques in the framework of learning from experimental data.

Before considering some popular and successful applications of NN models, it may be interesting to look at the wide range of problems that classical (linear) statistics attempted to solve. More details on these and many others, may be found in the standard statistical literature (e.g., Anderson 1958; Johnson and Wichern 1982).

- Effects of drugs on sleeping time
- Pulmonary function modeling by measuring oxygen consumption
- Comparison of head lengths and breadths of brothers
- Classification of the Brahman, Artisan, and Korwa groups based on physical measurements
- Classification of two species of flies using data on biting flies
- Battery-failure data dependence and regression
- Financial and market analyses (bankruptcy, stock market prediction, bonds, goods transportation cost data, production cost data)
- Study of love and marriage using data on the relationships and feelings of couples
- Air pollution data classification, college test scores classification and prediction, crude oil consumption modeling, degree of relation among 11 languages.

This is only a short list, but it shows the wide diversity of problems to which linear statistics has been successfully applied. In many instances, linear models perform well. In fact, whenever there are linear (or slightly nonlinear) dependencies in regression problems or when separation functions between the classes are (closely) linear, one can obtain very good results by applying conventional statistical tools.

Today, equipped with powerful computing techniques and high-performance sensors and actuators, we want to solve much more complex (highly nonlinear and high-dimensional) problems. However, this is even more risky endeavor than solving a variety of classical linear problems; this book introduces the reader to the very challenging and promising field of nonlinear classification and regression based on learning from experimental data. In addition, it presents, as a third constituent of soft computing, fuzzy logic modeling as a tool for embedding structured human knowledge into workable algorithms.

To begin, it may be helpful to look at a few successful developments and applications of neural networks and fuzzy logic paradigms. The success of these applications spurred widespread acceptance of these novel and powerful nonlinear modeling techniques. This short review is far from conclusive. It discusses only a few of the important supervised learning NN models as well as some early pioneering applications of FL models in solving practical problems.

The construction of the first learning machine, called the perceptron, by F. Rosenblatt in late 1960s is certainly a milestone in the history of NNs (see chapter 3). This was the first model of a machine that learns from experimental data, and this is when mathematical analysis of learning from data began. The early perceptron was designed for solving pattern recognition problems, that is, classification tasks.

At the same time, a philosophy of statistical learning theory was being developed by V. Vapnik and A. Chervonenkis (1968). Unlike the experimental approach of Rosenblatt, their work formulated essential theoretical concepts: Vapnik-Chervonenkis entropy and the Vapnik-Chervonenkis dimension, which in 1974 resulted in a novel inductive principle called structural risk minimization. At this early stage, these tools were also applied to classification problems (see chapter 2).

Concurrently, B. Widrow and M. Hoff developed the first adaptive learning rule for solving linear regression problems: the least mean square (LMS) learning rule, also known as the delta learning rule (see chapter 3). It was a rule for training a (neural) processing unit called the adaline (adaptive linear neuron) for adaptive signal filtering and adaptive equalization. Hence, this linear neuron was performing linear regression tasks.

By the mid-1980s a lot of progress had been made in developing specialized hardware and software for solving real-life problems without the relevant theoretical concepts being applied to the (mostly experimental) supervised learning machines. (Many unsupervised learning algorithms and approaches were also developed during that period.) Then, about that time, a breakthrough in learning from data and in neural network development came when several authors (Le Cun 1985; Parker 1985; Rumelhart, Hinton, and Williams 1986) independently proposed a gradient method, called error backpropagation, for training hidden layer weights (see section 4.1).

Independently, continuing their research in the field of statistical learning theory, Vapnik and Chervonenkis found the necessary and sufficient conditions for consistency of the empirical risk minimization inductive principle in 1989. In this way, all the theory needed for powerful learning networks was established, and in the early 1990s Vapnik and his coworkers developed support vector machines aimed at solving nonlinear classification and regression problems (see chapter 2).

A lot of effort was devoted in the late 1980s to developing so-called regularization networks, also known as radial basis function networks (Powell 1987; Broomhead and Lowe 1988; Poggio and Girosi 1989 and later). These networks have firm theoretical roots in Tikhonov's regularization theory (see chapter 5). A few well-known and successful applications are described here. The common feature of all the models (functions or machines) in these applications is that they learn complex, high-dimensional, nonlinear functional dependencies between given input and output variables from training data.

One of the first successful applications was the NETtalk project (Sejnowski and Rosenberg 1987), aimed at training a neural network to pronounce English text consisting of seven consecutive characters from written text, presented in a moving window that gradually scanned the text. Seven letters were chosen because linguistic studies have shown that the influence of the fourth and fifth distant letters on the pronunciation of the middle character is statistically small. To simplify the problem of speech synthesis, the NETtalk network recognized only 29 valid characters: the 26 alphabetic characters from A to Z and the comma, period, and space. No distinction was made between upper and lower case, and all characters that were not one of these 29 were ignored. Thus, the input was a $7 \times 29 = 203$-dimensional vector. The desired output was a phoneme code to be directed to a speech generator giving the pronunciation of the letter at the center of the input window. The network had 26 output units, each forming one of the 26 codes for the phoneme sound generation commands. The NETtalk network is an error backpropagation model having one hidden layer with 80 processing units (neurons). For the approach taken in this book, it is important to realize that such a structure represents a highly nonlinear mapping from a 203-dimensional space into a 26-dimensional space (i.e., NETtalk is an $\Re^{203} \to \Re^{26}$ mapping). The neural network was trained on 1,024 words and achieved intelligible speech after 10 training epochs and 95% accuracy after 50 epochs. (An epoch is a sweep through all the training data.)

Gorman and Sejnowski (1988) trained the same kind of multilayer perceptron to distinguish between reflected sonar signals from two kinds of objects lying at the bottom of the sea: rocks and metal cylinders. The input signal was the frequency spectrum (Fourier transform) of the reflected sonar signal. The network had 60 input units and 2 output neurons—one for rocks and one for cylinders. Note that this is a one-out-of-two classification (pattern recognition) problem. They varied the number of hidden layer (HL) neurons from zero (when an NN is without a hidden layer) to 24. Without any HL unit, the network achieved about 80% correct performance on training data. With two HL units, the network reached almost 100% accuracy on every training trial. There was no visible improvement in the results on increasing the number of HL units from 12 to 24. After training, the network was tested on new, previously unseen, data; with 12 HL neurons, it achieved about 85% correct classification. We are here particularly interested in mathematical side of the problem being solved. Therefore, note that the sonar signals recognition network performed a highly nonlinear mapping from a 60-dimensional space into a 2-dimensional space (an $\Re^{60} \to \Re^{2}$ mapping).

Pomerlau (1989) reported results on designing an NN-based car driver within the framework of the ALVINN (autonomous land vehicle in a neural network) project. The ALVINN network took road images from a camera (30×32 pixel image) and a laser range finder (8×32 pixel image) as inputs. Its output was a direction of the vehicle in order to follow the road. There were 29 neurons in a single hidden layer and 46 neurons in the output layer. Thus, the input vector was 1216-dimensional, and ALVINN represented a nonlinear functional mapping from a 1216-dimensional space into a 46-dimensional space (an $\Re^{1216} \rightarrow \Re^{46}$ mapping).

Handwritten character recognition is certainly one of the most interesting areas in which NNs have found wide application. One of the first reported applications is due to Le Cun and his colleagues (1989). The network input was a 16×16 array (i.e., a 256-dimensional vector) that received a pixel image of a particular handwritten digit scaled to a standard size. This signal was fed forward through three hidden layers to the output layer, which comprised ten output units, each signifying one of the digits 0–9. There were 64 neurons in the first HL, 16 in the second, and 30 in the third. The network used 9,300 ZIP codes (numerical postal codes). Approximately one quarter (2,000) was used for the test, and it was not seen during the training. This network, named LeNet1, was trained by error backpropagation. The error on the test set was 5.1%. It is interesting that this handwritten character recognition problem became a benchmark for various NN and SVM models. Recently, competitive results were reported by applying SVM with polynomial kernel functions, which achieved accuracy of 4% on test data. In the meantime, Le Cun and his colleagues developed LeNet5, which performed better than both of the previous models. Details of comparative results with other models can be found in Vapnik (1998). From the mathematical point of view, LeNet1 represented a nonlinear functional mapping from a 256-dimensional space into a 10-dimensional space (an $\Re^{256} \rightarrow \Re^{10}$ mapping).

These four early applications are only a few of the thousands now existing in many fields of human endeavor. An important point to note about all these applications is that the models all learn high-dimensional nonlinear mappings from training data that are usually sparse. This is the world of learning from data networks (models, functions, and machines), where, at the moment, no better alternatives exist.

A brief history of fuzzy logic and its applications follows. Fuzzy logic was first proposed by L. A. Zadeh in 1965. He elaborated on his ideas in a 1973 paper, which introduced the concept of linguistic variables, or fuzzy sets. Assilian and Mamdani were the first to implement fuzzy logic rules, for controlling a steam generator, in 1974. The first industrial application followed soon after: implementing fuzzy control in a cement kiln built in Denmark in 1975. Interestingly, at this early stage fuzzy systems were ignored in the country of their theoretical origins, the United States. One explanation is that people associated FL models and approaches with artificial intelligence, expert systems, and knowledge-based engineering, which at that point had not lived up to expectations. Such early (and erroneous) associations resulted in a lack of credibility for FL in U.S. industrial firms.

In Japan, without such prejudice, interest in fuzzy systems was much stronger. This might have been part of the so-called "invented here" syndrome, in which innovative ideas supposedly get more attention if they come from far away. In any case, Hitachi's first simulations, in 1985, demonstrated the superiority of fuzzy control systems for the Sendai railway. Within two years, fuzzy systems had been adopted to control accelerating, braking, and stopping of Sendai trains.

Another event helped promote interest in fuzzy systems in Japan. During an international meeting in 1987 in Tokyo, T. Yamakawa demonstrated the use of fuzzy control in an "inverted pendulum" experiment. This is a classic benchmark problem in controlling a nonlinear unstable system. He implemented a set of simple dedicated fuzzy logic chips for solving this nonlinear control task.

Following such demonstrations of FL models' capabilities, fuzzy systems were built into many Japanese consumer goods. Matsushita vacuum cleaners used four-bit FL controllers to adjust suction power according to dust sensor information. Hitachi washing machines implemented fuzzy controllers in load-weight, fabric-mix, and dirt sensors to automatically set the wash cycle for the best use of power, water, and detergent. Canon developed an auto-focusing camera that used a charge-coupled device to measure the clarity of the image in six regions of its field of view and with the information provided to determine if the image was in focus. The dedicated FL chip also tracked the rate of change of lens movement during focusing and controlled its speed to prevent overshoot. The camera's fuzzy control system used 12 inputs: six to obtain the current clarity data provided by the charge-coupled device and six to measure the rate of change of lens movement. The output was the position of the lens. The fuzzy control system used 13 rules and required 1.1 kilobytes of memory. However, for obvious reasons, the camera was not advertised as a "fuzzy" camera. Instead, the adjective "smart" was used, which (because of the application of smart fuzzy rules) this camera certainly was.

Another example of a consumer product incorporating fuzzy controllers is an industrial air conditioner designed by Mitsubishi that used 25 heating rules and 25 cooling rules. A temperature sensor provided input, and fuzzy controller outputs were fed to an inverter, a compressor valve, and a fan motor. According to Mitsubishi, compared to the previous design, the fuzzy controller heated and cooled five times faster, reduced power consumption by one quarter, increased temperature stability by a factor of two, and used fewer sensors.

Following the first successful applications, many others were reported in fields like character and handwriting recognition, optical fuzzy systems, robotics, voice-controlled helicopter flight, control of flow of powders in film manufacture, and elevator systems.

Work on fuzzy systems also proceeded in Europe and the United States, although not with the same enthusiasm as in Japan. In Europe, at the same time as FL was introduced for control purposes, H. Zimmermann and his coworkers found it useful in modeling human decision processes. However, they realized that the classical FL proposed by Zadeh was insufficient for modeling complex human decision processes, and they developed extensions, such as compensatory aggregation operators. As an immediate result of this, INFORM Corporation introduced a decision support system for banks in 1986.

The list of large European companies that started fuzzy logic task forces includes SGS-Thompson of France and Italy as well as Klöckner-Moeler, Siemens, and Daimler-Benz of Germany. SGS-Thompson invested $20 million in a fuzzy logic task force in Catania, Italy. This project primarily targeted FL hardware. Siemens started an FL task force at its central R&D facility in Munich. This task force emphasized expanding FL theory as well as supporting applications. The Siemens applications included washing machines, vacuum cleaners, automatic transmission systems, engine idle-speed controllers, traffic controllers, paper-processing systems, and H2-leakage diagnosis systems. A survey done in 1994 identified a total of 684 applications of fuzzy logic in Europe that were classified into four categories: industrial automation (44%), decision support and data analysis (30%), embedded control (19%), and process control (7%).

The Environmental Protection Agency in the United States has investigated fuzzy control for energy-efficient motors, and NASA has studied fuzzy control for automated space docking: simulations show that a fuzzy control system can greatly reduce fuel consumption. Firms such as Boeing, General Motors, Allen-Bradley, Chrysler, Eaton, and Whirlpool have worked on fuzzy logic for use in low-power refrigerators, improved automotive transmissions, and energy-efficient electric motors.

Research and development is continuing apace in fuzzy software design, fuzzy expert systems, and integration of fuzzy logic with neural networks in so-called neurofuzzy or fuzzyneuro systems. These issues are discussed in more detail later in the book.

1.2 Basic Tools of Soft Computing: Neural Networks, Fuzzy Logic Systems, and Support Vector Machines

In recent years, neural networks, fuzzy logic models, and support vector machines have been used in many different fields. This section primarily discusses NNs and FL models. SVMs are discussed in depth in chapter 2. However, because of a very high degree of resemblance between NNs and SVMs, almost all comments about the representational properties of NNs can also be applied to SVMs. Unlike their representational capabilities, the learning stages of these two modeling tools are different. Chapters 2, 3, and 4 clarify the differences.

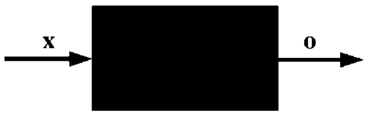
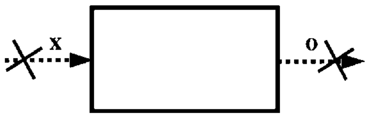
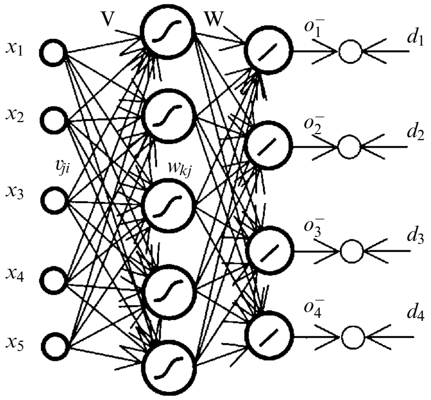
NNs and FL models are modeling tools. They perform in the same way after the learning stage of NNs or the embedding of human knowledge about some specific task of FL is finished. They are two sides of the same coin.[1] Whether the more appropriate tool for solving a given problem is an NN or an FL model depends upon the availability of previous knowledge about the system to be modeled and the amount of measured process data.

The classical NN and FL system paradigms lie at the two extreme poles of system modeling (see table 1.1). At the NN pole there is a black box design situation in which the process is entirely unknown but there are examples (measurements, records, observations, samples, data pairs). At the other pole (the FL model) the solution to the problem is known, that is, structured human knowledge (experience, expertise, heuristics) about the process exists. Then there is a white box situation. In short, the less previous knowledge exists, the more likely it is that an NN, not an FL, approach will be used to attempt a solution. The more knowledge available, the more suitable the problem will be for the application of fuzzy logic modeling. On the whole, both tools are aimed at solving pattern recognition (classification) and regression (multivariate function approximation) tasks.

For example, when they are applied in a system control area or the digital signal processing field, neural networks can be regarded as a nonlinear identification tool. This is the closest connection with a standard and well-developed field of estimation or identification of linear control systems. In fact, if the problem at hand is a linear one, an NN would degenerate into a single linear neuron, and in this case the weights of the neuron would correspond to the parameters of the plant's discrete transfer function $G(z)$ (see example 3.6). When applied to stock market predictions (see section 7.2) the approach will be the same as for linear dynamics identification, but the network will become a more complex structure. The underlying dependencies (if there are any) are usually far from being linear, and linear assumptions can no longer stand. A new, hidden layer of neurons will have to be added. In this way, the network can model nonlinear functions. This design step leads to a tremendous increase in modeling capacity, but there is a price: a nonlinear kind of learning will have to be performed, and this is generally not an easy task. However, this is the point where the world of neural networks and support vector machines begins.

In order to avoid too high (or too low) expectations for these new concepts of computing, particularly after they have been connected with intelligence, it might be useful to list some advantages and disadvantages that have been claimed for NNs and FL models (see tables 1.2 and 1.3). Because of the wide range of applications of

Table 1.1
Neural Networks, Support Vector Machines, and Fuzzy Logic Modeling as Examples of Modeling Approaches at Extreme Poles

Neural Networks and Support Vector Machines	Fuzzy Logic Models
	
Black Box	White Box
No previous knowledge, but there are measurements, observations, records, i.e., data pairs $\{\mathbf{x}_i, \mathbf{d}_i\}$ are known. Weights **V** and **W** are unknown.	Structured knowledge (experience, expertise, or heuristics). No data required. IF-THEN rules are the most typical examples of structured knowledge.
 V W x_1 x_2 x_3 x_4 x_5 v_{ji} w_{kj} o_1 o_2 o_3 o_4 d_1 d_2 d_3 d_4	*Example: Controlling the distance between two cars:* R1: IF the speed is *low* AND the distance is *small*, THEN the force on brake should be *small*. R2: IF the speed is *medium* AND the distance is *small*, THEN the force on brake should be *big*. R3: IF the speed is *high* AND the distance is *small*, THEN the force on brake should be *very big*.
Behind NNs and SVMs stands the idea of learning from the training data.	Behind FL stands the idea of embedding human knowledge into workable algorithms.

In many instances, we do have both *some knowledge* and *some data*.

This is the most common *gray box* situation covered by the paradigm of neuro-fuzzy or fuzzy-neuro models.

If we do not have any prior knowledge *and* we do not have any measurements (by all accounts, a very hopeless situation indeed), it may be hard to expect or believe that the problem at hand may be approached and solved easily. This is a *no-color box* situation.

Table 1.2
Some Advantages of Neural Networks and Fuzzy Logic Models

Neural Networks	Fuzzy Logic Models
Have the property of learning from the data, mimicking human learning ability	Are an efficient tool for embedding human (structured) knowledge into useful algorithms
Can approximate any multivariate nonlinear function	Can approximate any multivariate nonlinear function
Do not require deep understanding of the process or the problem being studied	Are applicable when mathematical model is unknown or impossible to obtain
Are robust to the presence of noisy data	Operate successfully under a lack of precise sensor information
Have parallel structure and can be easily implemented in hardware	Are useful at the higher levels of hierarchical control systems
Same NN can cover broad and different classes of tasks	Are appropriate tool in generic decision-making process

Table 1.3
Some Disadvantages of Neural Networks and Fuzzy Logic Models

Neural Networks	Fuzzy Logic Models
Need extremely long training or learning time (problems with local minima or multiple solutions) with little hope for many real-time applications.	Human solutions to the problem must exist, and this knowledge must be structured. Experts may have problems structuring the knowledge.
Do not uncover basic internal relations of physical variables, and do not increase our knowledge about the process.	Experts sway between extreme poles: too much aware in field of expertise, or tending to hide their knowledge.
Are prone to bad generalizations (with large number of weights, tendency to overfit the data; poor performance on previously unseen data during the test phase).	Number of rules increases exponentially with increase in the number of inputs and number of fuzzy subsets per input variable.
Little or no guidance is offered about NN structure or optimization procedure, or the type of NN to use for a particular problem.	Learning (changing membership functions' shapes and positions, or rules) is highly constrained; typically more complex than with NN.

these modeling tools, it is hard to prove or disprove these claims and counterclaims. Only a part of the answers will be found in this book. It is certain that everyone working with NNs, SVMs, and FL models will form their own opinions about these claims. However, the growing number of companies and products employing NNs and FL models and the increasing number of new neural network and fuzzy computing theories and paradigms show that despite the still many open questions NNs, SVMs, and FL models are already well-established engineering tools and are becoming a common computational means for solving many real-life tasks and problems.

1.2.1 Basics of Neural Networks

Artificial neural networks are software or hardware models inspired by the structure and behavior of biological neurons and the nervous system, but after this point of inspiration all resemblance to biologyical systems ceases.

There are about 50 different types of neural networks in use today. This book describes and discusses *feedforward* NNs with *supervised learning*. This section deals with the *representational capabilities* of NNs. It shows what NNs can model and how they can represent some specific underlying functions that generated training data. Chapters 3, 4, and 5 describe the *problem of learning*—how the best set of weights, which enables an NN to be a universal approximator, can be calculated (learned) by using these training data. Chapter 2 discusses much broader issues of statistical learning theory and in that framework presents support vector machines as approximating models with a powerful modeling capacity.

Feedforward neural networks are the models used most often for solving nonlinear classification and regression tasks by learning from data. In addition, feedforward NNs are mathematically very close, and sometimes even equivalent, to fuzzy logic models. Both NN and FL approximation techniques can be given graphical representation, which can be called a neural network or a fuzzy logic model. With such a representation of NN or FL tools, nothing new is added to approximation theory, but from the point of view of implementation (primarily in the sense of parallel and massive computing) this graphical representation is a desirable property.

There is a strong mathematical basis for developing NNs in the form of the famous Kolmogorov theorem (1957). This theorem encouraged many researchers but is still a source of controversy (see Girosi and Poggio 1989; Kurkova 1991). The Kolmogorov theorem states,

Given any continuous function $f\colon [0,1]^n \to \Re^m, f(\mathbf{x}) = \mathbf{y}, f$ can be implemented exactly by a network with n neurons (fan-out nodes) in an input layer, $(2n+1)$ neurons in the hidden layer, and m processing units (neurons) in the output layer.

However, the proof of this important theorem is not constructive in the sense that it cannot be used for network design. This is the reason this book pursues the standard constructive approaches developed in the framework of NNs or SVMs.

Artificial neural networks are composed of many computing units popularly (but perhaps misleadingly) called neurons. The strength of the connection, or link, between two neurons is called the weight. The values of the weights are true network parameters and the subjects of the learning procedure in NNs. Depending upon the problem, they have different physical meanings, and sometimes it is hard to find any physical meaning at all. Their geometrical meaning is much clearer. The weights define the positions and shapes of basis functions in neural network and fuzzy logic models.

The neurons are typically organized into layers in which all the neurons usually possess the same activation functions (AFs). The genuine neural networks are those with an input layer and at least two layers of neurons—a hidden layer (HL), and an output layer (OL)—provided that the HL neurons have nonlinear and differentiable AFs. Note that such an NN has two layers of adjustable weights that are typically organized as the elements of the weights matrices **V** and **W**. The matrix **V** is the matrix of the hidden layer weights and the matrix **W** comprises the output layer weights. For a single OL neuron, **W** degenerates into a weights vector **w**.

The nonlinear activation functions in the hidden layer neurons enable the neural network to be a universal approximator. Thus the nonlinearity of the AFs solves the problems of representation. The differentiability of the HL neurons' AFs makes possible the solution of nonlinear learning. (Today, using random optimization algorithms like the genetic algorithm, one may also think of learning HL weights in cases where the AFs of the hidden layer neurons are not differentiable. Fuzzy logic models are the most typical networks having nondifferentiable activation functions.)

Here, the input layer is not treated as a layer of neural processing units. The input units are merely fan-out nodes. Generally, there will not be any processing in the input layer, and although in its graphical representation it looks like a layer, the input layer is not a layer of neurons. Rather, it is an input vector, eventually augmented with a bias term, whose components will be fed to the next (hidden or output) layer of neural processing units. The OL neurons may be linear ones (for regression types of problems), or they can have sigmoidal activation functions (for classification or pattern recognition tasks). For NN-based adaptive control schemes or for (financial) times series analyses, OL neurons are typically linear units. An elementary (but powerful) feedforward neural network is shown in figure 1.1. This is a graphical representation of the approximation scheme (1.1):

$$o(\mathbf{x}, \mathbf{V}, \mathbf{w}, \mathbf{b}) = F(\mathbf{x}, \mathbf{V}, \mathbf{w}, \mathbf{b}) = \sum_{j=1}^{J} w_j \sigma_j(\mathbf{v}_j^T \mathbf{x} + b_j), \tag{1.1}$$

where σ_j stands for sigmoidal activation functions. This network is called a multilayer perceptron (see chapters 3 and 4). J corresponds to the number of HL neurons. By explicitly writing $o = o(\mathbf{x}, \mathbf{V}, \mathbf{w}, \mathbf{b})$ we deliberately stress the fact that the output from an NN depends upon the weights (unknown before learning) contained in $\mathbf{V}$, $\mathbf{w}$, and $\mathbf{b}$. Input vector $\mathbf{x}$, bias weights vector $\mathbf{b}$, HL weights matrix $\mathbf{V}$, and OL weights vector $\mathbf{w}$ are as follows:

$$\mathbf{x} = [x_1 \quad x_2 \quad \dots \quad x_n]^T. \tag{1.2}$$

$$\mathbf{V} = \begin{bmatrix} v_{11} & \dots & v_{1i} & \dots & v_{1n} \\ \vdots & & \vdots & & \vdots \\ v_{j1} & \dots & v_{ji} & \dots & v_{jn} \\ \vdots & & \vdots & & \vdots \\ v_{J1} & \dots & v_{Ji} & \dots & v_{Jn} \end{bmatrix}. \tag{1.3}$$

$$\mathbf{b} = [b_1 \quad b_2 \quad \dots \quad b_J]^T. \tag{1.4}$$

$$\mathbf{w} = [w_1 \quad w_2 \quad \dots \quad w_J \quad w_{J+1}]^T. \tag{1.5}$$

At this point, a few comments may be needed. Figure 1.1 represents a general structure of multilayer perceptrons, radial basis function (RBF) networks, and SVMs. In the case of a multilayer perceptron, x_{n+1} will be the constant term equal to 1, called *bias*. The bias weights vector $\mathbf{b}$ can simply be integrated into an HL weights matrix $\mathbf{V}$ as its last column. After such concatenation, (1.1) can be simplified to $o = \mathbf{w}^T \sigma(\mathbf{V}\mathbf{x})$.[2] For RBF networks, $x_{n+1} = 0$, meaning that there is no bias input. However, it can be used in an OL (see chapter 5).

For both multilayer perceptrons and RBF networks, the HL bias term may be used but is not generally needed. The HL activation functions shown in figure 1.1 are sigmoidal, indicating that this particular network represents a multilayer perceptron. However, the structures of multilayer perceptrons, RBF networks, and fuzzy logic models are the same or very similar. The basic distinction between sigmoidal and radial basis function networks is how the input to each particular neuron is calculated (see (1.6) and (1.7) and fig. 1.2).

The basic computation that takes place in an NN is very simple. After a specific input vector is presented to the network, the input signals to all HL neurons u_j are computed either as scalar (dot or inner) products between the weights vectors $\mathbf{v}_j$ and $\mathbf{x}$ (for a linear or sigmoidal AF) or as Euclidean distances between the centers $\mathbf{c}_j$ of the RBF and $\mathbf{x}$ (for RBF activation functions). A radial basis AF is typically parameterized by two sets of parameters: the center $\mathbf{c}$, which defines its position, and a second set of parameters that determines the shape (width or form) of an RBF. In the

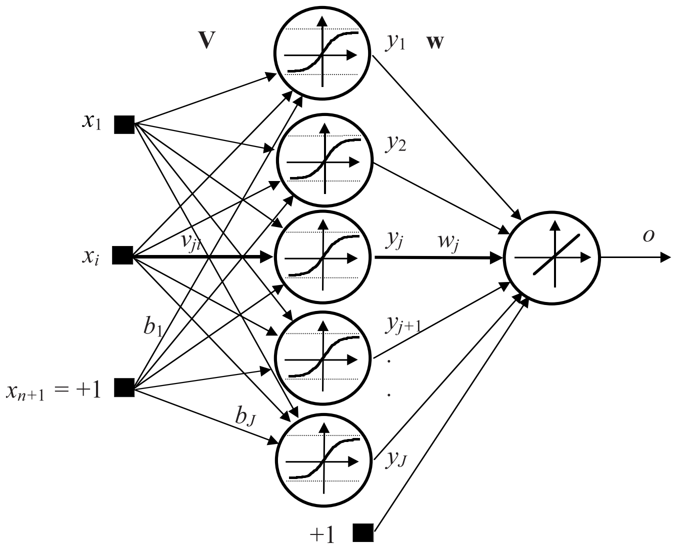


Figure 1.1
Feedforward neural network that can approximate any $\Re^n \to \Re^1$ nonlinear mapping.

case of a one-dimensional Gaussian function this second set of parameters is the standard deviation σ. (Do not confuse the standard deviation σ with the sigmoidal activation function σ given in (1.1) and shown in HL neurons in fig. 1.1.) In the case of a multivariate input vector $\mathbf{x}$ the parameters that define the shape of the hyper-Gaussian function are elements of a covariance matrix Σ. To put it simply, the ratios of HL bias weights and other HL weights of sigmoidal activation functions loosely correspond to the centers of RBF functions, whereas weights v_{ji}, which define the slope of sigmoidal functions with respect to each input variable, correspond to the width parameter of the RBF. Thus, the inputs to the HL neurons for sigmoidal AFs are given as

$$u_j = \mathbf{v}_j^T \mathbf{x}, \qquad j = 1, \dots, J, \tag{1.6}$$

and for RBF activation functions the distances are calculated as

$$r_j = \sqrt{(\mathbf{x} - \mathbf{c}_j)^T (\mathbf{x} - \mathbf{c}_j)}, \qquad u = f(r, \sigma). \tag{1.7}$$

In the case of three-dimensional input, when $\mathbf{x} = [x_1, x_2, x_3]^T$ and $\mathbf{c} = [c_1, c_2, c_3]^T$, the Euclidean distance r_j can be readily calculated as

$$r_j = \sqrt{(\mathbf{x} - \mathbf{c}_j)^T (\mathbf{x} - \mathbf{c}_j)} = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + (x_3 - c_3)^2}. \tag{1.8}$$

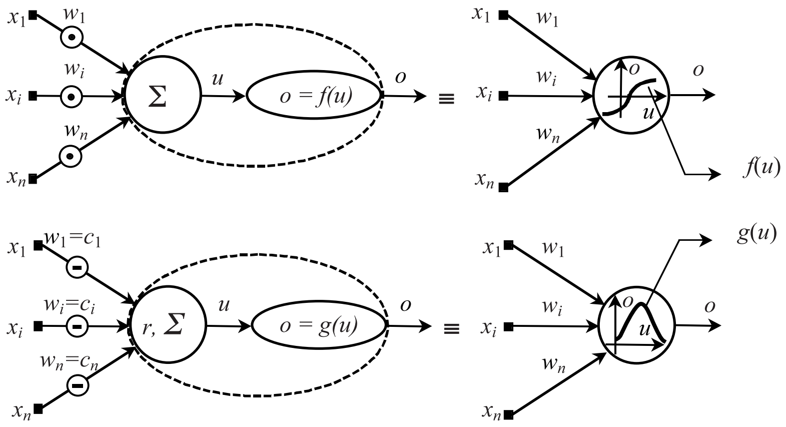


Figure 1.2
Formation of the input signal u to the neuron's activation function. *Top*, sigmoidal type activation function; input to the neuron is a scalar product $u = \mathbf{w}^T\mathbf{x}$. *Bottom*, RBF type activation function; input to the neuron is a distance r between $\mathbf{x}$ and the RBF center $\mathbf{c}$ ($r = \|\mathbf{x} - \mathbf{c}\|$), r depending on parameters of the shape or width of the RBF. For a Gaussian function, the covariance matrix $\mathbf{\Sigma}$ contains the shape parameters.

The output from each HL neuron depends on the type of activation function used. The most common activation functions in multilayer perceptrons are the squashing sigmoidal functions: the unipolar logistic function (1.9) and the bipolar sigmoidal, or tangent hyperbolic, function (1.10). These two AFs and their corresponding derivatives are presented in figure 4.3.

$$o = \frac{1}{1 + e^{-u}} \tag{1.9}$$

$$o = \tanh\left(\frac{u}{2}\right) = \frac{2}{1 + e^{-u}} - 1 \tag{1.10}$$

Figure 1.2 shows the basic difference in forming the input signals u to the AFs of the multilayer perceptron and an RBF neuron.

All three powerful nonlinear modeling tools—a multilayer perceptron, an RBF network, and an SVM—have the same structure. Thus, their representational capabilities are the same or very similar after successful training. All three models learn from a set of training data and try to be as good as possible in modeling typically sparse and noisy data pairs in high-dimensional space. (Note that none of the three adjectives used eases our learning from data task.) The output from these models is a hypersurface[3] in an $\Re^n \times \Re^m$ space, where n is the dimension of the input space and

Table 1.4
Basic Models and Their Error (Risk) Functions

Multilayer Perceptron	Radial Basis Function Network	Support Vector Machine
$E = \sum_{i=1}^{P} \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\textit{Closeness to data}}$	$E = \sum_{i=1}^{P} \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\textit{Closeness to data}} + \lambda \underbrace{\lVert \mathbf{P}f \rVert^2}_{\textit{Smoothness}}$	$E = \sum_{i=1}^{P} \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\textit{Closeness to data}} + \underbrace{\Omega(l, h, \eta)}_{\textit{Capacity of a machine}}$

m is the dimension of the output space. In trying to find the best model, one should be able to measure the accuracy or performance of the model. To do that one typically uses some measure of goodness, performance, or quality (see section 1.3).

This is where the basic difference between the three models lies. Each uses a different norm (error, risk, or cost function) that measures the goodness of the model, and the optimization of the different measures results in different models. The application of different norms also leads to different learning (optimization) procedures. This is one of the core issues in this book. Table 1.4 tabulates the basic norms (risk or error functionals) applied in developing the three basic networks.

1.2.2 Basics of Fuzzy Logic Modeling

Fuzzy logic lies at the opposite pole of system modeling with respect to the NN and SVM methods. It is a white box approach (see table 1.1) in the sense that it is assumed that there is already human knowledge about a solution. Therefore, the modeled system is known (i.e., white). On the application level, FL can be considered an efficient tool for embedding structured human knowledge into useful algorithms. It is a precious engineering tool developed to do a good job of trading off precision and significance. In this respect, FL models do what human beings have been doing for a very long time. As in human reasoning and inference, the truth of any statement, measurement, or observation is a matter of degree. This degree is expressed through the membership functions that quantify (measure) a degree of belonging of some (crisp) input to given fuzzy subsets.

The field of fuzzy logic is very broad and covers many mathematical and logical concepts underlying the applications in various fields. The basics of these conceptual foundations are described in chapter 6. In particular, the chapter presents the fundamental concepts of crisp and fuzzy sets, introduces basic logical operators of conjunction (AND), disjunction (OR), and implication (IF-THEN) within the realm of fuzzy logic (namely, T-norms and T-conorms), and discusses the equivalence of NNs and FL models. (However, a deep systematic exposition of the theoretical issues is outside the scope of this book.) Furthermore, fuzzy additive models (FAMs) are introduced, which are universal approximators in the sense that they can approxi-

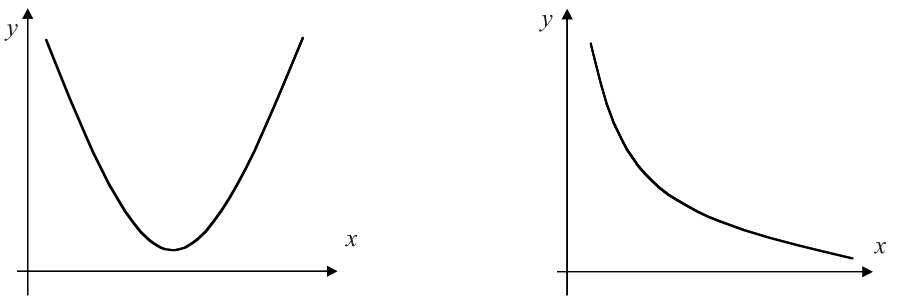


Figure 1.3
Two different nonlinear $\Re^1 \times \Re^1$ functions (mappings) to be modeled by a fuzzy additive model.

mate any multivariate nonlinear function on a compact domain to any degree of accuracy. This means that FAMs are dense in the space of continuous functions, and they share this very powerful property with NNs and SVMs.

This section discusses how FAMs approximate any (not analytically but verbally or linguistically) known functional dependency. A FAM is composed of a set of rules in the form of IF-THEN statements that express human knowledge about functional behavior. Suppose we want to model the two functions shown in figure 1.3. It is easy to model verbally the functional dependencies shown in figure 1.3. Both models would contain at least three IF-THEN rules. Using fewer rules would decrease the approximation accuracy, and using more rules would increase precision at the cost of more required computation time. This is the classical soft computing dilemma—a trade-off between imprecision and uncertainty on the one hand and low solution cost, tractability, and robustness on the other. The appropriate rules for the functions in figure 1.3 are as follows:

Left Graph	Right Graph
IF x is *low*, THEN y is *high.*	IF x is *low*, THEN y is *high.*
IF x is *medium*, THEN y is *low.*	IF x is *medium*, THEN y is *medium.*
IF x is *high*, THEN y is *high.*	IF x is *high*, THEN y is *low.*

These rules define three large rectangular patches that cover the functions. They are shown in figure 1.4 together with two possible approximators for each function.

Note that human beings do not (or only rarely) think in terms of nonlinear functions. We do not try to "draw these functions in our mind" or try to "see" them as geometrical artifacts. In general, we do not process geometrical figures, curves, surfaces, or hypersurfaces while performing tasks or expressing our knowledge. In addition, our expertise in or understanding of some functional dependencies is very

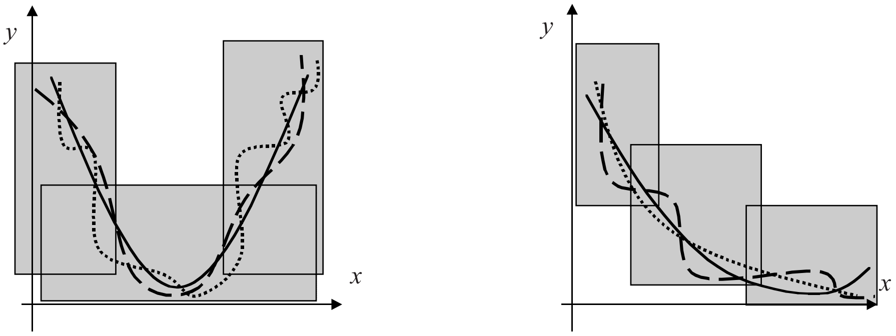


Figure 1.4
Two different functions (solid lines in both graphs) covered by three patches produced by IF-THEN rules and modeled by two possible approximators (dashed and dotted curves).

often not a structured piece of knowledge at all. We typically perform very complex tasks without being able to express how are we executing them. The curious reader should try, for example, to explain to a colleague in the form of IF-THEN rules how he is riding a bike, recognizing numerals, or surfing.

There are many steps, both heuristic and mathematical, between knowledge or expertise and a final fuzzy model. After all the design steps and computation have been completed, this final model is a very precisely defined nonlinear function. By choosing the complexity of the rule basis, one can control the precision of the fuzzy model and trade that off against solution costs. Thus, one first defines the most relevant input and output variables for a problem. In fuzzy logic terms, one must define the universes of discourse, i.e., the domains and the ranges of relevant variables. Then one specifies what is *low, medium, high, positive, zero, hot, cold,* and so on, in a given task. In fuzzy logic terms, one defines the fuzzy membership functions (fuzzy subsets or attributes) for the chosen input and output variables. Then one structures the knowledge in the form of IF-THEN rules, that is, fuzzy rules (one establishes the rule basis). The final stage is to perform the numerical part—applying some inference algorithm (e.g., SUM-PROD, MAX-MIN, SUM-MIN)—and to defuzzify the resulting (usually not-normal) fuzzy subsets. The last two steps are crisp and precise mathematical operations. *A* soft part in these calculations is the choice of membership functions as well as appropriate inference and defuzzification mechanisms. Again, there is a trade-off between simple and fast algorithms having low computational costs and the desired accuracy. Thus, the final approximating function depends upon many design decisions (only two out of many possible fuzzy approximators are shown in fig. 1.4). The design decisions include the number, shapes, and placements of the input and output membership functions, the inference mechanism applied, and the defuzzification method used.

Let us demonstrate the fuzzy modeling of a simple one-dimensional mapping $y = x^2$, $-3 < x < 0$. Choose four fuzzy *membership functions* (fuzzy subsets or attributes) for input and output variables as follows:

Input Variables	Output Variables
For $-3 < x < -2$, x is *very negative*.	For $4 < y < 10$, y is *large*.
For $-3 < x < -1$, x is *slightly negative*.	For $1 < y < 9$, y is *medium*.
For $-2 < x < 0$, x is *nearly zero*.	For $0 < y < 4$, y is *small*.
For $-1 < x < 0$, x is *very nearly zero*.	For $0 < y < 1$, y is *very small*.

These fuzzy membership functions are shown in figure 1.5. The *rule basis* for a fuzzy inference in the form of four IF-THEN rules is

R_1:	IF x is *very negative* (VN),	THEN y is *large* (L).
R_2:	IF x is *slightly negative* (SN),	THEN y is *medium* (M).
R_3:	IF x is *nearly zero* (NZ),	THEN y is *small* (S).
R_4:	IF x is *very nearly zero* (VNZ),	THEN y is *very small* (VS).

If one is not satisfied with the precision achieved, one should define more rules. This will be accomplished by a finer granulation (applying smaller patches) that can be realized by defining more membership functions. The fuzzy approximation that follows from a model with seven rules is shown in figure 1.6. The seven fuzzy membership functions (fuzzy subsets or attributes) for inputs and outputs, as well as the corresponding rule basis, are defined as follows:

Input Variables	Output Variables
For $-3.0 < x < -2.5$, x is *extremely far from zero*.	For $6.25 < y < 9$, y is *very large*.
For $-3.0 < x < -2.0$, x is *very far from zero*.	For $4 < y < 9$, y is *quite large*.
For $-2.5 < x < -1.5$, x is *quite far from zero*.	For $2.25 < y < 6.25$, y is *large*.
For $-2.0 < x < -1.0$, x is *far from zero*.	For $1 < y < 4$, y is *medium*.
For $-1.5 < x < -0.5$, x is *nearly zero*.	For $0.25 < y < 2.25$, y is *small*.
For $-1.0 < x < 0$, x is *very nearly zero*.	For $0 < y < 1$, y is *quite small*.
For $-0.5 < x < 0$, x is *extremely close to zero*.	For $0 < y < 0.25$, y is *very small*.

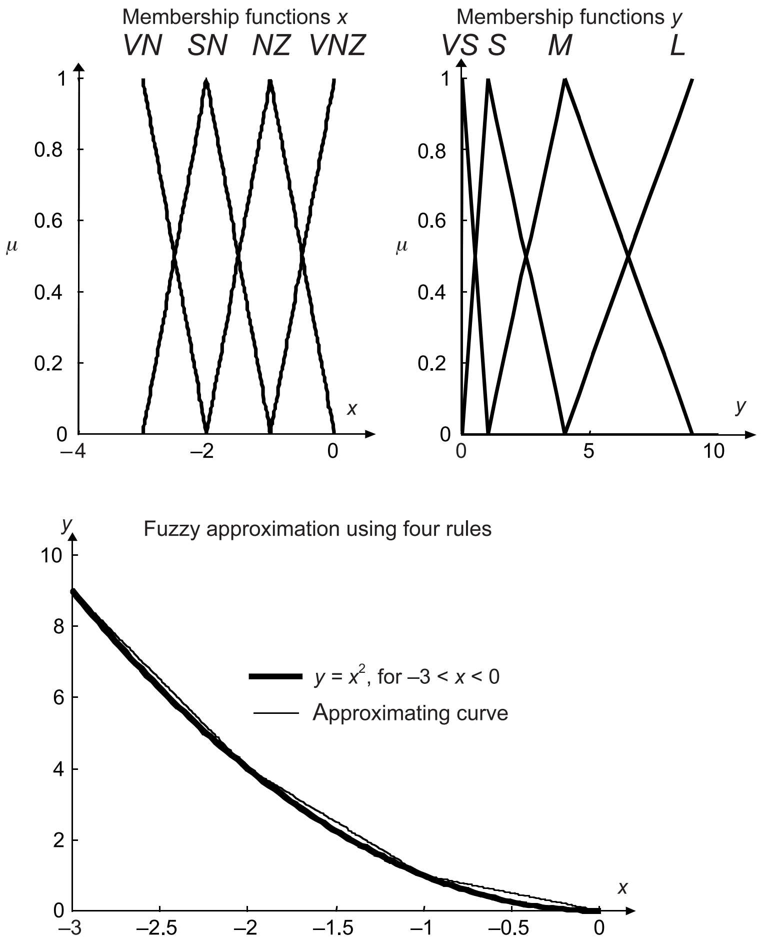


Figure 1.5
Modeling a one-dimensional mapping $y = x^2$ by a fuzzy model with four rules.

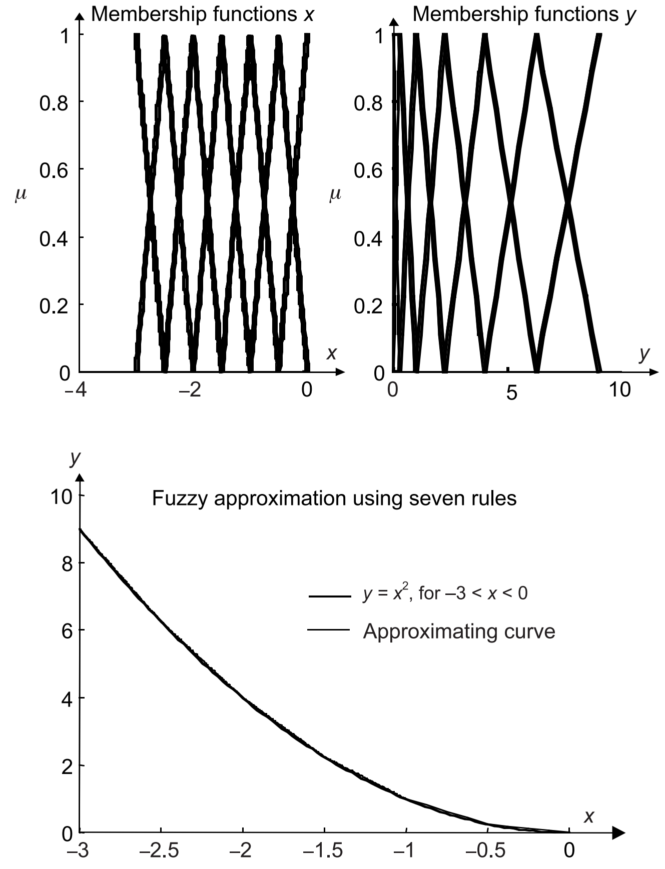


Figure 1.6
Modeling a one-dimensional mapping $y = x^2$ by a fuzzy model with seven rules.

R_1:	IF x is *extremely far from zero*,	THEN y is *very large*.
R_2:	IF x is *very far from zero*,	THEN y is *quite large*.
R_3:	IF x is *quite far from zero*,	THEN y is *large*.
R_4:	IF x is *far from zero*,	THEN y is *medium*.
R_5:	IF x is *nearly zero*,	THEN y is *small*.
R_6:	IF x is *very nearly zero*,	THEN y is *quite small*.
R_7:	IF x is *extremely close to zero*,	THEN y is *very small*

The fuzzy approximating function that results from a fuzzy additive model with seven rules is indistinguishable from an original and known functional dependency. Recall, however, that structured human knowledge is typically in the form of the (linguistically expressed) rule basis, not in the form of any mathematical expression. If one knew the mathematical expression, there would not be a need for designing a fuzzy model. One could simply use this known dependency in a crisp analytical form.

The fuzzy additive model can be represented graphically by a network like the one shown in figure 1.1. Section 6.2 discusses such structural equivalence. The resemblance, which follows from mild assumptions, can be readily seen in figure 6.25. The input membership functions of a FAM correspond to the HL activation (basis) functions in NNs, and the centers of the FAM's output membership functions are equivalent to the OL weights in an NN or SVM model.

1.3 Basic Mathematics of Soft Computing

The fields of learning from data and soft computing are mathematically well-founded disciplines. They are composed of several classical mathematics areas, shown as a "flower of learning and soft computing" in figure 1.7. One could say that both learning and soft computing are nothing but value-added applied mathematics and statistics, although this statement may be valid for many other fields as well. This book arose from a desire to show how the different areas depicted in figure 1.7, nicely connected, compose the powerful fields of learning from data and soft computing.

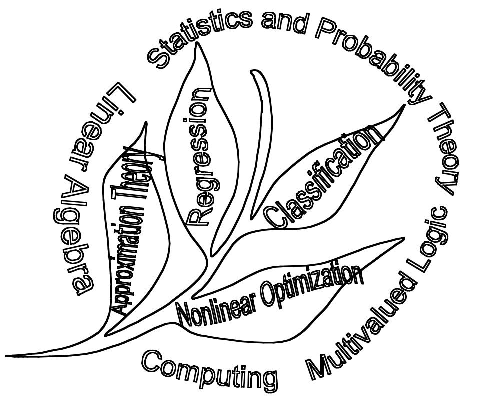


Figure 1.7
A "flower of learning and soft computing" and its basic mathematical constituents.

Here, *value-added* primarily means modern computer-based applications of standard and novel mathematical and statistical techniques. The statistical learning theory is presented in chapter 2, and fuzzy logic is the subject of chapter 6.

Two aspects of models (networks, machines, or mathematical functions) are analyzed here: what can these models represent, and how do we make them represent what we want them to? The first is *the problem of representation*, discussed in section 1.3.1. The second is *the problem of learning*, which boils down to some standard or novel nonlinear optimization techniques. Learning is the basic subject of this book, and it is analyzed in detail throughout. However, an elementary presentation of the need for, and the origins of, nonlinear optimization (learning or training) tools is given in section 1.3.2.

More specifically, section 1.3.1 presents the basics of classical and modern approaches to the approximation of multivariate functions, and section 1.3.2 introduces nonlinear optimization. Section 1.4 discusses the basics of classical statistical theory of regression and classification. Chapter 2 is devoted to the theoretically challenging area of support vector machines, which implement structural risk minimization, which is in turn developed within the framework of a new statistical learning theory. SVMs are also aimed at solving classification and regression problems, but unlike the theoretical approaches mentioned in section 1.4, which are based on known probability distributions, SVMs are the first mathematical models that do not assume any specific probability distributions and that learn from experimental data. Thus, sections 1.3 and 1.4 and chapter 2 roughly cover the basic theoretical constituents of the "learning from data" paradigm.

1.3.1 Approximation of Multivariate Functions

This section provides an elementary introduction to approximation theory. It also presents the basics of the theoretical interpolation/approximation[4] abilities of soft computing models: neural networks, support vector machines, and fuzzy logic models.

The classic one-dimensional problem is the approximation of a real continuous function $f(x)$ by an approximating function $f_a(x, \mathbf{w})$ having a fixed finite number of parameters w_i that are entries of a weights vector $\mathbf{w}$. (Later, $\mathbf{V}$ and $\mathbf{W}$ are used to denote matrices of hidden layer weights and output layer weights, respectively.) The basic problems in this book are somewhat different. One rarely tries to approximate some continuous univariate $f(x)$ or multivariate $f(\mathbf{x})$ function. The typical engineering problem to be solved is the interpolation or approximation of a set of P sparse and noisy training data points. However, to clarify the standard learning from data problem, classical issues from approximation theory are considered first. This section roughly follows Rice (1964) and Mason and Parks (1995).

Two major items in an approximation problem are the type of approximating function applied and the measure of goodness[5] of an approximation. This is also known as the question of choosing *form* and *norm*.

The choice of approximating function (form) is more important than the choice of a measure of goodness, that is, a distance function or norm that measures the distance between f and f_a. Unfortunately, there is no theoretical method of determining which out of many possible approximating functions will yield the best approximation. On the other hand, there are fortunately only a few feasible candidate functions in use or under investigation today. The most popular functions are tangent hyperbolic, a few radial basis functions (notably Gaussians and multiquadrics), polynomial functions, and three standard membership functions applied in fuzzy models (triangle, trapezoidal, and singleton). These functions are called activation, basis, and membership functions in multilayer perceptrons; radial basis function (RBF) or regularization networks, and fuzzy logic models. These models are, together with support vector machines, the most popular soft modeling and learning functions. Their mathematical forms follow.

A *multilayer perceptron* is a representative of nonlinear basis function expansion (approximation):

$$o = f_a(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{N} w_i \varphi_i(\mathbf{x}, \mathbf{v}_i), \tag{1.11}$$

where $\varphi_i(\mathbf{x}, \mathbf{v}_i)$ is a set of given functions (usually sigmoidal functions such as the logistic function or tangent hyperbolic—see (2.6) and chapter 3), o is the output from a model, and N is the number of hidden layer neurons. Both the output layer's weights w_i and the entries of the hidden layer weights vector $\mathbf{v}$ are free parameters that are subjects of learning.

An *RBF network* is a representative of a linear basis function expansion:

$$o = f_a(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{N} w_i \varphi_i(\mathbf{x}), \tag{1.12}$$

where $\varphi_i(\mathbf{x})$ is a fixed (chosen in advance) set of radial basis functions (e.g., Gaussians, splines, multiquadrics). Note that when the basis functions $\varphi_i(\mathbf{x})$ are not fixed, that is, when their positions $\mathbf{c}_i$ and shape parameters $\mathbf{\Sigma}$ are also subjects of learning ($\varphi_i = \varphi_i(\mathbf{x}, \mathbf{c}_i, \mathbf{\Sigma}_i)$), RBF networks become nonlinear approximation schemes. (See chapter 5 for a more detailed discussion of RBF networks.)

A *fuzzy logic model*, like an RBF network, can be a representative of a linear or nonlinear basis function expansion:

$$o = y = f_a(\mathbf{x}, \mathbf{c}, \mathbf{r}) = \frac{\sum_{i=1}^{N} G(\mathbf{x}, \mathbf{c}_i) r_i}{\sum_{i=1}^{N} G(\mathbf{x}, \mathbf{c}_i)}, \tag{1.13}$$

where N is the number of rules, r is the rules, and basis functions $G(\mathbf{x}, \mathbf{c}_i)$ are the input membership functions (attributes or fuzzy subsets) centered at $\mathbf{c}_i$ (see (2.7) and section 6.1).

In addition to these models, two classic one-dimensional approximation schemes are considered: a set of *algebraic polynomials*

$$f_a(x) = \sum_{i=0}^{n} w_i x^i = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_{n-1} x^{n-1} + w_n x^n \tag{1.14}$$

and a *truncated Fourier series*

$$\begin{aligned} f_a(x) = a_0 &+ a_1 \sin(x) + b_1 \cos(x) + a_2 \sin(2x) + b_2 \cos(2x) + \cdots \\ &+ a_n \sin(nx) + b_n \cos(nx). \end{aligned} \tag{1.15}$$

All but the first approximation scheme here are linear approximations. However, all given models are aimed at creating nonlinear approximating functions. Thus, the adjective *linear* is used because the parameters ($\mathbf{w}$, a_i, b_i, and r_i) that are subjects of learning enter linearly into the approximating functions. In other words, the approximation depends linearly upon weights that are the subjects of learning. This very important property of linear models leads to quadratic (convex) optimization problems with guaranteed global minimums when the L_2 norm is used.

The second major question to be answered is the choice of norm (the distance between the data and the approximating function $f_a(x, \mathbf{w})$). This choice is less important than the choice of form $f_a(x, \mathbf{w})$. If $f_a(x, \mathbf{w})$ is compatible with an underlying function $f(\mathbf{x})$ that produced the training data points, then almost any reasonable measure will lead to an acceptable approximation to $f(\mathbf{x})$. If $f_a(x, \mathbf{w})$ is not compatible with $f(\mathbf{x})$, none of the norms can improve bad approximations to $f(\mathbf{x})$. However, in many practical situations one norm of approximation is naturally preferred over another.

The norm of approximation is a measure of how well a specific approximation $f_a(\mathbf{x})$ of the given form matches the given set of noisy data. Norms are (positive) scalars used as measures of error, length, size, distance, and so on, depending on context. Here a norm usually represents an error. The most common mathematical class of norms in the case of a measured discrete data set is the L_p (Hölder) norm. The L_p norm is a p-norm of an error vector $\mathbf{e}$ given as

$$\|\mathbf{e}\|_p = \|\mathbf{d} - \mathbf{f}_a(\mathbf{x}, \mathbf{w})\|_p = \|\mathbf{d} - \mathbf{o}\|_p = \left(\sum_{i=1}^{P} |d_i - o_i|^p\right)^{1/p}, \tag{1.16}$$

where P indicates the size of the training data set, that is, the number of training data pairs, and $\mathbf{d}$ and $\mathbf{o}$ stand for P-dimensional vectors of desired and actual outputs of the neural network. Note that (1.16) is strictly valid for an $\Re^d \to \Re^1$ mapping, or for an NN with a single output layer neuron. For more OL neurons, a norm would be defined as a proper matrix norm. Assuming that the unknown underlying function $f(\mathbf{x})$ is given on a discrete data set containing P measurements $f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_P)$, the standard L_p norms in use are defined as

$$L_1\colon \quad \|f - f_a\|_1 = \sum_{i}^{P} |f(\mathbf{x}_i) - f_a(\mathbf{x}_i)| \qquad \text{(absolute value)} \tag{1.17}$$

$$L_2\colon \quad \|f - f_a\|_2 = \sum_{i}^{P} (|f(\mathbf{x}_i) - f_a(\mathbf{x}_i)|^2)^{1/2} \qquad \text{(Euclidean norm)} \tag{1.18}$$

$$L_\infty\colon \quad \|f - f_a\|_\infty = \max_i |f(\mathbf{x}_i) - f_a(\mathbf{x}_i)| \qquad \text{(Chebyshev, uniform, or infinity norm)} \tag{1.19}$$

The norms used during an optimization of the models are not only standard L_p norms. Usually rather more complex mathematical structures are applied in the form of cost or error functions that enforce the closeness to training data (most typically measured by an L_2 or L_1 norm) and that measure some other property of an approximating function (e.g., smoothness, model complexity, weights magnitude). These norms (actually variously defined functionals that do not possess the strict mathematical properties required for norms) are typically composed of two parts. The first component is usually a standard L_2 (or L_1) norm, and the second is some penalization term (see table 1.4 and equations (2.26)–(2.28)). Vapnik's ε-insensitive loss function (norm), which is particularly useful for regression problems, is introduced in chapter 2.

The Chebyshev (or uniform) norm is an L_p norm called an infinity norm by virtue of the identity

$$\lim_{p\to\infty} (|x_1|^p + |x_2|^p + \cdots + |x_p|^p)^{1/p} = \max_{1 \le i \le P} |x_i|. \tag{1.20}$$

The choice of the appropriate norm or a measure of the approximation's goodness depends primarily on the data and on the simplicity of approximation and optimization algorithms available. The L_2 norm is the best one for data corrupted with normally distributed (Gaussian) noise. In this case, it is known that the estimated parameters or weights obtained in L_2 norm correspond to the maximum-likelihood estimates. The L_1 norm is much better than the Euclidean norm for data that have outliers because the L_1 norm tends to ignore such data. The Chebyshev norm is ideal in the case of exact data with errors in a uniform distribution.

In many fields, particularly in signal processing and system identification and control, the L_2 or Euclidean norm is almost universally used, for two reasons. First, the assumption about the Gaussian character of the noise in a control systems environment is an acceptable and reasonable one. Second, the L_2 norm is mathematically simple and tractable.

Very often, a measure of goodness or closeness of approximation used during a learning stage does not satisfy (1.17)–(1.19) or other known properties of norms, and hence is not a norm. For example, the most common deviation from a "pure" L_2 norm is the use of the sum of error squares as a standard cost function in NN learning. The sum of error squares is the measure derived from the norm (it is a slightly changed version of a Euclidean norm without square root operation), but its minimization is equivalent to the minimization of the L_2 norm from which it is derived.

Now that the concepts of form (type of function used to approximate data) and norm (measure of closeness of approximation) have been introduced, a natural question is, what is the best approximation? Of course, in order to achieve better approximations, the approximants will generally have to be of higher and higher degree. An increase in degree, usually leads to overfitting, however. This problem is discussed in detail throughout the book.

Given the set S_n of approximating functions (say, polynomials of sixth order, or NNs with six HL neurons, or fuzzy models with six rules, $n = 6$), is there among the elements of S_n (among all possible polynomials of sixth order, or NNs with six HL neurons, or fuzzy models with six rules) one that is closer to given data points $f(\mathbf{x}_i)$, $i = 1, P$, than any other element (function, model, network) of S_n? If there is, it is known as the *best approximation* to $f(\mathbf{x})$.

Note, however, that the best approximation property depends upon the norm applied. Change the norm (the criterion of closeness of approximation) and the best approximation will change. Generally, the best approximation in the L_p norm is not the same as the best approximation in the L_q norm $(p \neq q)$. This is shown in the following one-dimensional continuous example (see also problems 1.8 and 1.9).

Example 1.1 Approximate $y = x^4$ over $[0, 1]$ by a straight line $f_a(x)$ so that

$\int_0^1 (x^4 - f_a(x))^2 \, dx$ is minimum,

$\int_0^1 (x^4 - f_a(x))^2 \, dx + \int_0^1 \left(\frac{d(x^4 - f_a(x))}{dx} \right)^2 dx$ is minimum,

$\max_{0 \le x \le 1} |x^4 - f_a(x)|$ is minimum.

Note that $y = x^4$ is given as a continuous function, not as a set of discrete points. Therefore, it is appropriate here to use norms defined over an interval that apply an integral operator instead of equations (1.17)–(1.19), which comprise a summing operator. Three different best approximations (solutions) corresponding to the given cost functions (norms) are

$f_a(x) = \frac{4x}{5} - \frac{1}{5}$ (solid line in fig. 1.8),

$f_a(x) = \frac{54x}{55} - \frac{16}{55}$ (dashed line in fig. 1.8),

$f_a(x) = x - 0.236$ (dotted line in fig. 1.8). ■[6]

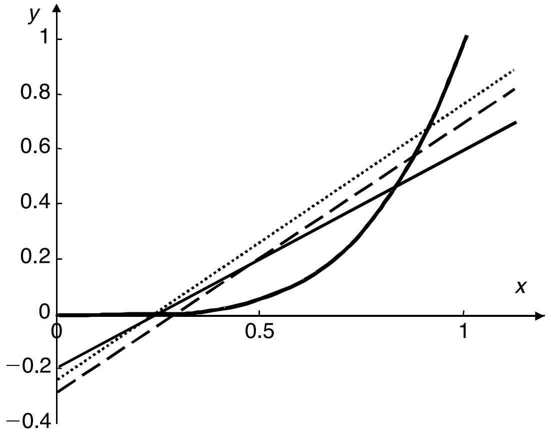


Figure 1.8
Best least square (solid), penalized least square (dashed), and uniform (dotted) linear approximations to x^4 on $[0, 1]$.

The best approximation is unique for all strict norms, that is, for $1 < p < \infty$. Thus, the best approximation with an L_2 norm is unique, but this property is not, for example, shared with L_1 and L_∞ norms. For more on the problem of a norm's uniqueness, see the related problems in the problems section of this chapter.

Before examining the approximation properties of models used in this book, one can consider some basic shortcomings of classical approximation schemes. Historically, there are two standard approximators: algebraic and trigonometric polynomials. The interested reader should consult the vast literature on the theory of approximation to find discussions on the existence and uniqueness of a solution, the best approximation calculation and its asymptotic properties, and the like. Here we start with polynomial approximators of a one-dimensional function given by discrete data. In trying to learn about the underlying dependency between inputs and outputs we are concerned with the error at all points in the range, not only at the sampled training data. Example 1.2 shows the deficiency of polynomial approximations. They behave badly in the proximity of domain boundaries even though they are perfect interpolators of given training data points.

Example 1.2 Approximate function $f(x) = 1/(1 + 25x^2)$ defined over the range $[-1, +1]$ and sampled at 21 equidistant points x_i, $i = 1, 21$, (i.e., $P = 21$) by polynomials of twelfth, sixteenth, and twentieth orders.

For this particular function (see fig. 1.9) it is found that for any point $x \neq x_i$, where $|x| > 0.75$, the error $|f_a(x) - f(x)|$ of the approximation increases without bound as

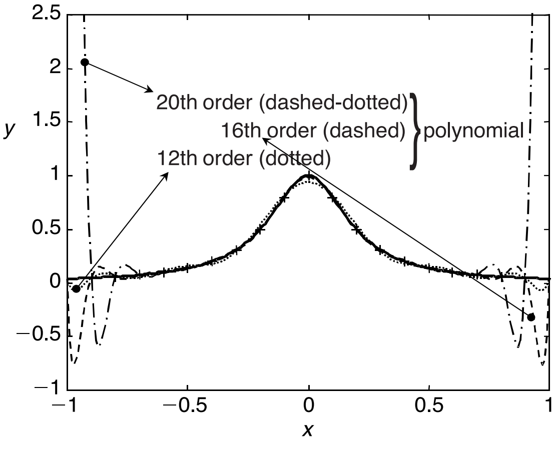


Figure 1.9
Polynomial approximations of a function $1/(1 + 25x^2)$.

the order of the approximating polynomials n increases. This is true even though $f_a(x_i) = f(x_i)$ when the order of polynomial $n = P - 1$, where P is the number of training data. In this example $f_a(x_i) = f(x_i)$ for a twentieth-order polynomial, which means that both L_1 and L_2 norms are equal to zero, wrongly suggesting perfect errorless approximation over the whole domain of input variable x. ■

When considering an error over a whole range, a more satisfactory objective is to make the maximum error as small as possible. This is the *minimax* or Chebyshev type of approximation where the error is defined by (1.19) and the function $f_a(x)$ is chosen so that L_∞ is minimized. It is in this context that the Chebyshev polynomials have found wide application.

There are also many other polynomials, notably a class of orthogonal polynomials, that can be used for approximations. All of them have similar deficiencies in the sense that as the number of training points P increases, the approximation improves near the center of the interval but shows pronounced oscillatory behavior at the ends of the interval.

Another popular classical approximation scheme involves rational functions (the ratio of two polynomials) given as

$$f_a(x, \mathbf{w}, \mathbf{v}) = \frac{\sum\limits_{i=0}^{n} w_i x^i}{\sum\limits_{i=0}^{m} v_i x^i}. \tag{1.21}$$

These functions are much more flexible, but they are nonlinear approximators (with respect to the denominator weights v_i) and learning of the weights v_i is not an easy task. They are of historical significance but are not used as basis functions in this book.

Both the polynomials and trigonometric sums have similar disadvantages in that they cannot take sharp bends followed by relatively flat behavior. Both functions are defined over the whole domain (i.e., they are globally acting activation functions), and they generally vary gently. Such characteristics can be circumvented by increasing the degree of these functions, but this has as a consequence wild behavior of the approximator close to boundaries (see fig. 1.9).

The best candidates for approximating functions that naturally originate from polynomials are the *piecewise polynomial functions*, the most popular being various *spline functions*. The spline functions are special cases of RBF functions; they are derived in chapter 5. They are defined by dividing the domain of the input variable into several intervals by a set of points called *joints* or *knots*. The approximating function is then a polynomial of specified degree between the knots. The approximation is linear for prespecified and fixed knots. However, the whole approximating scheme is nonlinear when the positions of knots are subjects of learning. The learning of knot positions, being a nonlinear optimization problem, is complex and generally not an easy task.

It is interesting to note that in the univariate case (for one-dimensional inputs) algebraic polynomials, trigonometric polynomials, and splines all have the property of providing a unique interpolation on a set of distinct points equal in number to the number of approximation parameters (weights). However, in the multivariate case it is not usually possible to guarantee a unique interpolant for these basis functions, and hence a best approximation is not necessarily unique. Problem 1.11 deals with the uniqueness of a polynomial approximation in a bivariate case. However, it deliberately emphasizes a kind of pathological case. It is important to be aware of possible pitfalls, but there are some very nice results in applying polynomials in support vector machines (see chapter 2) that exploit their properties. At the same time, RBFs uniquely interpolate any set of data on a distinct input set of training data points. This is but one of nice properties of RBF networks (RBF approximation models).

Figure 1.10 shows an interpolation of six ($P = 6$) random data points using a polynomial of fifth order. A perfect interpolation is achieved because the Vandermonde matrix **X** in (1.22) is nonsingular. (These matrices often occur in polynomial approximations, signal processing, and error-correcting codes.) However, this matrix

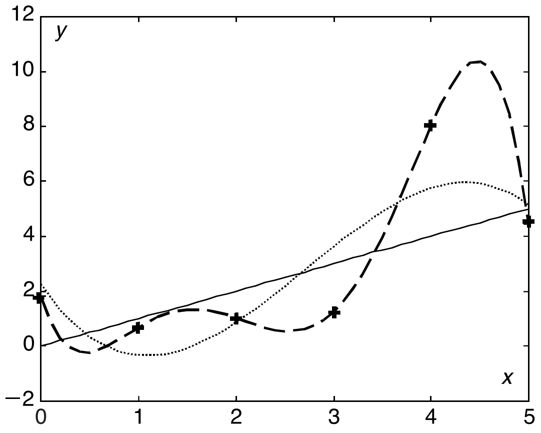


Figure 1.10
Interpolation and approximation polynomials of fifth order (dashed) and third order (dotted) to six highly noise-contaminated data points obtained by sampling the straight line (solid).

is notorious. Even for modest sizes of P, it is very ill conditioned (its determinant is very small), and solutions may be subject to severe numerical difficulties. The interpolation curve (dashed) is a solution of a linear system of equations, and it is a (unique) least-squares solution, as is the approximation curve (dotted). The latter is, however, a third-order polynomial approximation curve. It is also the best approximation in the least-squares sense but no longer passes through the training data points.

An *interpolating solution* results from solving the following system of linear equations:

$$w_0 + w_1x_i^1 + w_2x_i^2 + w_3x_i^3 + w_4x_i^4 + w_5x_i^5 = y_i, \qquad i = 1, 6,$$

which can be expressed in a matrix form as

$$\mathbf{Xw} = \mathbf{y} \Rightarrow \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 & x_2^5 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 & x_3^5 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 & x_4^5 \\ 1 & x_5 & x_5^2 & x_5^3 & x_5^4 & x_5^5 \\ 1 & x_6 & x_6^2 & x_6^3 & x_6^4 & x_6^5 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}. \tag{1.22}$$

In this case, input vector $\mathbf{x} = [0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5]^T$ and output vector $\mathbf{y} = [1.83 \quad 0.69 \quad 1.03 \quad 1.26 \quad 8.03 \quad 4.45]^T$. A nonsingular Vandermonde matrix $\mathbf{X}$ is

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 \\ 1 & 3 & 9 & 27 & 81 & 243 \\ 1 & 4 & 16 & 64 & 256 & 1024 \\ 1 & 5 & 25 & 125 & 625 & 3125 \end{bmatrix}$$

The solution vector $\mathbf{w}$ to (1.22) that ensures an interpolation is

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y} = [-10.84 \quad 18.57 \quad -11.60 \quad 2.99 \quad -0.26 \quad 1.83]^T.$$

An *approximating solution* using a polynomial of third order (shown in fig. 1.10 as a dotted curve) is obtained by solving the following overdetermined system of six equations in four unknowns:

$$w_0 + w_1x_i^1 + w_2x_i^2 + w_3x_i^3 = y_i, \qquad i = 1, 6,$$

or in matrix form,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \\ 1 & 5 & 25 & 125 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1.83 \\ 0.69 \\ 1.03 \\ 1.26 \\ 8.03 \\ 4.45 \end{bmatrix} \tag{1.23}$$

The best solution, a weights vector **w**, that results from an approximation in the least squares sense is obtained as follows:

$$\mathbf{w} = \mathbf{X}^{+}\mathbf{y} = [-5.17 \quad 2.96 \quad -0.36 \quad 2.25]^{T},$$

where $\mathbf{X}^{+}$ denotes the pseudoinversion of a rectangular matrix **X**.

The same set of training data points is interpolated by using *linear splines* and *cubic splines*; the interpolating curves are shown in figure 1.11.

The fifth column in a matrix **X** belonging to linear splines in (1.24) corresponds to the linear spline centered at $x = 4$ (thick dotted spline). Matrix **X** is also called a *design matrix*. The interpolation functions that are obtained by using spline functions as shown in figure 1.11 belong to radial basis functions network models. Chapter 5

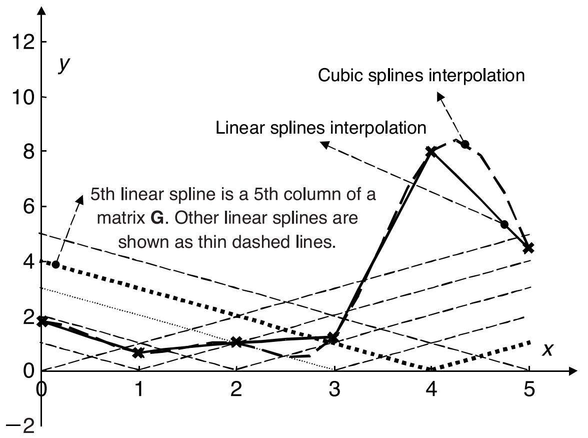


Figure 1.11
Interpolation by linear and cubic splines. Training data set is the same as in figure 1.10. Fifth linear spline corresponds to fifth column of a matrix **X** given in (1.24).

discusses these networks, also known as regularization networks. Chapter 5 also presents the origins of the linear and cubic splines applied here.

Corresponding systems of linear equations for both interpolations in matrix form and solution vectors **w** are as follows.

For linear splines,

$$\mathbf{X}\mathbf{w}_L = \mathbf{y} \Rightarrow \begin{bmatrix} 0 & 1 & 2 & 3 & \mathbf{4} & 5 \\ 1 & 0 & 1 & 2 & \mathbf{3} & 4 \\ 2 & 1 & 0 & 1 & \mathbf{2} & 3 \\ 3 & 2 & 1 & 0 & \mathbf{1} & 2 \\ 4 & 3 & 2 & 1 & \mathbf{0} & 1 \\ 5 & 4 & 3 & 2 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1.83 \\ 0.69 \\ 1.03 \\ 1.26 \\ 8.03 \\ 4.45 \end{bmatrix}. \tag{1.24}$$

For cubic splines,

$$\mathbf{X}\mathbf{w}_C = \mathbf{y} \Rightarrow \begin{bmatrix} 0 & 1 & 8 & 27 & 64 & 125 \\ 1 & 0 & 1 & 8 & 27 & 64 \\ 8 & 1 & 0 & 1 & 8 & 27 \\ 27 & 8 & 1 & 0 & 1 & 8 \\ 64 & 27 & 8 & 1 & 0 & 1 \\ 125 & 64 & 27 & 8 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1.83 \\ 0.69 \\ 1.03 \\ 1.26 \\ 8.03 \\ 4.45 \end{bmatrix}. \tag{1.25}$$

The solution vectors are

$$\mathbf{w}_L = [0.06 \quad 0.74 \quad -0.057 \quad 3.27 \quad -5.17 \quad 2.4163]^T.$$

$$\mathbf{w}_C = [0.59 \quad -1.66 \quad 2.55 \quad -4.43 \quad 3.39 \quad -0.9165]^T.$$

As long as the dimensionality of input vectors is not too high, many classical approximation tools may be appropriate for modeling training data points. However, in modern soft computing, the dimensionality of input vectors is very high; it may go to dozens of thousands. In such high-dimensional spaces, data are sparse, and modeling underlying dependencies is a formidable task. NNs, SVMs, and FL models are appropriate tools for such tasks. Their theoretical interpolation and approximation capacities are briefly discussed here. Chapters 2–6 treat these issues in more detail.

Both NNs and FL models are *universal approximators* in the sense that they can approximate any function to any degree of accuracy provided that there are enough hidden layer neurons or rules. The same can also be stated for SVMs. Without any doubt, such powerful approximating faculties are the foundation of, and theoretical justification for, the wide application of NNs and FL models.

Following are the classic Weierstrass theorem for the approximation by polynomials; the Cybenko-Hornik-Funahashi theorem (Cybenko 1989; Hornik, Stinchcombe, and White 1989; Funahashi 1989), which states identical abilities of sigmoidal functions; and the theorem for universal approximation properties using the Gaussian (radial basis) activation functions.

CLASSICAL WEIERSTRASS THEOREM The set $P_{[a,b]}$ of all polynomials

$$p(x) = \sum_{j=0}^{n} w_j x^j \tag{1.26}$$

is dense in $C[a,b]$. In other words, given $f \in C[a,b]$ and $\varepsilon > 0$, there is a polynomial p for which

$$|p(x) - f(x)| < \varepsilon, \quad \text{for all } x \in [a,b].$$

CYBENKO-HORNIK-FUNAHASHI THEOREM Let σ be any sigmoidal function and I_d the d-dimensional cube $[0,1]^d$. Then the finite sum of the form

$$f_a(\mathbf{x}) = \sum_{j=1}^{n} w_j \sigma_j(\mathbf{v}_j^T \mathbf{x} + b_j) \tag{1.27}$$

is dense in $C[I_d]$. In other words, given $f \in C[I_d]$ and $\varepsilon > 0$, there is a sum $f_a(\mathbf{x})$ for which

$$|f_a(\mathbf{x}) - f(\mathbf{x})| < \varepsilon, \quad \text{for all } \mathbf{x} \in I_d,$$

where w_j, $\mathbf{v}$, and b_j represent OL weights, HL weights, and bias weights of the hidden layer, respectively.

THEOREM FOR THE DENSITY OF GAUSSIAN FUNCTIONS Let G be a Gaussian function and I_d the d-dimensional cube $[0,1]^d$. Then the finite sum of the form

$$f_a(\mathbf{x}) = \sum_{j=1}^{n} w_j G(\|\mathbf{x} - \mathbf{c}_j\|) \tag{1.28}$$

is dense in $C[I_d]$. In other words, given $f \in C[I_d]$ and $\varepsilon > 0$, there is a sum $f_a(\mathbf{x})$ for which

$$|f_a(\mathbf{x}) - f(\mathbf{x})| < \varepsilon, \quad \text{for all } \mathbf{x} \in I_d,$$

where w_i and $\mathbf{c}_i$ represent OL weights and centers of HL multivariate Gaussian functions, respectively.[7]

The same results of universal approximation properties exist for fuzzy models, too. Results of this type can also be stated for many other different functions (notably trigonometric polynomials and various kernel functions). They are very common in approximation theory and hold under very weak assumptions. Density in the space of continuous functions is a necessary condition that every approximation scheme should satisfy.

However, the types of problems in this book are slightly different. In diverse tasks where NNs and SVMs are successfully applied, the problem is usually not one of approximating some continuous univariate $f(x)$ or multivariate $f(\mathbf{x})$ function over some interval. The typical engineering problem involves the interpolation or approximation of sets of P sparse and noisy training data points.

The NNs or SVMs will have to model the mapping of a finite training data set of P n-dimensional input training patterns $\mathbf{x}$ to the corresponding P m-dimensional output (desired or target) patterns $\mathbf{y}$. (These $\mathbf{y}$ are denoted by $\mathbf{d}$ during the training, where $\mathbf{d}$ stands for *desired.*) In other words, these models should model the dependency (or the underlying function, i.e., the hypersurface) $f\colon \Re^n \to \Re^m$. In the case of classification, the problem is to find the discriminant hyperfunctions that separate m classes in an n-dimensional space. The learning (adaptation, training phase) of our models corresponds to the linear or nonlinear optimization of a fitting procedure based on knowledge of the training data pairs. This is a task of hypersurface fitting in the generally high-dimensional space $\Re^n \otimes \Re^m$. RBF networks have a nice property that they can interpolate any set of P data points. The same is also true for fuzzy logic models, support vector machines, or multilayer perceptrons, and this powerful property is the basis for the existence of these novel modeling tools.

To *interpolate* the data means that the interpolating function $f_a(\mathbf{x}_p)$ must pass through each particular point in $\Re^n \otimes \Re^m$ space. Thus, an interpolating problem is stated as follows:

Given is a set of P measured (observed) data: $X = \{\mathbf{x}_p, \mathbf{d}_p, p = 1, \dots, P\}$ consisting of the input pattern vectors $\mathbf{x} \in \Re^n$ and output desired responses or targets $\mathbf{d} \in \Re^m$. An interpolating function is such that

$$f_a(\mathbf{x}_p) = \mathbf{d}_p, \qquad p = 1, \dots, P. \tag{1.29}$$

Note that an interpolating function is required to pass through each desired point $\mathbf{d}_p$. Thus, the cost or error function (norm) E that measures the quality of modeling (at this point, we use the sum of error squares) in the case of interpolation must be equal to zero

$$E = \sum_{p=1}^{P} \mathbf{e}_p^2 = \sum_{p=1}^{P} [\mathbf{d}_p - f_a(\mathbf{x}_p)]^2 = 0. \tag{1.30}$$

Strictly speaking, an interpolating function $f_a(\mathbf{x}_p)$ and therefore the error function E, are parameterized by approximation coefficients here called weights, and a more proper notation for these two functions would be $f_a(\mathbf{x}, \mathbf{w}, \mathbf{v})$ and $E(\mathbf{x}, \mathbf{w}, \mathbf{v})$. Weights $\mathbf{w}$ and $\mathbf{v}$ are typically a network's output layer weights and hidden layer weights, respectively. Writing this dependency explicitly stresses the fact that the weights will be subjected to an optimization procedure that should result in a good f_a and a small E. (Generally, weights are organized in matrices $\mathbf{W}$ and $\mathbf{V}$.)

Note that in order to interpolate data set X, the RBF network, for example, should have exactly P neurons in a hidden layer. Work with NNs typically involves sets of hundreds of thousands of patterns (measurements), which means that the size of such an interpolating network would have to be large. The numerical processing of matrices of such high order is very intractable. There is another important reason why the idea of data interpolation is usually not a good one. Real data are corrupted by noise, and interpolation of noisy data leads to the problem of overfitting. What we basically want NNs to do is to model the underlying function (dependency) and to filter out the noise contained in the training data. There are many different techniques for doing this, and some of these approaches are presented later. Chapter 2 is devoted to this problem of matching the complexity of a training data set to the capacity of an approximating model. The approach presented there also results in models with fewer processing units (HL neurons) than there are training patterns.

The number of neurons in a hidden layer and the parameters that define the shapes of these HL activation (basis) functions are the most important design parameters with respect to the approximation abilities of neural networks. Recall that both the number of input components (features) and the number of output neurons are in general determined by the very nature of the problem. At the same time, the number of HL neurons, which primarily determines the real representation power of a neural network and its generalization capacity, is a free parameter. In the case of general nonlinear regression performed by an NN, the main task is to model the underlying function between the given inputs and outputs and to filter out the disturbances contained in the noisy training data set. Similar statements can be made for pattern recognition (classification) problems. In the SVM field, one can say that the model complexity (capacity) should match the data complexity during training. Model capacity is most often controlled by the number of neurons in the hidden layer. In changing the number of HL nodes, two extreme solutions should be avoided: filtering out the underlying function (not enough HL neurons) and modeling the noise or overfitting the data (too many HL neurons). Therefore, there is a need to comment on appropriate measures of model quality.

In the theory of learning from data, the problems of measuring the model's performance are solved by using different approaches and inductive principles. Applying some simple norm, for instance, any L_p norm, is usually not good enough. Perfect performance on training data does not guarantee good performance on previously unseen inputs (see example 1.3). Various techniques aimed at resolving the trade-off between performance on training data and performance on previously unseen data are in use today. The concept of simultaneous minimization of a bias and a variance, known as the *bias-variance dilemma*, originated from the field of mathematical statistics (see chapter 4). Girosi analyzed the concepts of an approximation error and an estimation error (see section 2.3). Finally, in the field of SVMs one applies the structural risk minimization principle, which controls both the empirical risk and a confidence interval at the same time. In all three approaches, one tries to keep both components of an overall error or risk as low as possible. All these measures of the approximation performance of a model are similar in spirit but originate from different inductive principles, and they cannot be made equivalent.

One more classical statistical tool for resolving the trade-off between the performance on training data and the complexity of a model is the cross-validation technique. The basic idea of the cross-validation is founded on the fact that good results on the training data do not ensure good generalization capability. Generalization refers to the capacity of a neural network to give correct answers on previously unseen data. This set of previously unseen data is called a *test set* or *validation set* of patterns. The standard procedure to obtain this particular data set is to take out a part (say, one quarter) of all measured data, which will not be used during training but in the validation or test phase only. The higher the noise level in the data and the more complex the underlying function to be modeled, the larger the test set should be. Thus, in the cross-validation procedure the performance of the network is measured on the test or validation data set, ensuring the good generalization property of a neural network.

Example 1.3 demonstrates some basic phenomena during modeling of a noisy data set. It clearly shows why the idea of interpolation is not very sound.

Example 1.3 The dependency (plant or system to be identified) between two variables is given by $y = x + \sin(2x)$. Interpolate and approximate these data by an RBF network having Gaussian basis functions by using a highly corrupted training data set (25% Gaussian noise with zero mean) containing 36 measured patterns (x, d).

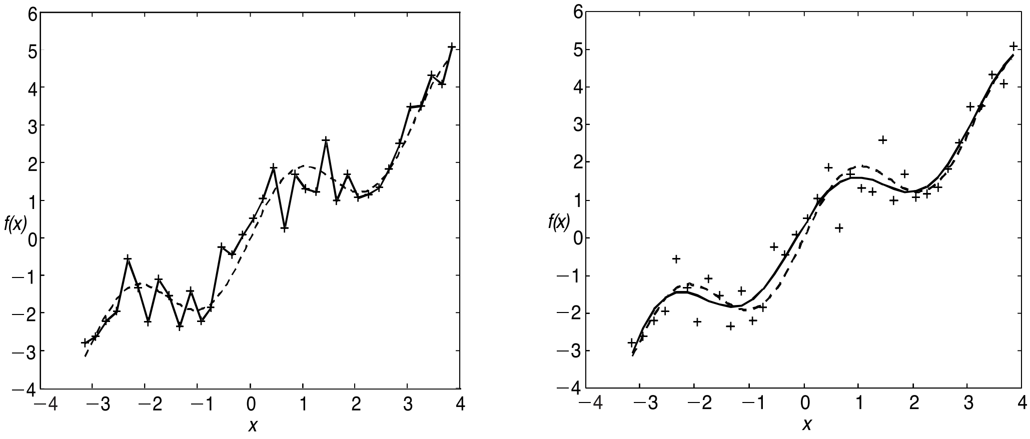


Figure 1.12
Modeling of noisy data by an RBF (regularization) network with Gaussian basis functions. *Left*, interpolation and overfitting of noisy data (36 hidden layer neurons). *Right*, approximation and smoothing of noisy data (8 hidden layer neurons). Underlying function (dashed) is $y = x + \sin(2x)$. Number of training patterns (crosses) $P = 36$.

Figure 1.12 shows the interpolation and the approximation solutions. Clearly, during the optimization of the network's size, one of the smoothing parameters is the number of HL neurons that should be small enough to filter out the noise and large enough to model the underlying function. This simple example of a one-dimensional mapping may serve as a good illustration of overfitting. It is clear that perfect performance on a training data set does not guarantee a good model (see left graph, where the interpolating function passes through the training data). The same phenomena will be observed while fitting multivariate hypersurfaces. ■

In order to avoid overfitting, one must relax an interpolation requirement like (1.29) while fitting noisy data and instead do an *approximation* of the training data set that can be expressed as

$$f_a(\mathbf{x}_p) \approx \mathbf{d}_p, \qquad p = 1, \dots, P. \tag{1.31}$$

In the case of approximation, the error or cost function is not required as $E = 0$. The requirement is only that the error function

$$E = \sum_{p=1}^{P} \mathbf{e}_p^2 = \sum_{p=1}^{P} [\mathbf{d}_p - f_a(\mathbf{x}_p)]^2$$

be small and the noise be filtered out as much as possible. Thus, approximation is related to interpolation but with the relaxed condition that an approximant $f_a(\mathbf{x}_p)$ does not have to go through all the training data points. Instead, it should approach the data set points as closely as possible, trying to minimize some measure of the error or disagreement between the approximated point $f_a(\mathbf{x}_p)$ and the desired value $\mathbf{d}$. These two concepts of curve fitting are readily seen in figure 1.12. In real technical applications the data set is usually colored or polluted with noise, and it is better to use approximation because it is a kind of smooth fit of noisy data. If one forces an approximating function to pass each noisy data point, one will easily get a model with high variance and poor generalization.

The interpolation and the approximation in example 1.3 were done by an RBF neural network as given in (1.28), with 36 and 8 neurons in the hidden layer, respectively. Gaussian functions (HL activation functions here) were placed symmetrically along the x-axis, each having the same standard deviation σ equal to double the distance between two adjacent centers. With such a choice of σ, a nice overlapping of the basis (activation) functions was obtained. Note that both parameters (centers c_i and standard deviations σ_i) of Gaussian bells were fixed during the calculation of the best output layer weights w_i. (In terms of NNs and FL models, the hidden layer weights and parameters that define positions and shapes of membership functions, respectively, were fixed or frozen during the fitting procedure.) In this way, such learning was the problem of linear approximation because the parameters w_i enter linearly into the expression for the approximating function $f_a(\mathbf{x})$. In other words, approximation error $e(\mathbf{w})$ depends linearly upon the parameters (here the OL weights w_i). Note that in this example the approximation function $f_a(\mathbf{x})$ represents physically the output from the single OL neuron, and the approximation error for a pth data pair (x_p, d_p) can be written as

$$e_p = e_p(\mathbf{w}) = d_p - f_a(x_p, \mathbf{w}) = d_p - o(x_p, \mathbf{w}). \tag{1.32}$$

Generally, x will be the $(n+1)$-dimensional vector $\mathbf{x}$. When approximation error e depends linearly upon the weights, the error function $E(\mathbf{w})$, defined as the sum of error squares, is a hyperparaboloidal bowl with a guaranteed single (global) minimum. The weights vector $\mathbf{w}^*$, which gives the minimal point $E_{\min} = E(\mathbf{w}^*)$, is the required solution, and in this case the approximating function $f_a(\mathbf{x})$ from (1.28) or (1.33) has a property of best approximation. Note that despite the fact that this approximation problem is linear, the approximating function $f_a(\mathbf{x})$ is nonlinear, resulting from the summation of weighted nonlinear basis function φ_i.

A variety of basis functions for approximation are available for use in NNs or FL models. (In the NN area, basis functions are typically called activation functions, and in the field of FL models, the most common names are membership functions, possibility distributions, attributes, fuzzy subsets, or degree-of-belonging functions.) The basic linear approximation scheme, as given by (1.12)–(1.15), can be rewritten as

$$f_a(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi_i(\mathbf{x}), \tag{1.33}$$

where N is the number of HL neurons, and $\mathbf{x}$ is an n-dimensional input vector. Equation (1.33) represents an $\Re^n \to \Re^1$ mapping, and $f_a(\mathbf{x})$ is a hypersurface in an $(n+1)$-dimensional space.

In the learning stage of a linear parameters model the weights w_i are calculated knowing training patterns $\{\mathbf{x}_i, d_i\}$, $i = 1, P$, and equation (1.33) is rewritten in the following matrix form for learning purposes:

$$\mathbf{d} = \mathbf{X}\mathbf{w} \Rightarrow \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{bmatrix} = \begin{bmatrix} \varphi_1(\mathbf{x}_1) & \varphi_2(\mathbf{x}_1) & \cdots & \varphi_N(\mathbf{x}_1) \\ \varphi_1(\mathbf{x}_2) & \varphi_2(\mathbf{x}_2) & \cdots & \varphi_N(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_1(\mathbf{x}_P) & \varphi_2(\mathbf{x}_P) & \cdots & \varphi_N(\mathbf{x}_P) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}, \tag{1.34}$$

where P is the number of training data pairs, and N is the number of neurons. Typically, $P > N$, meaning that $\mathbf{X}$ is a rectangular matrix and the solution vector $\mathbf{w}$ substituted in (1.33) produces an approximating hypersurface. When $P = N$, matrix $\mathbf{X}$ is square and $f_a(\mathbf{x})$ is an interpolating function passing through each training data point. It is assumed that none of the training data points coincide, i.e., $\mathbf{x}_i \neq \mathbf{x}_j$, $i = 1, P$, $j = 1, P$, $i \neq j$. In this case, and when $P = N$, a design matrix $\mathbf{X}$ is nonsingular.

The solution weights vector $\mathbf{w}$ is obtained from

$$\mathbf{w} = \mathbf{X}^+\mathbf{d}, \tag{1.35}$$

where $\mathbf{X}^+$ denotes pseudoinversion of a design matrix $\mathbf{X}$. The solution (1.35) is the least-squares solution. For $P = N$, $\mathbf{X}^+ = \mathbf{X}^{-1}$. Elements of a design matrix $\mathbf{X}$ are the values (scalars) of a basis function $\varphi_i(\mathbf{x})$ evaluated at the measured values $\mathbf{x}_i$ of the independent variable. The measured values of the dependent variable y, i.e., an unknown function $f(\mathbf{x})$, are the elements of a desired vector $\mathbf{d}$. Note that for an $\Re^n \to \Re^1$ mapping, a design matrix $\mathbf{X}$ is always a $(P \times N)$ array, independently of the dimensionality n of an input vector $\mathbf{x}$. When $P > N$, the system of linear equations (1.33) has more equations than unknowns (it is *overdetermined*). Equation (1.23) is an example of such an overdetermined system. An important and widely used method for solving overdetermined linear equation systems is the *method of least squares* (see a solution to (1.23) and (1.34)). Its application leads to relatively simple computations, and in many applications it can be motivated by statistical arguments.

Unlike the previous linear approximations, the one given in (1.27) represents a nonlinear multivariate approximation (now $\mathbf{x}$ is a vector and not a scalar):

$$f_a(\mathbf{x}) = \sum_{i=1}^{n} w_i \sigma_i(\mathbf{v}_i^T \mathbf{x} + b_i), \tag{1.36}$$

where σ_i typically denotes sigmoidal, (S-shaped) functions. (Note that biases b_i can be substituted into the corresponding weights vector $\mathbf{v}_i$ as the last or the first entries and are not necessarily expressed separately. Thus, bias b is meant whenever weights vector $\mathbf{v}$ is used). As before, $f_a(\mathbf{x})$ is a nonlinear function, and its characteristic as a nonlinear approximation results from the fact that $f_a(\mathbf{x})$ is no longer the weighted sum of *fixed* basis functions. The positions and the shapes of basis functions σ_i, defined as HL weights vectors $\mathbf{v}_i$ (and biases b_i), are also the subjects of the optimization procedure. The approximating function $f_a(\mathbf{x})$ and the error function E depend now on two sets of weights: linearly upon the OL weights vector $\mathbf{w}$ and nonlinearly upon the HL weights matrix $\mathbf{V}$, where the rows of $\mathbf{V}$ are weights vectors $\mathbf{v}_i$. If one wants to stress this fact, one may write these dependencies explicitly $f_a(\mathbf{x}, \mathbf{w}, \mathbf{V})$ and $E(\mathbf{w}, \mathbf{V})$. Now, the problem of finding the best set of weights is a nonlinear optimization problem, which is much more complex than the linear one. Basically, the optimization, or searching for the weights that result in the smallest error function $E(\mathbf{w}, \mathbf{V})$, will now be a lengthy iterative procedure that does not guarantee finding the global minimum. This problem is discussed in section 1.3.2 to show the need for, and the origins of, nonlinear optimization. Chapter 8 is devoted to the methods of nonlinear optimization, and these questions are discussed in detail there.

1.3.2 Nonlinear Error Surface and Optimization

Most of the complex, very sophisticated art of learning from data is the art of optimization. It is the second stage in building soft computing models, after decisions have been made about what form—approximating function, model, network type, or machine—to use. In this second stage, one first decides what is to be optimized, i.e. what norm should be used. There are many possible cost or error (risk) functionals that can be applied (see (2.26)–(2.28)). Then optimization aimed at finding the best weights to minimize the chosen norm or function can begin.

The previous section is devoted mostly to the problem of representation of our models. Here we present the origin of, and need for, a classic nonlinear optimization that is a basic learning tool. Nonlinear optimization is not the only tool currently used for training (learning, adapting, adjusting, or tuning) parameters of soft computing models. Several versions of massive search techniques are also in use, the most popular being genetic algorithms and evolutionary computing. But nonlinear optimization is still an important tool. From the material in this section the reader can understand why it was needed in the field of learning from data and how it came to be adopted for this purpose. Here a classic gradient algorithm is introduced without technicalities or detailed analysis. Chapter 8 is devoted to various nonlinear optimization techniques and discusses a few relevant algorithms in detail.

There are two, in general high-dimensional, spaces analyzed in the representational and learning parts of a model. Broadly speaking, the representational problem analyzes differences between two hypersurfaces in a $(\mathbf{x}, y)$ hyperspace, one being the approximated unknown function $f(\mathbf{x})$ given by sampled data pairs, and the other the approximating function $f_a(\mathbf{x})$. Both $f(\mathbf{x})$ and $f_a(\mathbf{x})$ lie over an n-dimensional space of input variable $\mathbf{x}$. During learning phase, however, it is more important to analyze an error hypersurface $E(\mathbf{w})$ that, unlike $f_a(\mathbf{x})$, lies over the weight space. Specifically, we follow how $E(\mathbf{w})$ changes (typically, how it decreases) with a change of weights vector $\mathbf{w}$. Both representational and learning space are introduced in example 1.4.

Example 1.4 The functional dependency between two variables is given by $y = 2x$. A training data set contains 21 measured patterns (x, d) sampled without noise. Approximate these data pairs by a linear model $y_a = w_1 x$ and show three different approximations for $w_1 = 0, 2$, and 4 as well as the dependency $E(w_1)$ graphically.

This example is a very simple one-dimensional learning problem that allows visualization of both a modeled function $y(x)$ and a cost function $E(w_1)$. Here $E(w_1)$ is derived from an L_2 norm. It is a sum of error squares. (Note that the graphical presentation of $E(\mathbf{w})$ would not have been possible with a simple quadratic function $y(x) = w_0 + w_1 x + w_2 x^2$ for approximation. $E(\mathbf{w})$ in this case would be a hypersurface lying over a three-dimensional weight space, i.e., it would have been a hypersurface in a four-dimensional space.) The right graph in figure 1.13, showing functional dependency $E(w_1)$, is relevant for a learning phase. All learning is about finding the optimal weight $w_1{}^*$ where the minimum[8] of a function $E(w_1)$ occurs. Even in this simple one-dimensional problem, the character of a quadratic curve $E(w)$ is the same for all linear in parameters models. Hence, this low-dimensional

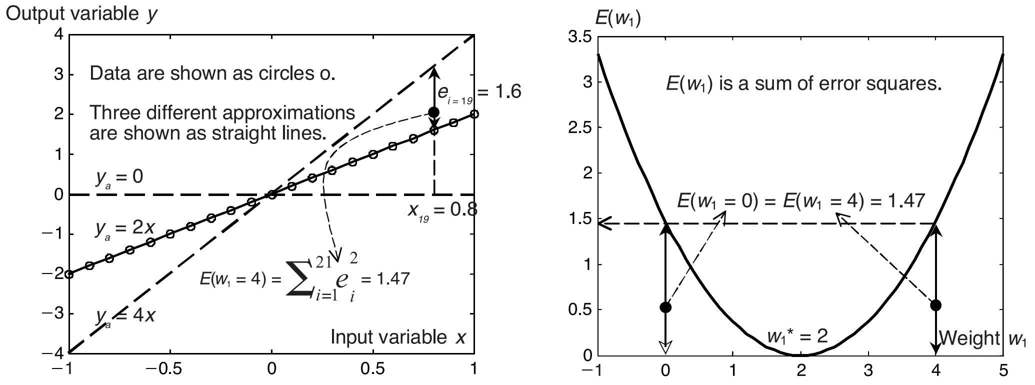


Figure 1.13
Modeling 21 data points obtained by sampling a straight line $y = 2x$ without noise. Three models are shown: a perfect interpolant when $w_1 = 2$, and two (bad) approximating lines with $w_1 = 0$ and $w_1 = 4$ that have the same sum of error squares. Number of training patterns $P = 21$.

example is an appropriate representative of all sum of error squares cost functions $E(w_1) = \sum_{i=1}^{P} e_i^2$. ■

$E(w_1, w_2)$ is a paraboloidal bowl when there are two weights to be learned and a paraboloidal hyperbowl for more than two weights. (See equations (3.45)–(3.48) of a quadratic hyperbowl that are obtained for a general linear neuron with n-dimensional input vector.) However, in all three cases, that is, for $n = 1, 2$ and $n > 2$, an important and desirable fact related to the learning task is that there is a single guaranteed global minimum $E_{\min}(\mathbf{w})$. Therefore, there is no risk of ending the learning in some local minimum, which is always a suboptimal solution.

In example 1.4, there is an interpolation for $w_1 = 2$, and the minimum $E(w_1 = 2) = 0$. It is always like that for all interpolating hypersurfaces $f_a(\mathbf{x})$. However, as already mentioned, the goal is not to interpolate data points. Thus, in the case of an approximating hypersurface $f_a(\mathbf{x})$ this minimal error $E_{\min}(\mathbf{w}) > 0$. Usually one is more interested in finding an optimal $\mathbf{w}^*$ that produces a minimum $E_{\min} = E(\mathbf{w}^*)$ than in knowing the exact value of this minimum.

Unfortunately, genuine soft models are nonlinear approximators in the sense that an error function (a norm or measure of model goodness) depends nonlinearly upon weights that are the subjects of learning. Thus, the error hypersurface is no longer a convex function, and a search for the best set of parameters (weights) that will ensure the best performance of the model is now a much harder and uncertain task than the search for a quadratic (convex) error function like the one in the right graph of figure 1.13.

Example 1.5 introduces a nonlinear nonconvex error surface. Again, for the sake of visualization, the example is low-dimensional. In fact, there are two weights only. It is clear that an error surface $E(w_1, w_2)$ depending on two weights is the last one that can be seen. No others of higher order can be visualized.

Example 1.5 Find a Fourier series model of the underlying functional dependency $y = 2.5\sin(1.5x)$. That the function is a sine is known, but its frequency and amplitude are unknown. Hence, using a training data set $\{\mathbf{x}, d\}$, system can be modeled with an NN model consisting of a single HL neuron (with a sine as an activation function) and a single linear OL neuron as given in figure 1.14.

Note that the NN shown in figure 1.14 is actually a graphical representation of a standard sine function $y = w_2\sin(w_1x)$. This figure could also be seen as a truncated Fourier series with a single term only.

There is a very important difference between classic Fourier series modeling and the NN modeling here, even when the activation functions are trigonometric. When sine and cosine functions are applied as basis functions in NN models, the goal is to learn both frequencies and amplitudes. Unlike in this nonlinear learning task, in classical Fourier series modeling one seeks to calculate amplitudes only. The frequencies are preselected as integer multiples of some user-selected base frequency. Therefore, because the frequencies are known, classical Fourier series learning is a linear problem.

The problem in this example is complex primarily because the error surface is nonconvex (see fig. 1.15). This is also a nice example of why and how the concepts of function, model, network, or machine are equivalent. A function $y = w_2\sin(w_1x)$ is shown as a network in figure 1.14, which is actually a model of this function. At the same time, this artifact is a machine that, as all machines do, processes (transforms) a

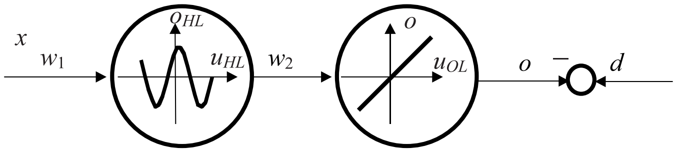


Figure 1.14
Neural network for modeling a data set obtained by sampling a function $y = 2.5\sin(1.5x)$ without noise. Amplitude $A = 2.5$ and frequency $\omega = 1.5$ are unknown to the model. The weights w_1 and w_2 that represent these two parameters should be learned from a training data set.

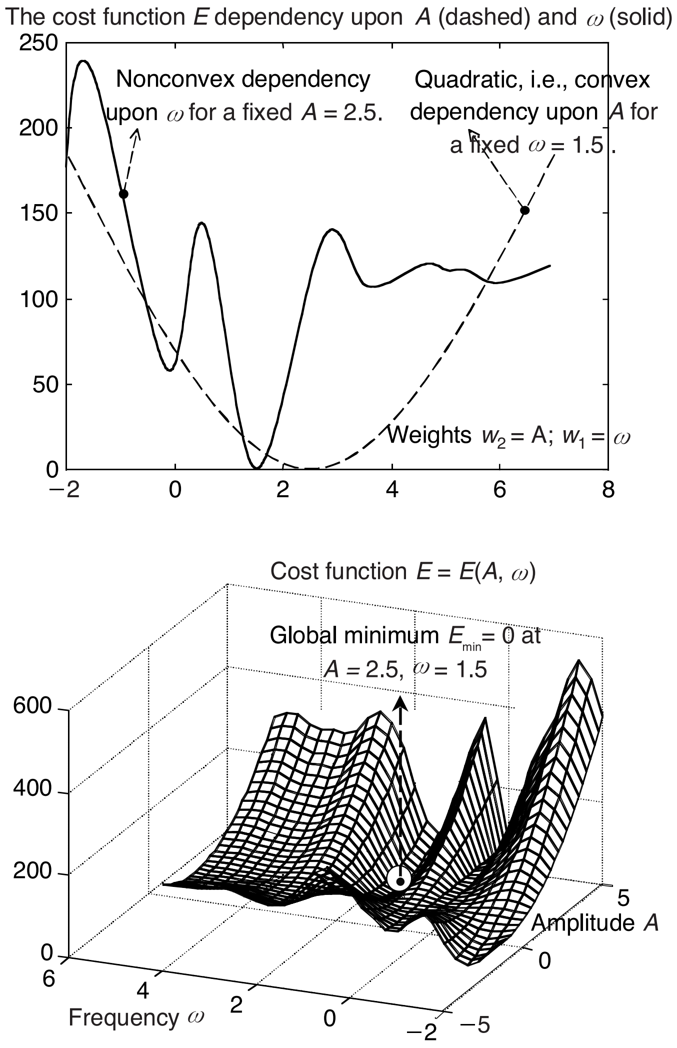


Figure 1.15
Dependence of an error function $E(w_1, w_2)$ upon weights while learning from training data sampled from a function $y = 2.5\ \sin(1.5x)$.

given input into some desirable product. Here, a given input is x, and a product is an output of the network o that "equals" (models) an underlying function y for correct values of the weights w_1 and w_2. Here, after a successful learning stage, the weights have very definite meanings: w_1 is a frequency ω, and w_2 corresponds to amplitude A. In this particular case, they are not just numbers.

Now, the error e can readily be expressed in terms of the weights as

$$e(\mathbf{w}) = d - o = d - w_2 \sin(w_1 x). \tag{1.37}$$

There is an obvious linear dependence upon weight w_2 and a nonlinear relation with respect to weight w_1. This nonlinear relation comes from the fact that error e "sees" the weight w_1 through the nonlinear sine function. The dependence of the cost function $E(w_1, w_2) = \frac{1}{2}\sum_{i=1}^{P} e_i^2$ on the weights is shown in figure 1.15. Note that an error function $E(w_1, w_2)$ is not an explicit function of input variable x. $E(w_1, w_2)$ is a scalar value calculated for given weights w_1 and w_2 over all training data points, that is, for all values of input variable x.

Note in figure 1.15 a detail relevant to learning: the error surface is no longer a convex function and with a standard gradient method for optimization, the training outcome is very uncertain. This means that besides ending in a global minimum the learning can get stuck at some local minima. In the top graph of figure 1.15, there are five local minima and one global minimum for a fixed value of amplitude $A = 2.5$, and a learning outcome is highly uncertain. Thus, even for known amplitude, learning of unknown frequency from a given training data set may have a very unsatisfactory outcome. Note that in a general high-dimensional ($N > 2$) case, $E(\mathbf{w})$ is a hilly hypersurface that cannot be visualized. There are many valleys (local minima), and it is difficult to control the optimization process. ■

One of the first, simplest, and most popular methods for finding the optimal weights vector $\mathbf{w}^*$ where either the global or local minimum of an error function $E(\mathbf{w})$ occurs is an iterative method based on the principle of going downhill to the lowest point of an error surface. This is the idea of the *method of steepest descent*, or *gradient method*. This basic method is introduced after example 1.6, which sheds more light on the origins of the nonlinear characteristics of an error cost function.

However, in considering the gradient-based learning algorithm, one should keep in mind the weakness of going downhill to find the lowest point of the error surface. Unless by chance one starts on the slope over the global minimum, one is unlikely to find the lowest point of a given hypersurface. All that can be done in the general case with plenty of local minima is to start at a number of random (or somehow well-chosen) initial places, then go downhill until there is no lower place to go, each time finding a local minimum. Then, from all the found local minima, one selects the lowest and takes the corresponding weights vector $\mathbf{w}$ as the best one, knowing that better local minima or the global minimum may have been missed.

Example 1.6 Consider a simple neural network having one neuron with a bipolar sigmoidal activation function as shown in figure 1.16. The activation function is

$$o = \frac{2}{1 + e^{-(w_1 x + w_2)}} - 1. \tag{1.38}$$

Assume the following learning task: using a training data set $\{\mathbf{x}, d\}$, learn the weights so that the network models the underlying unknown bipolar sigmoidal function

$$y = \frac{2}{1 + e^{-(ax + b)}} - 1. \tag{1.39}$$

Note that (1.38) is a standard representative of S-shaped functions σ_i given in (1.36). The solution is clear because the underlying function between the input x and the output y is known. However, for this network, the underlying function (1.39) is unknown, and the optimal weights of the neuron $w_{1\text{opt}} = a$, and $w_{2\text{opt}} = b$ should be found by using the training data set.

At this point, however, we are more interested in an error function whose minimum should evidently be at the point (a, b) in the weights' plane. Again use as an error function the sum of error squares

$$E(w_1, w_2) = \sum_{p=1}^{P} e_p^2 = \sum_{p=1}^{P} \left[d_p - \left(\frac{2}{1 + e^{-(w_1 x_p + w_2)}} - 1 \right) \right]^2, \tag{1.40}$$

where P is the number of patterns or data pairs used in training. As in (1.37), an error at some training data pair e_p is nonlinear. Here, it is nonlinear in terms of both unknown weights.

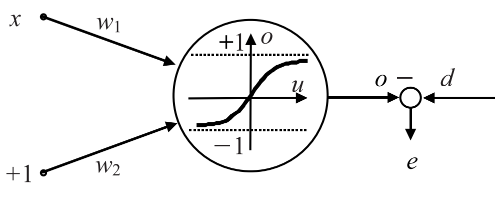


Figure 1.16
Simple neural network with a single neuron.

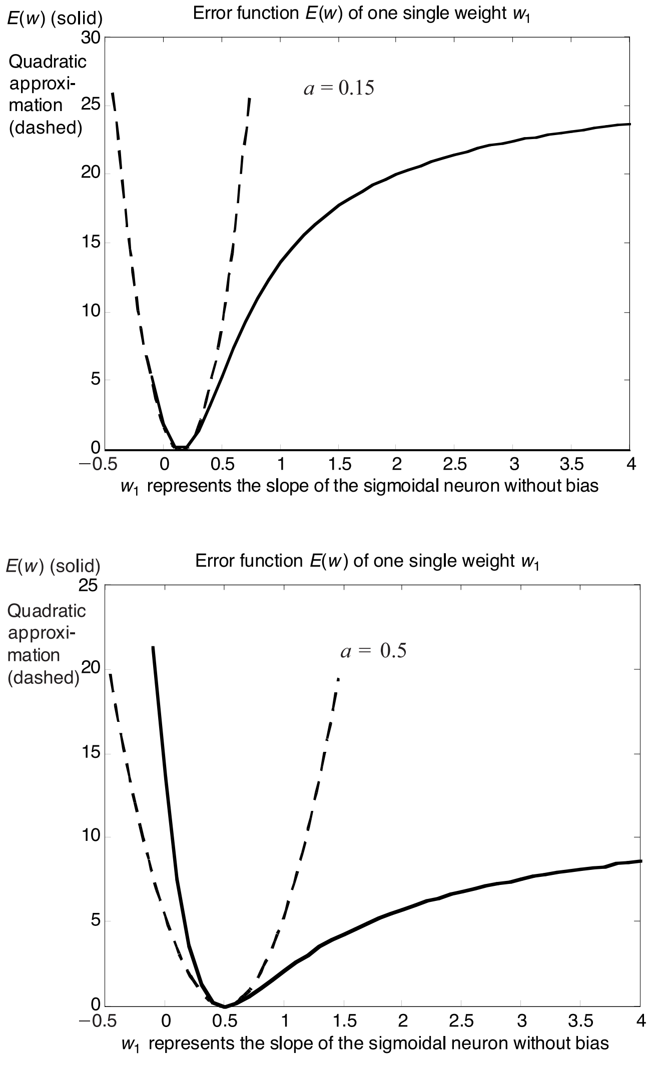
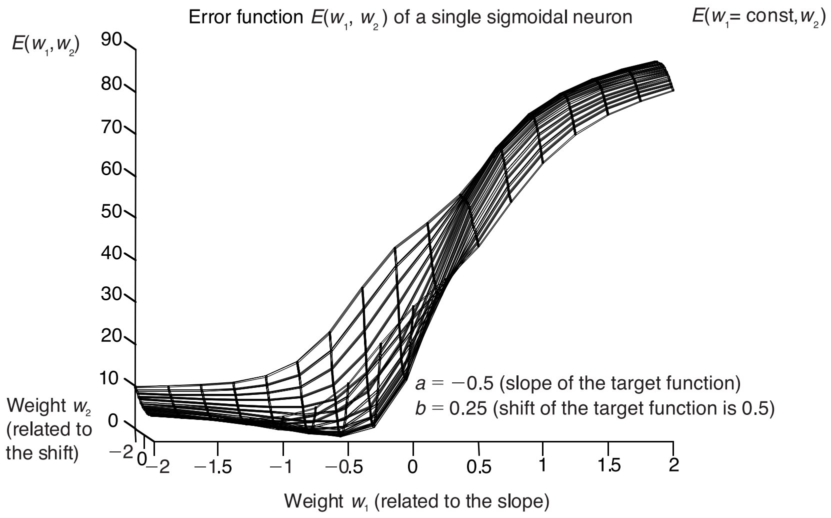


Figure 1.17
Nonlinear error curve and its quadratic approximation.



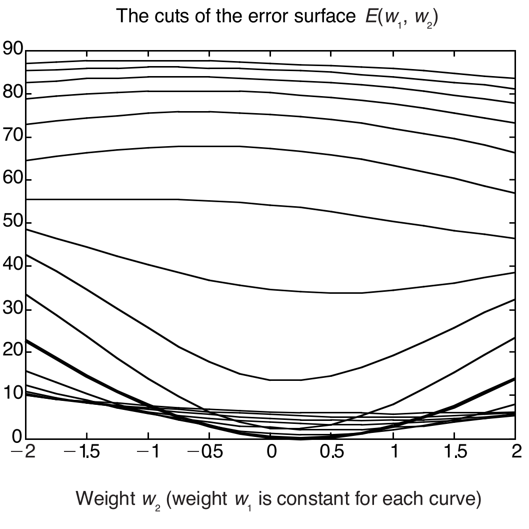


Figure 1.18
Nonlinear error surface and its cuts as the error curves for constant w_1.

It is easy to see in figure 1.17 that even in this simple example the actual shape of a nonlinear error curve $E(w_1, w_2)$ is both nonquadratic and nonconvex. To make the analysis even simpler, model the sigmoidal function with $b = 0$ first. Then $w_2 = 0$, and $E = E(w_1)$. This one-dimensional function shows the nonlinear character of E as well as the character of the quadratic approximation of E in the neighborhood of its minimum. (The quadratic approximation of an error function is a common assumption in proximity to a minimum. This can readily be seen in fig. 1.17). The figure also shows that the shape of E depends on the value of a (slope of the sigmoidal function to be approximated).

In this particular case, the error function E is a curve over the weight w_1 that has a single minimum exactly at $w_1 = a$. There is no saddle point, and all convergent iterative schemes for optimization, starting from any initial random weight w_{10}, will end up at this stationary point $w_1 = a$. Note that the shape of E, as well as its quadratic approximation, depends on the slope a of an approximated function. The smaller the slope a, the steeper the quadratic approximation will be. Expressed in mathematical terms, the curvature at $w_1 = a$, represented in a Hessian matrix[9] of second derivatives of E with respect to the weight, increases with the decrease of a. In this special case, when an error depends on a single weight only, that is, $E = E(w_1)$, the Hessian matrix is a $(1, 1)$ matrix, or a scalar. The same is true for the gradient of this one-dimensional error function. It is a scalar at any given point. Also note that a quadratic approximation to an error function $E(w_1)$ in proximity to an optimal weight value $w_{\text{opt}} = a$ may be seen as a good one.

Now, consider the case where the single neuron is to model the same sigmoidal function y, but with $b \neq 0$. This enables the function y from (1.39) to shift along the x-axis. The complexity of the problem increases dramatically. The error function $E = E(w_1, w_2)$ becomes a surface over the (w_1, w_2) plane. The gradient and the Hessian of E are no longer scalars but a $(2, 1)$ column vector and a $(2, 2)$ matrix, respectively.

Let us analyze the error surface $E(w_1, w_2)$ of the single neuron trying to model function (1.39), as shown in figure 1.18. The error surface in fig 1.18 has the form of a nicely designed driver's seat, and from the viewpoint of optimization is still a very desirable shape in the sense that there is only one minimum, which can be easily reached starting from almost any initial random point. ■

Now, we take up the oldest, and possibly the most utilized, nonlinear optimization algorithm: the gradient-based learning method. It is this method that is a foundation of the most popular learning method in the neural networks field, the error backpropagation method, which is discussed in detail in section 4.1.

A gradient of an error function $E(\mathbf{w})$ is a column vector of partial derivatives with respect to each of the n parameters in $\mathbf{w}$:

$$\mathbf{g} = \nabla E(\mathbf{w}) = \frac{\partial E}{\partial \mathbf{w}} = \begin{bmatrix} \dfrac{\partial E}{\partial w_1} & \dfrac{\partial E}{\partial w_2} & \cdots & \dfrac{\partial E}{\partial w_n} \end{bmatrix}^T. \tag{1.41}$$

An important property of a gradient vector is that its local direction is always the direction of steepest ascent. Therefore, the negative gradient shows the direction of steepest descent. The gradient changes its direction locally (from point to point) on the error hypersurface because the slope of this surface changes. Hence, if one is able to follow the direction of the local negative gradient, one should be led to a local minimum. Since all the nearby negative gradient paths lead to the same local minimum, it is not necessary to follow the negative gradient exactly.

The method of steepest descent exploits the negative gradient direction. It is an iterative method. Given the current point $\mathbf{w}_i$, the next point $\mathbf{w}_{i+1}$ is obtained by a one-dimensional search in the direction of $-\mathbf{g}(\mathbf{w}_i)$ (the gradient vector is evaluated at the current point $\mathbf{w}_i$):

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i \mathbf{g}(\mathbf{w}_i). \tag{1.42}$$

The initial point $\mathbf{w}_1$ is (randomly or more or less cleverly) chosen, and the learning rate η_i is determined by a linear search procedure or experimentally defined. The gradient method is very popular, but there are many ways it can be improved (see section 4.1 and chapter 8). The basic difficulties in applying it are, first, that it will always find a local minimum only, and second, that even though a one-dimensional search begins in the best direction, the direction of steepest descent is a local rather than a global property. Hence, frequent changes (calculations) of direction are often necessary, making the gradient method very inefficient for many problems.

Both these difficulties are readily seen in figure 1.19. Starting from a point A it is unlikely that an optimization, following gradient directions, can end up in the global minimum. Negative gradient vectors evaluated at points A, B, C, D, and E are along the gradient directions AA^*, BB^*, CC^*, DD^* and EE^*. Thus the error function $E(\mathbf{w})$ decreases at the fastest rate in direction AA^* at point A but not at point B. The direction of the fastest decrease at point B is BB^*, but this is not a steepest descent at point C, and so on.

In applying the first-order gradient method, convergence can be very slow, and many modifications have been proposed over the years to improve its speed. In the first-order methods only the first derivative of the error function, namely, the gradient $\nabla E(\mathbf{w})$, is used. The most common improvement is including in the algorithm the

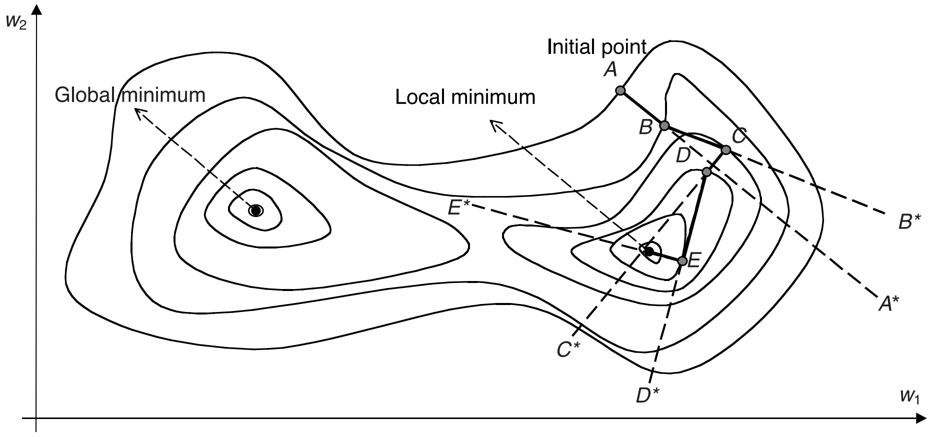


Figure 1.19
Contours of a nonlinear error surface $E(w_1, w_2)$ and steepest descent minimization.

second derivatives (i.e., Hessian matrix) that define the curvature of the function. This leads to a second-order Newton-Raphson method and to various quasi-Newton procedures (see section 4.1 and chapter 8).

The most troublesome parts of an error hypersurface are long, thin, curving valleys. In such valleys, the successive steps oscillate back and forth across the valley. For such elongated valleys the eigenvalues ratio $\lambda_{\max}/\lambda_{\min}$ of the corresponding Hessian matrix is much larger than 1. For such an area, using a Hessian matrix may greatly improve the convergence.

In applying the steepest descent method given by (1.42), the following question immediately arises: How large a step should be taken in the direction $-\mathbf{g}(\mathbf{w}_i)$ from one current point to the next. From (1.42) it is clear that a learning rate η_i determines the length of a step. A more important question is whether the choice of a learning rate η_i can make the whole gradient descent procedure an unstable process. On a one-dimensional quadratic error surface as shown in figure 1.20, the graphs clearly indicate that training diverges for learning rates $\eta > 2\eta_{\text{opt}}$.

For a quadratic one-dimensional error curve $E(w_1)$ the optimal learning rate can readily be calculated, and one can follow this calculation in figure 1.21. From (1.42), and when a learning rate is fixed ($\eta_i = \eta$), it follows that the weight change at the ith iteration step is

$$\Delta w_{1i} = \eta \frac{\partial E}{\partial w_1}\bigg|_i . \tag{1.43}$$

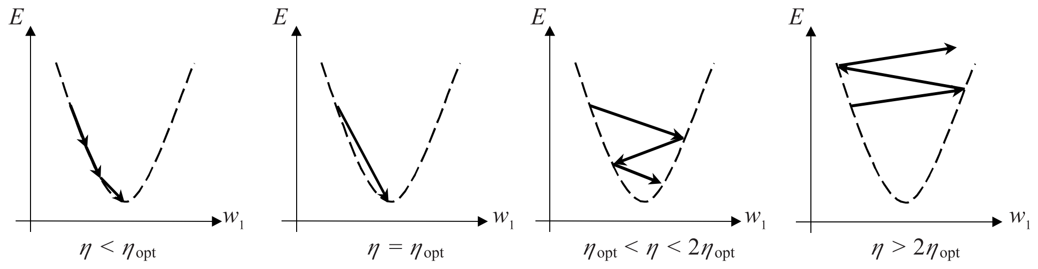

E
w_1
$\eta < \eta_{\text{opt}}$
$\eta = \eta_{\text{opt}}$
$\eta_{\text{opt}} < \eta < 2\eta_{\text{opt}}$
$\eta > 2\eta_{\text{opt}}$

Figure 1.20
Gradient descent for a one-dimensional quadratic error surface $E(w_1)$ and the influence of learning rate η size on the convergence of a steepest descent approach.

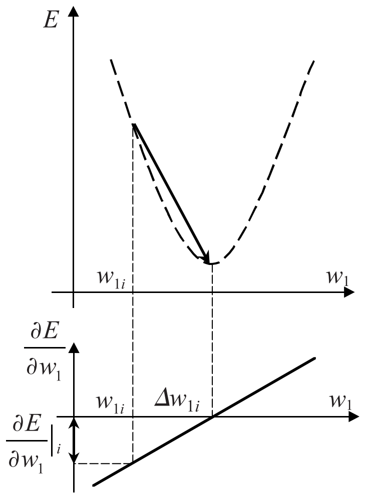

E
w_{1i}
w_1
$\frac{\partial E}{\partial w_1}$
w_{1i}
Δw_{1i}
w_1
$\frac{\partial E}{\partial w_1}\Big|_i$

Figure 1.21
Scheme for the calculation of an optimal learning rate η_{opt} for a one-dimensional quadratic error surface $E(w_1)$.

For a quadratic error surface one can exploit that

$$\left.\frac{\partial^2 E}{\partial w_1^2}\right|_i \Delta w_{1i} = \left.\frac{\partial E}{\partial w_1}\right|_i, \tag{1.44}$$

and combining (1.43) and (1.44), one obtains

$$\eta_{\text{opt}} = \left(\frac{\partial^2 E}{\partial w_{1i}^2}\right)^{-1}. \tag{1.45}$$

One reaches a minimum in a single step using this learning rate, but one must calculate a second derivative that is a scalar for an error function $E(w_1)$. When there are two or more (say, N) unknown weights, an error function is a hypersurface $E(\mathbf{w})$, and one must calculate the corresponding (N, N) symmetric Hessian matrix, defined as

$$\mathbf{H}(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 E(\mathbf{w})}{\partial w_1^2} & \frac{\partial^2 E(\mathbf{w})}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E(\mathbf{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 E(\mathbf{w})}{\partial w_2^2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_2 \partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 E(\mathbf{w})}{\partial w_N \partial w_1} & \frac{\partial^2 E(\mathbf{w})}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_N^2} \end{bmatrix} \tag{1.46}$$

The symmetry of $\mathbf{H}(\mathbf{w})$ follows from the fact that cross partial derivatives are independent of the order of differentiation:

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} = \frac{\partial^2 E(\mathbf{w})}{\partial w_j \partial w_i}, \qquad i, j = 1, 2, \ldots, N.$$

Note that $E(\mathbf{w})$ is a scalar and that on a general hypersurface both gradient $\mathbf{g} = \mathbf{g}(\mathbf{w})$ and Hessian matrix $\mathbf{H} = \mathbf{H}(\mathbf{w})$, that is, they depend on $\mathbf{w}$ (they are local properties) and do change over the domain space $\Re^N$.

Gradient descent in N dimensions can be viewed as N independent one-dimensional gradient descents along the eigenvectors of the Hessian. Convergence is obtained for $\eta < 2/\lambda_{\max}$, where $\lambda_{\max}$ is the largest eigenvalue of the Hessian. The optimal learning rate in N dimensions η_{opt} that yields the fastest convergence in the direction of highest curvature is $\eta_{\text{opt}} = 1/\lambda_{\max}$.

Note that in a one-dimensional case the optimal learning rate is inversely proportional to a second derivative of an error function. It is known that this derivative is a

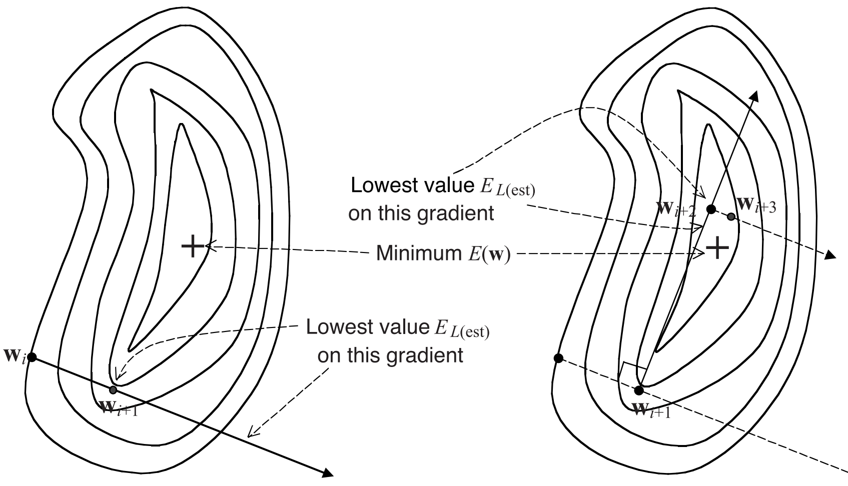


Figure 1.22
Gradient descent on a two-dimensional nonquadratic error surface $E(w_1, w_2)$. An optimal learning rate η_{opt} defines a minimum along the current negative gradient line.

measure of a curvature of a function. Equation (1.45) points to an interesting rule: the closer to a minimum, the higher the curvature and the smaller the learning rate must be. The maximum allowable learning rate for a one-dimensional quadratic error curve is

$$\eta_{\max} = 2\eta_{\text{opt}}. \tag{1.47}$$

For learning rates higher than $\eta_{\max}$, training does not converge (see fig. 1.20). In a general case, the error surface is not quadratic, and the previous considerations only indicate that there are constraints on the learning rate. They also show why η should decrease while approaching a (usually local) minimum of an error surface.

For a nonquadratic error surface (see figure 1.22), calculation of a Hessian at each step may be very time-consuming, and an optimal learning rate is found by a one-dimensional search as follows. The negative gradient at the ith step is perpendicular to the local contour curve and points in the direction of steepest descent. The best strategy is then to search along this direction for a local minimum. To do this, step forward applying equal-sized steps until three points are found, and calculate the corresponding values of the error functions. (A stricter presentation of Powell's quadratic interpolation method can be found in the literature.) Now use a quadratic approximation and estimate the minimum along a current gradient direction $E_{L(\text{est})}$.

For a quadratic surface, this minimum estimate $E_{L(\text{est})}$ is exact. For a nonquadratic error surface, it is an approximation of a minimum E_L only, but there is a little point in being very accurate because on a given slope above some (local) minimum of $E_{\min}(\mathbf{w})$, gradients at all points are nearly all directed toward this particular minimum. (Note the differences between the minimum of a nonconvex error surface $E_{\min}(\mathbf{w})$, the minimum along a current gradient direction E_L, and the minimum estimate along a current gradient direction $E_{L(\text{est})}$.)

At this minimum estimate point $E_{L(\text{est})}$, the current gradient line is tangent to the local level curve. Hence, at this point the new gradient is perpendicular to the current gradient, and the next search direction is orthogonal to the present one (see right graph in fig. 1.22). Repeating this search pattern obtains a local or, more desirable, a global minimum $E_{\min}(\mathbf{w})$ of the error function surface. Note that this procedure evaluates the error function $E(\mathbf{w})$ frequently but avoids frequent evaluation of the gradient.

Such gradient descent learning stops when a given stopping criterion is met. There are many different rules for stopping (see section 4.3.5).

The algorithm of a steepest descent is as follows:

1. Initialize some suitable starting point $\mathbf{w}_1$ (chosen at random or based on previous knowledge) and perform gradient descent at the ith iteration step ($i = 2, K$, where K denotes the iteration step when the stopping criterion is met; K is not known in advance) as in the following steps.

2. Compute the negative gradient in each j direction ($j = 1, N$, where N denotes the number of weights)

$$-g_{ji} = \nabla E(w_j)_i = \left.\frac{\partial E}{\partial w_j}\right|_i .$$

3. Step forward (applying equal-sized steps) until three points (a current point w_j, a middle point $w_j - bg_{ji}$, and a last point $w_j - cg_{ji}$) are found. Evaluate the error function for these three points. (For a nonquadratic surface, the middle point should have the lowest of the three values of the error function.)

4. Use quadratic approximation in each j direction with Powell's quadratic interpolation method to find the optimal learning rate

$$\eta_{\text{opt}} = \frac{1}{2}\left(\frac{(b^2 - c^2)E_a + (c^2 - a^2)E_b + (a^2 - b^2)E_c}{(b - c)E_a + (c - a)E_b + (a - b)E_c}\right)$$

where s is a step length, $a = 0$, $b = s$, $c = 2s$, $E_a = E(w_j - ag_{ji})$, $E_b = E(w_j - bg_{ji})$, and $E_c = E(w_j - cg_{ji})$. (See fig. 1.23.)

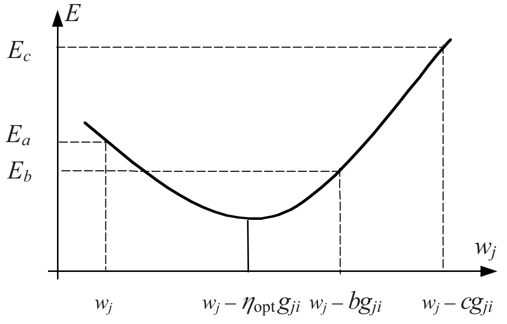


Figure 1.23
Quadratic interpolation about the middle point for a calculation of an optimal learning rate η_{opt} that defines a minimum $E(w_j)$ along the current negative gradient line.

5. Estimate the minimum along the current gradient direction for each j

$$w_{j_{i+1}} = w_{j_i} - \eta_{\text{opt}} \frac{\partial E}{\partial w_{j_i}}.$$

6. Evaluate error function $E(\mathbf{w}_{i+1})$, and if the stopping criterion is met, stop optimization; if not, return to step 2. In virtue of (1.45) as the iterations progress, we are closer to some local minimum, and it will usually be necessary to decrease the search step s, which will result in a smaller optimal learning rate η_{opt}.

Note that the steepest descent shown in figure 1.19 was not performed by applying the optimal learning rate. Had η_{opt} been used, the first descent would have ended up near point D. All sliding along the nonquadratic surface shown in figure 1.19 was done using $\eta < \eta_{\text{opt}}$.

A major shortcoming of the gradient method is that no account is taken of the second derivatives of $E(\mathbf{w})$, and yet the curvature of the function (which determines its behavior near the minimum) depends on these derivatives. There are many methods that partly overcome this disadvantage (see section 4.1 and chapter 8). At the same time, despite these shortcomings, the gradient descent method made a breakthrough in learning in neural networks in the late 1980s, and as mentioned, it is the foundation of the popular error backpropagation algorithm.

This concludes the basic introduction to approximation problems and the description of the need for nonlinear optimization tools in learning from data. Section 1.4 introduces the basics of classical regression and classification approaches that are based on known probability distributions. In this way, the reader will more easily be able to follow the learning from data methods when nothing or very little is known about the underlying dependency.

1.4 Learning and Statistical Approaches to Regression and Classification

There are many theories and definitions of what learning is, but the objective here is to consider how artificial systems, mathematical models, or generally, machines learn. Thus, in the framework of this book, a sound view may be that *learning is inferring functional dependencies* (regularities) *from a set of training examples* (data pairs, patterns, samples, measurements, observations, records).

In *supervised learning*, a set of training data pairs typically contains the inputs $\mathbf{x}_i$ and the desired outputs $y_i = d_i$. (A system's outputs y_i that are used in the training phase are also called desired values. Therefore, when referring to the training stage, this book alternatively uses both notations, y_i and d_i, where d_i stands for *desired.*) There are many different ways and various learning algorithms to extract underlying regularities between inputs and outputs. Successful learning ends in the values of some parameters of a learning machine[10] that capture these inherent dependencies. For a multilayer perceptron NN, these parameters are usually called the hidden and output layer weights. For a fuzzy logic model, they are the rules, as well as the parameters that describe the positions and shapes of the fuzzy subsets. And for a polynomial classifier, these parameters are the coefficients of a polynomial.

The choice of a particular type of learning machine depends on the kind of problem to be solved. They can be machines that learn system dependencies in order to predict future outcomes from observed data. For example, in control applications, signal processing, and financial markets, learning machines are used to predict various signals and stock prices based on past performance. In the case of optical character recognition and for other recognition tasks, learning machines are used to recognize (predict) particular alphabetic, numeric, or symbolic characters based on the data obtained by scanning a piece of paper. These examples involve predictions of two different types of outcome: *continuous variables* in control applications, signal processing, and stock markets, and *categorical variables* (class labels) in optical character or pattern recognition.

The prediction of continuous variables is known as *regression*, and the prediction of categorical variables is known as *classification*. Because of their utmost practical importance, this book takes up only regression and classification models. The third important problem in statistics, density estimation, is not the subject of investigation here. The basics of standard statistical techniques of regression and classification are presented first to aid in the understanding of inferring by using data. Traditionally, by using training patterns, mechanical fitting of the prespecified line, curve, plane, surface, or hypersurface solved these kinds of learning tasks. Here, these estimation problems are approached by using neural networks, fuzzy logic models, or support vector machines. Thus, sections 1.4.1 and 1.4.2, about the basic and classical theories of regression and classification, may give sound insights on learning from data problems.

1.4.1 Regression

The elementary presentation of regression is given using a two-dimensional case. In this way, vital concepts can be shown graphically, which should ease understanding of the true nature of the problem. Conceptually nothing changes in multivariate cases with higher dimensional inputs and outputs, but they cannot be visualized with the relevant hypercurves or hypersurfaces. First, a theoretical regression curve is defined that will later serve as a model for understanding the empirical regression curve.[11] The short definition for this curve states that *the theoretical regression curve is* (*a graph of*) *the mean of a conditional probability-density function* $P(y \mid x)$.

A geometrical insight into the theory of regression may be the easiest way to introduce the concepts that follow. In the two-dimensional case (where only two random variables are involved) the general joint probability-density function[12] $P(x, y)$ can be thought of as a surface $z = P(x, y)$ over the (x, y) plane. If this surface is intersected by a plane $x = x_i$, we obtain a curve $z = P(x_i, y)$ over the line $x = x_i$ in the (x, y) plane. The ordinates z of this curve are proportional to the conditional probability-density of y given $x = x_i$. If x has the fixed value x_i, then along the line $x = x_i$ in the (x, y) plane the mean (expected or average) value of y will determine a point whose ordinate is denoted by $\mu_{y|x_i}$. As different values of x are selected, different mean points along the corresponding vertical lines will be obtained. Hence, the ordinate $\mu_{y|x_i}$ of the mean point in the (x, y) plane is a function of the value of x_i selected. In other words, μ depends upon x, $\mu = \mu(x)$. The locus of all mean points will be the graph of $\mu_{y|x}$. This curve is called the *regression curve* of y on x. Figure 1.24 indicates the geometry of the typically nonlinear regression curve for a general density distribution (i.e., for the general joint probability-density function $P(x, y)$). Note that the surface $P(x, y)$ is not shown in this figure. Also, the meaning of the graph in figure 1.24 is that the peak of the conditional probability-density function $P(y \mid x)$ indicates that the most likely value of y given x_i is $\mu_{y|x_i}$. Analytically, the derivation of the regression curve is presented as follows.

Let x and y be random variables with a joint probability-density function $P(x, y)$. If this function is continuous in y, then the *conditional probability-density function* of y with respect to fixed x can be written as

$$P(y \mid x) = \frac{P(x, y)}{P(x)} = \frac{P(x, y)}{\int_{-\infty}^{+\infty} P(x, y)\, dy}, \tag{1.48}$$

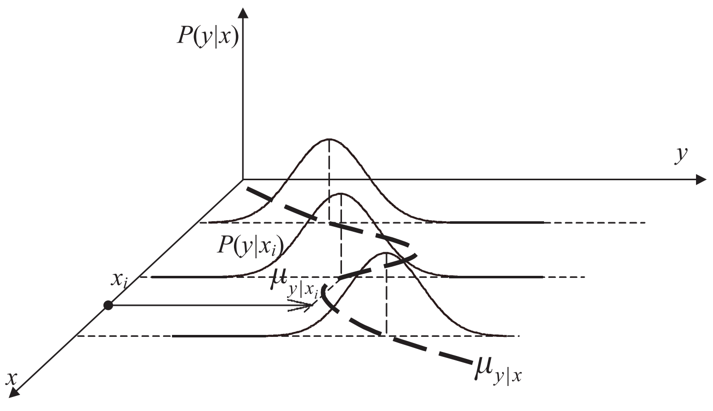


Figure 1.24
Geometry of the typical regression curve for a general density distribution.

where $P(x)$ represents the *marginal probability-density function* $P(x) = \int_{-\infty}^{+\infty} P(x,y)\,dy$. By using this function the regression curve is defined as the expectation of y for any value of x

$$\mu_{y|x} = \mathbb{E}(y \mid x) = \int_{-\infty}^{+\infty} yP(y \mid x)\,dy = \int_{-\infty}^{+\infty} y\frac{P(x,y)}{P(x)}\,dy = \frac{\int_{-\infty}^{+\infty} yP(x,y)\,dy}{P(x)}. \tag{1.49}$$

This function (1.49) is the regression curve of y on x. It can be easily shown that this regression curve gives the best estimation of y in the mean squared error sense. Note that there is no restriction on function $\mu_{y|x}$. Depending upon the joint probability-density function $P(x,y)$, this function belongs to a certain class, for example, the class of all linear functions or the class of all functions of a given algebraic or trigonometric polynomial form, and so on. Example 1.7 gives a simple illustration of how (1.49) applies.

Example 1.7 The joint probability-density function $P(x,y)$ is given as

$$P(x,y) = \begin{cases} 2-x-y, & 0<x<1, 0<y<1 \\ 0 & \text{elsewhere} \end{cases}.$$

Find the regression curve of y on x.

In order to find $\mu_{y|x}$, first find the marginal probability-density function

$$P(x) = \int_{-\infty}^{+\infty} P(x,y)\,dy = \int_0^1 (2-x-y)\,dy = \frac{3}{2} - x.$$

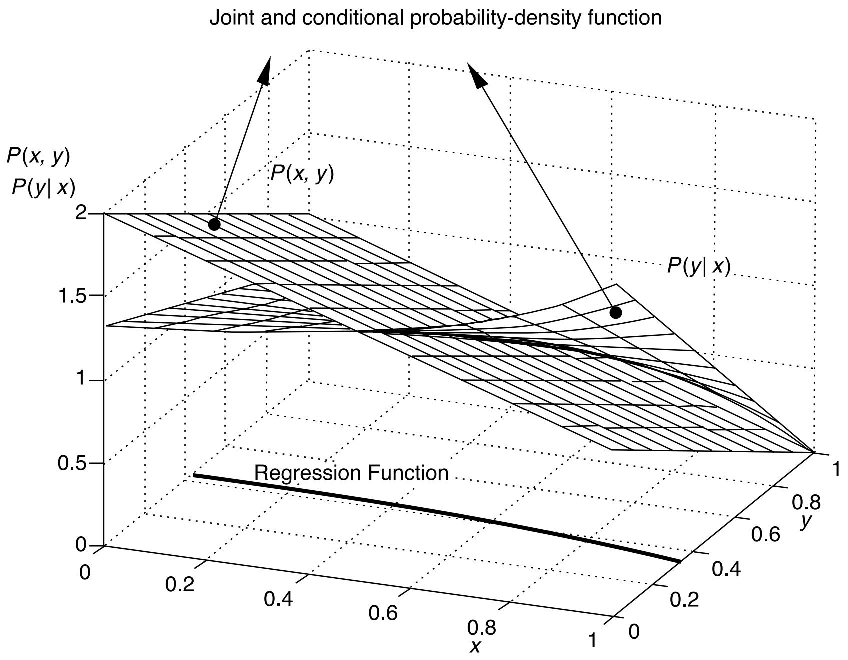


Figure 1.25
The joint probability-density function $P(x, y) = 2 - x - y$, the corresponding conditional probability-density function $P(y \mid x) = (2 - x - y)/(1.5 - x)$, and the regression function (curve) $\mu_{y|x}$.

From (1.49),

$$\begin{aligned}\mu_{y|x} &= \mathbb{E}(y \mid x)\\ &= \frac{\int_{-\infty}^{+\infty} yP(x, y)\, dy}{P(x)} = \int_0^1 y \frac{2 - x - y}{(3/2) - x}\, dy\\ &= \frac{1}{(3/2) - x} \int_0^1 [(2 - x)y - y^2]\, dy = \frac{3x - 4}{6x - 9}.\end{aligned}$$

Thus, the regression curve is the hyperbola. The joint probability-density function $P(x, y)$, the conditional probability-density function $P(y \mid x)$, and the regression curve $\mu_{y|x}$ are shown in figure 1.25. ■

Example 1.8 shows that the regression function for jointly normally distributed variables is linear, that is, a straight line. This is an interesting property that was heavily exploited in statistics. Linear regression and correlation analysis, which are closely related, are both very developed and widely used in diverse fields. The explanation for such broad application of these theories lies in the remarkable fact that under certain circumstances the probability distribution of the sum of independent random variables, each having an arbitrary (not necessarily normal) distribution, tends toward a normal probability distribution as the number of variables in the sum tends toward infinity. In statistics, this statement, together with the conditions under which the result can be proved, is known as the *central limit theorem*. These conditions are rarely tested in practice, but the empirically observed facts are that a joint probability-density function of a great many random variables closely approximates a normal distribution. The reason for the widespread occurrence of normal joint probability-density functions for random variables is certainly stated in the central limit theorem and in the fact that superposition may be common in nature.

Before proceeding to the next example remember that the joint probability-density function for two independent variables is $P(x, y) = P(x)P(y)$. If both variables are normally distributed, it follows that the normal bivariate (two-dimensional) joint probability-density function for independent random variables x and y is

$$P(x, y) = P(x)P(y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{1}{2}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right]\right). \tag{1.50}$$

If the variables x and y are not independently distributed, it is necessary to modify (1.50) to take into account the relationship between x and y. This is done in (1.51) by introducing a cross-product term in the exponent of (1.50). The *linear correlation coefficient* ρ of this term is defined as $\rho = \sigma_{xy}/\sigma_x\sigma_y$, where σ_{xy}, σ_x, and σ_y are the covariance and variances in directions x and y, respectively. ρ is equal to zero when x and y are independent, and equal to $+1$ or -1 when these two variables are deterministically connected. Equation (1.51) is defined for $-1 < \rho < +1$. For $\rho = \pm 1$, (1.51) does not have any sense. Note that the correlation coefficient ρ is defined for the linear dependence between two random variables, and it is a measure of the strength of this linear relationship. Thus, $\rho = 0$ does not imply that two variables are not closely related. It implies only that these variables are not linearly related. For nonlinearly depending variables, the linear correlation coefficient ρ as previously defined is equal to zero ($\rho = 0$).

Note also that the statistical functional relationships between two (or more) variables in general, and the correlation coefficient ρ in particular, are completely devoid of any cause-and-effect implications. For example, if one regresses (correlates) the size of a person's left hand (dependent variable y) to the size of her right hand (independent variable x), one will find that these two variables are highly correlated. But this does not mean that the size of a person's right hand *causes* a person's left hand to be large or small. Similarly, one can try to find the correlation between the death rate due to heart attack (infarction) and the kind of sports activity of a player at the moment of death. One will eventually find that the death rate while playing bowls or chess (low physical activity) is much higher than while taking part in boxing, soccer, or a triathlon (high physical activity). Despite this correlation, the conclusion that one is more likely to suffer heart attack while playing bowls, cards, or chess is wrong, for there is no direct cause-effect relationship between the correlated events of suffering an infarction and taking part in certain sports activities. It is far more likely that, typically, senior citizens are more involved in playing bowls and the cause of death is their age in the first instance. In short, note that two or more variables can be highly correlated without causation being implied.

Example 1.8 Consider two random variables that possess a bivariate normal joint probability-density function

$$P(x,y) = \frac{\exp\left(-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - 2\rho\left(\frac{x-\mu_x}{\sigma_x}\right)\left(\frac{y-\mu_y}{\sigma_y}\right) + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right]\right)}{2\pi\sigma_x\sigma_y\sqrt{(1-\rho^2)}}. \tag{1.51}$$

Show that both the marginal $(P(x), P(y))$ and the conditional $(P(y \mid x), P(x \mid y))$ probability-density functions are normal distributions. Show that the curve of regression is linear.

The marginal probability-density function is defined as $P(x) = \int_{-\infty}^{+\infty} P(x,y)\,dy$, where $P(x,y)$ is defined in (1.51). Simplify this integration by changing the variables to $u = (x-\mu_x)/\sigma_x$ and $v = (y - \mu_y/\sigma_y)$. Then $dy = \sigma_y\,dv$ and

$$P(x) = \frac{1}{2\pi\sigma_x\sqrt{(1-\rho^2)}}\int_{-\infty}^{+\infty}\exp\left(-\frac{1}{2(1-\rho^2)}(u^2 - 2\rho uv + v^2)\right)dv.$$

Adding and subtracting ρ^2u^2 to the exponent in order to complete the square in v gives

$$\begin{aligned} P(x) &= \frac{\exp(-(u^2/2))}{2\pi\sigma_x\sqrt{1-\rho^2)}}\int_{-\infty}^{+\infty}\exp\left(-\frac{1}{2(1-\rho^2)}(v-\rho u)^2\right)dv \\ &= \frac{\exp(-(u^2/2))}{2\pi\sigma_x}\int_{-\infty}^{+\infty}\exp(-(z^2/2))\,dz, \end{aligned}$$

where

$$z = \frac{v - \rho u}{\sqrt{1-\rho^2}} \quad \text{and} \quad dv = \sqrt{1-\rho^2}\,dz.$$

Substituting back the value of u in terms of x and inserting the value $\sqrt{2\pi}$ for this familiar integral, $P(x)$ finally reduces to

$$P(x) = \frac{\exp\left(-\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)^2\right)}{\sqrt{2\pi}\sigma_x}. \tag{1.52}$$

The corresponding result for $P(y)$ follows from symmetry, and (1.52) shows that the marginal distributions (probability-density functions) of a joint normal distribution are normal. Note that if one sets ρ equal to zero in (1.51), this equation reduces to (1.50), which is the joint normal distribution for two independent normal variables. Thus, if two normal variables are uncorrelated, they are independently distributed. Note, however, that from the preceding discussion of correlation, it should be clear that the lack of a linear correlation does not imply a lack of dependence (relationship) of every (notably nonlinear) kind between two, or more, variables.

For regression problems, the conditional probability distribution is of utmost importance, and in the case of the joint normal distribution it possesses interesting properties. In order to find $P(y \mid x)$, use the definition (1.48) as well as the substitutions u and v given previously, which yields

$$P(y \mid x) = \frac{\dfrac{\exp\left(-\frac{1}{2(1-\rho^2)}(u^2 - 2\rho uv + v^2)\right)}{2\pi\sigma_x\sigma_y\sqrt{(1-\rho^2)}}}{\dfrac{\exp(-(u^2/2))}{\sqrt{2\pi}\sigma_x}}$$

$$= \frac{\exp\left(-\frac{1}{2(1-\rho^2)}(v^2 - 2\rho uv + \rho^2 u^2)\right)}{\sqrt{2\pi}\sigma_y\sqrt{(1-\rho^2)}}$$

$$= \frac{\exp\left(-\frac{1}{2}\left(\frac{v-\rho u}{\sqrt{(1-\rho^2)}}\right)^2\right)}{\sqrt{2\pi}\sigma_y\sqrt{(1-\rho^2)}}.$$

Expressing u and v in terms of the original variables x and y and, in order to stress a dependence of y on the selected value of x, denoting y as y_x, the last expression reduces to

$$P(y\,|\,x) = \frac{\exp\left(-\frac{1}{2}\left(\frac{y_x - \mu_y - \rho\frac{\sigma_y}{\sigma_x}(x-\mu_x)}{\sqrt{(1-\rho^2)}}\right)^2\right)}{\sqrt{2\pi}\sigma_y\sqrt{(1-\rho^2)}}. \tag{1.53}$$

In order to find the regression curve $\mu_{y|x}$ defined in (1.49) as the expectation $\mu_{y|x} = E(y\,|\,x)$, note that x is the fixed variable in (1.53), and that this equation represents the normal density function for y_x. Hence, for given x, the mean of (1.53) is the sum of the second and third terms in the numerator of the exponent in (1.53). According to the definition of the regression curve, being the locus of the means of a conditional probability-density, the regression curve of y on x when x and y are jointly normally distributed is the straight line whose equation is

$$\mu_{y|x} = \mu_y + \rho\frac{\sigma_y}{\sigma_x}(x-\mu_x). \tag{1.54}$$

By symmetry a similar result holds for x and y interchanged, that is, for the curve of regression of x on y. The fact that the regression curve of two normally distributed variables is a straight line helps to justify the frequent use of linear regression models because variables that are approximately normally distributed are encountered frequently. ■

1.4.2 Classification

The standard statistical techniques for solving classification tasks cover the broad fields of pattern recognition and decision-making problems. Many artificial systems perform classification tasks: speech or character recognition systems, fault detection systems, readers of magnetic-strip codes on credit cards, readers of UPC bar codes, various alarm systems, and so on. In all these different systems the classifier is faced with different observations (measurements, records, patterns) that should be assigned meaning (class or category). *Classification* or *pattern recognition* is inferring meaning (category, class) from observations.

There are two basic stages in designing a classifier: the training phase and the test (generalization or application) phase. The most general schemes of these two stages are shown in figure 1.26.

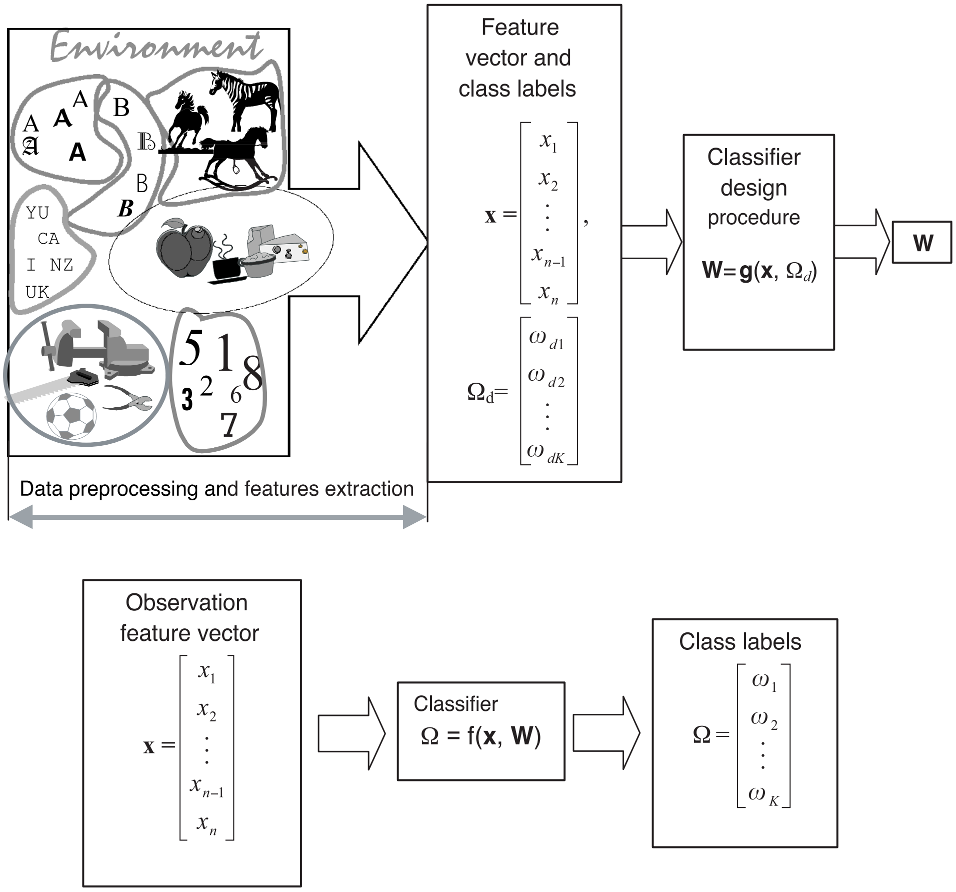

Environment
A A A A A
B B B B B
YU CA I NZ UK
5 1 8 3 2 6 7
Data preprocessing and features extraction
Feature vector and class labels
$\mathbf{x} = [x_1, x_2, \dots, x_{n-1}, x_n]^T$,
$\Omega_d = [\omega_{d1}, \omega_{d2}, \dots, \omega_{dK}]^T$
Classifier design procedure
$\mathbf{W} = \mathbf{g}(\mathbf{x}, \Omega_d)$
W
Observation feature vector
$\mathbf{x} = [x_1, x_2, \dots, x_{n-1}, x_n]^T$
Classifier
$\Omega = f(\mathbf{x}, \mathbf{W})$
Class labels
$\Omega = [\omega_1, \omega_2, \dots, \omega_K]^T$

Figure 1.26
Classification's training phase (*top*) and test (application) phase (*bottom*). The training phase, or classifier design, ends up in a set of parameters **W** that define the disjoint class regions.

During the training phase the classifier is given training patterns comprised of selected *training feature vectors* $\mathbf{x}$ and *desired class labels* Ω_d. The result of the training phase is the set of classifier's parameters that are called weights $\mathbf{W}$ here. These weights define the general discriminant functions that form the class boundaries between disjoint class or category regions. These class boundaries are points, curves, surfaces, and hypersurfaces in the case of one-, two-, three-, and higher-dimensional feature space, respectively. In the test phase, or later in applications, the classifier recognizes (classifies) the inputs in the form of (previously unseen) measured feature vectors $\mathbf{x}$.

Figure 1.26 indicates that classification is a very broad field. Human beings typically process visual, sound, tactile, olfactory, and taste signals. In science and engineering the goal is to understand and classify these and many other signals, notably different geometrical (shape and size) and temporal (time-dependent) signals. In order to do this the pattern recognition system should solve three basic problems: sensing desired variables, extracting relevant features, and based on these features, performing classification. While the first and second parts are highly problem-dependent, the classification procedure is a more or less general approach. Depending upon the specific problem to be solved, measurement (recording, observation) and features extraction would be done by different sensing devices: thermocouples, manometers, accelerometers, cameras, microphones, or other sensors. Today, using A/D converters, all these different signals would be transformed into digital form, and the relevant features would be extracted. It is clear that this preprocessing part is highly problem-dependent. A good features extractor for geometrical shape recognition would be of no use for speech recognition tasks or for fingerprint identification.

At the same time, the classification part is a more general tool. A pattern classifier deals with features and partitions (tessellates, carves up) the feature space into line segments, areas, volumes, and hypervolumes, called *decision regions*, in the case of one-, two-, three-, or higher-dimensional features, respectively. All feature vectors belonging to the same class are ideally assigned to the same category in a decision region. Decision regions are often single nonoverlapping volumes or hypervolumes. However, decision regions of the same class may also be disjoint, consisting of two or more nontouching regions.

Only the basics of the statistical approach to the problem of feature pattern classification are presented here. The objects are feature vectors $\mathbf{x}_i$ and class labels ω_i. The features extraction procedure is taken for granted in the hope that the ideal features extractor would produce the same feature vector $\mathbf{x}$ for each pattern in the same class and different feature vectors for patterns in different classes. In practice, because of the probabilistic nature of the recognition tasks, one must deal with stochastic (noisy) signals. Therefore, even in the case of pattern signals belonging to the same category, there will be different inputs to the features extractor that will always produce different feature vectors **x**, but, one hopes that the within-class variability is small relative to the between-class variability. In this section, the fundamentals of the Bayesian approach for classifying the handwritten numerals 1 and 0 are presented first. This is a simple yet important task of two-class (binary) classification (or dichotomization). This procedure is then generalized for multifeature and multiclass pattern classification. Despite being simple, these binary decision problems illustrate most of the concepts that underlie all decision theory.

There are many different but related criteria for designing classification decision rules. The six most frequently used decision criteria are maximum likelihood, Neyman-Pearson, probability-of-(classification)error, min-max, *maximum-a-posteriori* (*MAP*), known also as the *Bayes' decision criterion*, and finally, the *Bayes' risk decision criterion*. This book cannot cover all these approaches, and it concentrates on the Bayes' rule-based criteria only. We start with a seventh criterion, maximum-a-priori, in order to gradually introduce the reader to the MAP or Bayes' classification rule. The interested reader can check the following claims regarding the relationships among these criteria:

- The probability-of-(classification)-error decision criterion is equivalent to the MAP (Bayes') decision criterion; this is shown later in detail.
- For the same prior probabilities, $P(\omega_1) = P(\omega_2)$, the maximum likelihood decision criterion is equivalent to the probability-of-(classification)-error decision criterion, that is, to the MAP (Bayes') decision criterion.
- For the same conditional probability-densities, $P(x \mid \omega_1) = P(x \mid \omega_2)$, the maximum-a-priori criterion is equivalent to the MAP (Bayes') decision criterion.
- The Neyman-Pearson criterion is identical in form (which is actually a test of likelihood ratio against a threshold) to the maximum likelihood criterion. They differ in the values of thresholds and, when the threshold is equal to unity, the N-P criterion is equivalent to the maximum likelihood criterion.
- The Bayes' risk decision criterion represents a generalization of the probability-of-(classification)-error decision criterion, and for a 0-1 loss function these two classification methods are equivalent. This is shown later.

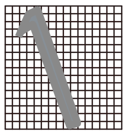
After the Bayes' (MAP) classification rule has been introduced, the subsequent sections examine an important concept in decision making: a *cost* or *loss* regarding a given classification. This leads to the classification schemes that minimize some *risk* function. This approach is important in all applications where misclassification of some classes is very costly (e.g., in medical or fault diagnosis and in investment decisions, but also in regression and standard classification problems where the risk would measure some error or discrepancy regarding desired values or misclassification of data).

Finally, the concepts of discriminant functions are introduced and an important class of problems is analyzed: classification of normally distributed classes that generally have quadratic decision boundaries. A more detailed treatment of these topics may be found in Cios, Pedrycz, and Swiniarski (1998, ch. 4) and in Schürmann (1996) as well as in classical volumes on decision and estimation (Melsa and Cohn 1978) or on classification (Duda and Hart 1973). The development here roughly follows Cios et al. and Melsa and Cohn.

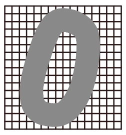
Bayesian Classification in the Case of Two Classes The Bayesian approach to classification assumes that the problem of pattern classification can be expressed in probabilistic terms and that the a priori probabilities $P(\omega_i)$ and the conditional probability-density functions $P(x \mid \omega_i)$, $i = 1, 2$, of feature pattern vectors are known. As is the case in regression, this initial assumption will generally not be fulfilled in practice. Nevertheless, a sound understanding of the classical Bayesian approach is fundamental to grasping basic concepts about learning from training data sets without knowledge of any probability distribution.

Assume recognition of two handwritten numerals (or any characters): 1 and 0. In the experiment, the optical device is supplied with typical samples (on, say, a 16×16 grid), as shown in figure 1.27. The 0's generally cover a larger total area of the grid than do the 1's, and the total area covered by the numeral is chosen as a suitable feature in this example.

The task here is to devise an algorithm for the classification of handwritten characters into two distinct classes: 1's and 0's. Assume that the characters emerge in



$$\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0.1 \\ 0.2 \\ \vdots \\ 0 \end{bmatrix} \quad \text{Feature } x_1 = \sum_{i=1}^{256} v_i = 4$$



$$\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0.1 \\ 1 \\ 0.4 \\ \vdots \\ 0 \end{bmatrix} \quad \text{Feature } x_1 = \sum_{i=1}^{256} v_i = 8$$

Figure 1.27
Typical samples for a two-class recognition with pattern vectors $\mathbf{v}_i$ and features x_i.

random sequence but that each can be only a 1 or a 0. In statistical terms, a *state of nature* (or class space) Ω, an emerged character, has only two distinct states—either it is "a 1" or "a 0":

$$\Omega = \{\omega_1, \omega_2\} = \{\text{"a 1"}, \text{"a 0"}\}. \tag{1.55}$$

Ω is a random variable taking two distinct values, ω_1 for a 1 and ω_2 for a 0. ω_i can be assigned a numerical coding, for example, $\omega_1 = 1$ (or 0, or -1, or any), and $\omega_2 = 0$ (or -1, or 1, or any). Note that a numeral is perceived as an object, an image, or a pattern. This pattern will then be analyzed considering its features. (There is a single feature, x_1, for this two-class task at the moment. In the next section, on multiclass classification, a second feature is introduced and the feature space becomes two-dimensional.) Since characters emerge in a random way, Ω is a random variable. So are the features, and the whole task is described in probabilistic terms.

The goal of the Bayesian method is to classify objects statistically in such a way as to minimize the probability of their misclassification. The classification ability of new patterns will depend on prior statistical information gathered from previously seen randomly appearing objects. In particular, such classification depends upon *prior* (a priori) *probabilities* $P(\omega_i)$ and on *conditional probability-density functions* $P(x \mid \omega_i)$, $i = 1, 2$. The prior probability $P(\omega_1)$ corresponds to the fraction n_{ω_1} of 1's in the total number of characters N. Therefore, the prior probabilities can be defined as

$$P(\omega_i) = \frac{n_{\omega_i}}{N}, \qquad i = 1, 2. \tag{1.56}$$

Thus, $P(\omega_i)$ denotes the *unconditional probability function* that an object belongs to class ω_i, without the help of any other information about this object in the form of feature measurements. A prior probability $P(\omega_i)$ represents prior knowledge (in probabilistic terms) of how likely it is that the pattern belonging to class i may appear even before its actual materialization. Thus, for example, if one knew from prior experiments that there are four times more 1's than 0's in the strings of numerals under observation, one would have $P(\omega_1) = 0.8$ and $P(\omega_2) = 0.2$. Note that the sum of prior probabilities is equal to 1:

$$\sum_{i=1}^{N} P(\omega_i) = 1. \tag{1.57}$$

Bayesian Classification Based on Prior Probabilities Only Let us start classifying under the most restricted assumption first. Suppose that the optical device is out of order and there is no information about feature x of a materialized numeral. Thus, the only available statistical knowledge of the character strings to be classified is the prior probabilities $P(\omega_1) = 0.8$ and $P(\omega_2) = 0.2$. It is difficult to believe that the classification will be very good with so little knowledge, but let us try to establish a decision strategy that should lead to the smallest misclassification error. The best and natural decision now is to assign the next character to the class having the higher prior probability. Therefore, with only the prior probabilities $P(\omega_1)$ and $P(\omega_2)$ known, the decision rule would be

Assign a character to

$$\begin{aligned} &\text{class } \omega_1 \quad \text{if} \quad P(\omega_1) > P(\omega_2), \quad \text{or to} \\ &\text{class } \omega_2 \quad \text{if} \quad P(\omega_2) > P(\omega_1). \end{aligned} \tag{1.58}$$

If $P(\omega_1) = P(\omega_2)$, both classes are equally likely, and either decision would be correct.

The task is to minimize the probability of a classification error, which can be expressed as

$$P(\text{classification error}) = \begin{cases} P(\omega_2) & \text{if we decide } \Omega = \omega_1, \\ P(\omega_1) & \text{if we decide } \Omega = \omega_2. \end{cases} \tag{1.59}$$

Thus, selecting a class with a bigger prior probability gives a smaller probability of classification error. If one chooses class ω_1 in this example without seeing any features, the probability of misclassification is $P(\omega_2) = 0.2$. This is the best classification strategy with so little information—$P(\omega_i)$ only—about the objects to be classified.

Frankly, one would never attempt to solve real-life classification problems with so little knowledge, and typical problems are those with available features.

Bayesian Classification Based on Prior Probabilities and Features It is clear that by including information on the total area covered by a numeral in the problem of classifying 1's and 0's, one can increase classification accuracy and consequently minimize the number of misclassified characters. Note that characters are stochastic images. Each person writes differently and writes the same characters differently each time. Thus, a feature x (the total area of a grid covered by a character) takes random values. This is a continuous variable over a given range, and experimentally by extracting features from 500 samples of each character, *discrete class-conditional probability-density functions* in the form of two histograms are obtained, as shown in figure 1.28. If the number of samples is increased to infinity, these discrete distribution densities converge into two *continuous class-conditional probability-density functions* $P(x \mid \omega_i)$, as shown in figure 1.28. $P(x \mid \omega_i)$ can also be called the *data*

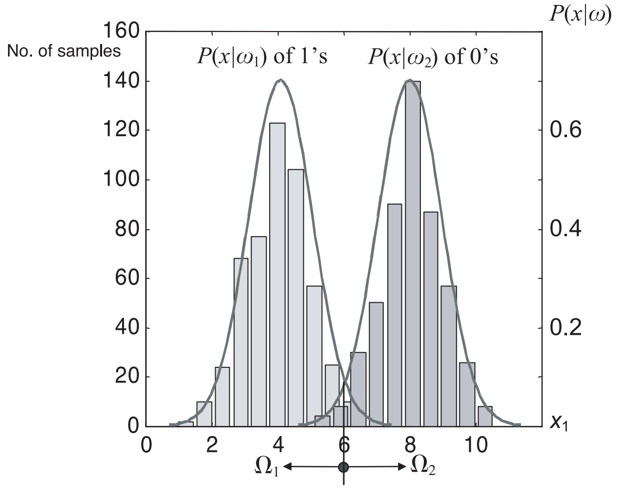


Figure 1.28
Typical histograms (*left ordinate*) and class-conditional probability-density functions $P(x \mid \omega_i)$ (*right ordinate*) for two-class recognition with a single feature x_1. The decision boundary, shown as a point $x_1 = 6$, is valid for equal prior probabilities $P(\omega_1) = P(\omega_2) = 0.5$.

generator's conditional probability-density functions or *the likelihood of class* ω_i with respect to the value x of a feature variable.

The probability distributions presented in figure 1.28 are fairly similar, but depending on the state of nature (the specific data generation mechanism), they can be rather different. The probability-density

$$P(x \mid \omega_i), \qquad i = 1, 2, \tag{1.60}$$

is the probability-density function for a value of a random feature variable x given that the pattern belongs to a class ω_i. The conditional probability-density functions $P(x \mid \omega_1)$ and $P(x \mid \omega_2)$ represent distributions of variability of a total area of the image covered by a 1 and a 0. These areas are thought to be different, and $P(x \mid \omega_1)$ and $P(x \mid \omega_2)$ may capture this difference in the case of 1's and 0's. Thus, information about this particular feature will presumably help in classifying these two numerals.

Remember that the joint probability-density function $P(\omega_i, x)$ is the probability-density that a pattern is in a class ω_i and has a feature variable value x. Recall also that the conditional probability function $P(\omega_i \mid x)$ denotes the probability (and not probability-density) that the pattern class is ω_i given that the measured value of the feature variable is x. The probability $P(\omega_i \mid x)$ is also called the *posterior* (a posteriori) *probability*, and its value depends on the a posteriori fact that a feature variable has a concrete value x. Because $P(\omega_i \mid x)$ is the probability function,

$$\sum_{i=1}^{2} P(\omega_i \mid x) = 1. \tag{1.61}$$

Now, use the relations

$$\begin{aligned} P(\omega_i, x) &= P(\omega_i \mid x)P(x), \qquad i = 1, 2, \\ P(\omega_i, x) &= P(x \mid \omega_i)P(\omega_i), \qquad i = 1, 2, \end{aligned} \tag{1.62}$$

where $P(x)$ denotes the *unconditional probability-density function* for a feature variable x

$$P(x) = \sum_{i=1}^{2} P(x \mid \omega_i)P(\omega_i) = P(x \mid \omega_1)P(\omega_1) + P(x \mid \omega_2)P(\omega_2). \tag{1.63}$$

The posterior probability $P(\omega_i \mid x)$ is sought for classifying the handwritten characters into corresponding classes. From equations (1.62) this probability can be expressed in the form of a Bayes' rule

$$P(\omega_i \mid x) = \frac{P(x \mid \omega_i)P(\omega_i)}{P(x)}, \qquad i = 1, 2, \tag{1.64}$$

or

$$P(\omega_i \mid x) = \frac{P(x \mid \omega_i)P(\omega_i)}{\sum_{i=1}^{2} P(x \mid \omega_i)P(\omega_i)} = \frac{P(x \mid \omega_i)P(\omega_i)}{P(x \mid \omega_1)P(\omega_1) + P(x \mid \omega_2)P(\omega_2)}. \tag{1.65}$$

The probability-density function $P(x)$ only scales the previous expressions, ensuring in this way that the sum of posterior probabilities is 1 ($P(\omega_1 \mid x) + P(\omega_2 \mid x) = 1$). The practicability of these Bayes' rules lies in the fact that the conditional probability function $P(\omega_i \mid x)$ can be calculated using $P(x \mid \omega_i)$ and $P(\omega_i)$, which can be estimated from data much more easily than $P(\omega_i \mid x)$ itself. Equipped with (1.65) and having the feature measurement x while knowing probabilities $P(\omega_i)$ and $P(x \mid \omega_i)$, one can calculate $P(\omega_i \mid x)$. Having the posterior probabilities $P(\omega_i \mid x)$, one can formulate the following classification decision rule based on both prior probability and observed features:

Assign a character to a class ω_i having the larger value of the posterior conditional probability $P(\omega_i \mid x)$ for a given feature x.

This is called the Bayes' classification rule, and it is the best classification rule for minimizing the probability of misclassification. In other words, this rule is the best one for minimizing the probability of classification error. In the case of two-class handwritten character recognition, for a given numeral with observed feature x, the conditional probability of the classification error is

$$P(\text{classification error} \mid x) = \begin{cases} P(\omega_2 \mid x) & \text{if we decide } \Omega = \omega_1, \\ P(\omega_1 \mid x) & \text{if we decide } \Omega = \omega_2. \end{cases} \tag{1.66}$$

Note that for equal prior probabilities $P(\omega_1) = P(\omega_2)$, the decision depends solely on the class-conditional probability-density functions $P(x \mid \omega_i)$, and the character is assigned to the class having the bigger $P(x \mid \omega_i)$. Thus, in this classification task, having $P(\omega_1) = P(\omega_2) = 0.5$, the decision boundary in figure 1.28 is at the intersecting point ($x = 6$) of the two class-conditional probability-density functions. (In the case of a one-dimensional feature, the decision regions are line segments, and the decision boundary is a point.)

Analyzing many differently written 1's and 0's will yield different feature values x, and it is important to see whether the Bayes' classification rule minimizes the *average probability of error*, because it should perform well for all possible patterns. This averaging is given by

$$\begin{aligned} P(\text{classification error}) &= \int_{-\infty}^{+\infty} P(\text{classification error}, x)\, dx \\ &= \int_{-\infty}^{+\infty} P(\text{classification error} \mid x) P(x)\, dx \end{aligned} \tag{1.67}$$

Clearly, if the classification rule as given by (1.66) minimizes the probability of misclassification for each x, then the average probability of error given by (1.67) will also be minimized. Thus, the Bayes' rule minimizes the *average* probability of a classification error. In the case of two-class classification,

$$P(\text{classification error} \mid x) = \min(P(\omega_1 \mid x), P(\omega_2 \mid x)). \tag{1.68}$$

Using Bayes' rule (1.64),

$$P(\text{classification error} \mid x) = \min\left(\frac{P(x \mid \omega_1)P(\omega_1)}{P(x)}, \frac{P(x \mid \omega_2)P(\omega_2)}{P(x)}\right). \tag{1.69}$$

Note that $P(x)$ is not relevant for the final decision. It is a scaling only, and in the case of two-class decisions the Bayes' classification rule becomes

Decide

$$\begin{aligned} &\text{class } \omega_1 \quad \text{if} \quad P(x \mid \omega_1)P(\omega_1) > P(x \mid \omega_2)P(\omega_2), \\ &\text{class } \omega_2 \quad \text{if} \quad P(x \mid \omega_2)P(\omega_2) > P(x \mid \omega_1)P(\omega_1). \end{aligned} \tag{1.70a}$$

By making such a decision the probability of a classification error and consequently the *average* probability of a classification error will be minimized. One obtains another common form of this rule by using the *likelihood ratio* $\Lambda(x) = P(x \mid \omega_1)/P(x \mid \omega_2)$:

Decide

$$\begin{aligned} &\text{class } \omega_1 \quad \text{if} \quad \Lambda(x) = \frac{P(x \mid \omega_1)}{P(x \mid \omega_2)} > \frac{P(\omega_2)}{P(\omega_1)}, \\ &\text{class } \omega_2 \quad \text{if} \quad \Lambda(x) = \frac{P(x \mid \omega_1)}{P(x \mid \omega_2)} < \frac{P(\omega_2)}{P(\omega_1)}. \end{aligned} \tag{1.70b}$$

The decision rule given by (1.70b) can also be rewritten as

$$\Lambda(x) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{P(\omega_2)}{P(\omega_1)}. \tag{1.70c}$$

For equal prior probabilities, a threshold of likelihood ratio is equal to 1, and this rule becomes identical to the maximum likelihood decision criterion.

A good practical point about this rule is that both the prior probabilities $P(\omega_i)$ and the class-conditional probability-density functions $P(x \mid \omega_i)$ can be more easily estimated from data than the posterior probability $P(\omega_i \mid x)$ on which the whole rule is based.

General Bayesian Classification Real pattern recognition problems today often involve patterns belonging to more than two classes and high-dimensional feature vectors. In the case of handwritten numerals there are ten different classes, and using a single feature as in the previous case of two-class classification, it would be relatively difficult to separate all ten numbers reliably. Suppose that in addition to the 1's and 0's, one wants to classify the handwritten number 8, as shown in figure 1.29.

Now, the single feature x_1 (the total area covered by the character) is insufficient to classify all three numbers, since the 0's and the 8's seem to cover almost the same total grid area. Defining another feature x_2 as the sum of the areas of the character on the diagonal grid cells and combining these two features in the two-dimensional feature vector may suffice for the classification of all three numerals.

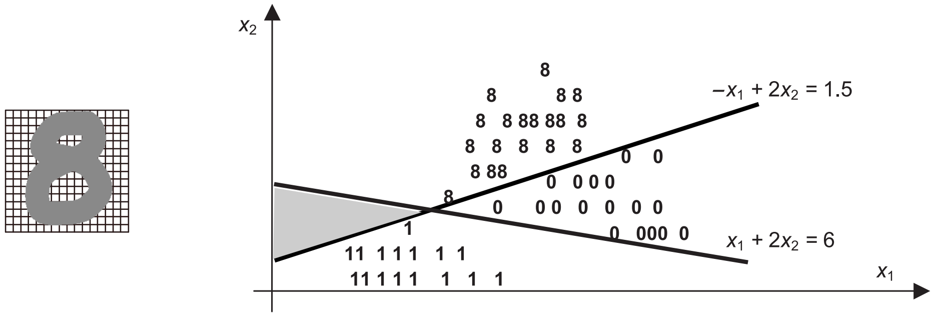


Figure 1.29
Left, typical sample of a handwritten number 8 on a 16 × 16 grid. *Right*, the decision regions and decision boundaries for a three-class character (1, 0, and 8) recognition problem.

Figure 1.29 depicts the positions of the training patterns in two-dimensional feature space. Despite its simplicity, this problem involves all the relevant concepts for solving problems with higher-dimensional feature vectors and more than three classes. By means of this introductory multifeature and multiclass example, the theory of classification is developed in general terms and later applied to cases involving normal or Gaussian distributions.

The two straight lines shown in figure 1.29 are the *decision boundary functions* that divide the feature space into disjoint decision regions. The latter can be readily associated with three given classes. The shaded area is a so-called indecision region in this problem, and the patterns falling in this region would be not assigned to any class.

When objects belong to more classes (say k, and for numerals $k = 10$), we have

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_k\}. \tag{1.71}$$

Now, Bayes' classification rule will be similar to the rule for two classes. In the case of multiclass and multifeature tasks, $P(\omega_i)$ denotes the prior probability that the given pattern belongs to a class ω_i, and it corresponds to the fraction of characters in an ith class. The class-conditional probability-density function is denoted for all k classes by $P(\mathbf{x} \mid \omega_i)$ and the joint probability-density function by $P(\omega_i, \mathbf{x})$, $i = 1, \ldots, k$. $P(\omega_i, \mathbf{x})$ is the probability-density that a pattern is in class ω_i and has a feature vector value $\mathbf{x}$. The conditional probability function $P(\omega_i \mid \mathbf{x})$ is the posterior probability that a pattern belongs to a class ω_i given that the observed value of a feature is $\mathbf{x}$, and

$$\sum_{i=1}^{k} P(\omega_i \mid \mathbf{x}) = 1. \tag{1.72}$$

As in the two-class case the prior and posterior probabilities are connected by

$$\begin{aligned} P(\omega_i, \mathbf{x}) &= P(\omega_i \,|\, \mathbf{x})P(\mathbf{x}), \qquad i = 1, 2, \ldots, k, \\ P(\omega_i, \mathbf{x}) &= P(\mathbf{x} \,|\, \omega_i)P(\omega_i), \qquad i = 1, 2, \ldots, k, \end{aligned} \tag{1.73}$$

where $P(\mathbf{x})$ is the unconditional probability-density function for a feature vector $\mathbf{x}$:

$$\begin{aligned} P(\mathbf{x}) &= \sum_{i=1}^{k} P(\mathbf{x} \,|\, \omega_i)P(\omega_i) \\ &= P(\mathbf{x} \,|\, \omega_1)P(\omega_1) + P(\mathbf{x} \,|\, \omega_2)P(\omega_2) + \cdots + P(\mathbf{x} \,|\, \omega_k)P(\omega_k). \end{aligned} \tag{1.74}$$

From (1.73) follows Bayes' theorem for a multifeature and multiclass case

$$P(\omega_i \,|\, \mathbf{x}) = \frac{P(\mathbf{x} \,|\, \omega_i)P(\omega_i)}{P(\mathbf{x})}, \qquad i = 1, 2, \ldots, k, \tag{1.75}$$

or

$$P(\omega_i \,|\, \mathbf{x}) = \frac{P(\mathbf{x} \,|\, \omega_i)P(\omega_i)}{\sum_{i=1}^{k} P(\mathbf{x} \,|\, \omega_i)P(\omega_i)} = \frac{P(\mathbf{x} \,|\, \omega_i)P(\omega_i)}{P(\mathbf{x} \,|\, \omega_1)P(\omega_1) + \cdots + P(\mathbf{x} \,|\, \omega_k)P(\omega_k)}. \tag{1.76}$$

Now, for a multifeature and multiclass case, Bayes' classification rule can be generalized as follows:

Assign a pattern to a class ω_i having the largest value of the posterior conditional probability $P(\omega_i \,|\, \mathbf{x})$ for a given feature $\mathbf{x}$.

In other words, assign a given pattern with an observed feature vector $\mathbf{x}$ to a class ω_i when

$$P(\omega_i \,|\, \mathbf{x}) > P(\omega_j \,|\, \mathbf{x}), \qquad j = 1, 2, \ldots, k, \ i \neq j. \tag{1.77}$$

Within the framework of learning from data it is much easier to estimate prior probability and class-conditional probability-density functions than the posterior probability itself. Therefore, a Bayes' classification rule (1.77) for a multifeature and multiclass case should be expressed as follows:

For a given feature vector $\mathbf{x}$, decide class ω_i if

$$P(\mathbf{x} \,|\, \omega_i)P(\omega_i) > P(\mathbf{x} \,|\, \omega_j)P(\omega_j), \qquad j = 1, 2, \ldots, k, \ i \neq j. \tag{1.78}$$

This final expression was obtained by using (1.75) after neglecting a scaling factor $P(\mathbf{x})$. Again, Bayes' classification rule is best for minimizing classification error.

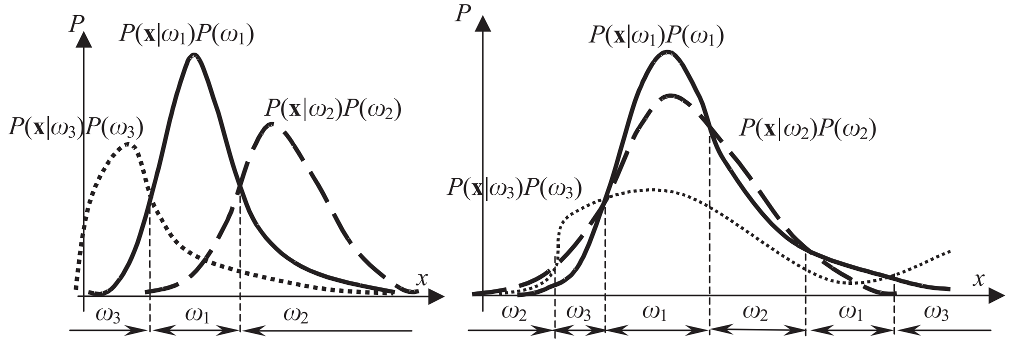


Figure 1.30
Bayes' classification rule for three classes may result in three single nonoverlapping decision regions (*left*) or in three nonoverlapping disjoint decision regions consisting of two (or generally more) nontouching regions (*right*). Other configurations of decision regions are possible, too.

Figure 1.30 illustrates (1.78) for three classes and, for the sake of clarity, only a single feature.

Statistical Classification Minimizing Risk For problems containing uncertainties, decisions are seldom based on probabilities alone. In most cases, one must be aware of the consequences (namely, the errors, potential profits or losses, penalties, or rewards involved). Thus, there is a need for combining probabilities and consequences, and for this reason, the concepts of *cost* or *loss*, and of *risk* (defined as expected loss) are introduced here. This is important in all decision-making processes. Introducing the minimization criterion involving potential loss into a classification decision made for a given true state of nature (for a given feature vector **x**) acknowledges the fact that misclassification of classes in some areas may be more costly than in others. The loss function can be very different in various applications, and its form depends upon the nature of the problem. Before considering the theory involving loss and risk functions, let us first study them inductively in example 1.9.

Example 1.9 Eleven boiler units in a plant are operating at different pressures. Three are operating at 101 bar, two at 102 bar, and others at 103, 105, 107, 110, 111, and 112 bar. A single process computer is, with the help of specific boiler pressure sensors (manometers), randomly reading the corresponding pressures, and the last eleven recorded samples are (101, 112, 101, 102, 107, 103, 105, 110, 102, 101, 111 bar). The pressures in the various boilers are mutually independent. In order to check a young engineer's understanding of this process and its random characteristics, his superior asks him to predict the next manometer reading under three different decision scenarios (l stands for a loss (function), r denotes a reward, and f designates a fine):

1. A reward of \$10 ($l = r = +10$) if the next reading is exactly the one he predicts, and a fine of \$1 ($l = f = -1$) if a different pressure is measured
2. A reward of \$10 ($l = r = +10$) if the next reading is exactly the one he predicts, and a fine equal in dollars to the size of his prediction error ($l = f = -|e|$)
3. A reward of \$10 ($l = r = +10$) if the next reading is exactly the one he predicts, and a fine equal in dollars to the square of his prediction error ($l = f = -(e^2)$)

The engineer needs to make a good decision because there are penalties for being wrong. His boss knows that if there were no penalty for being wrong or rewards for being right or close, nothing would be at stake and the engineer might just as well predict manometer readings of 50, 103.4, or 210.5 even though he knows that there will be no such readings. Therefore, in each case, the engineer should select the best possible decision to maximize expected profit (or to minimize expected loss).

Note that the different character of the loss functions in this example will lead to different decisions. In case 1, there is no reward for coming close to the correct manometer reading, so the size of the decision error does not matter. In case 2, the loss is proportional to the size of the error, and in case 3, the loss increases with the square of the error. (Note that the last fine resembles the sum-of-error-squares cost function.) What should the engineer decide in order to maximize expected profit?

In the first case, if he predicts a manometer reading of 101 (the mode, or most frequent observation, of the eleven samples), he stands to make \$10 with a probability of 3/11 and to lose \$1 with a probability of 8/11. Now, his expected profit (EP)[13] is

$$EP = \sum_{i=1}^{11} l_i p_i = 10\frac{3}{11} + (-1)\frac{8}{11} = \$2.$$

It can be easily verified that this is the best possible prediction given the loss functions $l = 10$ and $l = -1$ for a right and a wrong decision, respectively. Checking, for example, the prediction of 103 bar, one finds that $EP = \$0$. Note in this example that regardless of the value of the specific loss, the prediction of 101 bar will always be the best one, ensuring maximal profit or minimal loss. So, for example, if a correct prediction were rewarded by \$1 and a bad one fined by \$10, the expected profit would be negative:

$$EP = \sum_{i=1}^{11} l_i p_i = 1\frac{3}{11} + (-10)\frac{8}{11} = -\$7,$$

denoting the expected loss of \$7. Now, if the engineer predicts a reading of 103, the expected loss (or risk) is $1/11 - 10(10/11) = \$9$, and for a predicted reading of 102 the expected loss is \$8. Hence, the expected loss is again the smallest when 101 bar is predicted, given that the fine does not depend on the size of the estimation error. Note also that other predictions like 102.5 or 108 bar would now entail a *certain* loss of \$10. (Recall that the manometers can display only the integer values of the operating boiler pressures, and none is operating at these two pressures.) Thus, when there is no reward for coming close to some characteristics of the sample, the best decision is to use the *mode*, or the most frequent measurement.

In the second case, when the fine is proportional to the possible error, $l = f = -|e|$, it is the *median* (103 bar here) that maximizes the expected profit. Thus, if the engineer predicts that the next displayed reading will be 103, the fine will be \$2, \$1, \$2, \$4, \$7, \$8, or \$9, depending on whether the reading is 101, 102, 105, 107, 110, 111, or 112, and the expected profit is

$$EP = \sum_{i=1}^{11} l_i p_i = -2\frac{3}{11} - 1\frac{2}{11} + 10\frac{1}{11} - 2\frac{1}{11} - 4\frac{1}{11} - 7\frac{1}{11} - 8\frac{1}{11} - 9\frac{1}{11} = -\$2.55.$$

In other words, in this second case, the best decision cannot make any profit but would only entail the least possible loss of \$2.55. If the reward were \$38, the maximal expected profit would be \$0. Again, regardless of the reward assigned to the decision, the best possible prediction, given that the fine is proportional to the size of the prediction error, is a median (103 bar).

The expected fine or loss would be greater for any number other than the median. For instance, if the engineer predicts that the next reading will be 105, the *mean* of the eleven possible readings, the fine will be \$4, \$3, \$2, \$2, \$5, \$6, or \$7, depending on whether the reading is 101, 102, 103, 107, 110, 111, or 112, and the expected profit is

$$EP = \sum_{i=1}^{11} l_i p_i = -4\frac{3}{11} - 3\frac{2}{11} - 2\frac{1}{11} + 10\frac{1}{11} - 2\frac{1}{11} - 5\frac{1}{11} - 6\frac{1}{11} - 7\frac{1}{11} = -\$2.73.$$

Case 3 describes the scenario when the fine increases quadratically (rapidly) with the size of the error. This leads naturally to the *method of least squares*, which plays a very important role in statistical theory. It is easy to verify that for such a loss function the best possible prediction is the mean, 105, of the eleven sample manometer readings. The engineer finds that the fine will be \$16, \$9, \$4, \$4, \$25, \$36, or \$49, depending on whether the reading is 101, 102, 103, 107, 110, 111, or 112, and the expected profit is

$$EP = \sum_{i=1}^{11} l_i p_i = -16\frac{3}{11} - 9\frac{2}{11} - 4\frac{1}{11} + 10\frac{1}{11} - 4\frac{1}{11} - 25\frac{1}{11} - 36\frac{1}{11} - 49\frac{1}{11} = -\$12.10.$$

Again, in this third predicted scenario the best decision, to predict the mean, cannot make any profit but would only entail the least expected loss of \$12.10, given the reward of only \$10 for the correct prediction. It is left to the reader to verify this claim by calculating the expected profit (or loss) for any other possible decision. Note that, in the case when the fine increases quadratically with the size of the error, the expected profit is \$0 only if the reward is \$143.

The final decision (or simply a result) depends on the loss function used. As mentioned in section 1.3, the best solution depends upon the norm applied. Problems 1.8 and 1.9 illustrate this important observation in a nice graphical way.

The last two scenarios indicate that the reward defined by the engineer's superior is not very generous. But one can hope that using the correct prediction strategy—mode, median, and mean are the best decisions to maximize expected profit given various loss functions—will benefit the engineer more in his future professional life than his superior's present financial offer. ■

Now, these questions of the best decisions in classification tasks while minimizing risk (expected loss) can be set into a more general framework. First, define a loss function

$$L_{ji} = L(\text{decision class}_j \mid \text{true class}_i) \tag{1.79}$$

as a cost or penalty for assigning a pattern to a class ω_j when a true class is ω_i. In the case of an l-class classification problem, define an $l \times l$ loss matrix $\mathbf{L}$ as

$$\mathbf{L} = \begin{bmatrix} L_{11} & L_{12} & \dots & L_{1l} \\ L_{21} & L_{22} & \dots & L_{2l} \\ \vdots & \vdots & \vdots & \vdots \\ L_{l1} & L_{l2} & \dots & L_{ll} \end{bmatrix}. \tag{1.80}$$

A loss matrix $\mathbf{L}$, or the selection of the L_{ij}, is highly problem-dependent. At this point, selecting specific penalties or rewards is less important than understanding the concept of risk that originates from decision theory while combining probabilities with consequences (penalties or rewards). Recall that until now the best decision strategy was based only on the posterior probability $P(\omega_i \mid \mathbf{x})$, and using Bayes' rule,

$P(\omega_i \mid \mathbf{x})$ was expressed in terms of the prior probability $P(\omega_i)$ and a class-conditional probability-density $P(\mathbf{x} \mid \omega_i)$. Now, using the posterior probability $P(\omega_i \mid \mathbf{x})$ in a similar way as previously, one can define the *conditional risk*, or *expected (average) conditional loss*, associated with a decision that the observed pattern belongs to class ω_j when in fact it belongs to a class ω_i, $i = 1, 2, \ldots, l$; $i \neq j$:

$$R(\omega_j \mid \mathbf{x}) = \sum_{i=1}^{l} L(\text{decision class}_j \mid \text{true class}_i) P(\omega_i \mid \mathbf{x}) = \sum_{i=1}^{l} L_{ji} P(\omega_i \mid \mathbf{x}). \tag{1.81}$$

Thus, the conditional risk of making a decision ω_j, $R_j = R(\omega_j \mid \mathbf{x})$, is defined as the expectation of loss that is, through the use of $P(\omega_i \mid \mathbf{x})$, conditioned on the realization $\mathbf{x}$ of a feature vector. Hence, the best decision now should be a classification decision ω_j that minimizes the conditional risk $R_j, j = 1, 2, \ldots, l$. The *overall risk* is defined as the *expected loss* associated with a given classification decision and is considered for all possible realizations $\mathbf{x}$ of an n-dimensional feature vector from a feature vector space $\Re_{\mathbf{x}}$:

$$R = \int_{\Re_{\mathbf{x}}} R_j \, d\mathbf{x} = \int_{\Re_{\mathbf{x}}} R(\omega_j \mid \mathbf{x}) \, d\mathbf{x} = \int_{\Re_{\mathbf{x}}} \sum_{i=1}^{l} L_{ji} P(\omega_i \mid \mathbf{x}) \, d\mathbf{x}, \tag{1.82}$$

where the integral is calculated over an entire feature vector space $\Re_{\mathbf{x}}$.

The overall risk R is used as a classification criterion for the risk minimization while making a classification decision. The integral in (1.82) will be minimized if a classification decision ω_j minimizes the conditional risk $R(\omega_j \mid \mathbf{x})$ for each realization $\mathbf{x}$ of a feature vector.

This is a generalization of the Bayes' rule for minimization of a classification error (1.67), but here the minimization is of an overall risk R, or an expected loss. For this general classification problem we have the following Bayes' procedure and classification rule:

For a given feature vector $\mathbf{x}$ evaluate all conditional risks

$$R(\omega_j \mid \mathbf{x}) = \sum_{i=1}^{l} L_{ji} P(\omega_i \mid \mathbf{x})$$

for all possible classes ω_j, and choose a class (make a decision) ω_j for which the conditional risk $R(\omega_j \mid \mathbf{x})$ is minimal:

$$R(\omega_j \mid \mathbf{x}) < R(\omega_k \mid \mathbf{x}), \qquad k = 1, 2, \ldots, l, \; k \neq j. \tag{1.83}$$

Such classification decisions guarantee that the overall risk R will be minimal. This *minimal overall risk* is called *Bayes' risk*.

The last equation can be rewritten as

$$\sum_{i=1}^{l} L_{ji} P(\omega_i \mid \mathbf{x}) < \sum_{i=1}^{l} L_{ki} P(\omega_i \mid \mathbf{x}), \qquad k = 1, 2, \ldots, l, \ k \neq j, \tag{1.84}$$

and using Bayes' rule, (1.64),

$$P(\omega_i \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \omega_i) P(\omega_i)}{P(\mathbf{x})},$$

can be written as

$$\sum_{i=1}^{l} L_{ji} \frac{P(\mathbf{x} \mid \omega_i) P(\omega_i)}{P(\mathbf{x})} < \sum_{i=1}^{l} L_{ki} \frac{P(\mathbf{x} \mid \omega_i) P(\omega_i)}{P(\mathbf{x})}, \qquad k = 1, 2, \ldots, l, \ k \neq j. \tag{1.85}$$

Canceling the positive scaling factor $P(\mathbf{x})$ on both sides of this inequality yields the final practical form of Bayes' classification rule, which minimizes overall (Bayes') risk.

Choose a class (make a decision) ω_j for which

$$\sum_{i=1}^{l} L_{ji} P(\mathbf{x} \mid \omega_i) P(\omega_i) < \sum_{i=1}^{l} L_{ki} P(\mathbf{x} \mid \omega_i) P(\omega_i), \qquad k = 1, 2, \ldots, l, \ k \neq j. \tag{1.86}$$

For binary classification decision problems, Bayes' risk criterion (1.86) is given as follows. Let L_{ij} be the loss (cost) of making decision ω_i when ω_j is true. Then for the binary classification problem there are four possible losses:

L_{11} = loss (cost) of deciding ω_1 when, given $\mathbf{x}$, ω_1 is true,

L_{12} = loss (cost) of deciding ω_1 when, given $\mathbf{x}$, ω_2 is true,

L_{21} = loss (cost) of deciding ω_2 when, given $\mathbf{x}$, ω_1 is true,

L_{22} = loss (cost) of deciding ω_2 when, given $\mathbf{x}$, ω_2 is true,

or

$$\mathbf{L} = \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix}.$$

Note that there is nothing strange in associating a loss or cost with a correct decision. One can often set $L_{11} = L_{22} = 0$, but there will also be very common problems when both the correct and the wrong decisions are associated with certain costs. The Bayes' risk criterion will result in a (classification) decision when the expected loss or risk is minimal.

The risk (expected or average loss) that should be minimized is

$$R(\omega_1 \,|\, \mathbf{x}) = \sum_{i=1}^{2} L_{1i} P(\omega_i \,|\, \mathbf{x}) = L_{11} P(\omega_1 \,|\, \mathbf{x}) + L_{12} P(\omega_2 \,|\, \mathbf{x}),$$

$$R(\omega_2 \,|\, \mathbf{x}) = \sum_{i=1}^{2} L_{2i} P(\omega_i \,|\, \mathbf{x}) = L_{21} P(\omega_1 \,|\, \mathbf{x}) + L_{22} P(\omega_2 \,|\, \mathbf{x}),$$

or

$$\begin{aligned} R &= R(\omega_1 \,|\, \mathbf{x}) + R(\omega_2 \,|\, \mathbf{x}) \\ &= L_{11} P(\omega_1 \,|\, \mathbf{x}) + L_{12} P(\omega_2 \,|\, \mathbf{x}) + L_{21} P(\omega_1 \,|\, \mathbf{x}) + L_{22} P(\omega_2 \,|\, \mathbf{x}). \end{aligned} \tag{1.87}$$

The Bayes' risk formulation can be viewed as a generalization of a maximum-a-posteriori (MAP), or probability-of-error, decision criterion as given by (1.67). To show that, (1.67) can be rewritten as

$$\begin{aligned} P(\text{classification error}) &= \int_{-\infty}^{+\infty} P(\text{classification error} \,|\, x) P(x)\, dx \\ &= \int_{-\infty}^{+\infty} (P(\omega_1 \,|\, \mathbf{x}) + P(\omega_2 \,|\, \mathbf{x})) P(\mathbf{x})\, d\mathbf{x}. \end{aligned}$$

Clearly, by minimizing the probability of misclassification for each $\mathbf{x}$ (as the classification rule given by (1.66) requires), the average probability of error given by (1.67) (and by the preceding equation) will also be minimized.

At the same time, by assigning the losses $L_{11} = L_{22} = 0$ and $L_{12} = L_{21} = 1$, the risk (1.87) becomes

$$R = 0 \cdot P(\omega_1 \,|\, \mathbf{x}) + 1 \cdot P(\omega_2 \,|\, \mathbf{x}) + 1 \cdot P(\omega_1 \,|\, \mathbf{x}) + 0 \cdot P(\omega_2 \,|\, \mathbf{x}) = P(\omega_1 \,|\, \mathbf{x}) + P(\omega_2 \,|\, \mathbf{x}),$$

and this is exactly the argument of the preceding integral. In other words, by minimizing a risk, given a *zero-one loss function*, the average probability of classification error is also minimized.

Thus, for a zero-one loss function, a classification (decision) by minimizing risk is identical to the Bayes' (maximum-a-posteriori) classification decision criterion that minimizes the average (expected) probability of classification error. Note that in dichotomization (two classes only or binary) tasks, as long as $L_{ii} = 0$, that is, $L_{11} = L_{22} = 0$, the risk minimization criterion can be nicely expressed in a known form of a likelihood ratio

$$\Lambda(x) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{L_{12}}{L_{21}} \frac{P(\omega_2)}{P(\omega_1)}. \tag{1.88}$$

Hence, whenever the likelihood ratio $\Lambda(x) = P(x \mid \omega_1)/P(x \mid \omega_2)$ is larger than the product of the two ratios on the right-hand side of (1.88), the decision will be class 1. The last expression follows from (1.86), or after applying Bayes' theorem (1.64) in (1.87). Note that the costs L_{12} and L_{21} do not necessarily have to be equal to 1 now. Also, both the MAP (Bayes' decision criterion) and the maximum likelihood criterion are just special cases of the risk minimization criterion (1.88). Namely, the MAP rule follows when $L_{12} = L_{21} = 1$, and the maximum likelihood criterion results when $L_{12} = L_{21} = 1$ and $P(\omega_1) = P(\omega_2)$.

Finally, note that in more general multiclass problems, the zero-one loss matrix **L** (1.80) is given as

$$\mathbf{L} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & \cdots & 0 \end{bmatrix}. \tag{1.89}$$

Decision Regions and Discriminant Functions Pattern recognition systems perform multiclass, multifeature classification regardless of the type of decision rule applied. Recall that there are various decision rules that, depending on information about the patterns available, may be applied. Six different rules were listed at the beginning of section 1.4.2, and both the Bayes' rule for minimizing the average probability of error and the Bayes' rule for minimizing risk have been studied in more detail.

A pattern classifier assigns the feature vectors $\mathbf{x}$ to one of a number of possible classes ω_i, $i \in \{1, 2, \ldots, l\}$, and in this way partitions feature space into line segments, areas, volumes, and hypervolumes, which are decision regions $R_1, R_2, \ldots, R_l$, in the case of one-, two-, three-, or higher-dimensional features, respectively. All feature vectors belonging to the same class are ideally assigned to the same category in a decision region. The decision regions $\boldsymbol{R}_i$ are often single nonoverlapping volumes or hypervolumes, and decision regions of the same class may also be disjoint, consisting of two or more nontouching regions (see fig. 1.30). The boundaries between adjacent regions are called decision boundaries because classification decisions change

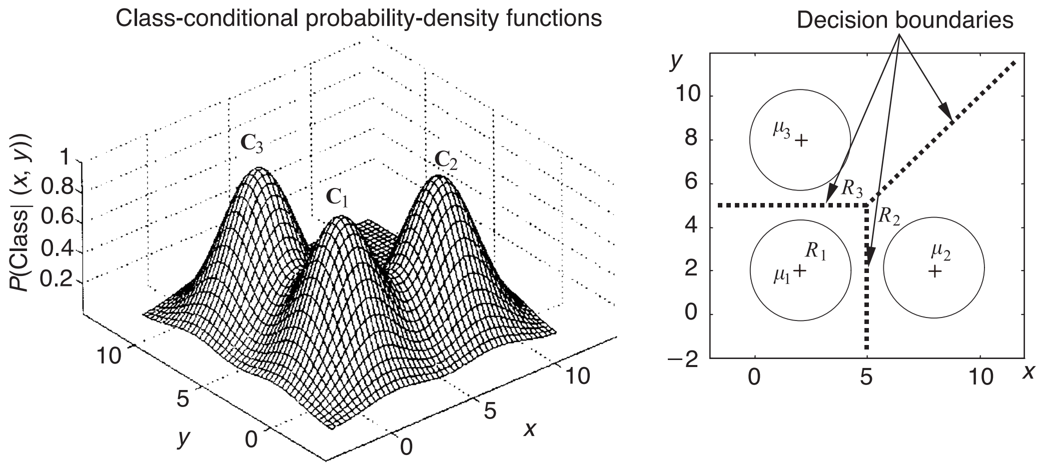


Figure 1.31
Class-conditional probability-density functions (*left*) and decision regions R_i and decision boundaries (*right*) for three classes result in three single nonoverlapping decision regions.

across boundaries. These class boundaries are points, straight lines or curves, planes or surfaces, and hyperplanes or hypersurfaces in the case of one-, two-, three- and higher-dimensional feature space, respectively. In the case of straight lines, planes, and hyperplanes, the decision boundaries are linear.

Figure 1.31 shows three class-conditional probability-density functions (likelihood functions) as well as the partitioning of two-dimensional feature space into three separated decision areas. The likelihood functions are three normal distributions with equal covariance matrices and centers at (2, 2), (8, 2) and (2, 8). Because of the equal covariance matrices, the three decision boundaries are straight lines (why this is so is explained later). In the case of three- and higher-dimensional feature space, the decision regions are volumes and hypervolumes, respectively, and visualization is no longer possible. Nevertheless, the classification decision procedures and the underlying theories are the same. The optimal classification strategy will most typically be the one that minimizes the probability of classification error, and the latter will be minimized if, for $P(\mathbf{x} \mid \omega_1)P(\omega_1) > P(\mathbf{x} \mid \omega_2)P(\omega_2)$, $\mathbf{x}$ is chosen to be in the region R_1.

More generally, classification decisions based on feature vector $\mathbf{x}$ may be stated using a set of explicitly defined *discriminant functions*

$$d_i(\mathbf{x}), \qquad i = 1, 2, \ldots, l, \tag{1.90}$$

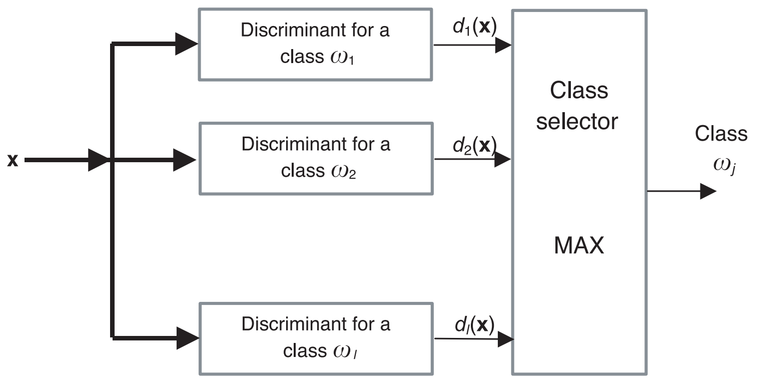

x
Discriminant for a class ω_1
Discriminant for a class ω_2
Discriminant for a class ω_l
$d_1(\mathbf{x})$
$d_2(\mathbf{x})$
$d_l(\mathbf{x})$
Class selector
MAX
Class ω_j

Figure 1.32
Discriminant classifier for multiclass pattern recognition.

where each discriminant is associated with a particular recognized class ω_i, $i = 1, 2, \ldots, l$.

The discriminant type of classifier (i.e., the classifier designed using the discriminant functions) assigns a pattern with feature vector $\mathbf{x}$ to a class ω_j for which the corresponding discriminant value d_j is the largest:

$$d_j(\mathbf{x}) > d_i(\mathbf{x}), \quad \text{for all } i = 1, 2, \ldots, l, \ i \neq j. \tag{1.91}$$

Such a discriminant classifier can be designed as a system comprising a set of l discriminants $d_i(\mathbf{x})$, $i = 1, 2, \ldots, l$, associated with each class ω_i, $i = 1, 2, \ldots, l$, along with a module that selects the largest-value discriminant as a recognized class (see fig. 1.32).

Note that the classification is based on the largest discriminant function $d_j(\mathbf{x})$ regardless how the corresponding discriminant functions are defined. Therefore, any monotonic function of a discriminant function $f(d(\mathbf{x}))$ will provide identical classification because of the fact that for the monotonic function $f(\cdot)$, the maximal $d_j(\mathbf{x})$ gives rise to the maximal $f(d_j(\mathbf{x}))$. It may be useful to understand this basic property of discriminant functions. If some $d_i(\mathbf{x})$, $i = 1, \ldots, l$, are the discriminant functions for a given classifier, so also are the functions $\ln d_i(\mathbf{x})$, $d_i(\mathbf{x}) + C$, or $Cd_i(\mathbf{x})$ for any class-independent constant C. This is widely explored in classification theory by using the natural logarithmic function $\ln d(\mathbf{x})$ as a discriminant function. Thus, in the case of a Bayes' classifier, instead of

$$d_i(\mathbf{x}) = P(\omega_i \,|\, \mathbf{x}) = P(\mathbf{x} \,|\, \omega_i)P(\omega_i)$$

the natural logarithm of $P(\omega_i \mid \mathbf{x})$ is used as a discriminant function, that is, the discriminant function is defined as

$$\begin{aligned} d_i(\mathbf{x}) &= \ln P(\omega_i \mid \mathbf{x}) = \ln(P(\mathbf{x} \mid \omega_i)P(\omega_i)) \\ &= \ln P(\mathbf{x} \mid \omega_i) + \ln P(\omega_i), \qquad i = 1, 2, \ldots, l. \end{aligned} \tag{1.92}$$

Discriminant functions define the decision boundaries that separate decision regions. Decision boundaries between neighboring regions R_j and R_i are obtained by equalizing the corresponding discriminant functions:

$$d_j(\mathbf{x}) = d_i(\mathbf{x}). \tag{1.93}$$

These boundaries between the decision regions are the points, lines or curves, planes or surfaces, and hyperplanes or hypersurfaces in the case of one-, two-, three-, and higher-dimensional feature vectors, respectively.

Depending upon the criteria for choosing which classification decision rule to apply, the discriminant function may be

$$d_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) \quad \text{in the case of Bayes' (MAP) classification,} \tag{1.94a}$$

$$d_i(\mathbf{x}) = P(\mathbf{x} \mid \omega_1) \quad \text{in the case of maximum likelihood classification,} \tag{1.94b}$$

$$d_i(\mathbf{x}) = P(\omega_i) \quad \text{in the case of maximum-a-priori classification,} \tag{1.94c}$$

$$d_i(\mathbf{x}) = -R(\omega_i \mid \mathbf{x}) \quad \text{in the case of Bayes' minimal risk classification,} \tag{1.94d}$$

$$i = 1, 2, \ldots, l.$$

Many other (not necessarily probabilistic) discriminant functions may also be defined. Note the minus sign in the last definition of the discriminant function, which denotes that the maximal value of the discriminant function $d_i(\mathbf{x})$ corresponds to the minimal conditional risk $R(\omega_i \mid \mathbf{x})$.

In the case of two-class or binary classification (dichotomization), instead of two discriminants $d_1(\mathbf{x})$ and $d_2(\mathbf{x})$ applied separately, typically a *dichotomizer* is applied, defined as

$$d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}). \tag{1.95}$$

A dichotomizer (1.95) calculates a value of a single discriminant function $d(\mathbf{x})$ and assigns a class according to the *sign* of this value. When $d_1(\mathbf{x})$ is larger than $d_2(\mathbf{x})$, $d(\mathbf{x}) > 0$ and a pattern $\mathbf{x}$ will be assigned to class 1, otherwise it is assigned to class 2. Thus, for Bayes' rule–based discriminant functions (1.94a), the dichotomizer for a binary classification is given as

$$d(\mathbf{x}) = P(\omega_1 \,|\, \mathbf{x}) - P(\omega_2 \,|\, \mathbf{x}) = P(\mathbf{x} \,|\, \omega_1)P(\omega_1) - P(\mathbf{x} \,|\, \omega_2)P(\omega_2).$$

Accordingly, the decision boundary between two classes is defined by

$$d_1(\mathbf{x}) = d_2(\mathbf{x}) \quad \text{or by} \quad d(\mathbf{x}) = 0. \tag{1.96}$$

Checking the geometrical meaning of (1.96) in the right graph in figure 1.33 for two-dimensional features, one sees that the separating function, or the decision boundary, is the intersecting curve (surface or hypersurface for three- and higher-dimensional features, respectively) between the dichotomizer $d(\mathbf{x})$ and the feature plane. Another useful form of the dichotomizer that uses a Bayes' (MAP) classification decision criterion can be obtained from (1.92) as follows:

$$\begin{aligned} d(\mathbf{x}) &= \ln P(\omega_1 \,|\, \mathbf{x}) - \ln P(\omega_2 \,|\, \mathbf{x}) = \ln(P(\mathbf{x} \,|\, \omega_1)P(\omega_1)) - \ln(P(\mathbf{x} \,|\, \omega_2)P(\omega_2)) \\ &= (\ln P(\mathbf{x} \,|\, \omega_1) + \ln P(\omega_1)) - (\ln P(\mathbf{x} \,|\, \omega_2) + \ln P(\omega_2)), \end{aligned}$$

or

$$d(\mathbf{x}) = \ln \frac{P(\mathbf{x} \,|\, \omega_1)}{P(\mathbf{x} \,|\, \omega_2)} + \ln \frac{P(\omega_1)}{P(\omega_2)}. \tag{1.97}$$

For a given feature vector $\mathbf{x}$, a dichotomizer $d(\mathbf{x})$ calculates a single value and assigns a class based on the sign of this value. In other words, because $d(\mathbf{x})$ is calculated as the difference $d_1(\mathbf{x}) - d_2(\mathbf{x})$, when $d(\mathbf{x}) > 0$ the pattern is assigned to class ω_1 and when $d(\mathbf{x}) < 0$ the pattern is assigned to class ω_2.

The next section takes up discriminant functions for normally distributed classes, which are very common. Solving classification tasks involving Gaussian examples can yield very useful closed-form expressions for calculating decision boundaries.

Discriminant Functions for Classification of Normally Distributed Classes In the case of normally distributed classes (Gaussian classes) discriminant functions are quadratic. These become linear (straight lines, planes, and hyperplanes for two-, three- and n-dimensional feature vectors, respectively) when the covariance matrices of corresponding classes are equal. The quadratic and linear classifiers belong to the group of *parametric classifiers* because they are defined in terms of Gaussian distribution parameters—mean vectors $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$.

Let us start with the simplest case of a binary classification problem in a one-dimensional feature space when two classes are generated by two Gaussian probability functions having the same variances $\sigma_1^2 = \sigma_2^2 = \sigma^2$ but different means $\mu_1 \neq \mu_2$ (see fig. 1.28, a classification of 1's and 0's). In this case, the class-conditional probability-density functions $P(x \mid \omega_i)$ are given as

$$P(x \mid \omega_1) = \frac{\exp\left(-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2\right)}{\sqrt{2\pi}\sigma_1}, \qquad P(x \mid \omega_2) = \frac{\exp\left(-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2\right)}{\sqrt{2\pi}\sigma_2},$$

and applying (1.97) results in

$$d(x) = \ln \frac{P(x \mid \omega_1)}{P(x \mid \omega_2)} + \ln \frac{P(\omega_1)}{P(\omega_2)} = \ln \frac{\exp\left(-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right)}{\exp\left(-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma}\right)^2\right)} + \ln \frac{P(\omega_1)}{P(\omega_2)},$$

or

$$d(x) = \frac{\mu_1 - \mu_2}{\sigma^2} x + \frac{\mu_2^2 - \mu_1^2}{2\sigma^2} + \ln \frac{P(\omega_1)}{P(\omega_2)}. \tag{1.98}$$

Hence, for a one-dimensional feature vector, given equal variances of normal class-conditional probability-density functions $P(x \mid \omega_i)$, the dichotomizer is linear (straight line). The decision boundary as defined by (1.96) or by $d(x) = 0$, is the point

$$x = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2}{\mu_2 - \mu_1} \ln \frac{P(\omega_1)}{P(\omega_2)}. \tag{1.99}$$

Note that in the equiprobable case when $P(\omega_1) = P(\omega_2)$, the decision boundary is a point x_{DB} in the middle of the class centers $x_{DB} = (\mu_1 + \mu_2)/2$. Otherwise, the decision boundary point x_{DB} is closer to the center of the less probable class. So, for example, if $P(\omega_1) > P(\omega_2)$, then $|\mu_2 - x_{DB}| < |\mu_1 - x_{DB}|$.

In the case of the multiclass classification problem in an n-dimensional feature space[14] when classes are generated according to Gaussian distributions with different covariance matrices $\mathbf{\Sigma}_1 \neq \mathbf{\Sigma}_2 \neq \cdots \neq \mathbf{\Sigma}_i$ and different means $\boldsymbol{\mu}_1 \neq \boldsymbol{\mu}_2 \neq \cdots \neq \boldsymbol{\mu}_i$, the class-conditional probability-density function $P(\mathbf{x} \mid \omega_i)$ is described by

$$P(\mathbf{x} \mid \omega_i) = \frac{1}{(2\pi)^{n/2} |\mathbf{\Sigma}_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right), \qquad i = 1, 2. \tag{1.100}$$

Now $\mathbf{x}$ and $\boldsymbol{\mu}_i$ are $(n, 1)$ vectors and the covariance matrices $\mathbf{\Sigma}_i$ are square and symmetric (n, n) matrices. $|\mathbf{\Sigma}|$ denotes the determinant of the covariance matrix. In the most general case for normally distributed classes, the discriminant function defined as $d(\mathbf{x}) = \ln P(\omega \mid \mathbf{x}) = \ln P(\mathbf{x} \mid \omega) P(\omega)$ becomes

$$d_i(\mathbf{x}) = \ln \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right) + \ln P(\omega_i), \quad i = 1, 2, \ldots, l,$$

or after expanding the logarithm,

$$d_i(\mathbf{x}) = -\frac{1}{2}\ln|\boldsymbol{\Sigma}_i| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{n}{2}\ln(2\pi) + \ln P(\omega_i), \qquad i = 1, 2, \ldots, l. \tag{1.101}$$

The constant term $n \ln(2\pi)/2$ is equal for all classes (i.e., it cannot change the classification), and consequently it can be eliminated in what results as a *quadratic discriminant function*,

$$d_i(\mathbf{x}) = -\frac{1}{2}\ln|\boldsymbol{\Sigma}_i| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) + \ln P(\omega_i), \qquad i = 1, 2, \ldots, l. \tag{1.102}$$

Decision boundaries (separation hypersurfaces) between classes i and j are the hyperquadratic functions in n-dimensional feature space (e.g., hyperspheres, hyperellipsoids, hyperparaboloids) for which $d_i(\mathbf{x}) = d_j(\mathbf{x})$. The specific form of discriminant functions and of decision boundaries is determined by the characteristics of covariance matrices.

The second term in (1.102) is the *Mahalanobis distance*, which calculates the distance between the feature vector $\mathbf{x}$ and the mean vector $\boldsymbol{\mu}_i$. Recall that for correlated features, covariance matrices $\boldsymbol{\Sigma}_i$ are nondiagonal symmetric matrices for which off-diagonal elements $\sigma_{ij}^2 \neq 0$, $i = 1, \ldots, l$, $j = 1, \ldots, l$, $i \neq j$.

The quadratic discriminant function is the most general discriminant in the case of normally distributed classes, and decisions are based on the Bayes' decision rule that minimizes the probability of error or the probability of misclassification. This is also known as the *minimum error rate classifier*.

The classification decision algorithm based on the quadratic Bayes' discriminants is now the following:

1. For given classes (feature vectors $\mathbf{x}$), calculate the mean vectors and the covariance matrices.
2. Compute the values of the quadratic discriminant functions (1.102) for each class.
3. Select a class ω_j for which $d_j(\mathbf{x}) = \max(d_i(\mathbf{x}))$, $i = 1, 2, \ldots, l$.

Example 1.10 illustrates how equation (1.102) applies to the calculation of quadratic discriminant functions. The computation of decision boundaries between two classes in the case of the two-dimensional feature vector $\mathbf{x}$ is also shown.

Example 1.10 Feature vectors $\mathbf{x}$ of two classes are generated by Gaussian distributions having parameters (covariance matrices $\boldsymbol{\Sigma}_i$ and means $\boldsymbol{\mu}_i$) as follows:

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.25 \end{bmatrix}, \qquad \boldsymbol{\mu}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.25 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{\mu}_2 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}.$$

Find the discriminant functions and decision boundaries. Classes are equiprobable ($P(\omega_i) = P(\omega) = P$). (Recall that the covariance matrices are diagonal when the features are statistically independent. The geometrical consequence of this fact can be seen in figure 1.34, where the principal axes of the contours of the Gaussian class densities are parallel to the feature axes.)

Quadratic discriminant functions for two given classes are defined by (1.102). The constant and the class-independent term $\ln P(\omega_i)$ can be ignored, and the two discriminants are obtained as

$$d_1(\mathbf{x}) = -\frac{1}{2}\ln|\Sigma_1| - \frac{1}{2}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 0.25 \end{bmatrix}^{-1} \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= 0.693 - \frac{1}{2}[x_1 \quad x_2]\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0.693 - 0.5(x_1^2 + 4x_2^2),$$

$$d_2(\mathbf{x}) = -\frac{1}{2}\ln|\Sigma_2| - \frac{1}{2}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0 \\ 2 \end{bmatrix}\right)^T \begin{bmatrix} 0.25 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0 \\ 2 \end{bmatrix}\right)$$

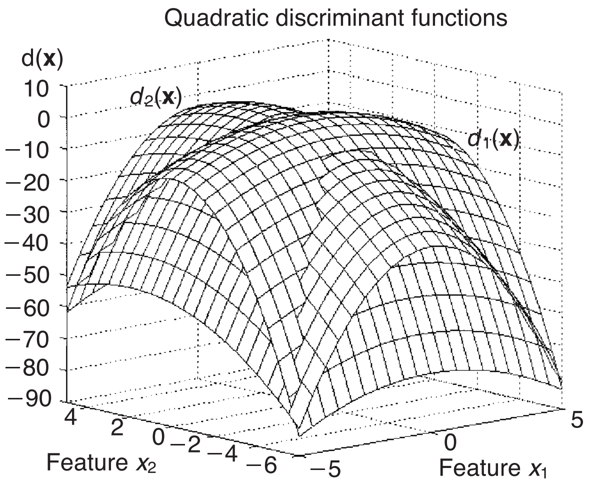
$$= 0.693 - \frac{1}{2}[x_1 \quad (x_2 - 2)]\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 - 2 \end{bmatrix} = 0.693 - 0.5(4x_1^2 + (x_2 - 2)^2).$$

Both discriminant functions and the dichotomizing discriminant function are shown in figure 1.33.

The decision boundary, or separation line, in the feature plane follows from $d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = 0$ as

$$d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = 1.5x_1^2 - 1.5x_2^2 - 2x_2 + 2 = 0.$$

Note that the decision regions in the feature plane, shown in figure 1.34, are nonoverlapping patches. The decision region for class 2 comprises two disjoint areas. It is not surprising that region R_2 is disjoint. This can be seen in figure 1.33, and it also



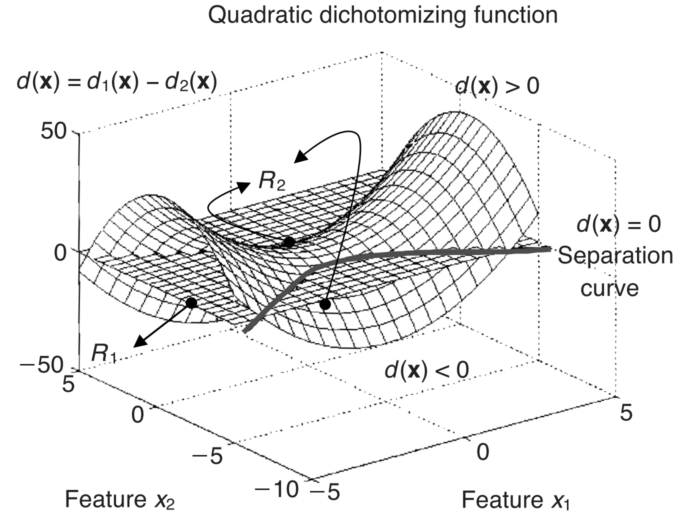


Figure 1.33
Classification of two Gaussian classes with different covariance matrices $\mathbf{\Sigma}_1 \neq \mathbf{\Sigma}_2$. *Top*, quadratic discriminant functions $d_1(\mathbf{x})$ and $d_2(\mathbf{x})$. *Bottom*, quadratic dichotomizing discriminant function $d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x})$. The decision boundary (separation curve) $d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = 0$ in the right graph is the intersection curve between the dichotomizer $d(\mathbf{x})$ and the feature plane. Note that the decision regions in the feature plane are nonoverlapping parts of the feature plane, but the decision region for class 2 comprises two disjoint areas.

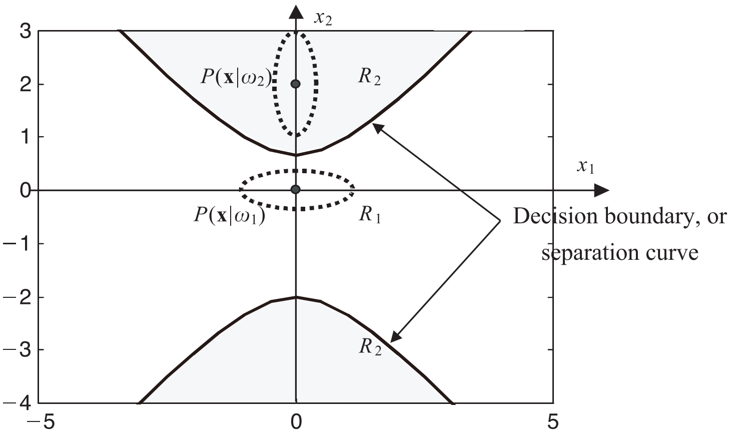


Figure 1.34
Quadratic decision boundary (separation curve) $d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = 0$ for two Gaussian classes with different covariance matrices $\boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$. This decision boundary is obtained by the intersection of the dichotomizing discriminant function $d(\mathbf{x})$ and the feature plane. Note that the decision region for class 2 comprises two disjoint areas.

follows from these considerations: the prior probabilities are equal, and the decision rule, being the minimum error rate classifier, chooses the class whose likelihood function $P(\mathbf{x} \mid \omega)$ is larger. Since the variance in the x_2 direction is larger for class 2 than for class 1, the class-conditional probability-density function (likelihood) $P(\mathbf{x} \mid \omega_2)$ is actually larger than $P(\mathbf{x} \mid \omega_1)$ for most of the lower half-plane in figure 1.34, despite the fact that all these points are closer (in the sense of the Euclidean distance) to the mean of class 1. The quadratic separation function shown in figure 1.34 is obtained from $d(\mathbf{x}) = 0$. All points in the feature plane for which $d(\mathbf{x}) < 0$ belong to class 1, and when $d(\mathbf{x}) > 0$ the specific pattern belongs to class 2. This can be readily checked analytically as well as in figures 1.33 and 1.34.

There are two simpler, widely used discriminant functions, or decision rules, that under certain assumptions follow from the quadratic discriminant function (1.102).

■

Linear Discriminant Function for Equal Covariance Matrices When the covariance matrices for all classes are equal ($\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, $i = 1, 2, \ldots, l$) so is the first term in (1.102) equal for all classes, and being class-independent, it can be dropped from (1.102), yielding a discriminant of the form

$$d_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) + \ln P(\omega_i), \qquad i = 1, 2, \ldots, l. \tag{1.103}$$

This discriminant function is linear, which can be readily seen from the expansion

$$\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)=\frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i-\frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\mathbf{x}+\frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i.$$

The quadratic term $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}$ is class-independent and can be dropped from this expansion. Furthermore, since the covariance matrix is symmetric, so is its inversion and, $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i=\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\mathbf{x}$. This results in a set of linear discriminant functions

$$d_i(\mathbf{x})=\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\mathbf{x}-\frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i+\ln P(\omega_i),\qquad i=1,2,\dots,l. \tag{1.104}$$

The classification decision algorithm for normally distributed classes having the same covariance matrix $\boldsymbol{\Sigma}$ is now the following:

1. For given classes (feature vectors $\mathbf{x}$), calculate the mean vectors and the covariance matrix $\boldsymbol{\Sigma}$.
2. Compute the values of the linear discriminant functions (1.104) for each class.
3. Select a class ω_j for which $d_j(\mathbf{x})=\max(d_i(\mathbf{x}))$, $i=1,2,\dots,l$.

Decision boundaries corresponding to $d_i(\mathbf{x})=d_j(\mathbf{x})$ are hyperplanes. In the case of a two-dimensional feature vector $\mathbf{x}$, these boundaries are straight lines in a feature plane. Linear discriminant functions and linear boundaries are closely related to neural network models (see chapter 3). Here the linear discriminant functions are presented in the "neural" form

$$d_i(\mathbf{x})=\mathbf{w}_i^T\mathbf{x}+w_{i0}, \tag{1.105}$$

where

$$\mathbf{w}_i^T=\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\quad\text{and}\quad w_{i0}=-\frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i+\ln P(\omega_i).$$

The decision boundary between classes $\boldsymbol{\Omega}_i$ and $\boldsymbol{\Omega}_j$ in neural form is given as a hyperplane,

$$d_i(\mathbf{x})-d_j(\mathbf{x})=\mathbf{w}\mathbf{x}+w_{ij0}=0, \tag{1.106}$$

where

$$\mathbf{w}=(\boldsymbol{\mu}_i^T-\boldsymbol{\mu}_j^T)\boldsymbol{\Sigma}^{-1}\quad\text{and}\quad w_{ij0}=-\frac{1}{2}(\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i-\boldsymbol{\mu}_j^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j)+\ln\frac{P(\omega_i)}{P(\omega_j)}.$$

It is straightforward to show that in the case of a one-dimensional feature vector x, equation (1.99), which defines the separation point, follows from (1.106). In the case of two feature patterns, (1.106) represents straight lines. Example 1.11 shows this.

Example 1.11 Figure 1.31 depicts the classification of three normally distributed classes that result in three linear separation lines and in the tessellation (partition) of a two-dimensional feature space (plane) into three separated decision regions. The likelihood functions of the three normal distributions have equal covariance matrices $\mathbf{\Sigma} = \mathbf{I}_2$ and centers (means) at $(2,2)$, $(8,2)$, and $(2,8)$. Check the validity of the right graph in figure 1.31 by applying (1.105) and (1.106) for the equiprobable classes.

Having identity covariance matrices and equiprobable classes (meaning that the last terms $\ln P(\omega_i)$ in (1.105) can be eliminated), three linear discriminant functions (planes) that follow from (1.105) are

$$d_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{i0} = \boldsymbol{\mu}_i^T\mathbf{x} - \tfrac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i, \qquad i = 1,2,3,$$

or

$$d_1(\mathbf{x}) = 2x_1 + 2x_2 - 4, \qquad d_2(\mathbf{x}) = 8x_1 + 2x_2 - 34, \qquad d_3(\mathbf{x}) = 2x_1 + 8x_2 - 34.$$

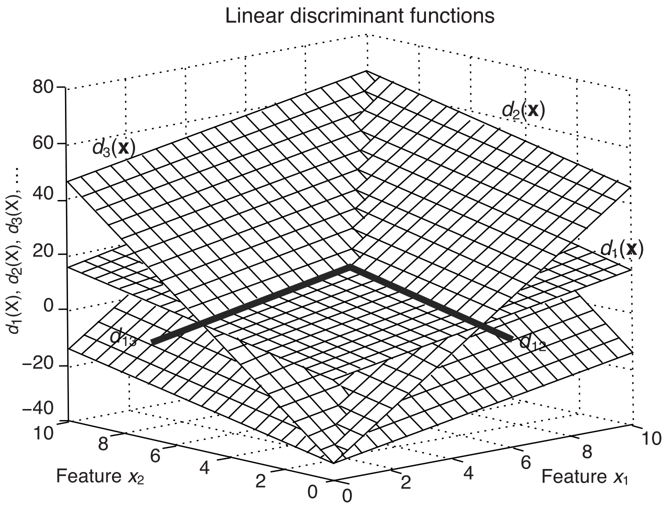


Figure 1.35
Linear discriminant functions (decision planes $d_i(\mathbf{x})$ in the case of two-dimensional features) for three Gaussian classes with the same covariance matrix $\mathbf{\Sigma} = \mathbf{I}_2$ as in figure 1.31. Three corresponding decision boundaries obtained as the planes' intersections divide the feature plane into three single nonoverlapping decision regions. (The two visible separation lines, d_{12} and d_{13}, are depicted as solid straight lines.

Similarly, the decision boundaries or the separation lines that follow from (1.106) are

$$d_{12}(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = -6x_1 + 30 = 0, \quad \text{or} \quad x_1 = 5,$$

$$d_{13}(\mathbf{x}) = d_1(\mathbf{x}) - d_3(\mathbf{x}) = -6x_2 + 30 = 0, \quad \text{or} \quad x_2 = 5,$$

$$d_{23}(\mathbf{x}) = d_2(\mathbf{x}) - d_3(\mathbf{x}) = 6x_1 - 6x_2 = 0, \quad \text{or} \quad x_2 = x_1.$$

Three discriminant planes $d_i(\mathbf{x})$, $i = 1, 2, 3$, together with the two visible decision boundary lines that separate given classes, are shown in figure 1.35. (All three separation lines are shown in the right graph in figure 1.31.) ■

Minimum Mahalanobis Distance and Euclidean Distance Classifiers Two particular cases for the classification of normally distributed (Gaussian) classes follow after applying several additional assumptions regarding their class distribution properties. If there are equal covariance matrices for all classes ($\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, $i = 1, 2, \dots, l$) and also equal prior probabilities for all classes ($P(\omega_i) = P(\omega) = P$), then the second term on the right-hand side of (1.103) can be eliminated. Additionally, being class-independent, the constant $1/2$ can be neglected, and this results in the following discriminant functions:

$$d_i(\mathbf{x}) = -(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i), \qquad i = 1, 2, \dots, l. \tag{1.107}$$

Thus, classification based on the maximization of these discriminants will assign a given pattern with a feature vector $\mathbf{x}$ to a class ω_k, for which the Mahalanobis distance $(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)$ of $\mathbf{x}$ to the mean vector $\boldsymbol{\mu}_k$ is the smallest. Note that because of the minus sign in (1.107), minimization of the Mahalanobis distance corresponds to maximization of the discriminant function $d(\mathbf{x})$. In other words, a pattern will be assigned to the closest class center $\boldsymbol{\mu}_k$ in the Mahalanobis sense. Note also that the Mahalanobis distance is relevant for correlated features, or when the off-diagonal elements of the covariance matrix Σ are not equal to zero (or when $\boldsymbol{\Sigma}$ is not a diagonal matrix, i.e., when $\sigma_{ij}^2 \neq 0$, $i = 1, \dots, l$, $j = 1, \dots, l$, $i \neq j$).

The classifier (1.107) is called a *minimum Mahalanobis distance classifier*. In the same way as (1.104), the Mahalanobis distance classifier can be given in linear form as

$$d_i(\mathbf{x}) = \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \tfrac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i, \qquad i = 1, 2, \dots, l. \tag{1.108}$$

Applying an even more restrictive assumption for the equiprobable classes ($P(\omega_i) = P(\omega) = P$), namely, assuming that the covariance matrices for all classes are not only equal but are also diagonal matrices, meaning that the features are statistically independent ($\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_n$, $i = 1, 2, \dots, l$), one obtains the simple discriminant functions

$$d_i(\mathbf{x}) = -\frac{(\mathbf{x}-\boldsymbol{\mu}_i)^T(\mathbf{x}-\boldsymbol{\mu}_i)}{\sigma^2} = -\frac{\|\mathbf{x}-\boldsymbol{\mu}_i\|^2}{\sigma^2} = -\|\mathbf{x}-\boldsymbol{\mu}_i\|^2, \qquad i = 1,2,\dots,l. \quad (1.109)$$

The class-independent coefficient variance σ^2 is neglected in the final stage of the classifier design.

Thus (1.109) represents the *minimum Euclidean distance classifier* because the discriminants (1.109) will assign a given pattern with a feature vector $\mathbf{x}$ to a class ω_k for which the *Euclidean distance* $\|\mathbf{x}-\boldsymbol{\mu}_k\|$ of $\mathbf{x}$ to the mean vector $\boldsymbol{\mu}_k$ is the smallest. As in the preceding case of a nondiagonal covariance matrix, and because of the minus sign, the minimal Euclidean distance will result in a maximal value for the discriminant function $d_i(\mathbf{x})$, as given in (1.109). In other words, a minimum (Euclidean) distance classifier assigns a pattern with feature $\mathbf{x}$ to the closest class center $\boldsymbol{\mu}_k$.

A linear form of the minimum distance classifier (neglecting a class-independent variance σ^2) is given as

$$d_i(\mathbf{x}) = \boldsymbol{\mu}_i^T\mathbf{x} - \tfrac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i, \qquad i = 1,2,\dots,l. \quad (1.110)$$

The algorithm for both the Mahalanobis and the Euclidean distance classifiers is the following. The mean vectors for all classes $\boldsymbol{\mu}_i$, $i = 1,2,\dots,l$, a feature vector $\mathbf{x}$, and a covariance matrix $\boldsymbol{\Sigma}$ are given.

1. Calculate the values of the corresponding distances between $\mathbf{x}$ and means $\boldsymbol{\mu}_i$ for all classes. The Mahalanobis distance for correlated classes is

$$D = (\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k), \qquad \boldsymbol{\Sigma} \text{ nondiagonal.}$$

The Euclidean distance for statistically independent classes is

$$D = \|\mathbf{x}-\boldsymbol{\mu}_k\|, \qquad \boldsymbol{\Sigma} \text{ diagonal.}$$

2. Assign the pattern to the class ω_k for which the distance D is minimal.

Both of these minimum distance classifiers are minimum error rate classifiers. In other words, for given assumptions, they are the Bayes' minimal probability of an error classifier. Furthermore, for both classifiers the mean vectors $\boldsymbol{\mu}_i$, $i = 1,\dots,l$, act as *templates* or *prototypes* for l classes. By measuring the distances between each new pattern $\mathbf{x}$ and these centers, each new feature vector $\mathbf{x}$ is matched to these templates. Hence, both the Mahalanobis and the Euclidean distance classifiers belong to the group of *template matching classifiers*.

Template matching is a classic, natural approach to pattern classification. Typically, noise-free versions of patterns are used as templates or as the means of the corresponding classes. To classify a previously unseen and noisy pattern, simply compare it to given templates (means) and assign the pattern to the closest template (mean, center). Template matching works well when the variations within a class are due to additive noise. But there are many other possible noises in classification and decision-making problems. For instance, regarding geometrically distorted patterns, some common distortions of feature vectors are translation, rotation, shearing, warping, expansion, contraction, and occlusion. For such patterns, more sophisticated techniques must be used. However, these are outside the scope of this book.

Limitations and Deficiencies of the Classical Bayesian Approach Almost all the methods and techniques presented in the preceding sections have a serious practical limitation. In order to apply the most general Bayes' minimum cost or minimum risk procedure (and related approaches) practically everything about the underlying analyzed process must be known. This includes the priors $P(\omega_i)$ and the class-conditional probability-densities (or likelihoods) $P(\mathbf{x} \mid \omega_i)$ as well as the costs of making errors $L(\omega_j \mid \omega_i)$. The fact that pattern recognition and regression problems are of random character and therefore expressed in probabilistic terms does not make the task simpler.

Suppose one wants to perform fault detection in engineering or in medical diagnosis. One must know how probable the different faults or symptoms (classes) are a priori ($P(\omega_i)$ are required). In other words, the prior probability of a system under investigation to experience different faults must be known. This is an intricate problem because very often it is difficult to distinguish priors $P(\omega_i)$ from class-conditional probability-densities $P(\mathbf{x} \mid \omega_i)$. One remedy is to use decision rules that do not contain priors (or to ignore them). The maximum likelihood classification rule is such a rule. Similarly, in regression, other approaches that require fewer assumptions can be tried, such as Markov estimators or the method of least squares.

The amount of assumed initial knowledge available on the process under investigation decreases in the following order: for the Bayes' procedure one should know everything; for the maximum likelihood approach one should know the class-conditional probability-density functions (likelihoods); for the Markov techniques in regression problems one should know the covariance matrix of noise; and for the least squares method one need only assume that random processes can be approximated sufficiently by the model chosen. But even in such a series of simplifications one must either know some distribution characteristics in advance or estimate the means, covariance matrices, or likelihoods (class-conditional probability densities) by using training patterns.

However, there are practical problems with density estimation approaches. To implement even the simplest Euclidean minimum distance classifier, one must know the mean vectors (centers or templates) $\boldsymbol{\mu}_i$, $i = 1, \ldots, l$, for all the classes. For this approach it is assumed that the underlying data-generating distribution is a Gaussian one, that the features are not correlated, and that the covariance matrices are equal. (This is too many assumptions for the simplest possible method.) To take into account eventually correlated features, or in considering the effects of scaling and linear transformation of data for which the Euclidean metric is not appropriate, the Mahalanobis metric should be used. However, in order to implement a minimum Mahalanobis distance classifier, both the mean vectors and the covariance matrices must be known. Recall that all that is typically available is training data and eventually some previous knowledge. Usually, this means estimating all these parameters from examples of patterns to be classified or regressed.

Yet this is exactly what both statistical learning theory (represented by support vector machines) and neural networks are trying to avoid. This book follows the approach of bypassing or dodging density estimation methods. Therefore, there is no explicit presentation of how to learn means, covariance matrices, or any other statistics from training data patterns. Instead, the discussion concerns SVM and NN tools that provide novel techniques for acquiring knowledge from training data (patterns, records, measurements, observations, examples, samples).

However, it should be stressed that the preceding approaches (which in pattern recognition problems most often result in quadratic or linear discriminant functions, decision boundaries and regions, and in regression problems result in linear approximating functions) still are and will remain very good theoretical and practical tools if the mentioned assumptions are valid. Very often, in modern real-world applications, many of these postulates are satisfied only approximately. However, even when these assumptions are not totally sound, quadratic and linear discriminants have shown acceptable performance as classifiers, as have linear approximators in regression problems. Because of their simple (linear or quadratic) structure, these techniques do not overfit the training data set, and for many regression tasks they may be good starting points or good first estimates of regression hyperplanes. In classification, they may indicate the structure of complex decision regions that are typically hypervolumes in n-dimensional space of features.

Problems

1.1. Let $\mathbf{x} = [-2 \quad 1]^T$, $\mathbf{y} = [-3 \quad 1]^T$, $\mathbf{v} = [4 \quad -3 \quad 2]^T$, $\mathbf{w} = [5 \quad 6 \quad -1]^T$.

a. Compute $\dfrac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}}$ and $\dfrac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}}\mathbf{x}$.

b. Calculate a unit vector $\mathbf{u}$ in the direction of $\mathbf{v}$.

c. Are vectors $\mathbf{v}$ and $\mathbf{w}$ orthogonal?

1.2. Calculate the L_1, L_2, and L_∞ norms of the following vectors:

a. $\mathbf{x} = [-2 \quad 1 \quad 3 \quad 2]^T$.

b. $\mathbf{y} = [-3 \quad 1]^T$.

c. $\mathbf{v} = [4 \quad -3 \quad 2]^T$.

d. $\mathbf{w} = [-5 \quad -6 \quad -1 \quad -3 \quad -5]^T$.

1.3. Let $x_1, x_2, \ldots, x_n$ be fixed numbers. The Vandermonde matrix $\mathbf{X}$ is

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x^{n-2} & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-2} & x_2^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-2} & x_n^{n-1} \end{bmatrix}.$$

Given $\mathbf{d} = [d_1 \quad d_2 \quad \ldots \quad d_n]^T$, suppose that $\mathbf{w} \in \Re^n$ satisfies $\mathbf{Xw} = \mathbf{d}$, and define the polynomial

$$y(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_{n-1} x^{n-1}.$$

a. Show that $y(x_1) = d_1, \ldots, y(x_n) = d_n$ (i.e., that the polynomial $y(x)$ passes through each training data point).

b. Show that when $x_1, x_2, \ldots, x_n$ are distinct, the columns of $\mathbf{X}$ are linearly independent.

c. Prove that if $x_1, x_2, \ldots, x_n$ are distinct numbers and $\mathbf{d}$ is an arbitrary vector, then there is an interpolating polynomial of degree $\le n-1$ for all pairs (x_1, d_1), $(x_2, d_2), \ldots, (x_n, d_n)$.

1.4. For the training data pairs $(1, 2)$, $(2, 1)$, $(5, 10)$ find

a. the interpolating polynomial of second order,

b. the best least squares approximating straight line.

1.5. Compute L_∞ and L_2 norms for the function $f(x) = (1 + x)^{-1}$ on the interval $[0, 1]$. (See hint in problem 1.6b.)

1.6. Find the best approximating straight lines to the curve $y = e^x$ such that

a. the L_2 (Euclidean) norm of the error function on the discrete set $x = [-1 \quad -0.5 \quad 0 \quad 0.5 \quad 1]$ is as small as possible,

b. the L_2 (Euclidean) norm of the error function on the interval $[-1, \quad 1]$ is as small as possible. (*Hint:* Use $\int_a^b |E(w)|^2\, dw$ for the L_2 (Euclidean) norm of the continuous error function on the interval $[a, b]$.)

1.7. Figure P1.1 shows the unit sphere of the L_2 norm in $\Re^2$.

a. Draw unit spheres of L_p norms for $p = 0.5, 1, 10$, and ∞. Comment on the geometrical meaning of these spheres.

b. Draw unit spheres of L_p norms for $p = 1, 2$, and ∞ in $\Re^3$.

1.8. Consider vector $\mathbf{x}$ in $\Re^2$ shown in figures P1.2a and P1.2b. Find the best approximation to $\mathbf{x}$ in a subspace SL (a straight line) in

a. L_1 norm,

b. L_2 norm,

c. L_∞ norm.

Draw your result and comment on the equality or difference of the best approximations in given norms.

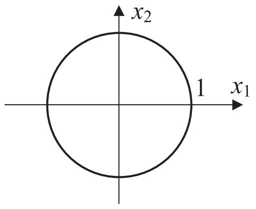


Figure P1.1
Graph for problem 1.7.

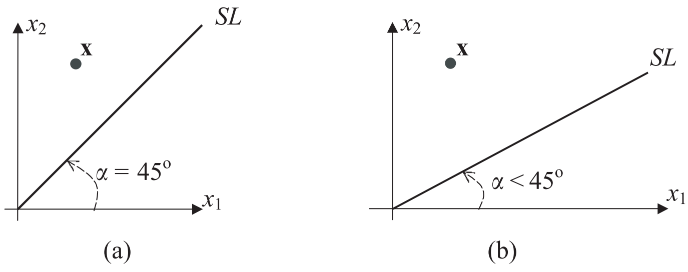


Figure P1.2
Graphs for problem 1.8.

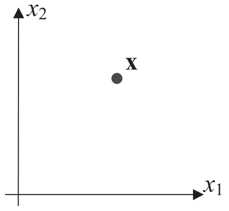


Figure P1.3
Graph for problem 1.9.

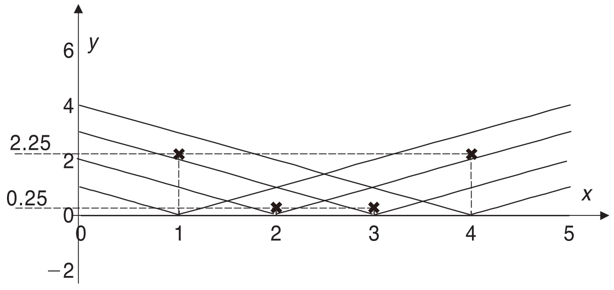
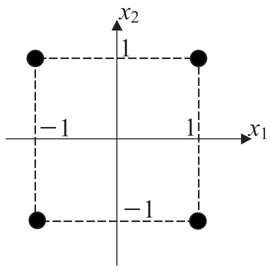


Figure P1.4
Graph for problem 1.10.

1.9. For a given vector $\mathbf{x}$ find a subspace (a straight line SL) of $\Re^2$ for which the best approximations to $\mathbf{x}$ in SL, in L_1 and L_2 norms, are equal. Comment on the best approximation in L_∞ norm in this straight line. (See figure P1.3.)

1.10. Consider four measurement points given in figure P1.4. Find the weights corresponding to four linear splines that ensure an interpolating (piecewise linear) curve $y_a(x)$.

1.11. It was shown that there is a unique polynomial interpolant for a one-dimensional input x (i.e., when $y = y(x)$). There is no theorem about a unique polynomial interpolation for an $\Re^n \to \Re^1$ mapping $(n > 1)$ (i.e., for $y = y(\mathbf{x})$). Check whether four given points in figures P1.5a and P 1.5b can be uniquely interpolated by a bilinear polynomial $y_a = w_1 + w_2x_1 + w_3x_2 + w_4x_1x_2$. Training data pairs are given as



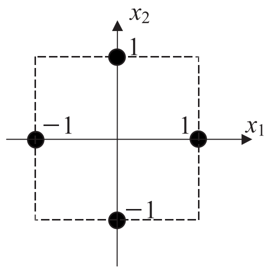


Figure P1.5
Graphs for problem 1.11.

Figure 1.5a				Figure 1.5b			
d_1	d_2	d_3	d_4	d_1	d_2	d_3	d_4
1	-1	-1	1	1	0	-1	0
1	1	-1	-1	0	1	0	-1

(*Hint:* For both problems write down a system of four equations in four unknowns and check whether there is a unique solution.)

1.12. Many phenomena are subject to seasonal fluctuations (e.g., plant growth, monthly sales of products, airline ticket sales). To model such periodic problems, the preferred approximating scheme is

$$y_a = w_0 + w_1 x + w_2 \sin \frac{2\pi x}{12}.$$

a. Show this model graphically as a network.

b. Give the design matrix **X**.

c. Is the approximation problem linear?

1.13. (a) Expand a function $f(\mathbf{x})$ about $\mathbf{x}_0$ in a Taylor series, and show this expansion graphically as a ("neural") network, retaining the first three terms only. An input to the network is $\Delta\mathbf{x}$, and the output is Δf.

(b) Show as a network a Taylor expansion of a function e^{kx}, retaining the first four terms only.

1.14. Determine the gradient vector and the Hessian matrix for each error function $E(\mathbf{w})$:

(a) $E(\mathbf{w}) = 3w_1 w_2^2 + 4e^{w_1 w_2}$.

(b) $E(\mathbf{w}) = w_1^{w_2} + \ln w_1 w_2$.

(c) $E(\mathbf{w}) = w_1^2 + w_2^2 + w_3^2$.

(d) $E(\mathbf{w}) = \ln(w_1^2 + w_1 w_2 + w_2^2)$.

1.15. The gradient path for minimizing the elliptic paraboloid $E(\mathbf{w}) = w_1^2 + 2w_2^2$ is $w_2 = e^c w_1^2$. Both the level curves (contours) and the gradient path are shown in figure P1.6. Show that the gradient path is orthogonal to the level curves. (*Hint:* Use the fact that if the curves are orthogonal, then their normal vectors are also orthogonal.)

1.16. Show that the optimal learning rate η_{opt} for minimizing the quadratic error function

$$E(\mathbf{w}) = 0.5\mathbf{w}^T\mathbf{Q}\mathbf{w} \text{ is } \eta_{\text{opt}} = -\left.\frac{\mathbf{w}^T\mathbf{Q}^2\mathbf{w}}{\mathbf{w}^T\mathbf{Q}^3\mathbf{w}}\right|_i.$$

(*Hint:* Express the value of the error function at step $i+1$, and minimize this expression with respect to learning rate η. Use that for quadratic forms $\mathbf{Q} = \mathbf{Q}^T$.)

1.17. Perform two iterations of the optimal gradient algorithm to find the minimum of $E(\mathbf{w}) = 2w_1^2 + 2w_1 w_2 + 5w_2^2$. The starting point is $\mathbf{w}_0 = [2 \; -2]^T$. Draw the contours, and show your learning path graphically. Check the orthogonality of gradient vectors at the initial point and at the next one. (*Hint:* Express $E(\mathbf{w})$ in matrix form, and use the expression for η_{opt} from problem 1.16.)

1.18. Which of the four given functions in figure P1.7 are probability-density functions? (*Hint:* Use that $\int_{-\infty}^{+\infty} P(x)\,dx = 1$.)

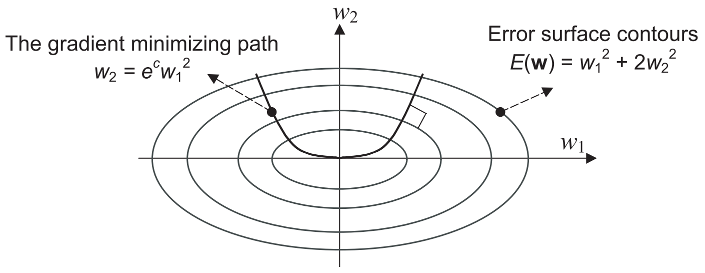

w_2
The gradient minimizing path
$w_2 = e^c w_1^2$
Error surface contours
$E(\mathbf{w}) = w_1^2 + 2w_2^2$
w_1

Figure P1.6
Graph for problem 1.15.

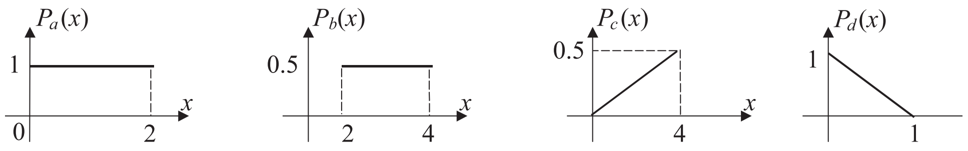


Figure P1.7
Graphs for problem 1.18.

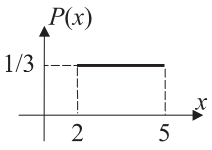


Figure P1.8
Graph for problem 1.19.

1.19. Consider a continuous variable that can take on values on the interval from 2 to 5. Its probability-density function is shown in figure P1.8. (This is called the uniform density.) What is the probability that the random variable will take on a value

a) between 3 and 4,

b) less than 2.8,

c) greater than 4.3,

d) between 2.6 and 4.4?

1.20. The three graphs in figures P1.9a, P1.9b, and P1.9c show uniform, normal, and triangular probability-density functions, respectively. Find the constants C_i for each function.

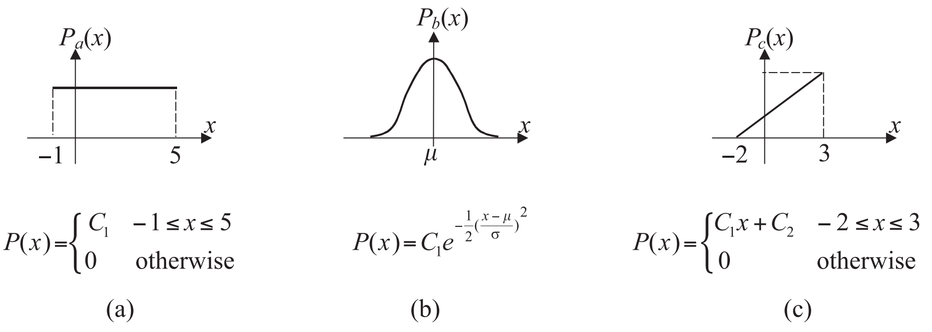
1.21. A random variable x is defined by its probability-density function

$$P(x) = \begin{cases} \dfrac{C}{8} e^{-x/(C+2)} & x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

a) Calculate constant C and comment on the results.

b) Find a mean μ_x and a variance σ_x^2.

c) Calculate the probability $P(-1 \leq x < 2)$.



$$P(x)=\begin{cases} C_1 & -1 \le x \le 5 \\ 0 & \text{otherwise} \end{cases}$$

(a)

$$P(x)=C_1 e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

(b)

$$P(x)=\begin{cases} C_1 x + C_2 & -2 \le x \le 3 \\ 0 & \text{otherwise} \end{cases}$$

(c)

Figure P1.9
Graphs for problem 1.20.

1.22. Let x and y have the joint probability-density function

$$P(x,y) = \begin{cases} 5 & 0 < x < y < 1, \\ 0 & \text{otherwise.} \end{cases}$$

a) Find the marginal and conditional probability-density functions $P(x)$, $P(y)$, and $P(x,y)$.

b) Find the regression function $x = f(y)$ (i.e., find the conditional mean of x given $y = y_0$).

1.23. Consider two random variables defined as $|x| \le 4$ and $0 \le y \le 2 - 0.5|x|$. Let them have a joint probability-density function

$$P(x,y) = \begin{cases} C & \text{over the sample space,} \\ 0 & \text{elsewhere.} \end{cases}$$

a) Draw the joint probability-density function.

b) Find C.

c) Are random variables x and y independent?

d) Calculate a correlation coefficient ρ.

1.24. The joint probability-density function of two variables is given as

$$P(x,y) = \begin{cases} q & x \ge 0, y \ge 0, x^2 + y^2 \le 1, \\ 0 & \text{elsewhere.} \end{cases}$$

a) Draw the sample space.

b) Find q.

c) Calculate the marginal probability-density function $P(x)$.

d) Find a mean μ_x and a variance σ_x^2.

Hint:

$$\int \sqrt{a^2 - x^2}\, dx = \frac{1}{2}\left(x\sqrt{a^2 - x^2} + a^2 \arcsin\frac{x}{a}\right)$$

$$\int x\sqrt{a^2 - x^2}\, dx = -\frac{1}{3}\sqrt{(a^2 - x^2)^3}$$

$$\int x^2\sqrt{a^2 - x^2}\, dx = -\frac{x}{4}\sqrt{(a^2 - x^2)^3} + \frac{a^2}{8}\left(x\sqrt{a^2 - x^2} + a^2 \arcsin\frac{x}{a}\right)$$

1.25. Let two random variables be defined by the joint probability-density function

$$P(x,y) = \begin{cases} k & 0 \le x \le 1,\ 1 \le x + y \le 2, \\ 0 & \text{otherwise.} \end{cases}$$

a) Draw the graph of $P(x,y)$.

b) Calculate k, $P(x)$, and $P(y)$.

c) Find μ_x, μ_y, $\sigma_{xy} = E\{xy\}$, σ_x^2, and σ_y^2.

d) Are x and y dependent variables? Are they correlated?

1.26. Consider two random variables having the joint probability-density function

$$P(x,y) = \begin{cases} x + y & 0 < x < 1,\ 0 < y < 1, \\ 0 & \text{elsewhere.} \end{cases}$$

Find the correlation coefficient ρ of x and y.

1.27. The joint probability-density function $P(x,y)$ is given as

$$P(x,y) = \begin{cases} \frac{1}{2}xy & 0 < x < 2,\ 0 < y < x, \\ 0 & \text{otherwise.} \end{cases}$$

a) Draw the sample space in an (x,y) plane.

b) Find the marginal probability-density functions $P(x)$ and $P(y)$.

1.28. Find the equation of a regression curve $\mu_{y|x} = y = f(\mathrm{x})$ in problem 1.27.

1.29. A theoretical correlation coefficient is defined as $\rho = \sigma_{xy}/\sigma_x\sigma_y$. Apply this expression to example 1.7 and find ρ.

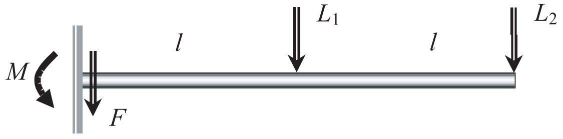


Figure P1.10
Graph for problem 1.30.

1.30. A beam in figure P1.10 is subjected to two random loads L_1 and L_2, which are statistically independent with means and standard deviations μ_1, σ_1, and μ_2, σ_2, respectively. Are the shear force F and bending moment M correlated? Find the correlation coefficient. For $\sigma_1 = \sigma_2$, is there no correlation at all? Are F and M just correlated, or are they highly correlated? Are they causally related? (*Hint:* $F = L_1 + L_2$, $M = lL_1 + 2lL_2$. Find the means and standard deviations of force and moment, calculate the correlation coefficient, and discuss your result.)

1.31. The probability-density function $P(\mathbf{x})$ of a multivariate n-dimensional Gaussian distribution is given by

$$P(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp(-0.5(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}))$$

that is, it is parameterized by the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. For a two-dimensional vector $\mathbf{x}$, $\boldsymbol{\mu} = [\mu_1 \quad \mu_2]^T$ and $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_2^2 \end{bmatrix}$. Sketch the contours of $P(\mathbf{x})$ in an (x_1, x_2) plane for the following four distributions:

a) $\boldsymbol{\mu} = [2 \quad 3]^T$, $\sigma_{12} = \sigma_{21} = 0$, $\sigma_{11} = \sigma_{22} = \sigma$.

b) $\boldsymbol{\mu} = [-2 \quad 2]^T$, $\sigma_{12} = \sigma_{21} = 0$, $\sigma_{11} > \sigma_{22}$.

c) $\boldsymbol{\mu} = [-3 \quad -3]^T$, $\sigma_{12} = \sigma_{21} = 0$, $\sigma_{11} < \sigma_{22}$.

d) $\boldsymbol{\mu} = [3 \quad -2]^T$, $\sigma_{12} = \sigma_{21} > 0$.

1.32. Find the equations of $P(\mathbf{x})$, and of the contours for which $P(\mathbf{x}) = 0.5$, for the following three two-dimensional Gaussian distributions:

a) $\boldsymbol{\mu} = [2 \quad 2]^T$, $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

b) $\boldsymbol{\mu} = [2 \quad 2]^T$, $\boldsymbol{\Sigma} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$.

c) $\boldsymbol{\mu} = [2 \quad 2]^T$, $\boldsymbol{\Sigma} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}$.

1.33. Given the covariance matrix and mean vector of a four-dimensional normal distribution

$$\boldsymbol{\Sigma} = \begin{bmatrix} 15 & 3 & 1 & 0 \\ 3 & 16 & 6 & -2 \\ 1 & 6 & 4 & 1 \\ 0 & -2 & 1 & 3 \end{bmatrix}, \qquad \boldsymbol{\mu} = [10 \quad 0 \quad -10 \quad 1]^T$$

determine the probability-density function $P(\mathbf{x})$. (*Hint:* Calculate $\boldsymbol{\Sigma}^{-1}$ and $|\boldsymbol{\Sigma}|$. Do not leave the exponent of $P(\mathbf{x})$ in matrix form.)

1.34. Consider a three-dimensional Gaussian probability-density function

$$P(\mathbf{x}) = \frac{\sqrt{3}}{16\pi^{3/2}} \exp\left(-\frac{1}{8}\left(2x_1^2 + 4x_2^2 - 2x_2(x_3+5) + (x_3+5)^2\right)\right).$$

a) Find the covariance matrix $\boldsymbol{\Sigma}$. (*Hint:* Note that in this problem $P = P_1(x_1) \cdot P_2(x_2, x_3)$, meaning that $\sigma_{12} = \sigma_{13} = 0$).

b) Determine the locus of points for which the probability-density is 0.01.

1.35. Consider the multivariate n-dimensional Gaussian probability-density function $P(\mathbf{x})$ for which the covariance matrix $\boldsymbol{\Sigma}$ is diagonal (i.e., $\sigma_{ij} = 0$, $i \neq j$).

a) Show that $P(\mathbf{x})$ can be expressed as

$$P(\mathbf{x}) = \frac{1}{\prod_{i=1}^{n} \sigma_i \sqrt{(2\pi)}} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 \right).$$

b) What are the contours of constant probability-density? Show graphically the contours for $\mathbf{x} = [x_1 \quad x_2]^T$, $\boldsymbol{\mu} = [-2 \quad 3]^T$, and $\sigma_2 = 2\sigma_1$.

c) Show that the expression in the exponent of $P(\mathbf{x})$ in (a) is a Mahalanobis distance.

1.36. Consider three classes with Gaussian class-conditional probability-density functions having the same covariance matrix $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$, $i = 1, 2, 3$, and with means $\boldsymbol{\mu}_1 = [0 \quad 2]^T$, $\boldsymbol{\mu}_2 = [4 \quad 1]^T$, $\boldsymbol{\mu}_3 = [1 \quad 0]^T$.

a) Draw the contours of $P_i(\mathbf{x})$ in an (x_1, x_2) plane.

b) Find both the decision (discriminant) functions and the equations of decision boundaries. Draw the boundaries in the graph in (a).

1.37. The calculation of a Bayes's classifier requires knowledge of means and covariance matrices. However, typically only training data are known. The following two feature patterns ($\mathbf{x} \in \Re^2$) from the two equiprobable normally (Gaussian) distributed classes (class 1 = 0, and class 2 = 1) have been drawn:

Class 1

x_1	x_2	d
1	2	0
2	2	0
2	3	0
3	1	0
3	2	0

Class 2

x_1	x_2	d
6	8	1
7	8	1
8	7	1
8	8	1
7	9	1

a) Find discriminant (decision) functions for both classes.

b) Calculate the dichotomizing function. Draw (in an (x_1, x_2) plane) the training data pairs, the dichotomizing function, and the intersections of the two discriminant functions with an (x_1, x_2) plane.

c) Test the performance of your dichotomizer by classifying the previously unseen pattern $\mathbf{x}_1 = [4 \quad 1]^T$, $\mathbf{x}_1 = [6 \quad 7]^T$. (*Hint:* Calculate the empirical means and covariance matrices from the data first, and then apply appropriate equations for calculation of discriminant functions. Use

$$\boldsymbol{\Sigma}_{i(\text{est})} = \frac{\sum_{p=1}^{P}(\mathbf{x}_p^{\text{class } i} - \boldsymbol{\mu}^{\text{class } i})(\mathbf{x}_p^{\text{class } i} - \boldsymbol{\mu}^{\text{class } i})^T}{(P-1)}$$

for a covariance matrix calculation. Subscript (est) denotes an estimate.)

1.38. A two-dimensional random vector $\mathbf{y}$ has the probability-density function

$$P(\mathbf{y}) = \begin{cases} \dfrac{1}{a^2} & 0 \le y_1,\ y_2 \le a, \\ 0 & \text{otherwise.} \end{cases}$$

Another two-dimensional random vector **x** related to **y** has the conditional density function

$$P(\mathbf{x}\,|\,\mathbf{y}) = \frac{1}{2\pi\sigma_1\sigma_2}\exp\left(-\left(\frac{(x_1-y_1)^2}{2\sigma_1^2}+\frac{(x_2-y_2)^2}{2\sigma_2^2}\right)\right).$$

Find the posterior probability-density function $P(\mathbf{y}\,|\,\mathbf{x})$. Is there a closed form solution? (*Hint:* Find the joint probability-density function $P(\mathbf{x},\mathbf{y})$ first, and use it for computing $P(\mathbf{y}\,|\,\mathbf{x})$. If this problem seems to be too difficult, go to the next problem and return to this one later.)

1.39. Assign the feature $x = 0.6$ to the one of two equiprobable classes by using the maximum likelihood decision rule associated with classes that are given by the following class-conditional probability-density functions:

$$P(x\,|\,\omega_1) = \left(\frac{1}{2\pi}\right)^{0.5}\exp\left(-\frac{x^2}{2}\right) \quad \text{and} \quad P(x\,|\,\omega_2) = \left(\frac{1}{2\pi}\right)^{0.5}\exp\left(-\frac{(x-1)^2}{2}\right).$$

Draw the class-conditional probability-density functions, and show the decision boundary. (*Hint:* Assuming equal prior probability-densities ($P(\omega_1) = P(\omega_2)$), the maximum likelihood decision rule is equivalent to the MAP (Bayes') decision criterion.)

1.40. Determine the maximum likelihood decision rule associated with the two equiprobable classes given by the following class-conditional probability-density functions:

$$P(x\,|\,\omega_1) = \left(\frac{1}{2\pi}\right)^{0.5}\exp\left(-\frac{x^2}{2}\right) \quad \text{and} \quad P(x\,|\,\omega_2) = \left(\frac{1}{8\pi}\right)^{0.5}\exp\left(-\frac{x^2}{8}\right).$$

Draw the class-conditional probability-density functions and show the decision boundaries. (*Hint:* Note that the means are equal.)

1.41. The class-conditional probability-density functions of two classes are given by

$$P(x\,|\,\omega_1) = \left(\frac{1}{2\pi}\right)^{0.5}\exp\left(-\frac{x^2}{2}\right) \quad \text{and} \quad P(x\,|\,\omega_2) = \left(\frac{1}{2\pi}\right)^{0.5}\exp\left(-\frac{(x-1)^2}{2}\right).$$

Prior probability for a class 1 is $P(\omega_1) = 0.25$. Find the decision boundary by minimizing a probability of classification error, that is, use the MAP (Bayes') decision criterion. (*Hint:* Use (1.70b).)

1.42. The class-conditional probability-density functions of two classes are given by

$$P(x\,|\,\omega_1) = \frac{1}{2}\exp(-|x|) \quad \text{and} \quad P(x\,|\,\omega_2) = \exp(-2|x|).$$

Prior probability for a class 1 is $P(\omega_1) = 0.25$, and the losses are given as $L_{11} = L_{22} = 0$, $L_{12} = 1$, and $L_{21} = 2$. Find the decision boundaries by minimizing Bayes' risk. (*Hint:* Use (1.86).)

1.43. The class-conditional probability-density functions of two classes are given by

$$P(x\,|\,\omega_1) = 2\exp(-2x) \quad \text{and} \quad P(x\,|\,\omega_2) = \exp(-x),$$

both for $x \geq 0$; otherwise both are equal to zero).

a) Find the maximum likelihood decision rule.

b) Find the minimal probability of error decision rule. Prior probability for a class 1 is $P(\omega_1) = 2/3$.

c) Find the minimal risk decision rule for equally probable classes and with the losses $L_{11} = 0$, $L_{22} = 1$, $L_{12} = 2$, and $L_{21} = 3$.

(Solving this problem, you will confirm that here (as in the case of function approximation tasks), the best solution depends upon the norm applied. Note that in (a) the best means maximization, and in (b) and (c) the best is the minimizing solution.)

1.44. Find the posterior probability-density functions $P(\omega_1\,|\,x)$ and $P(\omega_2\,|\,x)$ for the two equiprobable one-dimensional normally distributed classes given by likelihood functions (class-conditional probability-density functions that are also known as data generation mechanisms)

$$P(x\,|\,\omega_1) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right],$$

$$P(x\,|\,\omega_2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma}\right)^2\right].$$

(*Hint:* Start with Bayes' rule, plug in the given likelihoods, and find the desired posterior probability-density functions in terms of distribution means and standard deviation.)

1.45. Derive the posterior probability-density function $P(\omega_1\,|\,x)$ for the likelihood functions defined in problem 1.44 but having different variances ($\sigma_1 \neq \sigma_2$).

1.46. Find the posterior probability-density function $P(\omega_1 \mid \mathbf{x})$ for a binary (two classes only) classification problem when $\mathbf{x}$ is an n-dimensional vector. The two Gaussian multivariate class-conditional probability-densities have arbitrary mean vectors $\boldsymbol{\mu}_i$, $i = 1, 2$, equal prior probabilities, and the same covariance matrix Σ. Sketch the classification problem in an (x_1, x_2) plane, and explain your graph for the case of an identity covariance matrix ($\boldsymbol{\Sigma} = \mathbf{I}$). (*Hint:* Start by plugging the multivariate Gaussian likelihood functions into Bayes' rule. At the end of the derivation, similar to the previous one-dimensional case, the posterior probability-density function $P(\omega_1 \mid \mathbf{x})$ should have a form of logistic function and should be expressed in terms of vector quantities comprising means and covariance matrix.)

Simulation Experiments

The simulation experiments in chapter 1 have the purpose of familiarizing the reader with interpolation/approximation, that is, nonlinear regression. (Nonlinear regression and classification are the core problems of soft computing.) The programs used in chapter 2 on support vector machines cover both classification and regression by applying the SVM technique. There is no need for a manual here because all routines are simple (if anything is simple about programming). The experiments are aimed at reviewing many basic facets of regression (notably problems of over- and underfitting, the influence of noise, and the smoothness of approximation). The first two approximators are classic ones, namely, one-dimensional algebraic polynomials and Chebyshev polynomials. The last three are radial basis function approximators: linear splines, cubic splines, and Gaussian radial basis functions. In addition, there is a fuzzy model that applies five different membership functions.

Be aware of the following facts about the program `aproxim`:

1. It is developed for interpolation/approximation problems.
2. It is designed for one-dimensional input data ($y = f(x)$).
3. It is user-friendly, even for beginners in using MATLAB, but you must cooperate. It prompts you to select, to define, or to choose different things.

Experiment with the program `aproxim` as follows:

1. Launch MATLAB.
2. Connect to directory `learnsc` (at the `matlab` prompt, type `cd learnsc` ⟨RETURN⟩). `learnsc` is a subdirectory of `matlab`, as `bin`, `toolbox`, and `uitools` are. While typing `cd learnsc`, make sure that your working directory is `matlab`, not `matlab/bin`, for example.

3. Type `start` ⟨RETURN⟩.
4. The pop-up menu will prompt you to choose between several models. Choose 1D Approximation.
5. The pop-up box offers five approximators (networks, models, or machines). Click on one.
6. Follow the prompting pop-up menus to make some modeling choices. Note that if you want to have your function polluted by 15% noise, type `0.15`.

Now perform the following experiments (start with the `demo` function):

1. Look at the difference between the interpolation and approximation. Add 25% noise (noise $= 0.25$), and in order to do interpolation, choose the model order for polynomials to be $n = P - 1$, where P equals the number of training data. (The number will be printed on the screen.) For radial basis functions, interpolation will take place when you choose $t = 1$. Choosing $t = 1$ means that you are placing one radial basis function at each training data point. If you want to approximate the given function, you select $n < P - 1$ and $t > 1$ for polynomials and RBFs, respectively. It is clear that $t < P$.

Experiment type 1 Start with any `demo` function with 25% noise. Interpolate it first. Reduce the order of the polynomial gradually, and observe the changes in modeling quality. Always check the final error of models. Note that if there is a lot of noise, lower-order models do filter the noise out. But don't overdo it. At some stage, further decreasing the model's order (or the capacity of the model) leads to underfitting. This means that you are starting to filter out both the noise and the underlying function.

Experiment type 2 Now repeat experiment 1 with an RBF model. Controlling the capacity of your model to avoid overfitting noisy data is different for polynomials than for RBFs. The order of the model controls the polynomial capacity, and the number of basis functions is the smoothing parameter (the parameter for capacity control) for RBFs. It is not the only parameter, however. This is one of the topics in chapter 5.

Compare the smoothness of linear splines approximators and cubic splines approximators. When using Gaussian basis functions, you will have to choose k_σ. Choosing, for example, the value for this coefficient $k_\sigma = 2$, you define a standard deviation of Gaussian bells $\sigma = 2\Delta c$. This means that σ of all bells is equal to two distances between the Gaussian bell centers. For good (smooth) approximations, $0.75 < \sigma < 10$. This is both a broad and an approximate span for σ values. σ is typically the subject of learning. However, you may try experimenting with various values for the standard deviation σ. You will be also prompted to choose an RBF with bias or without it. Choose both, and compare the results. Many graphs will be displayed, but they are not that complicated to read. Try to understand them. More is explained in chapter 5.

2. Look at the effects of different noise levels on various interpolators or approximators. Note that noise = 1.0 means that there is 100% noise. For many practical situations, this is too high a noise level. On the other hand, in many recognition tasks, pollution by noise may be even higher. Repeat all experiments from (1) with a different noise level.

3. You are now ready to define your own functions and to perform experiments that you like or are interested in.