

Notation

I have made an effort to keep a consistent mathematical notation throughout the book. Although every symbol is defined in the text prior to its use, it may be convenient for the reader to have the list of major symbols summarized together. The list is presented below:

- Vectors are denoted with **boldface** letters, such as \mathbf{x} .
- Matrices are denoted with capital letters, such as A .
- The determinant of a matrix is denoted as $\det\{A\}$, and sometimes as $|A|$.
- A diagonal matrix with elements a_1, a_2, \dots, a_l in its diagonal is denoted as $A = \text{diag}\{a_1, a_2, \dots, a_l\}$.
- The identity matrix is denoted as I .
- The trace of a matrix is denoted as $\text{trace}\{A\}$.
- Random variables are denoted with roman fonts, such as x , and their corresponding values with *mathmode* letters, such as x .
- Similarly, random vectors are denoted with roman **boldface**, such as \mathbf{x} , and the corresponding values as \mathbf{x} . The same is true for random matrices, denoted as X and their values as X .
- Probability values for discrete random variables are denoted by capital P , and probability density functions (PDFs), for continuous random variables, are denoted by lower case p .
- The vectors are assumed to be column-vectors. In other words,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{bmatrix}, \text{ or } \mathbf{x} = \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(l) \end{bmatrix}.$$

That is, the i th element of a vector can be represented either with a subscript, x_i , or as $x(i)$.

- Matrices are written as

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1l} \\ \vdots & \vdots & \ddots & \vdots \\ x_{l1} & x_{l2} & \dots & x_{ll} \end{bmatrix}, \text{ or } X = \begin{bmatrix} X(1, 1) & X(1, 2) & \dots & X(1, l) \\ \vdots & \vdots & \ddots & \vdots \\ X(l, 1) & X(l, 2) & \dots & X(l, l) \end{bmatrix}.$$

- Transposition of a vector is denoted as \mathbf{x}^T and the Hermitian transposition as \mathbf{x}^H .
- Complex conjugation of a complex number is denoted as x^* and also $\sqrt{-1} := j$. The symbol “:=” denotes definition.
- The sets of real, complex, integer, and natural numbers are denoted as \mathbb{R} , \mathbb{C} , \mathbb{Z} , and \mathbb{N} , respectively.
- Sequences of numbers (vectors) are denoted as x_n (\mathbf{x}_n) or $x(n)$ ($\mathbf{x}(n)$) depending on the context.
- Functions are denoted with lower case letters, e.g., f , or in terms of their arguments, e.g., $f(x)$ or sometimes as $f(\cdot)$, if no specific argument is used, to indicate a function of a single argument, or $f(\cdot, \cdot)$ for a function of two arguments and so on.

INTRODUCTION

1

CONTENTS

1.1	The Historical Context.....	1
1.2	Artificial Intelligence and Machine Learning.....	2
1.3	Algorithms Can Learn What Is Hidden in the Data.....	4
1.4	Typical Applications of Machine Learning.....	6
	Speech Recognition	6
	Computer Vision.....	6
	Multimodal Data	6
	Natural Language Processing	7
	Robotics	7
	Autonomous Cars	7
	Challenges for the Future.....	8
1.5	Machine Learning: Major Directions.....	8
1.5.1	Supervised Learning	8
	Classification	9
	Regression.....	11
1.6	Unsupervised and Semisupervised Learning	11
1.7	Structure and a Road Map of the Book.....	12
	References.....	16

1.1 THE HISTORICAL CONTEXT

During the period that covers, roughly, the last 250 years, humankind has lived and experienced three *transforming* revolutions, which have been powered by technology and science. The *first industrial* revolution was based on the use of water and steam and its origins are traced to the end of the 18th century, when the first organized factories appeared in England. The *second industrial* revolution was powered by the use of electricity and mass production, and its “birth” is traced back to around the turn of the 20th century. The *third industrial* revolution was fueled by the use of electronics, information technology, and the adoption of automation in production. Its origins coincide with the end of the Second World War.

Although difficult for humans, including historians, to put a stamp on the age in which they themselves live, more and more people are claiming that the *fourth industrial* revolution has already started and is fast transforming everything that we know and learned to live with so far. The fourth industrial revolution builds upon the third one and is powered by the *fusion* of a number of technologies, e.g.,

computers and communications (internet), and it is characterized by the convergence of the *physical*, *digital*, and *biological* spheres.

The terms artificial intelligence (AI) and machine learning are used and spread more and more to denote the type of automation technology that is used in the production (industry), in the distribution of goods (commerce), in the service sector, and in our economic transactions (e.g., banking). Moreover, these technologies affect and shape the way we socialize and interact as humans via social networks, and the way we entertain ourselves, involving games and cultural products such as music and movies.

A distinct qualitative difference of the fourth, compared to the previous industrial revolutions, is that, before, it was the manual skills of humans that were gradually replaced by “machines.” In the one that we are currently experiencing, mental skills are also replaced by “machines.” We now have automatic answering software that runs on computers, less people are serving us in banks, and many jobs in the service sector have been taken over by computers and related software platforms. Soon, we are going to have cars without drivers and drones for deliveries. At the same time, new jobs, needs, and opportunities appear and are created. The labor market is fast changing and new competences and skills are and will be required in the future (see, e.g., [22,23]).

At the center of this historical happening, as one of the key enabling technologies, lies a discipline that deals with data and whose goal is to extract information and related knowledge that is hidden in it, in order to make predictions and, subsequently, take decisions. That is, the goal of this discipline is to *learn* from data. This is analogous to what humans do in order to reach decisions. Learning through the senses, personal experience, and the knowledge that propagates from generation to generation is at the heart of human intelligence. Also, at the center of any scientific field lies the development of models (often called theories) in order to explain the available experimental evidence. In other words, *data* comprise a major source of *learning*.

1.2 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

The title of the book refers to *machine learning*, although the term *artificial intelligence* is used more and more, especially in the media but also by some experts, to refer to any type of algorithms and methods that perform tasks that traditionally required human intelligence. Being aware that definitions of terms can never be exact and there is always some “vagueness” around their respective meanings, I will still attempt to clarify what I mean by machine learning and in which aspects this term means something different from AI. No doubt, there may be different views on this.

Although the term machine learning was popularized fairly recently, as a scientific field it is an old one, whose roots go back to statistics, computer science, information theory, signal processing, and automatic control. Examples of some related names from the past are statistical learning, pattern recognition, adaptive signal processing, system identification, image analysis, and speech recognition. What all these disciplines have in common is that they process data, develop models that are *data-adaptive*, and subsequently make predictions that can lead to decisions. Most of the basic theories and algorithmic tools that are used today had already been developed and known before the dawn of this century. With a “small” yet important difference: the available data, as well as the computer power prior to 2000, were not enough to use some of the more elaborate and complex models that had been developed. The terrain started changing after 2000, in particular around 2010. Large data sets were gradually created and the computer power became affordable to allow the use of more complex mod-

els. In turn, more and more applications adopted such algorithmic techniques. “Learning from data” became the new trend and the term machine learning prevailed as an umbrella for such techniques.

Moreover, the big difference was made with the use and “rediscovery” of what is today known as *deep neural networks*. These models offered impressive predictive accuracies that had never been achieved by previous models. In turn, these successes paved the way for the adoption of such models in a wide range of applications and also ignited intense research, and new versions and models have been proposed. These days, another term that is catching up is “data science,” indicating the emphasis on how one can develop robust machine learning and computational techniques that deal efficiently with large-scale data.

However, the main rationale, which runs the spine of all the methods that come under the machine learning umbrella, remains the same and it has been around for many decades. The main concept is to estimate a set of parameters that describe the model, using the available data and, in the sequel, to make *predictions* based on *low-level information and signals*. One may easily argue that there is not much *intelligence* built in such approaches. No doubt, deep neural networks involve much more “intelligence” than their predecessors. They have the potential to *optimize the representation* of their low-level input information to the computer.

The term “representation” refers to the way in which related information that is hidden in the input data is quantified/coded so that it can be subsequently processed by a computer. In the more technical jargon, each piece of such information is known as a feature (see also Section 1.5.1). As discussed in detail in Chapter 18, where neural networks (NNs) are defined and presented in detail, what makes these models distinctly different from other data learning methods is their *multilayer* structure. This allows for the “building” up of a *hierarchy* of representations of the input information at various abstraction levels. Every layer builds upon the previous one and the higher in hierarchy, the more abstract the obtained representation is. This structure offers to neural networks a significant performance advantage over alternative models, which restrict themselves to a single representation layer. Furthermore, this single-level representation was rather hand-crafted and designed by the users, in contrast to the deep networks that “learn” the representation layers from the input data via the use of optimality criteria.

Yet, in spite of the previously stated successes, I share the view that we are still very far from what an intelligent machine should be. For example, once trained (estimating the parameters) on one data set, which has been developed for a specific task, it is not easy for such models to *generalize* to other tasks. Although, as we are going to see in Chapter 18, advances in this direction have been made, we are still very far from what human intelligence can achieve. When a child sees one cat, readily recognizes another one, even if this other cat has a different color or if it turns around. Current machine learning systems need thousands of images with cats, in order to be trained to “recognize” one in an image. If a human learns to ride a bike, it is very easy to transfer this knowledge and learn to ride a motorbike or even to drive a car. Humans can easily transfer knowledge from one task to another, without forgetting the previous one. In contrast, current machine learning systems lack such a generalization power and tend to forget the previous task once they are trained to learn a new one. This is also an open field of research, where advances have also been reported.

Furthermore, machine learning systems that employ deep networks can even achieve superhuman prediction accuracies on data similar to those with which they have been trained. This is a significant achievement, not to be underestimated, since such techniques can efficiently be used for dedicated jobs; for example, to recognize faces, to recognize the presence of various objects in photographs, and also to annotate images and produce text that is related to the content of the image. They can recognize

speech, translate text from one language to another, detect which music piece is currently playing in the bar, and whether the piece belongs to the jazz or to the rock musical genre. At the same time, they can be fooled by carefully constructed examples, known as adversarial examples, in a way that no human would be fooled to produce a wrong prediction (see Chapter 18).

Concerning AI, the term “artificial intelligence” was first coined by John McCarthy in 1956 when he organized the first dedicated conference (see, e.g., [20] for a short history). The concept at that time, which still remains a goal, was whether one can build an intelligent machine, realized on software and hardware, that can possess *human-like* intelligence. In contrast to the field of machine learning, the concept for AI was not to focus on low-level information processing with emphasis on predictions, but on the high-level *cognitive* capabilities of humans to *reason* and *think*. No doubt, we are still very far from this original goal. Predictions are, indeed, part of intelligence. Yet, intelligence is much more than that. Predictions are associated with what we call *inductive* reasoning. Yet what really differentiates human from the animals intelligence is the power of the human mind to form *concepts* and *create conjectures* for explaining data and more general the World in which we live. *Explanations* comprise a high-level facet of our intelligence and constitute the basis for scientific theories and the creation of our civilization. They are assertions concerning the “why”’s and the “how”’s related to a task, e.g., [5,6,11].

To talk about AI, at least as it was conceived by pioneers such as Alan Turing [16], systems should have built-in capabilities for *reasoning* and giving *meaning*, e.g., in language processing, to be able to infer *causality*, to model efficient representations of *uncertainty*, and, also, to pursue long-term goals [8]. Possibly, towards achieving these challenging goals, we may have to understand and implement notions from the theory of mind, and also build machines that implement self-awareness. The former psychological term refers to the understanding that others have their own beliefs and intentions that justify their decisions. The latter refers to what we call *consciousness*. As a last point, recall that human intelligence is closely related to feelings and emotions. As a matter of fact, the latter seem to play an important part in the *creative* mental power of humans (e.g., [3,4,17]). Thus, in this more theoretical perspective AI still remains a *vision for the future*.

The previous discussion should not be taken as an attempt to get involved with philosophical theories concerning the nature of human intelligence and AI. These topics comprise a field in itself, for more than 60 years, which is much beyond the scope of this book. My aim was to make the newcomer in the field aware of some views and concerns that are currently being discussed.

In the more practical front, for the early years, the term AI was used to refer to techniques built around knowledge-based systems that sought to hard-code knowledge in terms of formal languages, e.g., [13]. Computer “reasoning” was implemented via a set of logical inference rules. In spite of the early successes, such methods seem to have reached a limit, see, e.g., [7]. It was the alternative path of machine learning, via learning from data, that gave a real push into the field. These days, the term AI is used as an umbrella to cover all methods and algorithmic approaches that are related to the machine intelligence discipline, with machine learning and knowledge-based techniques being parts of it.

1.3 ALGORITHMS CAN LEARN WHAT IS HIDDEN IN THE DATA

It has already been emphasized that data lie at the heart of machine learning systems. Data are the *beginning*. It is the information hidden in the data, in the form of underlying regularities, correlations,

or structure, which a machine learning system tries to “learn.” Thus, irrespective of how intelligent a software algorithm is designed to be, it cannot learn more than what the data which it has been trained on allow.

Collecting the data and building the data set on which an “intelligent” system is going to be trained is highly critical. Building data sets that address human needs and developing systems that are going to make decisions on issues, where humans and their lives are involved, requires special attention, and above all, responsibility. This is not an easy issue and good intentions are not enough. We are all products of the societies in which we live, which means that our beliefs, to a large extent, are formed by the prevailing social stereotypes concerning, e.g., gender, racial, ethnic, religious, cultural, class-related, and political views. Most importantly, most of these beliefs take place and exist at a subconscious level. Thus, sampling “typical” cases to form data sets may have a strong flavor of *subjectivity* and introduce biases. A system trained on such data can affect lives, and it may take time for this to be found out. Furthermore, our world is fast changing and these changes should continuously be reflected in the systems that make decisions on our behalf. Outsourcing our lives to computers should be done cautiously and above all in an ethical framework, which is much wider and general than the set of the existing legal rules.

Of course, although this puts a burden on the shoulders of the individuals, governments, and companies that develop data sets and “intelligent” systems, it cannot be left to their good will. On the one hand, a specialized legal framework that guides the *design, implementation, and use* of such platforms and systems is required to protect our ethical standards and social values. Of course, this does not concern only the data that are collected but also the overall system that is built. Any system that replaces humans should be (a) transparent, (b) fair, and (c) accurate. Not that the humans act, necessarily, according to what the previous three terms mean. However, humans can, also, reason and discuss, we have feelings and emotions, and we do not just perform predictions.

On the other hand, this may be the time when we can develop and build more “objective” systems; that is, to go beyond human subjectivity. However, such “objectivity” should be based on science, rules, criteria, and principles, which are not yet here. As Michael Jordan [8] puts it, the development of such systems will require perspectives from the *social sciences* and *humanities*. Currently, such systems are built following an ad hoc rather than a principled way. Karl Popper [12], one of the most influential philosophers of science, stressed that all knowledge creation is theory-laden. Observations are never free of an underlying theory or explanation. Even if one believes that the process begins with observations, the act of observing requires a *point of view* (see, also, [1,6]).

If I take the liberty to make a bit of a science fiction (something trendy these days), when AI, in the context of its original conception, is realized, then data sampling and creation of data sets for training could be taken care of by specialized algorithms. Maybe such algorithms will be based on scientific principles that in the meantime will have been developed. After all, this may be the time of dawn for the emergence of a new scientific/engineering field that integrates in a principle way data-focused disciplines. To this end, another statement of Karl Popper may have to be implemented, i.e., that of *falsification*, yet in a slightly abused interpretation. An emphasis on building intelligent systems should be directed on criticism and experimentations for finding evidence that *refutes* the principles, which were employed for their development. Systems can only be used if they survive the falsification test.

1.4 TYPICAL APPLICATIONS OF MACHINE LEARNING

It is hard to find a discipline in which machine learning techniques for “learning” from data have not been applied. Yet, there are some areas, which can be considered as typical applications, maybe due to their economic and social impact. Examples of such applications are summarized below.

SPEECH RECOGNITION

Speech is the primary means of communication among humans. Language and speech comprise major attributes that differentiate humans from animals. Speech recognition has been one of the main research topics whose roots date back to the early 1960s. The goal of speech recognition is to develop methods and algorithms that enable the recognition and the subsequent representation in a computer of spoken language. This is an interdisciplinary field involving signal processing, machine learning, linguistics, and computer science.

Examples of speech recognition tasks and related systems that have been developed over the years range from the simplest isolated word recognition, where the speaker has to wait between utterances, to the more advanced continuous speech recognizers. In the latter, the user can speak almost naturally, and concurrently the computer can determine the content. Speaker recognition is another topic, where the system can identify the speaker. Such systems are used, for example, for security purposes.

Speech recognition embraces a wide spectrum of applications. Some typical cases where speech recognizers have been used include automatic call processing in telephone networks, query-based information systems, data entry, voice dictation, robotics, as well as assistive technologies for people with special needs, e.g., blind people.

COMPUTER VISION

This is a discipline that has been inspired by the human visual system. Typical tasks that are addressed within the computer vision community include the automatic extraction of edges from images, representation of objects as compositions of smaller structures, object detection and recognition, optical flow, motion estimation, inference of shape from various cues, such as shading and texture, and three-dimensional reconstruction of scenes from multiple images. Image morphing, that is, changing one image to another through a seamless transition, and image stitching, i.e., creating a panoramic image from a number of images, are also topics in the computer vision research. More recently, there is more and more interaction between the field of computer vision and that of graphics.

MULTIMODAL DATA

Both speech recognition and computer vision process information that originates from single *modalities*. However, humans perceive the natural world in a *multimodal way*, via their multiple senses, e.g., vision, hearing, and touch. There is complementary information in each one of the involved modalities that the human brain exploits in order to understand and perceive the surrounding world.

Inspired by that, *multimedia* or multimodal understanding, via *cross-media integration*, has given birth to a related field whose goal is to improve the performance in the various scientific tasks that arise in problems that deal with multiple modalities. An example of modality blending is to combine

together image/video, speech/audio, and text. A related summary that also touches issues concerning the mental processes of human *sensation, perception*, and *cognition* is presented in [9].

NATURAL LANGUAGE PROCESSING

This is the discipline that studies the processing of a language using computers. An example of a natural language processing (NLP) task is that of SPAM detection. Currently, the NLP field is an area of intense research with typical topics being the development of automatic translation algorithms and software, sentiment analysis, text summarization, and authorship identification. Speech recognition has a strong affinity with NLP and, strictly speaking, could be considered as a special subtopic of it. Two case studies related to NLP are treated in the book, one in Chapter 11 concerning authorship identification and one in Chapter 18 related to neural machine translation (NMT).

ROBOTICS

Robots are used to perform tasks in the manufacturing industry, e.g., in an assembly line for car production, or by space agencies to move objects in the space. More recently, the so-called social robots are built to interact with people in their social environment. For example, social robots are used to benefit hospitalized children [10].

Robots have been used in situations that are difficult or dangerous for humans, such as bomb detonation and work in difficult and hazardous environments, e.g., places of high heat, deep oceans, and areas of high radiation. Robots have also been developed for teaching.

Robotics is an interdisciplinary field that, besides machine learning, includes disciplines such as mechanical engineering, electronic engineering, computer science, computer vision, and speech recognition.

AUTONOMOUS CARS

An autonomous or *self-driving* car is a vehicle that can move around with no or little human intervention. Most of us have used self-driving trains in airports. However, these operate in a very well-controlled environment. Autonomous cars are designed to operate in the city streets and in motorways. This field is also of interdisciplinary nature, where areas such as radar, lidar, computer vision, automatic control, sensor networks, and machine learning meet together. It is anticipated that the use of self-driving cars will reduce the number of accidents, since, statistically, most of the accidents occur because of human errors, due to alcohol, high speed, stress, fatigue, etc.

There are various levels of automation that one can implement. At level 0, which is the category in which most of the cars currently operate, the driver has the control and the automated built-in system may issue warnings. The higher the level, the more autonomy is present. For example, at level 4, the driver would be first notified whether conditions are safe, and then the driver can decide to switch the vehicle into the autonomous driving mode. At the highest level, level 5, the autonomous driving requires absolutely no human intervention [21].

Besides the aforementioned examples of notable machine learning applications, machine learning has been applied in a wide range of other areas, such as healthcare, bioinformatics, business, finance, education, law, and manufacturing.

CHALLENGES FOR THE FUTURE

In spite of the impressive advances that have been achieved in machine learning, there are a number of challenges for the foreseeable future, besides the long-term ones that were mentioned before, while presenting AI. In the Berkeley report [18], the following list of challenges are summarized:

- Designing systems that learn *continually* by interacting with a *dynamic environment*, while making decisions that are *timely, robust, and secure*.
- Designing systems that enable personalized applications and services, yet do not compromise users' *privacy and security*.
- Designing systems that can train on data sets owned by different organizations without compromising their *confidentiality*, and in the process provide AI capabilities that span the boundaries of potentially competing organizations.
- Developing domain-specific architectures and software systems to address the performance needs of future applications, including custom chips, edge-cloud systems to efficiently process data at the edge, and techniques for abstracting and sampling data.

Besides the above more technology-oriented challenges, important social challenges do exist. The new technologies are influencing our daily lives more and more. In principle, they offer the potential, much more than ever before, to manipulate and shape beliefs, views, interests, entertainment, customs, and culture, independent of the societies. Moreover, they offer the potential for accessing personal data that in the sequel can be exploited for various reasons, such as economic, political, or other malicious purposes. As M. Schaake, a member of the European Parliament, puts it, "When algorithms affect human rights, public values or public decision-making, we need oversight and transparency." However, what was said before should not mobilize technophobic reactions. On the contrary, human civilization has advanced because of leaps in science and technology. All that is needed is social sensitivity, awareness of the possible dangers, and a related legal "shielding."

Putting it in simple words, as Henri Bergson said [2], history is not deterministic. History is a *creative evolution*.

1.5 MACHINE LEARNING: MAJOR DIRECTIONS

As has already been stated before, machine learning is the scientific field whose goal is to develop methods and algorithms that "learn" from data; that is, to extract information that "resides" in the data, which can subsequently be used by the computer to perform a task. To this end, the starting point is an available data set. Depending on the type of information that one needs to acquire, in the context of a specific task, different types of machine learning have been developed. They are described below.

1.5.1 SUPERVISED LEARNING

Supervised learning refers to the type of machine learning where all the available data have been *labeled*. In other words, data are represented in pairs of *observations*, e.g., (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, where each \mathbf{x}_n is a vector or, in general, a set of variables. The variables in \mathbf{x}_n are called the input variables, also known as the *independent variables* or *features*, and the respective vector is known as the *feature vector*. The variables y_n are known as the output or *dependent* or *target* or *label* variables.

In some cases, y_n can also be a vector. The goal of learning is to obtain/estimate a functional mapping to, given the value of the input variables, predict the value of the respective output one. Two “pillars” of supervised learning are the classification and the regression tasks.

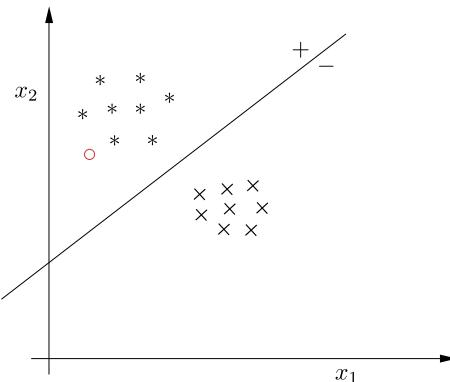
Classification

The goal in classification is to assign a *pattern* to one of a set of possible classes, whose number is considered to be known. For example, in X-ray mammography, we are given an image where a region indicates the existence of a tumor. The goal of a computer-aided diagnosis system is to predict whether this tumor corresponds to the *benign* or the *malignant* class. Optical character recognition (OCR) systems are also built around a classification system, in which the image corresponding to each letter of the alphabet has to be recognized and assigned to one of the 26 (for the Latin alphabet) classes; see Example 18.3 for a related case study. Another example is the prediction of the authorship of a given text. Given a text written by an unknown author, the goal of a classification system is to predict the author among a number of authors (classes); this application is treated in Section 11.15. The receiver in a digital communications system can also be viewed as a classification system. Upon receiving the transmitted data, which have been contaminated by noise and also by other transformations imposed by the transmission channel (Chapter 4), the receiver has to reach a decision on the value of the originally transmitted symbols. For example, in a binary transmitted sequence, the original symbols belong either to the +1 or to the -1 class. This task is known as *channel equalization*.

The first step in designing any machine learning task is to decide how to represent each pattern in the computer. This is achieved during the preprocessing stage; one has to “encode” related information that resides in the raw data (e.g., image pixels or strings of words) in an efficient and information-rich way. This is usually done by transforming the raw data into a new space and representing each pattern by a vector, $\mathbf{x} \in \mathbb{R}^l$. This comprises the feature vector and its l elements the corresponding feature values. In this way, each pattern becomes a single point in an l -dimensional space, known as the *feature space* or the *input space*. We refer to this transformation of the raw data as the *feature generation* or *feature extraction* stage. One starts with generating some large value, K , of possible features and eventually selects the l most informative ones via an optimizing procedure known as the *feature selection* stage. As we will see in Section 18.12, in the context of convolutional neural networks, the previous two stages are merged together and the features are obtained and optimized in a combined way, together with the estimation of the functional mapping, which was mentioned before.

Having decided upon the input space in which the data are represented, one has to train a classifier; that is, a predictor. This is achieved by first selecting a set of N data points/samples/examples, whose class is known, and this comprises the *training set*. This is the set of observation pairs, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, where y_n is the (output) variable denoting the class in which \mathbf{x}_n belongs, and it is known as the corresponding *class label*; the class labels take values over a *discrete* set, e.g., $\{1, 2, \dots, M\}$, for an M -class classification task. For example, for a two-class classification task, $y_n \in \{-1, +1\}$ or $y_n \in \{0, +1\}$. To keep our discussion simple, let us focus on the two-class case. Based on the training data, one then designs a function, f , which is used to predict the output label, given the input feature vector, \mathbf{x} . In general, we may need to design a set of such functions.

Once the function, f , has been designed, the system is ready to make predictions. Given a pattern whose class is unknown, we obtain the corresponding feature vector, \mathbf{x} , from the raw data. Depending on the value of $f(\mathbf{x})$, the pattern is classified in one of the two classes. For example, if the labels take the values ± 1 , then the predicted label is obtained as $\hat{y} = \text{sgn}\{f(\mathbf{x})\}$. This operation defines the

**FIGURE 1.1**

The classifier (linear in this simple case) has been designed in order to separate the training data into two classes. The graph (straight line) of the linear function, $f(\mathbf{x}) = 0$, has on its positive side the points coming from one class and on its negative side those of the other. The “red” point, whose class is unknown, is classified in the same class as the “star” points, since it lies on the positive side of the line.

classifier. If the function f is linear (nonlinear) we say that the respective classification task is linear (nonlinear) or, in a slight abuse of terminology, that the classifier is a linear (nonlinear) one.

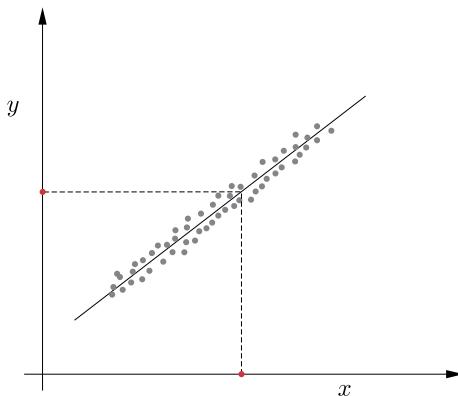
Fig. 1.1 illustrates the classification task. Initially, we are given the set of points, each one representing a pattern in the two-dimensional space (two features used, x_1, x_2). Stars belong to one class, and the crosses to the other, in a two-class classification task. These are the training points, which are used to obtain a classifier. For our very simple case, this is achieved via a linear function,

$$f(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \theta_0, \quad (1.1)$$

whose graph, for all the points such that $f(\mathbf{x}) = 0$, is the straight line shown in the figure. The values of the parameters $\theta_1, \theta_2, \theta_0$ are obtained via an estimation method based on the training set. This phase, where a classifier is *estimated*, is also known as the *training* or *learning* phase.

Once a classifier has been “learned,” we are ready to perform predictions, that is, to predict the class label of a pattern \mathbf{x} . For example, we are given the point denoted by the red circle, whose class is unknown to us. According to the classification system that has been designed, this belongs to the same class as the points denoted by stars, which all belong to, say, class +1. Indeed, every point on one side of the straight line will give a positive value, $f(\mathbf{x}) > 0$, and all the points on its other side will give a negative value, $f(\mathbf{x}) < 0$. The predicted label, \hat{y} , for the point denoted with the red circle will then be $\hat{y} = \text{sgn}\{f(\mathbf{x})\} > 0$, and it is classified in class +1, to which the star points belong.

Our discussion to present the classification task was based on features that take numeric values. Classification tasks where the features are of categorical type do exist and are of major importance, too.

**FIGURE 1.2**

In a regression task, once a function (linear in this case) f has been designed, for its graph to fit the available training data set, given a new (red) point, x , the prediction of the associated output (red) value is given by $y = f(x)$.

Regression

Regression shares to a large extent the feature generation/selection stage, as described before; however, now the output variable, y , is *not* discrete, but it takes values in an interval in the real axis or in a region in the complex numbers' plane. Generalizations to vector-valued outputs are also possible. Our focus here is on real variables. The regression task is basically a function (curve/surface) fitting problem.

We are given a set of training samples, (y_n, \mathbf{x}_n) , $y_n \in \mathbb{R}$, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, and the task is to estimate a function f , whose graph fits the data. Once we have found such a function, when a new sample \mathbf{x} , outside the training set, arrives, we can predict its output value. This is shown in Fig. 1.2. The training data in this case are the gray points. Once the function fitting task has been completed, given a new point x (red), we are ready to predict its output value as $\hat{y} = f(\mathbf{x})$. In the simple case of the figure, the function f is linear and thus its graph is a straight line.

The regression task is a generic one that embraces a number of problems. For example, in financial applications, one can predict tomorrow's stock market prices given current market conditions and other related information. Each piece of information is a measured value of a corresponding feature. Signal and image restoration come under this common umbrella of tasks. Signal and image denoising can also be seen as a special type of the regression task. Deblurring of a blurred image can also be treated as regression (see Chapter 4).

1.6 UNSUPERVISED AND SEMISUPERVISED LEARNING

The goal of supervised learning is to establish a functional relationship between the input and output variables. To this end, labeled data are used, which comprise the set of output–input pairs, on which the learning of the unknown mapping is performed.

In the antipode of supervised learning lies *unsupervised learning*, where only input variables are provided. No output or label information is available. The aim of unsupervised learning is to unravel the structure that underlies the given set of data. This is an important part in data learning methods. Unsupervised learning comes under a number of facets.

One of the most important types of unsupervised learning is that of *clustering*. The goal of any clustering task is to unravel the way in which the points in a data set are grouped assuming that such a group structure exists. As an example, given a set of newspaper articles, one may want to group them together according to how similar their content is. As a matter of fact, at the heart of any clustering algorithm lies the concept of similarity, since patterns that belong to the same group (cluster) are assumed to be more similar than patterns that belong to different clusters.

One of the most classical clustering schemes, the so-called k -means clustering, is presented and discussed in Section 12.6.1. However, clustering is not a main topic of this book, and the interested reader may look at more specialized references (e.g., [14,15]).

Another type of unsupervised learning is *dimensionality reduction*. The goal is also to reveal a particular structure of the data, which is of a different nature than that of the groupings. For example, although the data may be represented in a high-dimensional space, they may lie around a lower-dimensional subspace or a manifold. Such methods are very important in machine learning for compressed representations or computational reduction reasons. Dimensionality reduction methods are treated in detail in Chapter 19.

Probability distribution estimation can also be considered as a special case of unsupervised learning. Probabilistic modeling is treated extensively in Chapters 12, 13, 15, and 16.

More recently, unsupervised learning is used for data generation. The so-called generative adversarial networks (GANs) comprise a new way of dealing with this old topic, by employing game theoretic arguments, and they are treated in Chapter 18.

Semisupervised lies in between supervised and unsupervised learning. In semisupervised learning, there are labeled data but not enough to get a good estimate of the output–input dependence. The existence of a number of unlabeled patterns can assist the task, since it can reveal additional structure of the input data that can be efficiently utilized. Semisupervised learning is treated in, e.g., [14].

Finally, another type of learning, which is increasingly gaining in importance, is the so-called *reinforcement learning* (RL). This is also an old field, with origins in automatic control. At the heart of this type of learning lies a set of rules and the goal is to learn sequences of actions that will lead an *agent* to achieve its goal or to maximize its objective function. For example, if the agent is a robot, the goal may be to move from point A to point B. Intuitively, RL attempts to learn actions by *trial and error*. In contrast to supervised learning, optimal actions are not learned from labels but from what is known as a *reward*. This scalar value informs the system whether the outcome of whatever it did was right or wrong. Taking actions that maximize the reward is the goal of RL.

Reinforcement learning is beyond the scope of this book and the interested reader may consult, e.g., [19].

1.7 STRUCTURE AND A ROAD MAP OF THE BOOK

In the discussion above, we saw that seemingly different applications, e.g., authorship identification and channel equalization, as well as financial prediction and image deblurring, can be treated in a unified

framework. Many of the techniques that have been developed for machine learning are no different than techniques used in statistical signal processing or adaptive signal processing. Filtering comes under the general framework of regression (Chapter 4), and “adaptive filtering” is the same as “online learning” in machine learning. As a matter of fact, as will be explained in more detail, this book can serve the needs of more than one advanced graduate or postgraduate courses.

Over the years, a large number of techniques have been developed, in the context of different applications. Most of these techniques belong to one of two schools of thought. In one of them, the involved parameters that define an unknown function, for example, $\theta_1, \theta_2, \theta_0$ in Eq. (1.1), are treated as random variables. Bayesian learning builds upon this rationale. Bayesian methods learn distributions that describe the randomness of the involved parameters/variables. According to the other school, parameters are treated as nonrandom variables. They correspond to a fixed, yet unknown value. We will refer to such parameters as deterministic. This term is justified by the fact that, in contrast to random variables, if the value of a nonrandom variable is known, then its value can be “predicted” exactly. Learning methods that build around deterministic variables focus on optimization techniques to obtain estimates of the corresponding values. In some cases, the term “frequentist” is used to describe the latter type of techniques (see Chapter 12).

Each of the two previous schools of thought has its pros and cons, and I firmly believe that there is always more than one road that leads to the “truth.” Each can solve some problems more efficiently than the other. Maybe in a few years, the scene will be more clear and more definite conclusions can be drawn. Or it may turn out, as in life, that the “truth” is somewhere in the middle.

In any case, every newcomer to the field has to learn the basics and the classics. That is why, in this book, all major directions and methods will be discussed, in an equally balanced manner, to the greatest extent possible. Of course, the author, being human, could not avoid giving some more emphasis to the techniques with which he is most familiar through his own research. This is healthy, since writing a book is a means of sharing the author’s expertise and point of view with the readers. This is why I strongly believe that a new book does not serve to replace previous ones, but to complement previously published points of view.

Chapter 2 is an introduction to probability and statistics. Random processes are also discussed. Readers who are familiar with such concepts can bypass this chapter. On the other hand, one can focus on different parts of this chapter. Readers who would like to focus on statistical signal processing/adaptive processing can focus more on the random processes part. Those who would like to follow a probabilistic machine learning point of view would find the part presenting the various distributions more important. In any case, the multivariate normal (Gaussian) distribution is a must for those who are not yet familiar with it.

Chapter 3 is an overview of the parameter estimation task. This is a chapter that presents an overview of the book and defines the main concepts that run across its pages. This chapter has also been written to stand alone as an introduction to machine learning. Although it is my feeling that all of it should be read and taught, depending on the focus of the course and taking into account the omnipresent time limitations, one can focus more on the parts of her or his interest. Least-squares and ridge regression are discussed alongside the maximum likelihood method and the presentation of the basic notion of the Bayesian approach. In any case, the parts dealing with the definition of the inverse problems, the bias–variance tradeoff, and the concepts of generalization and regularization are a must.

Chapter 4 is dedicated to the mean-square error (MSE) linear estimation. For those following a statistical signal processing course, the whole chapter is important. The rest of the readers can bypass

the parts related to complex-valued processing and also the part dealing with computational complexity issues, since this is only of importance if the input data are random processes. Bypassing this part will not affect reading later parts of the chapter that deal with the MSE estimation of linear models, the Gauss–Markov theorem, and the Kalman filtering.

Chapter 5 introduces the stochastic gradient descent family of algorithms. The first part, dealing with the stochastic approximation method, is a must for every reader. The rest of the chapter, which deals with the least-mean-squares (LMS) algorithm and its offsprings, is more appropriate for readers who are interested in a statistical signal processing course, since these families are suited for tracking time-varying environments. This may not be the first priority for readers who are interested in classification and machine learning tasks with data whose statistical properties are not time-varying.

Chapter 6 is dedicated to the least-squares (LS) method, which is of interest to all readers in machine learning and signal processing. The latter part, dealing with the total least-squares method, can be bypassed in a first reading. Emphasis is also put on ridge regression and its geometric interpretation. Ridge regression is important to the newcomer, since he/she becomes familiar with the concept of regularization; this is an important aspect in any machine learning task, tied directly with the generalization performance of the designed predictor.

I have decided to compress the part dealing with fast LS algorithms, which are appropriate when the input is a random process/signal that imposes a special structure on the involved covariance matrices, into a discussion section. It is the author's feeling that this is of no greater interest than it was a decade or two ago. Also, the main idea, that of a highly structured covariance matrix that lies behind the fast algorithms, is discussed in some detail in Chapter 4, in the context of Levinson's algorithm and its lattice and lattice-ladder by-products.

Chapter 7 is a must for any machine learning course. Important classical concepts, including classification in the context of the Bayesian decision theory, nearest neighbor classifiers, logistic regression, Fisher's discriminant analysis and decision trees are discussed. Courses on statistical signal processing can also accommodate the first part of the chapter dealing with the classical Bayesian decision theory.

The aforementioned six chapters comprise the part of the book that deals with more or less classical topics. The rest of the chapters deal with more advanced techniques and can fit with any course dealing with machine learning or statistical/adaptive signal processing, depending on the focus, the time constraints, and the background of the audience.

Chapter 8 deals with convexity, a topic that is receiving more and more attention recently. The chapter presents the basic definitions concerning convex sets and functions and the notion of projection. These are important tools used in a number of recently developed algorithms. Also, the classical projections onto convex sets (POCS) algorithm and the set-theoretic approach to online learning are discussed as an alternative to gradient descent-based schemes. Then, the task of optimization of nonsmooth convex loss functions is introduced, and the family of proximal mapping, alternating direction method of multipliers (ADMM), and forward backward-splitting methods are presented. This is a chapter that can be used when the emphasis of the course is on optimization. Employing nonsmooth loss functions and/or nonsmooth regularization terms, in place of the squared error and its ridge regression relative, is a trend of high research and practical interest.

Chapters 9 and 10 deal with sparse modeling. The first of the two chapters introduces the main concepts and ideas and the second deals with algorithms for batch as well for as online learning scenarios. Also, in the second chapter, a case study in the context of time-frequency analysis is discussed. Depending on time constraints, the main concepts behind sparse modeling and compressed sensing can

be taught in a related course. These two chapters can also be used as a specialized course on sparsity on a postgraduate level.

Chapter 11 deals with learning in reproducing kernel Hilbert spaces and nonlinear techniques. The first part of the chapter is a must for any course with an emphasis on classification. Support vector regression and support vector machines are treated in detail. Moreover, a course on statistical signal processing with an emphasis on nonlinear modeling can also include material and concepts from this chapter. A case study dealing with authorship identification is discussed at the end of this chapter.

Chapters 12 and 13 deal with Bayesian learning. Thus, both chapters can be the backbone of a course on machine learning and statistical signal processing that intends to emphasize Bayesian methods. The former of the chapters deals with the basic principles and it is an introduction to the expectation-maximization (EM) algorithm. The use of this celebrated algorithm is demonstrated in the context of two classical applications: linear regression and Gaussian mixture modeling for probability density function estimation. The second chapter deals with approximate inference techniques, and one can use parts of it, depending on the time constraints and the background of the audience. Sparse Bayesian learning and the relevance vector machine (RVM) framework are introduced. At the end of this chapter, nonparametric Bayesian techniques such as the Chinese restaurant process (CRP), the Indian buffet process (IBP), and Gaussian processes are discussed. Finally, a case study concerning hyperspectral image unmixing is presented. Both chapters, in their full length, can be used as a specialized course on Bayesian learning.

Chapters 14 and 17 deal with Monte Carlo sampling methods. The latter chapter deals with particle filtering. Both chapters, together with the two previous ones that deal with Bayesian learning, can be combined in a course whose emphasis is on statistical methods of machine learning/statistical signal processing.

Chapters 15 and 16 deal with probabilistic graphical models. The former chapter introduces the main concepts and definitions, and at the end it introduces the message passing algorithm for chains and trees. This chapter is a must for any course whose emphasis is on probabilistic graphical models. The latter of the two chapters deals with message passing algorithms on junction trees and then with approximate inference techniques. Dynamic graphical models and hidden Markov models (HMMs) are introduced. The Baum–Welch and Viterbi schemes are derived as special cases of message passaging algorithms by treating the HMM as a special instance of a junction tree.

Chapter 18 deals with neural networks and deep learning. In the second edition, this chapter has been basically rewritten to accommodate advances in this topic that have taken place after the first edition was published. This chapter is also a must in any course with an emphasis on classification. The chapter starts from the early days of the perceptron algorithm and perceptron rule and moves on to the most recent advances in deep learning. The feed-forward multilayer architecture is introduced and a number of stochastic gradient-type algorithmic variants, for training networks, are presented. Different types of nonlinearities and cost functions are discussed and their interplay with respect to the vanishing/exploding phenomenon in the gradient propagation are presented. Regularization and the dropout method are discussed. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are reviewed in some detail. The notions of attention mechanism and adversarial examples are presented. Deep belief networks, GANs, and variational autoencoders are considered in some detail. Capsule networks are introduced and, at the end, a discussion on transfer learning and multitask learning is provided. The chapter concludes with a case study related to neural machine translation (NMT).

Chapter 19 is on dimensionality reduction techniques and latent variable modeling. The methods of principal component analysis (PCA), canonical correlations analysis (CCA), and independent component analysis (ICA) are introduced. The probabilistic approach to latent variable modeling is discussed, and the probabilistic PCA (PPCA) is presented. Then, the focus turns to dictionary learning and to robust PCA. Nonlinear dimensionality reduction techniques such as kernel PCA are discussed, along with classical manifold learning methods: local linear embedding (LLE) and isometric mapping (ISOMAP). Finally, a case study in the context of functional magnetic resonance imaging (fMRI) data analysis, based on ICA, is presented.

Each chapter starts with the basics and moves on to cover more recent advances in the corresponding topic. This is also true for the whole book and the first six chapters cover more classical material.

In summary, we provide the following suggestions for different courses, depending on the emphasis that the instructor wants to place on various topics.

- Machine learning with emphasis on classification:
 - Main chapters: 3, 7, 11, and 18.
 - Secondary chapters: 12 and 13, and possibly the first part of 6.
- Statistical signal processing:
 - Main chapters: 3, 4, 6, and 12.
 - Secondary chapters: 5 (first part) and 13–17.
- Machine learning with emphasis on Bayesian techniques:
 - Main chapters: 3 and 12–14.
 - Secondary chapters: 7, 15, and 16, and possibly the first part of 6.
- Adaptive signal processing:
 - Main chapters: 3–6.
 - Secondary chapters: 8, 9, 10, 11, 14, and 17.

I believe that the above suggestions of following various combinations of chapters is possible, since the book has been written in such a way as to make individual chapters as self-contained as possible.

At the end of most of the chapters, there are computer exercises, mainly based on the various examples given in the text. The exercises are given in MATLAB® and the respective code is available on the book’s website. Moreover, all exercises are provided, together with respective codes, in Python and are also available on the book’s website. Some of the exercises in Chapter 18 are in the context of TensorFlow.

The solutions manual as well as all the figures of the book are available on the book’s website.

REFERENCES

- [1] R. Bajcsy, Active perception, *Proceedings of the IEEE* 76 (8) (1988) 966–1005.
- [2] H. Bergson, *Creative Evolution*, MacMillan, London, 1922.
- [3] A. Damasio, *Descartes’ Error: Emotion, Reason, and the Human Brain*, Penguin, 2005 (paperback reprint).
- [4] S. Dehaene, H. Lau, S. Kouider, What is consciousness, and could machines have it?, *Science* 358 (6362) (2017) 486–492.
- [5] D. Deutsch, *The Fabric of Reality*, Penguin, 1998.
- [6] D. Deutsch, *The Beginning of Infinity: Explanations That Transform the World*, Allen Lane, 2011.
- [7] H. Dreyfus, *What Computers Still Can’t Do*, M.I.T. Press, 1992.
- [8] M. Jordan, Artificial intelligence: the revolution hasn’t happened yet, <https://medium.com/@mijordan3/artificial-intelligence-the-revolution-hasnt-happened-yet-5e1d5812e1e7>, 2018.

- [9] P. Maragos, P. Gros, A. Katsamanis, G. Papandreou, Cross-modal integration for performance improving in multimedia: a review, in: P. Maragos, A. Potamianos, P. Gros (Eds.), *Multimodal Processing and Interaction: Audio, Video, Text*, Springer, 2008.
- [10] MIT News, Study: social robots can benefit hospitalized children, <http://news.mit.edu/2019/social-robots-benefit-sick-children-0626>.
- [11] J. Pearl, D. Mackenzie, *The Book of Why: The New Science of Cause and Effect*, Basic Books, 2018.
- [12] K. Popper, *The Logic of Scientific Discovery*, Routledge Classics, 2002.
- [13] S. Russell, P. Norvig, *Artificial Intelligence*, third ed., Prentice Hall, 2010.
- [14] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, Amsterdam, 2009.
- [15] S. Theodoridis, A. Pikrakis, K. Koutroumbas, D. Cavouras, *Introduction to Pattern Recognition: A MATLAB Approach*, Academic Press, Amsterdam, 2010.
- [16] A. Turing, Computing machinery intelligence, MIND 49 (1950) 433–460.
- [17] N. Schwarz, I. Skurnik, Feeling and thinking: implications for problem solving, in: J.E. Davidson, R. Sternberg (Eds.), *The Psychology of Problem Solving*, Cambridge University Press, 2003, pp. 263–292.
- [18] I. Stoica, et al., A Berkeley View of Systems Challenges for AI, Technical Report No. UCB/EECS-2017-159, 2017.
- [19] R.C. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.
- [20] The history of artificial intelligence, University of Washington, <https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf>.
- [21] USA Department of Transportation Report, https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13069a_ads2.0_0906179a_tag.pdf.
- [22] The Changing Nature of Work, World Bank Report, 2019, <http://documents.worldbank.org/curated/en/816281518818814423/pdf/2019-WDR-Report.pdf>.
- [23] World Economic Forum, The future of jobs, http://www3.weforum.org/docs/WEF_Future_of_Jobs_2018.pdf, 2018.

PROBABILITY AND STOCHASTIC PROCESSES

2

CONTENTS

2.1	Introduction	20
2.2	Probability and Random Variables	20
2.2.1	Probability	20
	<i>Relative Frequency Definition</i>	21
	<i>Axiomatic Definition</i>	21
2.2.2	Discrete Random Variables	22
	<i>Joint and Conditional Probabilities</i>	22
	<i>Bayes Theorem</i>	23
2.2.3	Continuous Random Variables	24
2.2.4	Mean and Variance	25
	<i>Complex Random Variables</i>	27
2.2.5	Transformation of Random Variables	28
2.3	Examples of Distributions	29
2.3.1	Discrete Variables	29
	<i>The Bernoulli Distribution</i>	29
	<i>The Binomial Distribution</i>	30
	<i>The Multinomial Distribution</i>	31
2.3.2	Continuous Variables	32
	<i>The Uniform Distribution</i>	32
	<i>The Gaussian Distribution</i>	32
	<i>The Central Limit Theorem</i>	36
	<i>The Exponential Distribution</i>	37
	<i>The Beta Distribution</i>	37
	<i>The Gamma Distribution</i>	38
	<i>The Dirichlet Distribution</i>	39
2.4	Stochastic Processes	41
2.4.1	First- and Second-Order Statistics	42
2.4.2	Stationarity and Ergodicity	43
2.4.3	Power Spectral Density	46
	<i>Properties of the Autocorrelation Sequence</i>	46
	<i>Power Spectral Density</i>	47
	<i>Transmission Through a Linear System</i>	48
	<i>Physical Interpretation of the PSD</i>	50
2.4.4	Autoregressive Models	51
2.5	Information Theory	54
2.5.1	Discrete Random Variables	56
	<i>Information</i>	56

<i>Mutual and Conditional Information</i>	56
<i>Entropy and Average Mutual Information</i>	58
2.5.2 Continuous Random Variables.....	59
<i>Average Mutual Information and Conditional Information</i>	61
<i>Relative Entropy or Kullback–Leibler Divergence</i>	61
2.6 Stochastic Convergence	61
Convergence Everywhere.....	62
Convergence Almost Everywhere	62
Convergence in the Mean-Square Sense	62
Convergence in Probability	63
Convergence in Distribution	63
Problems	63
References	65

2.1 INTRODUCTION

The goal of this chapter is to provide the basic definitions and properties related to probability theory and stochastic processes. It is assumed that the reader has attended a basic course on probability and statistics prior to reading this book. So, the aim is to help the reader refresh her/his memory and to establish a common language and a commonly understood notation.

Besides probability and random variables, random processes are briefly reviewed and some basic theorems are stated. A number of key probability distributions that will be used later on in a number of chapters are presented. Finally, at the end of the chapter, basic definitions and properties related to information theory and stochastic convergence are summarized.

The reader who is familiar with all these notions can bypass this chapter.

2.2 PROBABILITY AND RANDOM VARIABLES

A random variable, x , is a variable whose variations are due to chance/randomness. A random variable can be considered as a function, which assigns a value to the outcome of an experiment. For example, in a coin tossing experiment, the corresponding random variable, x , can assume the values $x_1 = 0$ if the result of the experiment is “heads” and $x_2 = 1$ if the result is “tails.”

We will denote a random variable with a lower case roman, such as x , and the values it takes once an experiment has been performed, with mathmode italics, such as x .

A random variable is described in terms of a set of *probabilities* if its values are of a discrete nature, or in terms of a *probability density function* (PDF) if its values lie anywhere within an interval of the real axis (noncountably infinite set). For a more formal treatment and discussion, see [4,6].

2.2.1 PROBABILITY

Although the words “probability” and “probable” are quite common in our everyday vocabulary, the mathematical definition of probability is not a straightforward one, and there are a number of different definitions that have been proposed over the years. Needless to say, whatever definition is adopted,

the end result is that the properties and rules which are derived remain the same. Two of the most commonly used definitions are the following

Relative Frequency Definition

The probability $P(A)$ of an event A is the limit

$$P(A) = \lim_{n \rightarrow \infty} \frac{n_A}{n}, \quad (2.1)$$

where n is the number of total trials and n_A the number of times event A occurred. The problem with this definition is that in practice in any physical experiment, the numbers n_A and n can be large, yet they are always finite. Thus, the limit can only be used as a *hypothesis* and not as something that can be attained experimentally. In practice, often, we use

$$P(A) \approx \frac{n_A}{n} \quad (2.2)$$

for large values of n . However, this has to be used with caution, especially when the probability of an event is very small.

Axiomatic Definition

This definition of probability is traced back to 1933 to the work of Andrey Kolmogorov, who found a close connection between probability theory and the mathematical theory of sets and functions of a real variable, in the context of measure theory, as noted in [5].

The probability $P(A)$ of an event is a nonnegative number assigned to this event, or

$$P(A) \geq 0. \quad (2.3)$$

The probability of an event C which is certain to occur is equal to one, i.e.,

$$P(C) = 1. \quad (2.4)$$

If two events A and B are mutually exclusive (they cannot occur simultaneously), then the probability of occurrence of either A or B (denoted as $A \cup B$) is given by

$$P(A \cup B) = P(A) + P(B). \quad (2.5)$$

It turns out that these three defining properties, which can be considered as the respective *axioms*, suffice to develop the rest of the theory. For example, it can be shown that the probability of an impossible event is equal to zero, e.g., [6].

The previous two approaches for defining probability are not the only ones. Another interpretation, which is in line with the way we are going to use the notion of probability in a number of places in this book in the context of *Bayesian learning*, has been given by Cox [2]. There, probability was seen as a measure of *uncertainty* concerning an event. Take, for example, the uncertainty whether the Minoan civilization was destroyed as a consequence of the earthquake that happened close to the island of Santorini. This is obviously not an event whose probability can be tested with repeated trials. However, putting together historical as well as scientific evidence, we can quantify our expression of uncertainty

concerning such a conjecture. Also, we can modify the degree of our uncertainty once more historical evidence comes to light due to new archeological findings. Assigning numerical values to represent degrees of belief, Cox developed a set of axioms encoding common sense properties of such beliefs, and he came to a set of rules equivalent to the ones we are going to review soon; see also [4].

The origins of probability theory are traced back to the middle 17th century in the works of Pierre Fermat (1601–1665), Blaise Pascal (1623–1662), and Christian Huygens (1629–1695). The concepts of probability and the mean value of a random variable can be found there. The original motivation for developing the theory seems not to be related to any purpose for “serving society”; the purpose was to serve the needs of gambling and games of chance!

2.2.2 DISCRETE RANDOM VARIABLES

A discrete random variable x can take any value from a finite or *countably* infinite set \mathcal{X} . The probability of the event, “ $x = x \in \mathcal{X}$,” is denoted as

$$P(x = x) \quad \text{or simply } P(x). \quad (2.6)$$

The function P is known as the *probability mass function* (PMF). Being a probability of events, it has to satisfy the first axiom, so $P(x) \geq 0$. Assuming that no two values in \mathcal{X} can occur simultaneously and that after any experiment a single value will always occur, the second and third axioms combined give

$$\sum_{x \in \mathcal{X}} P(x) = 1. \quad (2.7)$$

The set \mathcal{X} is also known as the *sample* or *state space*.

Joint and Conditional Probabilities

The *joint probability* of two events, A, B , is the probability that both events occur simultaneously, and it is denoted as $P(A, B)$. Let us now consider two random variables, x, y , with sample spaces $\mathcal{X} = \{x_1, \dots, x_{n_x}\}$ and $\mathcal{Y} = \{y_1, \dots, y_{n_y}\}$, respectively. Let us adopt the relative frequency definition and assume that we carry out n experiments and that each one of the values in \mathcal{X} occurred $n_1^x, \dots, n_{n_x}^x$ times and each one of the values in \mathcal{Y} occurred $n_1^y, \dots, n_{n_y}^y$ times. Then,

$$P(x_i) \approx \frac{n_i^x}{n}, \quad i = 1, 2, \dots, n_x, \quad \text{and} \quad P(y_j) \approx \frac{n_j^y}{n}, \quad j = 1, 2, \dots, n_y.$$

Let us denote by n_{ij} the number of times the values x_i and y_j occurred simultaneously. Then, $P(x_i, y_j) \approx \frac{n_{ij}}{n}$. Simple reasoning dictates that the total number, n_i^x , that value x_i occurred is equal to

$$n_i^x = \sum_{j=1}^{n_y} n_{ij}. \quad (2.8)$$

Dividing both sides in the above by n , the following *sum rule* readily results.

$$P(x) = \sum_{y \in \mathcal{Y}} P(x, y) : \text{ sum rule.} \quad (2.9)$$

The *conditional probability* of an event A , given another event B , is denoted as $P(A|B)$, and it is defined as

$$P(A|B) := \frac{P(A, B)}{P(B)} : \text{ conditional probability,} \quad (2.10)$$

provided $P(B) \neq 0$. It can be shown that this is indeed a probability, in the sense that it respects all three axioms [6]. We can better grasp its physical meaning if the relative frequency definition is adopted. Let n_{AB} be the number of times that both events occurred simultaneously, and let n_B be the number of times event B occurred, out of n experiments. Then we have

$$P(A|B) = \frac{n_{AB}}{n} \frac{n}{n_B} = \frac{n_{AB}}{n_B}. \quad (2.11)$$

In other words, the conditional probability of an event A , given another event B , is the relative frequency that A occurred, not with respect to the total number of experiments performed, but relative to the times event B occurred.

Viewed differently and adopting similar notation in terms of random variables, in conformity with Eq. (2.9), the definition of the conditional probability is also known as the *product rule* of probability, written as

$$P(x, y) = P(x|y)P(y) : \text{ product rule.} \quad (2.12)$$

To differentiate from the joint and conditional probabilities, probabilities $P(x)$ and $P(y)$ are known as *marginal probabilities*. The product rule is generalized in a straightforward way to l random variables, i.e.,

$$P(x_1, x_2, \dots, x_l) = P(x_l|x_{l-1}, \dots, x_1)P(x_{l-1}, \dots, x_1),$$

which recursively leads to the product

$$P(x_1, x_2, \dots, x_l) = P(x_l|x_{l-1}, \dots, x_1)P(x_{l-1}|x_{l-2}, \dots, x_1) \dots P(x_1).$$

Statistical independence: Two random variables are said to be statistically independent if and only if their joint probability is equal to the product of the respective marginal probabilities, i.e.,

$$P(x, y) = P(x)P(y). \quad (2.13)$$

Bayes Theorem

The Bayes theorem is a direct consequence of the product rule and the symmetry property of the joint probability, $P(x, y) = P(y, x)$, and it is stated as

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} : \text{ Bayes theorem,} \quad (2.14)$$

where the marginal, $P(x)$, can be written as

$$P(x) = \sum_{y \in \mathcal{Y}} P(x, y) = \sum_{y \in \mathcal{Y}} P(x|y)P(y),$$

and it can be considered as the normalizing constant of the numerator on the right-hand side in Eq. (2.14), which guarantees that summing up $P(y|x)$ with respect to all possible values of $y \in \mathcal{Y}$ results in one.

The Bayes theorem plays a central role in machine learning, and it will be the basis for developing Bayesian techniques for estimating the values of unknown parameters.

2.2.3 CONTINUOUS RANDOM VARIABLES

So far, we have focused on discrete random variables. Our interest now turns to the extension of the notion of probability to random variables which take values on the real axis, \mathbb{R} .

The starting point is to compute the probability of a random variable, x , to lie in an interval, $x_1 < x \leq x_2$. Note that the two events, $x \leq x_1$ and $x_1 < x \leq x_2$, are mutually exclusive. Thus, we can write that

$$P(x \leq x_1) + P(x_1 < x \leq x_2) = P(x \leq x_2). \quad (2.15)$$

We define the *cumulative distribution function* (CDF) of x as

$$\boxed{F_x(x) := P(x \leq x) : \text{ cumulative distribution function.}} \quad (2.16)$$

Then, Eq. (2.15) can be written as

$$P(x_1 < x \leq x_2) = F_x(x_2) - F_x(x_1). \quad (2.17)$$

Note that F_x is a monotonically increasing function. Furthermore, if it is continuous, the random variable x is said to be of a *continuous* type. Assuming that it is also differentiable, we can define the *probability density function* (PDF) of x as

$$\boxed{p_x(x) := \frac{dF_x(x)}{dx} : \text{ probability density function,}} \quad (2.18)$$

which then leads to

$$P(x_1 < x \leq x_2) = \int_{x_1}^{x_2} p_x(x) dx. \quad (2.19)$$

Also,

$$F_x(x) = \int_{-\infty}^x p_x(z) dz. \quad (2.20)$$

Using familiar arguments from calculus, the PDF can be interpreted as

$$\Delta P(x < x \leq x + \Delta x) \approx p_x(x)\Delta x, \quad (2.21)$$

which justifies its name as a “density” function, being the probability (ΔP) of x lying in a small interval Δx , divided by the length of this interval. Note that as $\Delta x \rightarrow 0$ this probability tends to zero. Thus, the probability of a continuous random variable taking any single value is zero. Moreover, since $P(-\infty < x < +\infty) = 1$, we have

$$\int_{-\infty}^{+\infty} p_x(x) dx = 1. \quad (2.22)$$

Usually, in order to simplify notation, the subscript x is dropped and we write $p(x)$, unless it is necessary for avoiding possible confusion. Note, also, that we have adopted the lower case “ p ” to denote a PDF and the capital “ P ” to denote a probability.

All previously stated rules concerning probabilities are readily carried out for the case of PDFs, in the following way:

$$p(x|y) = \frac{p(x, y)}{p(y)}, \quad p(x) = \int_{-\infty}^{+\infty} p(x, y) dy. \quad (2.23)$$

2.2.4 MEAN AND VARIANCE

Two of the most common and useful quantities associated with any random variable are the respective mean value and variance. The mean value (or sometimes called expected value) is denoted as

$$\mathbb{E}[x] := \int_{-\infty}^{+\infty} xp(x) dx : \text{mean value}, \quad (2.24)$$

where for discrete random variables the integration is replaced by summation ($\mathbb{E}[x] = \sum_{x \in \mathcal{X}} x P(x)$).

The variance is denoted as σ_x^2 and it is defined as

$$\sigma_x^2 := \int_{-\infty}^{+\infty} (x - \mathbb{E}[x])^2 p(x) dx : \text{variance}, \quad (2.25)$$

where integration is replaced by summation for discrete variables. The variance is a measure of the spread of the values of the random variable around its mean value.

The definition of the mean value is generalized for any function $f(x)$, i.e.,

$$\mathbb{E}[f(x)] := \int_{-\infty}^{+\infty} f(x) p(x) dx. \quad (2.26)$$

It is readily shown that the mean value with respect to two random variables, y, x , can be written as the product

$$\mathbb{E}_{x,y}[f(x, y)] = \mathbb{E}_x [\mathbb{E}_{y|x}[f(x, y)]], \quad (2.27)$$

where $\mathbb{E}_{y|x}$ denotes the mean value with respect to $p(y|x)$. This is a direct consequence of the definition of the mean value and the product rule of probabilities.

Given two random variables x, y , their *covariance* is defined as

$$\text{cov}(x, y) := \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])], \quad (2.28)$$

and their *correlation* as

$$r_{xy} := \mathbb{E}[xy] = \text{cov}(x, y) + \mathbb{E}[x]\mathbb{E}[y]. \quad (2.29)$$

A *random vector* is a collection of random variables, $\mathbf{x} = [x_1, \dots, x_l]^T$, and $p(\mathbf{x})$ is the joint PDF (probability mass for discrete variables),

$$p(\mathbf{x}) = p(x_1, \dots, x_l). \quad (2.30)$$

The *covariance matrix* of a random vector \mathbf{x} is defined as

$$\boxed{\text{Cov}(\mathbf{x}) := \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] : \text{ covariance matrix}}, \quad (2.31)$$

or

$$\text{Cov}(\mathbf{x}) = \begin{bmatrix} \text{cov}(x_1, x_1) & \dots & \text{cov}(x_1, x_l) \\ \vdots & \ddots & \vdots \\ \text{cov}(x_l, x_1) & \dots & \text{cov}(x_l, x_l) \end{bmatrix}. \quad (2.32)$$

Another symbol that will be used to denote the covariance matrix is Σ_x . Similarly, the *correlation matrix* of a random vector \mathbf{x} is defined as

$$\boxed{R_{\mathbf{x}} := \mathbb{E}[\mathbf{x}\mathbf{x}^T] : \text{ correlation matrix}}, \quad (2.33)$$

or

$$\begin{aligned} R_{\mathbf{x}} &= \begin{bmatrix} \mathbb{E}[x_1, x_1] & \dots & \mathbb{E}[x_1, x_l] \\ \vdots & \ddots & \vdots \\ \mathbb{E}[x_l, x_1] & \dots & \mathbb{E}[x_l, x_l] \end{bmatrix} \\ &= \text{Cov}(\mathbf{x}) + \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{x}^T]. \end{aligned} \quad (2.34)$$

Often, subscripts are dropped in other to simplify notation, and the corresponding symbols, e.g., r , Σ and R are used instead, unless it is necessary to avoid confusion when different random variables are involved.¹ Both the covariance and correlation matrices have a very rich structure, which will be exploited in various parts of this book to lead to computational savings whenever they are present in calculations. For the time being, observe that both are symmetric and positive semidefinite. The

¹ Note that in the subsequent chapters, to avoid having bold letters in subscripts, which can be cumbersome when more than one vector variables are involved, the notation has been slightly relaxed. For example, R_x is used in place of $R_{\mathbf{x}}$. For the sake of uniformity, the same applies to the rest of the variables corresponding to correlations, variances and covariance matrices.

symmetry, $\Sigma = \Sigma^T$, is readily deduced from the definition. An $l \times l$ symmetric matrix A is called *positive semidefinite* if

$$\mathbf{y}^T A \mathbf{y} \geq 0, \quad \forall \mathbf{y} \in \mathbb{R}^l. \quad (2.35)$$

If the inequality is a strict one, the matrix is said to be *positive definite*. For the covariance matrix, we have

$$\mathbf{y}^T \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] \mathbf{y} = \mathbb{E}[(\mathbf{y}^T (\mathbf{x} - \mathbb{E}[\mathbf{x}]))^2] \geq 0,$$

and the claim has been proved.

Complex Random Variables

A complex random variable, $\mathbf{z} \in \mathbb{C}$, is a sum

$$\mathbf{z} = \mathbf{x} + j\mathbf{y}, \quad (2.36)$$

where \mathbf{x}, \mathbf{y} are real random variables and $j := \sqrt{-1}$. Note that for complex random variables, the PDF *cannot* be defined since inequalities of the form $\mathbf{x} + j\mathbf{y} \leq \mathbf{x} + j\mathbf{y}$ have no meaning. When we write $p(\mathbf{z})$, we mean the joint PDF of the real and imaginary parts, expressed as

$$p(\mathbf{z}) := p(\mathbf{x}, \mathbf{y}). \quad (2.37)$$

For complex random variables, the notions of mean and covariance are defined as

$$\mathbb{E}[\mathbf{z}] := \mathbb{E}[\mathbf{x}] + j\mathbb{E}[\mathbf{y}], \quad (2.38)$$

and

$$\text{cov}(\mathbf{z}_1, \mathbf{z}_2) := \mathbb{E}[(\mathbf{z}_1 - \mathbb{E}[\mathbf{z}_1])(\mathbf{z}_2 - \mathbb{E}[\mathbf{z}_2])^*], \quad (2.39)$$

where “*” denotes complex conjugation. The latter definition leads to the variance of a complex variable,

$$\sigma_z^2 = \mathbb{E}[|\mathbf{z} - \mathbb{E}[\mathbf{z}]|^2] = \mathbb{E}[|z|^2] - |\mathbb{E}[z]|^2. \quad (2.40)$$

Similarly, for complex random vectors, $\mathbf{z} = \mathbf{x} + j\mathbf{y} \in \mathbb{C}^l$, we have

$$p(\mathbf{z}) := p(x_1, \dots, x_l, y_1, \dots, y_l), \quad (2.41)$$

where $x_i, y_i, i = 1, 2, \dots, l$, are the components of the involved real vectors, respectively. The covariance and correlation matrices are similarly defined as

$$\text{Cov}(\mathbf{z}) := \mathbb{E}[(\mathbf{z} - \mathbb{E}[\mathbf{z}])(\mathbf{z} - \mathbb{E}[\mathbf{z}])^H], \quad (2.42)$$

where “ H ” denotes the Hermitian (transposition and conjugation) operation.

For the rest of the chapter, we are going to deal mainly with real random variables. Whenever needed, differences with the case of complex variables will be stated.

2.2.5 TRANSFORMATION OF RANDOM VARIABLES

Let \mathbf{x} and \mathbf{y} be two random vectors, which are related via the vector transform,

$$\mathbf{y} = f(\mathbf{x}), \quad (2.43)$$

where $f : \mathbb{R}^l \mapsto \mathbb{R}^l$ is an *invertible* transform. That is, given \mathbf{y} , $\mathbf{x} = f^{-1}(\mathbf{y})$ can be uniquely obtained. We are given the joint PDF, $p_{\mathbf{x}}(\mathbf{x})$, of \mathbf{x} and the task is to obtain the joint PDF, $p_{\mathbf{y}}(\mathbf{y})$, of \mathbf{y} .

The Jacobian matrix of the transformation is defined as

$$J(\mathbf{y}; \mathbf{x}) := \frac{\partial(y_1, y_2, \dots, y_l)}{\partial(x_1, x_2, \dots, x_l)} := \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_l} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_l}{\partial x_1} & \cdots & \frac{\partial y_l}{\partial x_l} \end{bmatrix}. \quad (2.44)$$

Then, it can be shown (e.g., [6]) that

$$p_{\mathbf{y}}(\mathbf{y}) = \frac{p_{\mathbf{x}}(\mathbf{x})}{|\det(J(\mathbf{y}; \mathbf{x}))|} \Big|_{\mathbf{x}=f^{-1}(\mathbf{y})}, \quad (2.45)$$

where $|\det(\cdot)|$ denotes the absolute value of the determinant of a matrix. For real random variables, as in $\mathbf{y} = f(\mathbf{x})$, Eq. (2.45) simplifies to

$$p_{\mathbf{y}}(y) = \frac{p_{\mathbf{x}}(x)}{\left| \frac{dy}{dx} \right|} \Big|_{x=f^{-1}(y)}. \quad (2.46)$$

The latter can be graphically understood from Fig. 2.1. The following two events have equal probabilities:

$$P(x < x \leq x + \Delta x) = P(y + \Delta y < y \leq y), \quad \Delta x > 0, \quad \Delta y < 0.$$

Hence, by the definition of a PDF we have

$$p_{\mathbf{y}}(y) |\Delta y| = p_{\mathbf{x}}(x) |\Delta x|, \quad (2.47)$$

which leads to Eq. (2.46).

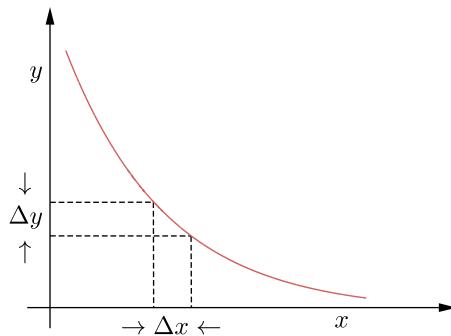
Example 2.1. Let us consider two random vectors that are related via the linear transform

$$\mathbf{y} = A\mathbf{x}, \quad (2.48)$$

where A is invertible. Compute the joint PDF of \mathbf{y} in terms of $p_{\mathbf{x}}(\mathbf{x})$.

The Jacobian of the transformation is easily computed and given by

$$J(\mathbf{y}; \mathbf{x}) = \begin{bmatrix} a_{11} & \dots & a_{1l} \\ a_{21} & \dots & a_{2l} \\ \vdots & \ddots & \vdots \\ a_{l1} & \dots & a_{ll} \end{bmatrix} = A.$$

**FIGURE 2.1**

Note that by the definition of a PDF, $p_y(y)|\Delta y| = p_x(x)|\Delta x|$.

Hence,

$$p_y(y) = \frac{p_x(A^{-1}y)}{|\det(A)|}. \quad (2.49)$$

2.3 EXAMPLES OF DISTRIBUTIONS

In this section, some notable examples of distributions are provided. These are popular for modeling the random nature of variables met in a wide range of applications, and they will be used later in this book.

2.3.1 DISCRETE VARIABLES

The Bernoulli Distribution

A random variable is said to be distributed according to a Bernoulli distribution if it is binary, $\mathcal{X} = \{0, 1\}$, with

$$P(x=1) = p, \quad P(x=0) = 1 - p.$$

In a more compact way, we write $x \sim \text{Bern}(x|p)$, where

$P(x) = \text{Bern}(x|p) := p^x(1-p)^{1-x}.$

(2.50)

Its mean value is equal to

$$\mathbb{E}[x] = 1p + 0(1-p) = p \quad (2.51)$$

and its variance is equal to

$$\sigma_x^2 = (1-p)^2 p + p^2(1-p) = p(1-p). \quad (2.52)$$

The Binomial Distribution

A random variable x is said to follow a binomial distribution with parameters n, p , and we write $x \sim \text{Bin}(x|n, p)$, if $\mathcal{X} = \{0, 1, \dots, n\}$ and

$$P(x = k) := \text{Bin}(k|n, p) = \binom{n}{k} p^k (1 - p)^{n-k}, \quad k = 0, 1, \dots, n, \quad (2.53)$$

where by definition

$$\binom{n}{k} := \frac{n!}{(n - k)!k!}. \quad (2.54)$$

For example, this distribution models the times that heads occurs in n successive trials, where $P(\text{Heads}) = p$. The binomial is a generalization of the Bernoulli distribution, which results if in Eq. (2.53) we set $n = 1$. The mean and variance of the binomial distribution are (Problem 2.1)

$$\mathbb{E}[x] = np \quad (2.55)$$

and

$$\sigma_x^2 = np(1 - p). \quad (2.56)$$

Fig. 2.2A shows the probability $P(k)$ as a function of k for $p = 0.4$ and $n = 9$. Fig. 2.2B shows the respective cumulative distribution. Observe that the latter has a staircase form, as is always the case for discrete variables.

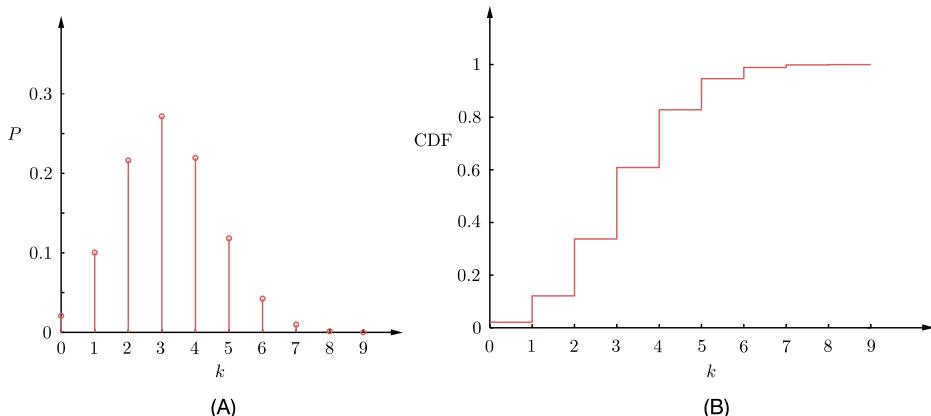


FIGURE 2.2

(A) The probability mass function (PMF) for the binomial distribution for $p = 0.4$ and $n = 9$. (B) The respective cumulative distribution function (CDF). Since the random variable is discrete, the CDF has a staircase-like graph.

The Multinomial Distribution

This is a generalization of the binomial distribution if the outcome of each experiment is not binary but can take one out of K possible values. For example, instead of tossing a coin, a die with K sides is thrown. Each one of the possible K outcomes has probability P_1, P_2, \dots, P_K , respectively, to occur, and we denote

$$\mathbf{P} = [P_1, P_2, \dots, P_K]^T.$$

After n experiments, assume that x_1, x_2, \dots, x_K times sides $x = 1, x = 2, \dots, x = K$ occurred, respectively. We say that the random (discrete) vector,

$$\mathbf{x} = [x_1, x_2, \dots, x_K]^T, \quad (2.57)$$

follows a multinomial distribution, $\mathbf{x} \sim \text{Mult}(\mathbf{x}|n, \mathbf{P})$, if

$$P(\mathbf{x}) = \text{Mult}(\mathbf{x}|n, \mathbf{P}) := \binom{n}{x_1, x_2, \dots, x_K} \prod_{k=1}^K P_k^{x_k}, \quad (2.58)$$

where

$$\binom{n}{x_1, x_2, \dots, x_K} := \frac{n!}{x_1! x_2! \dots x_K!}.$$

Note that the variables x_1, \dots, x_K are subject to the constraint

$$\sum_{k=1}^K x_k = n,$$

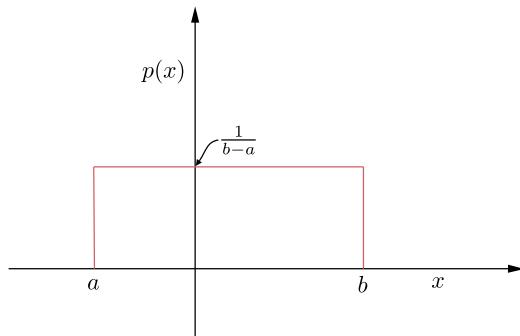
and also

$$\sum_{k=1}^K P_k = 1.$$

The mean value, the variances, and the covariances are given by

$$\mathbb{E}[\mathbf{x}] = n\mathbf{P}, \quad \sigma_{x_k}^2 = n P_k(1 - P_k), \quad k = 1, 2, \dots, K, \quad \text{cov}(x_i, x_j) = -n P_i P_j, \quad i \neq j. \quad (2.59)$$

The special case of the multinomial, where only one experiment, $n = 1$, is performed, is known as the *categorical* distribution. The latter can be considered as the generalization of the Bernoulli distribution.

**FIGURE 2.3**

The PDF of a uniform distribution $\mathcal{U}(a, b)$.

2.3.2 CONTINUOUS VARIABLES

The Uniform Distribution

A random variable x is said to follow a *uniform* distribution in an interval $[a, b]$, and we write $x \sim \mathcal{U}(a, b)$, with $a > -\infty$ and $b < +\infty$, if

$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{if } a \leq x \leq b, \\ 0, & \text{otherwise.} \end{cases} \quad (2.60)$$

Fig. 2.3 shows the respective graph. The mean value is equal to

$$\mathbb{E}[x] = \frac{a+b}{2}, \quad (2.61)$$

and the variance is given by (Problem 2.2)

$$\sigma_x^2 = \frac{1}{12}(b-a)^2. \quad (2.62)$$

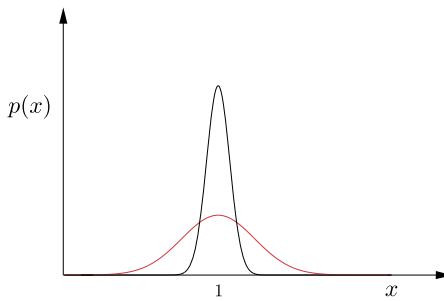
The Gaussian Distribution

The Gaussian or normal distribution is one among the most widely used distributions in all scientific disciplines. We say that a random variable x is *Gaussian* or *normal* with parameters μ and σ^2 , and we write $x \sim \mathcal{N}(\mu, \sigma^2)$ or $\mathcal{N}(x|\mu, \sigma^2)$, if

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \quad (2.63)$$

It can be shown that the corresponding mean and variance are

$$\mathbb{E}[x] = \mu \quad \text{and} \quad \sigma_x^2 = \sigma^2. \quad (2.64)$$

**FIGURE 2.4**

The graphs of two Gaussian PDFs for $\mu = 1$ and $\sigma^2 = 0.1$ (red) and $\sigma^2 = 0.01$ (gray).

Indeed, by the definition of the mean value, we have

$$\begin{aligned}\mathbb{E}[x] &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} x \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx \\ &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} (y + \mu) \exp\left(-\frac{y^2}{2\sigma^2}\right) dy.\end{aligned}\quad (2.65)$$

Due to the symmetry of the exponential function, performing the integration involving y gives zero and the only surviving term is due to μ . Taking into account that a PDF integrates to one, we obtain the result.

To derive the variance, from the definition of the Gaussian PDF, we have

$$\int_{-\infty}^{+\infty} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sqrt{2\pi}\sigma. \quad (2.66)$$

Taking the derivative of both sides with respect to σ , we obtain

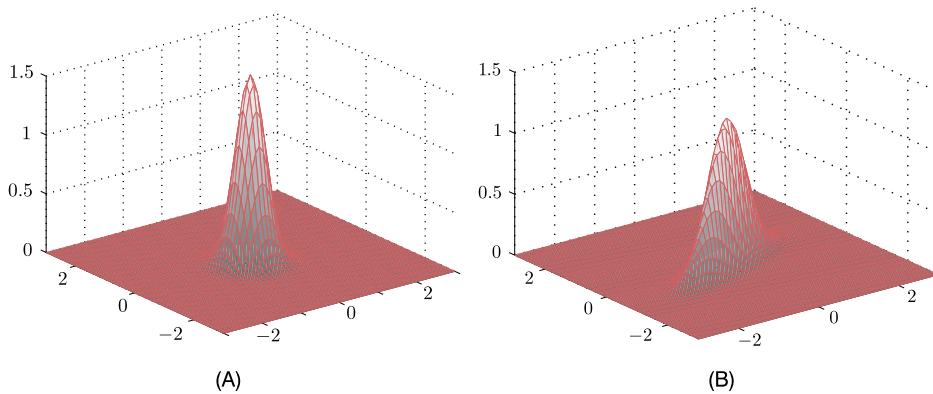
$$\int_{-\infty}^{+\infty} \frac{(x-\mu)^2}{\sigma^3} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sqrt{2\pi} \quad (2.67)$$

or

$$\sigma_x^2 := \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} (x-\mu)^2 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sigma^2, \quad (2.68)$$

which proves the claim.

Fig. 2.4 shows the graph for two cases, $\mathcal{N}(x|1, 0.1)$ and $\mathcal{N}(x|1, 0.01)$. Both curves are symmetrically placed around the mean value $\mu = 1$. Observe that the smaller the variance is, the sharper around the mean value the PDF becomes.

**FIGURE 2.5**

The graphs of two-dimensional Gaussian PDFs for $\mu = \mathbf{0}$ and different covariance matrices. (A) The covariance matrix is diagonal with equal elements along the diagonal. (B) The corresponding covariance matrix is nondiagonal.

The generalization of the Gaussian to vector variables, $\mathbf{x} \in \mathbb{R}^l$, results in the so-called *multivariate Gaussian* or *normal* distribution, $\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, which is defined as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{l/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right): \quad \text{Gaussian PDF,} \quad (2.69)$$

where $|\cdot|$ denotes the determinant of a matrix. It can be shown (Problem 2.3) that the respective mean values and the covariance matrix are given by

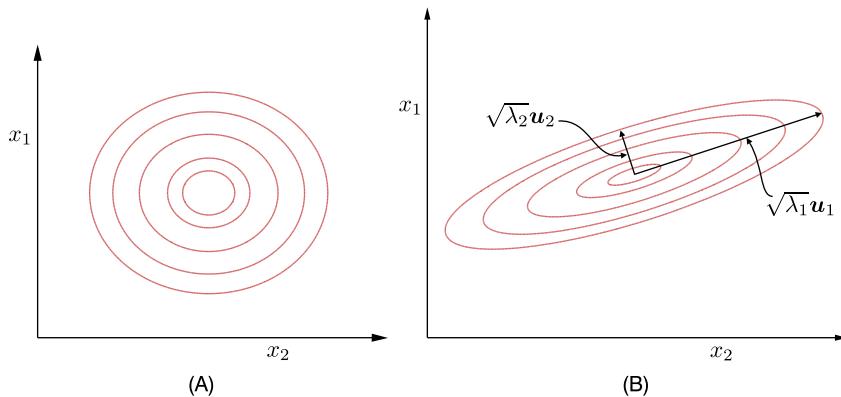
$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} \quad \text{and} \quad \boldsymbol{\Sigma}_{\mathbf{x}} = \boldsymbol{\Sigma}. \quad (2.70)$$

Fig. 2.5 shows the two-dimensional normal PDF for two cases. Both share the same mean vector, $\boldsymbol{\mu} = \mathbf{0}$, but they have different covariance matrices,

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.1 & 0.01 \\ 0.01 & 0.2 \end{bmatrix}. \quad (2.71)$$

Fig. 2.6 shows the corresponding isovalue contours for equal probability density values. In Fig. 2.6A, the contours are circles, corresponding to the symmetric PDF in Fig. 2.5A with covariance matrix $\boldsymbol{\Sigma}_1$. The one shown in Fig. 2.6B corresponds to the PDF in Fig. 2.5B associated with $\boldsymbol{\Sigma}_2$. Observe that, in general, the isovalue curves are ellipses/hyperellipsoids. They are centered at the mean vector, and the orientation of the major axis as well their exact shape is controlled by the eigenstructure of the associated covariance matrix. Indeed, all points $\mathbf{x} \in \mathbb{R}^l$, which score the same probability density value, obey

$$(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) = \text{constant} = c. \quad (2.72)$$

**FIGURE 2.6**

The isovalue contours for the two Gaussians of Fig. 2.5. The contours for the Gaussian in Fig. 2.5A are circles, while those corresponding to Fig. 2.5B are ellipses. The major and minor axes of the ellipse are determined by the eigenvectors/eigenvalues of the respective covariance matrix, and they are proportional to $\sqrt{\lambda_1}c$ and $\sqrt{\lambda_2}c$, respectively. In the figure, they are shown for the case of $c = 1$. For the case of the diagonal matrix, with equal elements along the diagonal, all eigenvalues are equal, and the ellipse becomes a circle.

We know that the covariance matrix besides being positive definite is also *symmetric*, $\Sigma = \Sigma^T$. Thus, its eigenvalues are real and the corresponding eigenvectors can be chosen to form an orthonormal basis (Appendix A.2), which leads to its diagonalization,

$$\Sigma = U \Lambda U^T, \quad (2.73)$$

with

$$U := [\mathbf{u}_1, \dots, \mathbf{u}_l], \quad (2.74)$$

where $\mathbf{u}_i, i = 1, 2, \dots, l$, are the orthonormal eigenvectors, and

$$\Lambda := \text{diag}\{\lambda_1, \dots, \lambda_l\} \quad (2.75)$$

comprise the respective eigenvalues. We assume that Σ is invertible, hence all eigenvalues are positive (being positive definite it has positive eigenvalues, Appendix A.2). Due to the orthonormality of the eigenvectors, matrix U is orthogonal as expressed in $UU^T = U^TU = I$. Thus, Eq. (2.72) can now be written as

$$\mathbf{y}^T \Lambda^{-1} \mathbf{y} = c, \quad (2.76)$$

where we have used the linear transformation

$$\mathbf{y} := U^T(\mathbf{x} - \boldsymbol{\mu}), \quad (2.77)$$

which corresponds to a rotation of the axes by U and a translation of the origin to μ . Eq. (2.76) can be written as

$$\frac{y_1^2}{\lambda_1} + \cdots + \frac{y_l^2}{\lambda_l} = c, \quad (2.78)$$

where it can be readily observed that it is an equation describing a (hyper)ellipsoid in the \mathbb{R}^l . From Eq. (2.77), it is easily seen that it is centered at μ and that the major axes of the ellipsoid are parallel to u_1, \dots, u_l (plug in place of x the standard basis vectors, $[1, 0, \dots, 0]^T$, etc.). The sizes of the respective axes are controlled by the corresponding eigenvalues. This is shown in Fig. 2.6B. For the special case of a diagonal covariance with equal elements across the diagonal, all eigenvalues are equal to the value of the common diagonal element and the ellipsoid becomes a (hyper)sphere (circle) as shown in Fig. 2.6A.

The Gaussian PDF has a number of nice properties, which we are going to discover as we move on in this book. For the time being, note that if the covariance matrix is diagonal,

$$\Sigma = \text{diag}\{\sigma_1^2, \dots, \sigma_l^2\},$$

that is, when the covariance of all the elements $\text{cov}(x_i, x_j) = 0, i, j = 1, 2, \dots, l$, then the random variables comprising \mathbf{x} are statistically *independent*. In general, this is not true. Uncorrelated variables are not necessarily independent; independence is a much stronger condition. This is true, however, if they follow a multivariate Gaussian. Indeed, if the covariance matrix is diagonal, then the multivariate Gaussian is written as

$$p(\mathbf{x}) = \prod_{i=1}^l \frac{1}{\sqrt{2\pi\sigma_i}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right). \quad (2.79)$$

In other words,

$$p(\mathbf{x}) = \prod_{i=1}^l p(x_i), \quad (2.80)$$

which is the condition for statistical independence.

The Central Limit Theorem

This is one of the most fundamental theorems in probability theory and statistics and it partly explains the popularity of the Gaussian distribution. Consider N mutually *independent* random variables, each following its own distribution with mean values μ_i and variances $\sigma_i^2, i = 1, 2, \dots, N$. Define a new random variable as their sum,

$$\mathbf{x} = \sum_{i=1}^N \mathbf{x}_i. \quad (2.81)$$

Then the mean and variance of the new variable are given by

$$\mu = \sum_{i=1}^N \mu_i \quad \text{and} \quad \sigma^2 = \sum_{i=1}^N \sigma_i^2. \quad (2.82)$$

It can be shown (e.g., [4,6]) that as $N \rightarrow \infty$ the distribution of the normalized variable

$$z = \frac{x - \mu}{\sigma} \quad (2.83)$$

tends to the *standard* normal distribution, and for the corresponding PDF we have

$$p(z) \xrightarrow{N \rightarrow \infty} \mathcal{N}(z|0, 1). \quad (2.84)$$

In practice, even summing up a relatively small number, N , of random variables, one can obtain a good approximation to a Gaussian. For example, if the individual PDFs are smooth enough and each random variable is *independent and identically distributed* (i.i.d.), a number N between 5 and 10 can be sufficient. The term i.i.d. will be used a lot in this book. The term implies that successive samples of a random variable are drawn independently from the same distribution that describes the respective variable.

The Exponential Distribution

We say that a random variable follows an exponential distribution with parameter $\lambda > 0$, if

$$p(x) = \begin{cases} \lambda \exp(-\lambda x), & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \quad (2.85)$$

The distribution has been used, for example, to model the time between arrivals of telephone calls or of a bus at a bus stop. The mean and variance can be easily computed by following simple integration rules, and they are

$$\mathbb{E}[x] = \frac{1}{\lambda}, \quad \sigma_x^2 = \frac{1}{\lambda^2}. \quad (2.86)$$

The Beta Distribution

We say that a random variable, $x \in [0, 1]$, follows a beta distribution with positive parameters, a, b , and we write, $x \sim \text{Beta}(x|a, b)$, if

$$p(x) = \begin{cases} \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1}, & \text{if } 0 \leq x \leq 1, \\ 0, & \text{otherwise,} \end{cases} \quad (2.87)$$

where $B(a, b)$ is the beta function, defined as

$$B(a, b) := \int_0^1 x^{a-1} (1-x)^{b-1} dx. \quad (2.88)$$

The mean and variance of the beta distribution are given by (Problem 2.4)

$$\mathbb{E}[x] = \frac{a}{a+b}, \quad \sigma_x^2 = \frac{ab}{(a+b)^2(a+b+1)}. \quad (2.89)$$

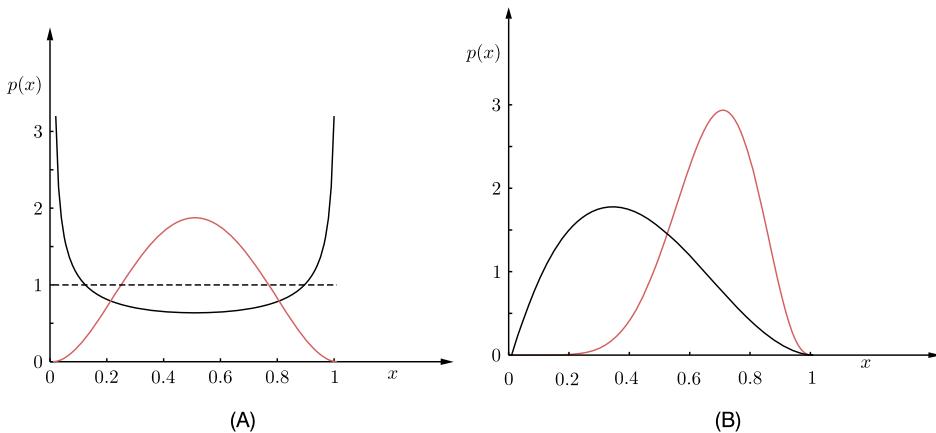


FIGURE 2.7

The graphs of the PDFs of the beta distribution for different values of the parameters. (A) The dotted line corresponds to $a = 1, b = 1$, the gray line to $a = 0.5, b = 0.5$, and the red one to $a = 3, b = 3$. (B) The gray line corresponds to $a = 2, b = 3$, and the red one to $a = 8, b = 4$. For values $a = b$, the shape is symmetric around 1/2. For $a < 1, b < 1$, it is convex. For $a > 1, b > 1$, it is zero at $x = 0$ and $x = 1$. For $a = 1 = b$, it becomes the uniform distribution. If $a < 1$, $p(x) \rightarrow \infty, x \rightarrow 0$ and if $b < 1$, $p(x) \rightarrow \infty, x \rightarrow 1$.

Moreover, it can be shown (Problem 2.5) that

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}, \quad (2.90)$$

where Γ is the gamma function defined as

$$\Gamma(a) = \int_0^\infty x^{a-1} e^{-x} dx. \quad (2.91)$$

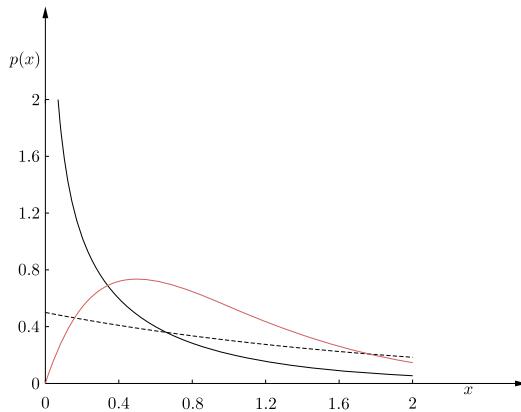
The beta distribution is very flexible and one can achieve various shapes by changing the parameters a, b . For example, if $a = b = 1$, the uniform distribution results. If $a = b$, the PDF has a symmetric graph around 1/2. If $a > 1, b > 1$, then $p(x) \rightarrow 0$ both at $x = 0$ and $x = 1$. If $a < 1$ and $b < 1$, it is convex with a unique minimum. If $a < 1$, it tends to ∞ as $x \rightarrow 0$, and if $b < 1$, it tends to ∞ for $x \rightarrow 1$. Figs. 2.7A and B show the graph of the beta distribution for different values of the parameters.

The Gamma Distribution

A random variable follows the gamma distribution with positive parameters a, b , and we write $x \sim \text{Gamma}(x|a, b)$, if

$$p(x) = \begin{cases} \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}, & x > 0, \\ 0, & \text{otherwise.} \end{cases}$$

(2.92)

**FIGURE 2.8**

The PDF of the gamma distribution takes different shapes for the various values of the following parameters: $a = 0.5, b = 1$ (full line gray), $a = 2, b = 0.5$ (red), $a = 1, b = 2$ (dotted).

The mean and variance are given by

$$\mathbb{E}[x] = \frac{a}{b}, \quad \sigma_x^2 = \frac{a}{b^2}. \quad (2.93)$$

The gamma distribution also takes various shapes by varying the parameters. For $a < 1$, it is strictly decreasing and $p(x) \rightarrow \infty$ as $x \rightarrow 0$ and $p(x) \rightarrow 0$ as $x \rightarrow \infty$. Fig. 2.8 shows the resulting graphs for various values of the parameters.

Remarks 2.1.

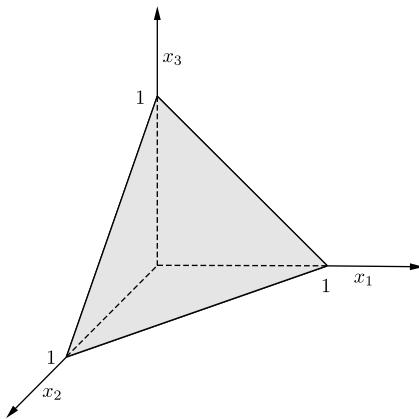
- Setting in the gamma distribution a to be an integer (usually $a = 2$), the *Erlang* distribution results. This distribution is used to model waiting times in queueing systems.
- The *chi-squared* is also a special case of the gamma distribution, and it is obtained if we set $b = 1/2$ and $a = v/2$. The chi-squared distribution results if we sum up v squared normal variables.

The Dirichlet Distribution

The Dirichlet distribution can be considered as the multivariate generalization of the beta distribution. Let $\mathbf{x} = [x_1, \dots, x_K]^T$ be a random vector, with components such as

$$0 \leq x_k \leq 1, \quad k = 1, 2, \dots, K, \quad \text{and} \quad \sum_{k=1}^K x_k = 1. \quad (2.94)$$

In other words, the random variables lie on $(K - 1)$ -dimensional *simplex*, Fig. 2.9. We say that the random vector \mathbf{x} follows a Dirichlet distribution with (positive) parameters $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_K]^T$, and we

**FIGURE 2.9**

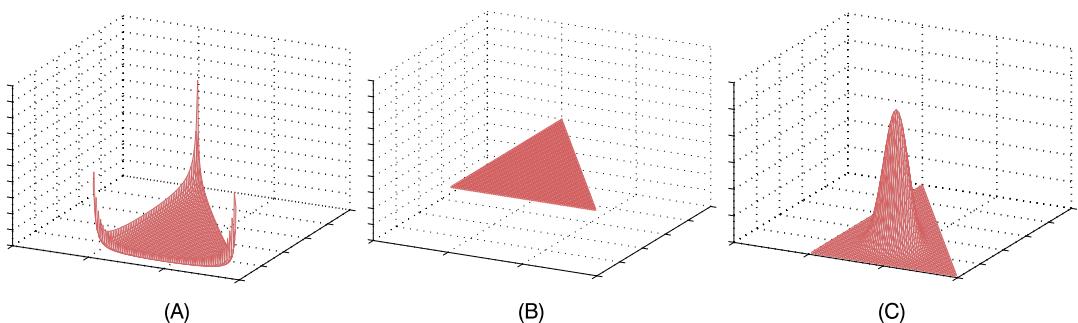
The two-dimensional simplex in \mathbb{R}^3 .

write $\mathbf{x} \sim \text{Dir}(\mathbf{x}|\boldsymbol{\alpha})$, if

$$p(\mathbf{x}) = \text{Dir}(\mathbf{x}|\boldsymbol{\alpha}) := \frac{\Gamma(\bar{\alpha})}{\Gamma(a_1)\dots\Gamma(a_K)} \prod_{k=1}^K x_k^{a_k-1}, \quad (2.95)$$

where

$$\bar{\alpha} = \sum_{k=1}^K a_k. \quad (2.96)$$

**FIGURE 2.10**

The Dirichlet distribution over the two-dimensional simplex for (A) $(0.1, 0.1, 0.1)$, (B) $(1, 1, 1)$, and (C) $(10, 10, 10)$.

The mean, variance, and covariances of the involved random variables are given by (Problem 2.7), i.e.,

$$\mathbb{E}[\mathbf{x}] = \frac{1}{\bar{a}}\mathbf{a}, \quad \sigma_{x_k}^2 = \frac{a_k(\bar{a} - a_k)}{\bar{a}^2(\bar{a} + 1)}, \quad \text{cov}(x_i, x_j) = -\frac{a_i a_j}{\bar{a}^2(\bar{a} + 1)}, \quad i \neq j. \quad (2.97)$$

Fig. 2.10 shows the graph of the Dirichlet distribution for different values of the parameters, over the respective two-dimensional simplex.

2.4 STOCHASTIC PROCESSES

The notion of a random variable has been introduced to describe the result of a random experiment whose outcome is a single value, such as heads or tails in a coin tossing experiment, or a value between one and six when throwing the die in a backgammon game.

In this section, the notion of a *stochastic process* is introduced to describe random experiments where the outcome of each experiment is a function or a sequence; in other words, the outcome of each experiment is an infinite number of values. In this book, we are only going to be concerned with stochastic processes associated with sequences. Thus, the result of a random experiment is a *sequence*, u_n (or sometimes denoted as $u(n)$), $n \in \mathbb{Z}$, where \mathbb{Z} is the set of integers. Usually, n is interpreted as a time index, and u_n is called a *time series*, or in signal processing jargon, a *discrete-time signal*. In contrast, if the outcome is a function, $u(t)$, it is called a *continuous-time signal*. We are going to adopt the time interpretation of the free variable n for the rest of the chapter, without harming generality.

When discussing random variables, we used the notation x to denote the random variable, which assumes a value, x , from the sample space once an experiment is performed. Similarly, we are going to use u_n to denote the specific sequence resulting from a single experiment and the roman font, u_n , to denote the corresponding *discrete-time* random process, that is, the rule that assigns a specific sequence as the outcome of an experiment. A stochastic process can be considered as a family or *ensemble* of sequences. The individual sequences are known as *sample sequences* or simply as *realizations*.

For our notational convention, in general, we are going to reserve different symbols for processes and random variables. We have already used the symbol u and not x ; this is only for pedagogical reasons, just to make sure that the reader readily recognizes when the focus is on random variables and when it is on random processes. In signal processing jargon, a stochastic process is also known as a *random signal*. Fig. 2.11 illustrates the fact that the outcome of an experiment involving a stochastic process is a sequence of values.

Note that fixing the time to a specific value, $n = n_0$, makes u_{n_0} a random variable. Indeed, for each random experiment we perform, a single value results at time instant n_0 . From this perspective, a random process can be considered the collection of infinite random variables, $\{u_n, n \in \mathbb{Z}\}$. So, is there a need to study a stochastic process separate from random variables/vectors? The answer is yes, and the reason is that we are going to allow certain time dependencies among the random variables, corresponding to different time instants, and study the respective effect on the time evolution of the random process. Stochastic processes will be considered in Chapter 5, where the underlying time dependencies will be exploited for computational simplifications, and in Chapter 13 in the context of Gaussian processes.

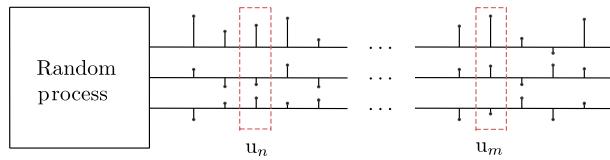


FIGURE 2.11

The outcome of each experiment, associated with a *discrete-time* stochastic process, is a *sequence* of values. For each one of the realizations, the corresponding values obtained at any instant (e.g., n or m) comprise the outcomes of a corresponding random variable, u_n or u_m , respectively.

2.4.1 FIRST- AND SECOND-ORDER STATISTICS

For a stochastic process to be fully described, one must know the joint PDFs (PMFs for discrete-valued random variables)

$$p(u_n, u_m, \dots, u_r; n, m, \dots, r), \quad (2.98)$$

for all possible combinations of random variables, u_n, u_m, \dots, u_r . Note that, in order to emphasize it, we have explicitly denoted the dependence of the joint PDFs on the involved time instants. However, from now on, this will be suppressed for notational convenience. Most often, in practice, and certainly in this book, the emphasis is on computing first- and second-order statistics only, based on $p(u_n)$ and $p(u_n, u_m)$. To this end, the following quantities are of particular interest.

Mean at time n :

$$\mu_n := \mathbb{E}[u_n] = \int_{-\infty}^{+\infty} u_n p(u_n) du_n. \quad (2.99)$$

Autocovariance at time instants n, m :

$$\text{cov}(n, m) := \mathbb{E}[(u_n - \mathbb{E}[u_n])(u_m - \mathbb{E}[u_m])]. \quad (2.100)$$

Autocorrelation at time instants n, m :

$$r(n, m) := \mathbb{E}[u_n u_m]. \quad (2.101)$$

Note that for notational simplicity, subscripts have been dropped from the respective symbols, e.g., $r(n, m)$ is used instead of the more formal notation, $r_u(n, m)$. We refer to these mean values as *ensemble* averages to stress that they convey statistical information over the ensemble of sequences that comprise the process.

The respective definitions for complex stochastic processes are

$$\text{cov}(n, m) = \mathbb{E}[(u_n - \mathbb{E}[u_n])(u_m - \mathbb{E}[u_m])^*], \quad (2.102)$$

and

$$r(n, m) = \mathbb{E}[u_n u_m^*]. \quad (2.103)$$

2.4.2 STATIONARITY AND ERGODICITY

Definition 2.1 (*Strict-sense stationarity*). A stochastic process u_n is said to be *strict-sense stationary* (SSS) if its statistical properties are invariant to a shift of the origin, or if $\forall k \in \mathbb{Z}$

$$p(u_n, u_m, \dots, u_r) = p(u_{n-k}, u_{m-k}, \dots, u_{r-k}), \quad (2.104)$$

and for any possible combination of time instants, $n, m, \dots, r \in \mathbb{Z}$.

In other words, the stochastic processes u_n and u_{n-k} are described by the same joint PDFs of all orders. A weaker version of stationarity is that of the m th-order stationarity, where joint PDFs involving up to m variables are invariant to the choice of the origin. For example, for a second-order ($m = 2$) stationary process, we have $p(u_n) = p(u_{n-k})$ and $p(u_n, u_r) = p(u_{n-k}, u_{r-k}), \forall n, r, k \in \mathbb{Z}$.

Definition 2.2 (*Wide-sense stationarity*). A stochastic process u_n is said to be *wide-sense stationary* (WSS) if the mean value is constant over all time instants and the autocorrelation/autocovariance sequences depend on the difference of the involved time indices, or

$$\mu_n = \mu, \quad \text{and} \quad r(n, n - k) = r(k). \quad (2.105)$$

Note that WSS is a weaker version of the second-order stationarity; in the latter case, all possible second-order statistics are independent of the time origin. In the former, we only require the autocorrelation (autocovariance) and the mean value to be independent of the time origin. The reason we focus on these two quantities (statistics) is that they are of major importance in the study of linear systems and in the mean-square estimation, as we will see in Chapter 4.

Obviously, an SSS process is also WSS but, in general, not the other way around. For WSS processes, the autocorrelation becomes a *sequence* with a *single* time index as the free parameter; thus its value, which measures a relation of the random variables at two time instants, depends *solely on how much these time instants differ*, and not on their specific values.

From our basic statistics course, we know that given a random variable x , its mean value can be approximated by the sample mean. Carrying out N successive independent experiments, let $x_n, n = 1, 2, \dots, N$, be the obtained values, known as *observations*. The *sample mean* is defined as

$$\hat{\mu}_N := \frac{1}{N} \sum_{n=1}^N x_n. \quad (2.106)$$

For large enough values of N , we expect the sample mean to be close to the true mean value, $\mathbb{E}[x]$. In a more formal way, this is guaranteed by the fact that $\hat{\mu}_N$ is associated with an *unbiased* and *consistent* estimator. We will discuss such issues in Chapter 3; however, we can refresh our memory at this point. Every time we repeat the N random experiments, different samples result and hence a different estimate $\hat{\mu}_N$ is computed. Thus, the values of the estimates define a new random variable, $\hat{\mu}_n$, known as the estimator. This is unbiased, because it can easily be shown that

$$\mathbb{E}[\hat{\mu}_N] = \mathbb{E}[x], \quad (2.107)$$

and it is consistent because its variance tends to zero as $N \rightarrow +\infty$ (Problem 2.8). These two properties guarantee that, with high probability, for large values of N , $\hat{\mu}_N$ will be close to the true mean value.

To apply the concept of sample mean approximation to random processes, one must have at her/his disposal a number of N realizations, and compute the sample mean at different time instants “across the process,” using *different* realizations, representing the ensemble of sequences. Similarly, sample mean arguments can be used to approximate the autocovariance/autocorrelation sequences. However, this is a costly operation, since now each experiment results in an infinite number of values (a sequence of values). Moreover, it is common in practical applications that only one realization is available to the user.

To this end, we will now define a special type of stochastic processes, where the sample mean operation can be significantly simplified.

Definition 2.3 (Ergodicity). A stochastic process is said to be *ergodic* if the complete statistics can be determined by any one of the realizations.

In other words, if a process is ergodic, every single realization carries identical statistical information and it can describe the entire random process. Since from a single sequence only one set of PDFs can be obtained, we conclude that *every ergodic process is necessarily stationary*. A nonstationary process has infinite sets of PDFs, depending upon the choice of the origin. For example, there is only one mean value that can result from a single realization and be obtained as a (time) average over the values of the sequence. Hence, the mean value of a stochastic process that is ergodic must be constant for all time instants, or independent of the time origin. The same is true for all higher-order statistics.

A special type of ergodicity is that of the *second-order* ergodicity. This means that only statistics up to a second order can be obtained from a single realization. Second-order ergodic processes are necessarily WSS. For second-order ergodic processes, the following are true:

$$\mathbb{E}[u_n] = \mu = \lim_{N \rightarrow \infty} \hat{\mu}_N, \quad (2.108)$$

where

$$\hat{\mu}_N := \frac{1}{2N+1} \sum_{n=-N}^N u_n.$$

Also,

$$\text{cov}(k) = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{n=-N}^N (u_n - \mu)(u_{n-k} - \mu), \quad (2.109)$$

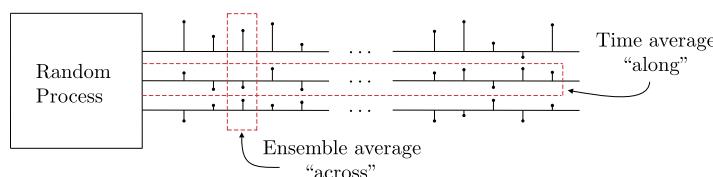


FIGURE 2.12

For ergodic processes, the common mean value, for all time instants (ensemble averaging “across” the process), is computed as the time average “along” the process.

where both limits are in the mean-square sense; that is,

$$\lim_{N \rightarrow \infty} \mathbb{E} [|\hat{\mu}_N - \mu|^2] = 0,$$

and similarly for the autocovariance. Note that often, ergodicity is only required to be assumed for the computation of the mean and covariance and not for all possible second-order statistics. In this case, we talk about *mean-ergodic* and *covariance-ergodic* processes.

In summary, when ergodic processes are involved, ensemble averages “across the process” can be obtained as time averages “along the process”; see Fig. 2.12.

In practice, when only a finite number of samples from a realization is available, then the mean and covariance are approximated as the respective sample means.

An issue is to establish conditions under which a process is mean-ergodic or covariance-ergodic. Such conditions do exist, and the interested reader can find such information in more specialized books [6]. It turns out that the condition for mean-ergodicity relies on second-order statistics and the condition for covariance-ergodicity on fourth-order statistics.

It is very common in statistics as well as in machine learning and signal processing to subtract the mean value from the data during the *preprocessing stage*. In such a case, we say that the data are *centered*. The resulting new process has now *zero mean* value, and the covariance and autocorrelation sequences coincide. From now on, we will assume that the mean is known (or computed as a sample mean) and then subtracted. Such a treatment simplifies the analysis without harming generality.

Example 2.2. The goal of this example is to construct a process that is WSS yet not ergodic. Let a WSS process, u_n ,

$$\mathbb{E}[u_n] = \mu,$$

and

$$\mathbb{E}[u_n u_{n-k}] = r_u(k).$$

Define the process

$$v_n := a u_n, \quad (2.110)$$

where a is a random variable taking values in $\{0, 1\}$, with probabilities $P(0) = P(1) = 0.5$. Moreover, a and u_n are statistically independent. Then, we have

$$\mathbb{E}[v_n] = \mathbb{E}[au_n] = \mathbb{E}[a]\mathbb{E}[u_n] = 0.5\mu, \quad (2.111)$$

and

$$\mathbb{E}[v_n v_{n-k}] = \mathbb{E}[a^2] \mathbb{E}[u_n u_{n-k}] = 0.5r_u(k). \quad (2.112)$$

Thus, v_n is WSS. However, it is not ergodic. Indeed, some of the realizations will be equal to zero (when $a = 0$), and the mean value and autocorrelation, which will result from them as time averages, will be zero, which is different from the ensemble averages.

2.4.3 POWER SPECTRAL DENSITY

The Fourier transform is an indispensable tool for representing in a compact way, in the frequency domain, the variations that a function/sequence undergoes in terms of its free variable (e.g., time). Stochastic processes are inherently related to time. The question that is now raised is whether stochastic processes can be described in terms of a Fourier transform. The answer is affirmative, and the vehicle to achieve this is via the autocorrelation sequence for processes that are at least WSS. Prior to providing the necessary definitions, it is useful to summarize some common properties of the autocorrelation sequence.

Properties of the Autocorrelation Sequence

Let u_n be a WSS process. Its autocorrelation sequence has the following properties, which are given for the more general complex-valued case:

- *Property I.*

$$r(k) = r^*(-k), \quad \forall k \in \mathbb{Z}. \quad (2.113)$$

This property is a direct consequence of the invariance with respect to the choice of the origin. Indeed,

$$r(k) = \mathbb{E}[u_n u_{n-k}^*] = \mathbb{E}[u_{n+k} u_n^*] = r^*(-k).$$

- *Property II.*

$$r(0) = \mathbb{E}[|u_n|^2]. \quad (2.114)$$

That is, the value of the autocorrelation at $k = 0$ is equal to the mean-square of the magnitude of the respective random variables. Interpreting the square of the magnitude of a variable as its energy, $r(0)$ can be interpreted as the corresponding (average) power.

- *Property III.*

$$r(0) \geq |r(k)|, \quad \forall k \neq 0. \quad (2.115)$$

The proof is provided in Problem 2.9. In other words, the correlation of the variables, corresponding to two different time instants, cannot be larger (in magnitude) than $r(0)$. As we will see in Chapter 4, this property is essentially the Cauchy–Schwarz inequality for the inner products (see also Appendix of Chapter 8).

- *Property IV.* The autocorrelation sequence of a stochastic process is *positive definite*. That is,

$$\sum_{n=1}^N \sum_{m=1}^N a_n a_m^* r(n, m) \geq 0, \quad \forall a_n \in \mathbb{C}, \quad n = 1, 2, \dots, N, \quad \forall N \in \mathbb{Z}. \quad (2.116)$$

Proof. The proof is easily obtained by the definition of the autocorrelation,

$$0 \leq \mathbb{E}\left[\left|\sum_{n=1}^N a_n u_n\right|^2\right] = \sum_{n=1}^N \sum_{m=1}^N a_n a_m^* \mathbb{E}[u_n u_m^*], \quad (2.117)$$

which proves the claim. Note that strictly speaking, we should say that it is semipositive definite. However, the “positive definite” name is the one that has survived in the literature. This property will be useful when introducing *Gaussian processes* in Chapter 13. \square

- *Property V.* Let u_n and v_n be two WSS processes. Define the new process

$$z_n = u_n + v_n.$$

Then,

$$r_z(k) = r_u(k) + r_v(k) + r_{uv}(k) + r_{vu}(k), \quad (2.118)$$

where the *cross-correlation* between two jointly WSS stochastic processes is defined as

$$r_{uv}(k) := \mathbb{E}[u_n v_{n-k}^*], \quad k \in \mathbb{Z}: \quad \text{cross-correlation.} \quad (2.119)$$

The proof is a direct consequence of the definition. Note that if the two processes are *uncorrelated*, as when $r_{uv}(k) = r_{vu}(k) = 0$, then

$$r_z(k) = r_u(k) + r_v(k).$$

Obviously, this is also true if the processes u_n and v_n are independent and of zero mean value, since then $\mathbb{E}[u_n v_{n-k}^*] = \mathbb{E}[u_n] \mathbb{E}[v_{n-k}^*] = 0$. It should be stressed here that uncorrelatedness is a weaker condition and it *does not* necessarily imply independence; the opposite is true for zero mean values.

- *Property VI.*

$$r_{uv}(k) = r_{vu}^*(-k). \quad (2.120)$$

The proof is similar to that of Property I.

- *Property VII.*

$$r_u(0)r_v(0) \geq |r_{uv}(k)|^2, \quad \forall k \in \mathbb{Z}. \quad (2.121)$$

The proof is also given in Problem 2.9.

Power Spectral Density

Definition 2.4. Given a WSS stochastic process u_n , its *power spectral density* (PSD) (or simply the *power spectrum*) is defined as the Fourier transform of its autocorrelation sequence,

$$S(\omega) := \sum_{k=-\infty}^{\infty} r(k) \exp(-j\omega k): \quad \text{power spectral density.} \quad (2.122)$$

Using the Fourier transform properties, we can recover the autocorrelation sequence via the *inverse Fourier transform* in the following manner:

$$r(k) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\omega) \exp(j\omega k) d\omega. \quad (2.123)$$

Due to the properties of the autocorrelation sequence, the PSD has some interesting and useful properties, from a practical point of view.

Properties of the PSD

- The PSD of a WSS stochastic process is a *real* and *nonnegative* function of ω . Indeed, we have

$$\begin{aligned}
 S(\omega) &= \sum_{k=-\infty}^{+\infty} r(k) \exp(-j\omega k) \\
 &= r(0) + \sum_{k=-\infty}^{-1} r(k) \exp(-j\omega k) + \sum_{k=1}^{\infty} r(k) \exp(-j\omega k) \\
 &= r(0) + \sum_{k=1}^{+\infty} r^*(k) \exp(j\omega k) + \sum_{k=1}^{\infty} r(k) \exp(-j\omega k) \\
 &= r(0) + 2 \sum_{k=1}^{+\infty} \text{Real}\{r(k) \exp(-j\omega k)\}, \tag{2.124}
 \end{aligned}$$

which proves the claim that PSD is a real number.² In the proof, Property I of the autocorrelation sequence has been used. We defer the proof concerning the nonnegativity to the end of this section.

- The area under the graph of $S(\omega)$ is proportional to the power of the stochastic process, as expressed by

$$\mathbb{E}[|u_n|^2] = r(0) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\omega) d\omega, \tag{2.125}$$

which is obtained from Eq. (2.123) if we set $k = 0$. We will come to the physical meaning of this property very soon.

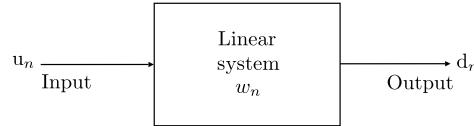
Transmission Through a Linear System

One of the most important tasks in signal processing and systems theory is the *linear filtering operation* on an *input* time series (*signal*) to generate another *output* sequence. The block diagram of the filtering operation is shown in Fig. 2.13. From the linear system theory and signal processing basics, it is established that for a class of linear systems known as *linear time-invariant*, the input–output relation is given via the elegant *convolution* between the input sequence and the *impulse response* of the filter,

$$d_n = w_n * u_n := \sum_{i=-\infty}^{+\infty} w_i^* u_{n-i} : \text{ convolution sum,} \tag{2.126}$$

where $\dots, w_0, w_1, w_2, \dots$ are the parameters comprising the impulse response describing the filter [8]. In case the impulse response is of finite duration, for example, w_0, w_1, \dots, w_{l-1} , and the rest of the

² Recall that if $z = a + jb$ is a complex number, its real part $\text{Real}\{z\} = a = \frac{1}{2}(z + z^*)$.

**FIGURE 2.13**

The linear system (filter) is excited by the input sequence (signal) u_n and provides the output sequence (signal) d_n .

values are zero, then the convolution can be written as

$$d_n = \sum_{i=0}^{l-1} w_i^* u_{n-i} = \mathbf{w}^H \mathbf{u}_n, \quad (2.127)$$

where

$$\mathbf{w} := [w_0, w_1, \dots, w_{l-1}]^T, \quad (2.128)$$

and

$$\mathbf{u}_n := [u_n, u_{n-1}, \dots, u_{n-l+1}]^T \in \mathbb{R}^l. \quad (2.129)$$

The latter is known as the *input vector* of order l and at time n . It is interesting to note that this is a random vector. However, its elements are part of the stochastic process at *successive* time instants. This gives the respective autocorrelation matrix certain properties and a rich structure, which will be studied and exploited in Chapter 4. As a matter of fact, this is the reason that we used different symbols to denote processes and general random vectors; thus, the reader can readily remember that when dealing with a process, the elements of the involved random vectors have this *extra structure*. Moreover, observe from Eq. (2.126) that if the impulse response of the system is zero for negative values of the time index n , this guarantees *causality*. That is, the output depends only on the values of the input at the current and previous time instants, and there is no dependence on future values. As a matter of fact, this is also a necessary condition for causality; that is, if the system is causal, then its impulse response is zero for negative time instants [8].

Theorem 2.1. *The PSD of the output d_n of a linear time-invariant system, when it is excited by a WSS stochastic process u_n , is given by*

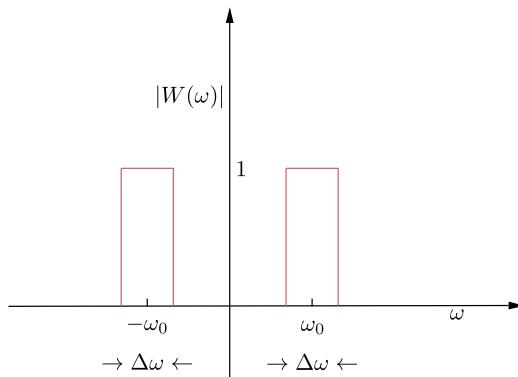
$$S_d(\omega) = |W(\omega)|^2 S_u(\omega), \quad (2.130)$$

where

$$W(\omega) := \sum_{n=-\infty}^{+\infty} w_n \exp(-j\omega n). \quad (2.131)$$

Proof. First, it is shown (Problem 2.10) that

$$r_d(k) = r_u(k) * w_k * w_{-k}^*. \quad (2.132)$$

**FIGURE 2.14**

An ideal bandpass filter. The output contains frequencies only in the range of $|\omega - \omega_o| < \Delta\omega/2$.

Then, taking the Fourier transform of both sides, we obtain Eq. (2.130). To this end, we used the well-known properties of the Fourier transform,

$$r_u(k) * w_k \longmapsto S_u(\omega)W(\omega), \quad \text{and} \quad w_{-k}^* \longmapsto W^*(\omega).$$

□

Physical Interpretation of the PSD

We are now ready to justify why the Fourier transform of the autocorrelation sequence was given the specific name of “power spectral density.” We restrict our discussion to real processes, although similar arguments hold true for the more general complex case. Fig. 2.14 shows the magnitude of the Fourier transform of the impulse response of a very special linear system. The Fourier transform is unity for any frequency in the range $|\omega - \omega_o| \leq \frac{\Delta\omega}{2}$ and zero otherwise. Such a system is known as *bandpass filter*. We assume that $\Delta\omega$ is very small. Then, using Eq. (2.130) and assuming that within the intervals $|\omega - \omega_o| \leq \frac{\Delta\omega}{2}$, $S_u(\omega) \approx S_u(\omega_o)$, we have

$$S_d(\omega) = \begin{cases} S_u(\omega_o), & \text{if } |\omega - \omega_o| \leq \frac{\Delta\omega}{2}, \\ 0, & \text{otherwise.} \end{cases} \quad (2.133)$$

Hence,

$$\Delta P := \mathbb{E}[|d_n|^2] = r_d(0) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S_d(\omega) d\omega \approx S_u(\omega_o) \frac{\Delta\omega}{\pi}, \quad (2.134)$$

due to the symmetry of the PSD ($S_u(\omega) = S_u(-\omega)$). Hence,

$$\frac{1}{\pi} S_u(\omega_o) = \frac{\Delta P}{\Delta\omega}. \quad (2.135)$$

In other words, the value $S_u(\omega_o)$ can be interpreted as the power density (power per frequency interval) in the frequency (spectrum) domain.

Moreover, this also establishes what was said before that the PSD is a *nonnegative* real function for any value of $\omega \in [-\pi, +\pi]$ (the PSD, being the Fourier transform of a sequence, is periodic with period 2π , e.g., [8]).

Remarks 2.2.

- Note that for any WSS stochastic process, there is only one autocorrelation sequence that describes it. However, the converse is not true. A single autocorrelation sequence can correspond to more than one WSS process. Recall that the autocorrelation is the mean value of the product of random variables. However, many random variables can have the same mean value.
- We have shown that the Fourier transform, $S(\omega)$, of an autocorrelation sequence, $r(k)$, is nonnegative. Moreover, if a sequence $r(k)$ has a nonnegative Fourier transform, then it is positive definite and we can *always* construct a WSS process that has $r(k)$ as its autocorrelation sequence (e.g., [6, pages 410, 421]). Thus, the *necessary and sufficient condition for a sequence to be an autocorrelation sequence is the nonnegativity of its Fourier transform*.

Example 2.3 (White noise sequence). A stochastic process η_n is said to be *white noise* if the mean and its autocorrelation sequence satisfy

$$\mathbb{E}[\eta_n] = 0 \quad \text{and} \quad r(k) = \begin{cases} \sigma_\eta^2, & \text{if } k = 0, \\ 0, & \text{if } k \neq 0. \end{cases} : \quad \text{white noise,} \quad (2.136)$$

where σ_η^2 is its variance. In other words, all variables at different time instants are uncorrelated. If, in addition, they are independent, we say that it is *strictly white noise*. It is readily seen that its PSD is given by

$$S_\eta(\omega) = \sigma_\eta^2. \quad (2.137)$$

That is, it is constant, and this is the reason it is called white noise, analogous to white light, whose spectrum is equally spread over all wavelengths.

2.4.4 AUTOREGRESSIVE MODELS

We have just seen an example of a stochastic process, namely, white noise. We now turn our attention to generating WSS processes via appropriate modeling. In this way, we will introduce controlled correlation among the variables, corresponding to the various time instants. We focus on the real data case, to simplify the discussion.

Autoregressive processes are among the most popular and widely used models. An autoregressive process of order l , denoted as AR(l), is defined via the following *difference equation*:

$$u_n + a_1 u_{n-1} + \cdots + a_l u_{n-l} = \eta_n : \quad \text{autoregressive process,} \quad (2.138)$$

where η_n is a white noise process with variance σ_η^2 .

As is always the case with any difference equation, one starts from some initial conditions and then generates samples recursively by plugging into the model the input sequence samples. The input

samples here correspond to a white noise sequence and the initial conditions are set equal to zero, $u_{-1} = \dots = u_{-l} = 0$.

There is no need to mobilize mathematics to see that such a process is not stationary. Indeed, time instant $n = 0$ is distinctly different from all the rest, since it is the time in which initial conditions are applied. However, the effects of the initial conditions tend asymptotically to zero if all the roots of the corresponding characteristic polynomial,

$$z^l + a_1 z^{l-1} + \dots + a_l = 0,$$

have magnitude *less than unity* (the solution of the corresponding homogeneous equation, without input, tends to zero) [7]. Then, it can be shown that asymptotically, the AR(l) becomes WSS. This is the assumption that is usually adopted in practice, which will be the case for the rest of this section. Note that the mean value of the process is zero (try it).

The goal now becomes to compute the corresponding autocorrelation sequence, $r(k)$, $k \in \mathbb{Z}$. Multiplying both sides in Eq. (2.138) with u_{n-k} , $k > 0$, and taking the expectation, we obtain

$$\sum_{i=0}^l a_i \mathbb{E}[u_{n-i} u_{n-k}] = \mathbb{E}[\eta_n u_{n-k}], \quad k > 0,$$

where $a_0 := 1$, or

$$\sum_{i=0}^l a_i r(k-i) = 0. \quad (2.139)$$

We have used the fact that $\mathbb{E}[\eta_n u_{n-k}]$, $k > 0$, is zero. Indeed, u_{n-k} depends recursively on $\eta_{n-k}, \eta_{n-k-1}, \dots$, which are all uncorrelated to η_n , since this is a white noise process. Note that Eq. (2.139) is a difference equation, which can be solved provided we have the initial conditions. To this end, multiply Eq. (2.138) by u_n and take expectations, which results in

$$\sum_{i=0}^l a_i r(i) = \sigma_\eta^2, \quad (2.140)$$

since u_n recursively depends on η_n , which contributes the σ_η^2 term, and η_{n-1}, \dots , which result to zeros. Combining Eqs. (2.140)–(2.139) the following *linear* system of equations results:

$$\begin{bmatrix} r(0) & r(1) & \dots & r(l) \\ r(1) & r(0) & \dots & r(l-1) \\ \vdots & \vdots & \vdots & \vdots \\ r(l) & r(l-1) & \dots & r(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_l \end{bmatrix} = \begin{bmatrix} \sigma_\eta^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \quad (2.141)$$

These are known as the *Yule–Walker equations*, whose solution results in the values $r(0), \dots, r(l)$, which are then used as the initial conditions to solve the difference equation in (2.139) and obtain $r(k)$, $\forall k \in \mathbb{Z}$.

Observe the special structure of the matrix in the linear system. This type of matrix is known as *Toeplitz*, and this is the property that will be exploited to efficiently solve such systems, which result when the autocorrelation matrix of a WSS process is involved; see Chapter 4.

Besides the autoregressive models, other types of stochastic models have been suggested and used. The *autoregressive-moving average* (ARMA) model of order (l, m) is defined by the difference equation

$$u_n + a_1 u_{n-1} + \dots + a_l u_{n-l} = b_1 \eta_n + \dots + b_m \eta_{n-m}, \quad (2.142)$$

and the *moving average* model of order m , denoted as MA(m), is defined as

$$u_n = b_1 \eta_n + \dots + b_m \eta_{n-m}. \quad (2.143)$$

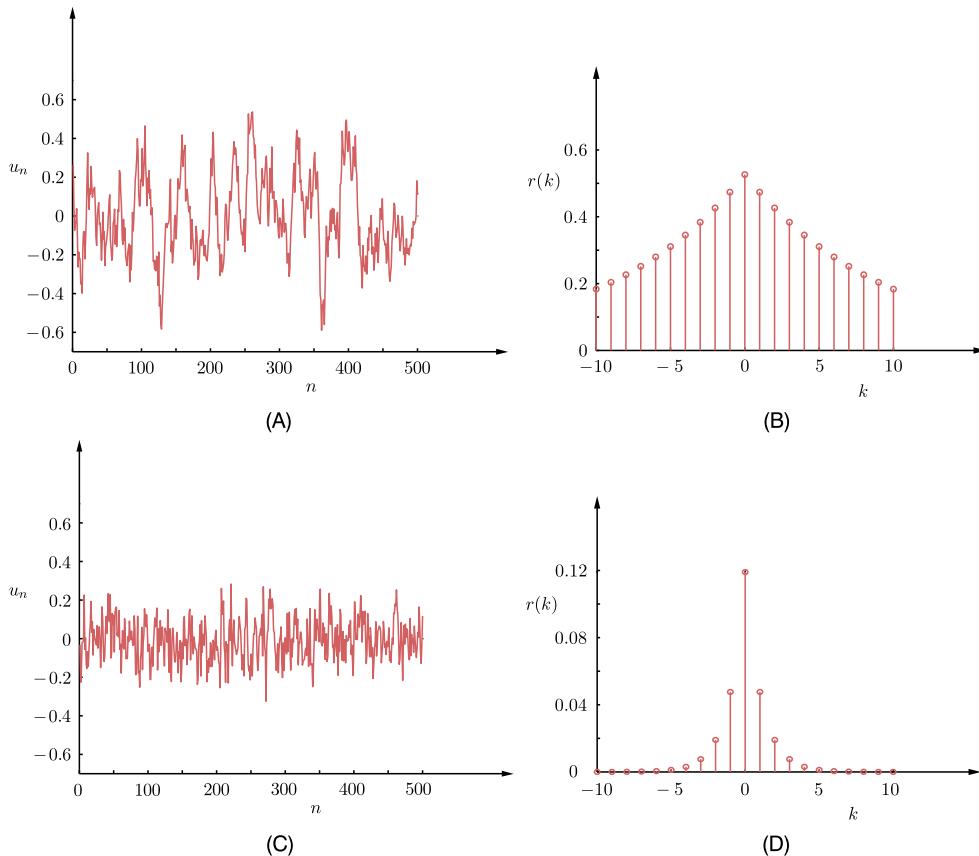


FIGURE 2.15

- (A) The time evolution of a realization of the AR(1) with $a = -0.9$ and (B) the respective autocorrelation sequence.
 (C) The time evolution of a realization of the AR(1) with $a = -0.4$ and (D) the corresponding autocorrelation sequence.

Note that the AR(1) and the MA(m) models can be considered as special cases of the ARMA(l, m). For a more theoretical treatment of the topic, see [1].

Example 2.4. Consider the AR(1) process,

$$u_n + au_{n-1} = \eta_n.$$

Following the general methodology explained before, we have

$$\begin{aligned} r(k) + ar(k-1) &= 0, \quad k = 1, 2, \dots, \\ r(0) + ar(1) &= \sigma_\eta^2. \end{aligned}$$

Taking the first equation for $k = 1$ together with the second one readily results in

$$r(0) = \frac{\sigma_\eta^2}{1 - a^2}.$$

Plugging this value into the difference equation, we recursively obtain

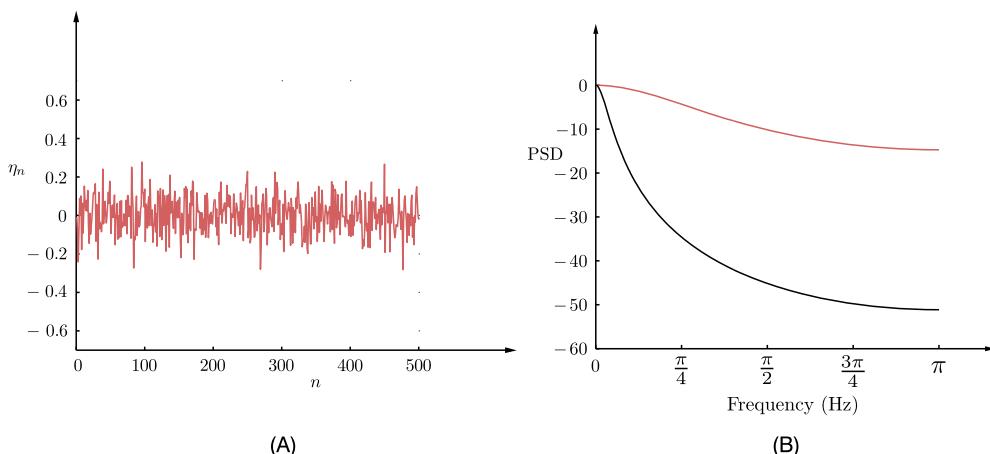
$$r(k) = (-a)^{|k|} \frac{\sigma_\eta^2}{1 - a^2}, \quad k = 0, \pm 1, \pm 2, \dots, \quad (2.144)$$

where we used the property $r(k) = r(-k)$. Observe that if $|a| > 1$, $r(0) < 0$, which is meaningless. Also, $|a| < 1$ guarantees that the magnitude of the root of the characteristic polynomial ($z_* = -a$) is smaller than one. Moreover, $|a| < 1$ guarantees that $r(k) \rightarrow 0$ as $k \rightarrow \infty$. This is in line with common sense, since variables that are far away must be uncorrelated.

Fig. 2.15 shows the time evolution of two AR(1) processes (after the processes have converged to be stationary) together with the respective autocorrelation sequences, for two cases, corresponding to $a = -0.9$ and $a = -0.4$. Observe that the larger the magnitude of a , the smoother the realization becomes and time variations are *slower*. This is natural, since nearby samples are highly correlated and so, on average, they tend to have similar values. The opposite is true for small values of a . For comparison purposes, Fig. 2.16A is the case of $a = 0$, which corresponds to a white noise. Fig. 2.16B shows the PSDs corresponding to the two cases of Fig. 2.15. Observe that the faster the autocorrelation approaches zero, the more spread out the PSD is, and vice versa.

2.5 INFORMATION THEORY

So far in this chapter, we have looked at some basic definitions and properties concerning probability theory and stochastic processes. In the same vein, we will now focus on the basic definitions and notions related to *information theory*. Although information theory was originally developed in the context of communications and coding disciplines, its application and use has now been adopted in a wide range of areas, including machine learning. Notions from information theory are used for establishing cost functions for optimization in parameter estimation problems, and concepts from information theory are employed to estimate unknown probability distributions in the context of constrained optimization tasks. We will discuss such methods later in this book.

**FIGURE 2.16**

(A) The time evolution of a realization from a white noise process. (B) The PSDs in dBs, for the two AR(1) sequences of Fig. 2.15. The red one corresponds to $a = -0.4$ and the gray one to $a = -0.9$. The smaller the magnitude of a , the closer the process is to a white noise, and its PSD tends to increase the power with which high frequencies participate. Since the PSD is the Fourier transform of the autocorrelation sequence, observe that the broader a sequence is in time, the narrower its Fourier transform becomes, and vice versa.

The father of information theory is *Claude Elwood Shannon* (1916–2001), an American mathematician and electrical engineer. He founded information theory with the landmark paper “A mathematical theory of communication,” published in the Bell System Technical Journal in 1948. However, he is also credited with founding digital circuit design theory in 1937, when, as a 21-year-old Master’s degree student at the Massachusetts Institute of Technology (MIT), he wrote his thesis demonstrating that electrical applications of Boolean algebra could construct and resolve any logical, numerical relationship. So he is also credited as a father of digital computers. Shannon, while working for the national defense during the Second World War, contributed to the field of cryptography, converting it from an art to a rigorous scientific field.

As is the case for probability, the notion of information is part of our everyday vocabulary. In this context, an event carries information if it is either unknown to us, or if the probability of its occurrence is very low and, in spite of that, it happens. For example, if one tells us that the sun shines bright during summer days in the Sahara desert, we could consider such a statement rather dull and useless. On the contrary, if somebody gives us news about snow in the Sahara during summer, that statement carries a lot of information and can possibly ignite a discussion concerning climate change.

Thus, trying to formalize the notion of information from a mathematical point of view, it is reasonable to define it in terms of the negative logarithm of the probability of an event. If the event is certain to occur, it carries zero information content; however, if its probability of occurrence is low, then its information content has a large positive value.

2.5.1 DISCRETE RANDOM VARIABLES

Information

Given a discrete random variable x , which takes values in the set \mathcal{X} , the *information* associated with any value $x \in \mathcal{X}$ is denoted as $I(x)$ and it is defined as

$$I(x) = -\log P(x) : \text{ information associated with } x = x \in \mathcal{X}. \quad (2.145)$$

Any base for the logarithm can be used. If the natural logarithm is chosen, information is measured in terms of *nats* (natural units). If the base 2 logarithm is employed, information is measured in terms of *bits* (binary digits). Employing the logarithmic function to define information is also in line with common sense reasoning that the information content of two statistically independent events should be the sum of the information conveyed by each one of them individually; $I(x, y) = -\log P(x, y) = -\log P(x) - \log P(y)$.

Example 2.5. We are given a binary random variable $x \in \mathcal{X} = \{0, 1\}$, and we assume that $P(1) = P(0) = 0.5$. We can consider this random variable as a source that generates and emits two possible values. The information content of each one of the two equiprobable events is

$$I(0) = I(1) = -\log_2 0.5 = 1 \text{ bit.}$$

Let us now consider another source of random events, which generates *code words* comprising k binary variables together. The output of this source can be seen as a random vector with binary-valued elements, $\mathbf{x} = [x_1, \dots, x_k]^T$. The corresponding probability space, \mathcal{X} , comprises $K = 2^k$ elements. If all possible values have the same probability, $1/K$, then the information content of each possible event is equal to

$$I(x_i) = -\log_2 \frac{1}{K} = k \text{ bits.}$$

We observe that in the case where the number of possible events is larger, the information content of each individual one (assuming equiprobable events) becomes larger. This is also in line with common sense reasoning, since if the source can emit a large number of (equiprobable) events, then the occurrence of any one of them carries more information than a source that can only emit a few possible events.

Mutual and Conditional Information

Besides marginal probabilities, we have already been introduced to the concept of conditional probability. This leads to the definition of mutual information.

Given two discrete random variables, $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, the information content provided by the occurrence of the event $y = y$ about the event $x = x$ is measured by the *mutual information*, denoted as $I(x; y)$ and defined by

$$I(x; y) := \log \frac{P(x|y)}{P(x)} : \text{ mutual information.} \quad (2.146)$$

Note that if the two variables are statistically independent, then their mutual information is zero; this is most reasonable, since observing y says nothing about x . On the contrary, if by observing y it is certain

that x will occur, as when $P(x|y) = 1$, then the mutual information becomes $I(x; y) = I(x)$, which is again in line with common reasoning. Mobilizing our now familiar product rule, we can see that

$$I(x; y) = I(y; x).$$

The *conditional information* of x given y is defined as

$$I(x|y) = -\log P(x|y) : \quad \text{conditional information.} \quad (2.147)$$

It is straightforward to show that

$$I(x; y) = I(x) - I(x|y). \quad (2.148)$$

Example 2.6. In a communications channel, the source transmits binary symbols, x , with probability $P(0) = P(1) = 1/2$. The channel is noisy, so the received symbols y may have changed polarity, due to noise, with the following probabilities:

$$\begin{aligned} P(y = 0|x = 0) &= 1 - p, \\ P(y = 1|x = 0) &= p, \\ P(y = 1|x = 1) &= 1 - q, \\ P(y = 0|x = 1) &= q. \end{aligned}$$

This example illustrates in its simplest form the effect of a *communications channel*. Transmitted bits are hit by noise and what the receiver receives is the noisy (possibly wrong) information. The task of the receiver is to decide, upon reception of a sequence of symbols, which was the originally transmitted one.

The goal of our example is to determine the mutual information about the occurrence of $x = 0$ and $x = 1$ once $y = 0$ has been observed. To this end, we first need to compute the marginal probabilities,

$$P(y = 0) = P(y = 0|x = 0)P(x = 0) + P(y = 0|x = 1)P(x = 1) = \frac{1}{2}(1 - p + q),$$

and similarly,

$$P(y = 1) = \frac{1}{2}(1 - q + p).$$

Thus, the mutual information is

$$\begin{aligned} I(0; 0) &= \log_2 \frac{P(x = 0|y = 0)}{P(x = 0)} = \log_2 \frac{P(y = 0|x = 0)}{P(y = 0)} \\ &= \log_2 \frac{2(1 - p)}{1 - p + q}, \end{aligned}$$

and

$$I(1; 0) = \log_2 \frac{2q}{1 - p + q}.$$

Let us now consider that $p = q = 0$. Then $I(0; 0) = 1$ bit, which is equal to $I(x = 0)$, since the output specifies the input with certainty. If on the other hand $p = q = 1/2$, then $I(0; 0) = 0$ bits, since the noise can randomly change polarity with equal probability. If now $p = q = 1/4$, then $I(0; 0) = \log_2 \frac{3}{2} = 0.587$ bits and $I(1; 0) = -1$ bit. Observe that the mutual information can take negative values, too.

Entropy and Average Mutual Information

Given a discrete random variable $x \in \mathcal{X}$, its *entropy* is defined as the average information over all possible outcomes,

$$H(x) := - \sum_{x \in \mathcal{X}} P(x) \log P(x) : \text{ entropy of } x. \quad (2.149)$$

Note that if $P(x) = 0$, $P(x) \log P(x) = 0$, by taking into consideration that $\lim_{x \rightarrow 0} x \log x = 0$.

In a similar way, the *average mutual information* between two random variables, x, y , is defined as

$$\begin{aligned} I(x; y) &:= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) I(x; y) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \frac{P(x|y)}{P(x)} \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \frac{P(x|y) P(y)}{P(x) P(y)} \end{aligned}$$

or

$$I(x; y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \frac{P(x, y)}{P(x) P(y)} : \text{ average mutual information.} \quad (2.150)$$

It can be shown that

$$I(x; y) \geq 0,$$

and it is zero if x and y are statistically independent (Problem 2.12).

In comparison, the *conditional entropy* of x given y is defined as

$$H(x|y) := - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log P(x|y) : \text{ conditional entropy.} \quad (2.151)$$

It is readily shown, by taking into account the probability product rule, that

$$I(x; y) = H(x) - H(x|y). \quad (2.152)$$

Lemma 2.1. *The entropy of a random variable $x \in \mathcal{X}$ takes its maximum value if all possible values $x \in \mathcal{X}$ are equiprobable.*

Proof. The proof is given in Problem 2.14. □

In other words, the entropy can be considered as a measure of randomness of a source that emits symbols randomly. The maximum value is associated with the maximum uncertainty of what is going to be emitted, since the maximum value occurs if all symbols are equiprobable. The smallest value of the entropy is equal to zero, which corresponds to the case where all events have zero probability with the exception of one, whose probability to occur is equal to one.

Example 2.7. Consider a binary source that transmits the values 1 or 0 with probabilities p and $1 - p$, respectively. Then the entropy of the associated random variable is

$$H(x) = -p \log_2 p - (1 - p) \log_2(1 - p).$$

Fig. 2.17 shows the graph for various values of $p \in [0, 1]$. Observe that the maximum value occurs for $p = 1/2$.

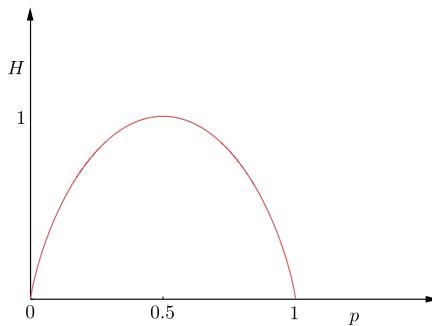


FIGURE 2.17

The maximum value of the entropy for a binary random variable occurs if the two possible events have equal probability, $p = 1/2$.

2.5.2 CONTINUOUS RANDOM VARIABLES

All the definitions given before can be generalized to the case of continuous random variables. However, this generalization must be made with caution. Recall that the probability of occurrence of any single value of a random variable that takes values in an interval in the real axis is zero. Hence, the corresponding information content is infinite.

To define the entropy of a continuous variable x , we first *discretize* it and form the corresponding discrete variable x_Δ , i.e.,

$$x_\Delta := n\Delta, \text{ if } (n-1)\Delta < x \leq n\Delta, \quad (2.153)$$

where $\Delta > 0$. Then,

$$P(x_\Delta = n\Delta) = P(n\Delta - \Delta < x \leq n\Delta) = \int_{(n-1)\Delta}^{n\Delta} p(x) dx = \Delta \bar{p}(n\Delta), \quad (2.154)$$

where $\bar{p}(n\Delta)$ is a number between the maximum and the minimum value of $p(x)$, $x \in (n\Delta - \Delta, n\Delta]$ (such a number exists by the mean value theorem of calculus). Then we can write

$$H(x_\Delta) = - \sum_{n=-\infty}^{+\infty} \Delta \bar{p}(n\Delta) \log(\Delta \bar{p}(n\Delta)), \quad (2.155)$$

and since

$$\sum_{n=-\infty}^{+\infty} \Delta \bar{p}(n\Delta) = \int_{-\infty}^{+\infty} p(x) dx = 1,$$

we obtain

$$H(x_\Delta) = -\log \Delta - \sum_{n=-\infty}^{+\infty} \Delta \bar{p}(n\Delta) \log(\bar{p}(n\Delta)). \quad (2.156)$$

Note that $x_\Delta \rightarrow x$ as $\Delta \rightarrow 0$. However, if we take the limit in Eq. (2.156), then $-\log \Delta$ goes to infinity. This is the crucial difference compared to the discrete variables.

The entropy for a continuous random variable x is defined as the limit

$$H(x) := \lim_{\Delta \rightarrow 0} (H(x_\Delta) + \log \Delta),$$

or

$$H(x) = - \int_{-\infty}^{+\infty} p(x) \log p(x) dx : \text{ entropy.} \quad (2.157)$$

This is the reason that the entropy of a continuous variable is also called *differential entropy*.

Note that the entropy is still a measure of randomness (uncertainty) of the distribution describing x . This is demonstrated via the following example.

Example 2.8. We are given a random variable $x \in [a, b]$. Of all the possible PDFs that can describe this variable, find the one that maximizes the entropy.

This task translates to the following constrained optimization task:

$$\begin{aligned} &\text{maximize with respect to } p : H = - \int_a^b p(x) \ln p(x) dx, \\ &\text{subject to: } \int_a^b p(x) dx = 1. \end{aligned}$$

The constraint guarantees that the function to result is indeed a PDF. Using calculus of variations to perform the optimization (Problem 2.15), it turns out that

$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{if } x \in [a, b], \\ 0, & \text{otherwise.} \end{cases}$$

In other words, the result is the uniform distribution, which is indeed the most random one since it gives no preference to any particular subinterval of $[a, b]$.

We will come to this method of estimating PDFs in Section 12.8.1. This elegant method for estimating PDFs comes from Jaynes [3,4], and it is known as the *maximum entropy method*. In its more general form, more constraints are involved to fit the needs of the specific problem.

Average Mutual Information and Conditional Information

Given two continuous random variables, the average mutual information is defined as

$$I(x; y) := \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy \quad (2.158)$$

and the conditional entropy of x , given y ,

$$H(x|y) := - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} p(x, y) \log p(x|y) dx dy. \quad (2.159)$$

Using standard arguments and the product rule, it is easy to show that

$$I(x; y) = H(x) - H(x|y) = H(y) - H(y|x). \quad (2.160)$$

Relative Entropy or Kullback–Leibler Divergence

The *relative entropy* or *Kullback–Leibler divergence* is a quantity that has been developed within the context of information theory for measuring similarity between two PDFs. It is widely used in machine learning optimization tasks when PDFs are involved; see Chapter 12. Given two PDFs, p and q , their Kullback–Leibler divergence, denoted as $\text{KL}(p||q)$, is defined as

$$\text{KL}(p||q) := \int_{-\infty}^{+\infty} p(x) \log \frac{p(x)}{q(x)} dx : \quad \text{Kullback–Leibler divergence.} \quad (2.161)$$

Note that

$$I(x; y) = \text{KL}(p(x, y)||p(x)p(y)).$$

The Kullback–Leibler divergence is *not* symmetric, i.e., $\text{KL}(p||q) \neq \text{KL}(q||p)$, and it can be shown that it is a nonnegative quantity (the proof is similar to the proof that the mutual information is non-negative; see Problem 12.7 of Chapter 12). Moreover, it is zero if and only if $p = q$.

Note that all we have said concerning entropy and mutual information is readily generalized to the case of random vectors.

2.6 STOCHASTIC CONVERGENCE

We will close this memory refreshing tour of the theory of probability and related concepts with some definitions concerning convergence of sequences of random variables.

Consider a sequence of random variables

$$x_0, x_1, \dots, x_n \dots$$

We can consider this sequence as a discrete-time stochastic process. Due to the randomness, a realization of this process, as shown by

$$x_0, x_1, \dots, x_n \dots,$$

may converge or may not. Thus, the notion of convergence of random variables has to be treated carefully, and different interpretations have been developed.

Recall from our basic calculus that a sequence of numbers, x_n , converges to a value x if $\forall \epsilon > 0$ there exists a number $n(\epsilon)$ such that

$$|x_n - x| < \epsilon, \quad \forall n \geq n(\epsilon). \quad (2.162)$$

CONVERGENCE EVERYWHERE

We say that a random sequence *converges everywhere* if every realization, x_n , of the random process converges to a value x , according to the definition given in Eq. (2.162). Note that every realization converges to a different value, which itself can be considered as the outcome of a random variable x , and we write

$$x_n \xrightarrow{n \rightarrow \infty} x. \quad (2.163)$$

It is common to denote a realization (outcome) of a random process as $x_n(\zeta)$, where ζ denotes a specific experiment.

CONVERGENCE ALMOST EVERYWHERE

A weaker version of convergence, compared to the previous one, is the *convergence almost everywhere*. Consider the set of outcomes ζ , such that

$$\lim x_n(\zeta) = x(\zeta), \quad n \rightarrow \infty.$$

We say that the sequence x_n converges almost everywhere if

$$P(x_n \rightarrow x) = 1, \quad n \rightarrow \infty. \quad (2.164)$$

Note that $\{x_n \rightarrow x\}$ denotes the event comprising *all* the outcomes such that $\lim x_n(\zeta) = x(\zeta)$. The difference with the convergence everywhere is that now it is allowed to a finite or countably infinite number of realizations (that is, to a set of zero probability) not to converge. Often, this type of convergence is referred to as *almost sure* convergence or convergence *with probability 1*.

CONVERGENCE IN THE MEAN-SQUARE SENSE

We say that a random sequence x_n converges to the random variable x in the *mean-square sense* if

$$\mathbb{E} \left[|x_n - x|^2 \right] \rightarrow 0, \quad n \rightarrow \infty. \quad (2.165)$$

CONVERGENCE IN PROBABILITY

Given a random sequence x_n , a random variable x , and a nonnegative number ϵ , then $\{|x_n - x| > \epsilon\}$ is an event. We define the new sequence of numbers, $P(|x_n - x| > \epsilon)$. We say that x_n converges to x *in probability* if the constructed sequence of numbers tends to zero,

$$P(|x_n - x| > \epsilon) \rightarrow 0, \quad n \rightarrow \infty, \quad \forall \epsilon > 0. \quad (2.166)$$

CONVERGENCE IN DISTRIBUTION

Given a random sequence x_n and a random variable x , let $F_n(x)$ and $F(x)$ be the CDFs, respectively. We say that x_n converges to x *in distribution* if

$$F_n(x) \rightarrow F(x), \quad n \rightarrow \infty, \quad (2.167)$$

for every point x of continuity of $F(x)$.

It can be shown that if a random sequence converges either almost everywhere or in the mean-square sense, then it necessarily converges in probability, and if it converges in probability, then it necessarily converges in distribution. The converse arguments are not necessarily true. In other words, the weakest version of convergence is that of convergence in distribution.

PROBLEMS

- 2.1** Derive the mean and variance for the binomial distribution.
- 2.2** Derive the mean and variance for the uniform distribution.
- 2.3** Derive the mean and covariance matrix of the multivariate Gaussian.
- 2.4** Show that the mean and variance of the beta distribution with parameters a and b are given by

$$\mathbb{E}[x] = \frac{a}{a+b}$$

and

$$\sigma_x^2 = \frac{ab}{(a+b)^2(a+b+1)}.$$

Hint: Use the property $\Gamma(a+1) = a\Gamma(a)$.

- 2.5** Show that the normalizing constant in the beta distribution with parameters a, b is given by

$$\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}.$$

- 2.6** Show that the mean and variance of the gamma PDF

$$\text{Gamma}(x|a, b) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}, \quad a, b, x > 0,$$

LEARNING IN PARAMETRIC MODELING: BASIC CONCEPTS AND DIRECTIONS

3

CONTENTS

3.1	Introduction	67
3.2	Parameter Estimation: the Deterministic Point of View	68
3.3	Linear Regression	71
3.4	Classification	75
	Generative Versus Discriminative Learning	78
3.5	Biased Versus Unbiased Estimation	80
	3.5.1 Biased or Unbiased Estimation?	81
3.6	The Cramér–Rao Lower Bound	83
3.7	Sufficient Statistic	87
3.8	Regularization	89
	Inverse Problems: III-Conditioning and Overfitting	91
3.9	The Bias–Variance Dilemma	93
	3.9.1 Mean-Square Error Estimation	94
	3.9.2 Bias–Variance Tradeoff	95
3.10	Maximum Likelihood Method	98
	3.10.1 Linear Regression: the Nonwhite Gaussian Noise Case	101
3.11	Bayesian Inference	102
	3.11.1 The Maximum a Posteriori Probability Estimation Method	107
3.12	Curse of Dimensionality	108
3.13	Validation	109
	Cross-Validation	111
3.14	Expected Loss and Empirical Risk Functions	112
	Learnability	113
3.15	Nonparametric Modeling and Estimation	114
Problems	114	
	MATLAB® Exercises	119
References	119	

3.1 INTRODUCTION

Parametric modeling is a theme that runs across the spine of this book. A number of chapters focus on different aspects of this important problem. This chapter provides basic definitions and concepts related to the task of learning when parametric models are mobilized to describe the available data.

As has already been pointed out in the introductory chapter, a large class of machine learning problems ends up as being equivalent to a function estimation/approximation task. The function is “learned” during the learning/training phase by digging in the information that resides in the available training data set. This function relates the so-called input variables to the output variable(s). Once this functional relationship is established, one can in turn exploit it to *predict* the value(s) of the output(s), based on measurements from the respective input variables; these predictions can then be used to proceed to the decision making phase.

In parametric modeling, the aforementioned functional dependence that relates the input to the output is defined via a set of parameters, whose number is *fixed* and a-priori known. The values of the parameters are unknown and have to be estimated based on the available input–output observations. In contrast to the parametric, there are the so-called *nonparametric* methods. In such methods, parameters may still be involved to establish the input–output relationship, yet their number is *not* fixed; it depends on the size of the data set and it grows with the number of observations. Nonparametric methods will also be treated in this book (e.g., Chapters 11 and 13). However, the emphasis in the current chapter lies on parametric models.

There are two possible paths to deal with the uncertainty imposed by the unknown values of the involved parameters. According to the first one, parameters are treated as *deterministic* nonrandom variables. The task of learning is to obtain estimates of their unknown values. For each one of the parameters a single value estimate is obtained. The other approach has a stronger statistical flavor. The unknown parameters are treated as *random variables* and the task of learning is to *infer* the associated probability distributions. Once the distributions have been learned/inferred, one can use them to make predictions. Both approaches are introduced in the current chapter and are treated in more detail later on in various chapters of the book.

Two of the major machine learning tasks, namely, regression and classification, are presented and the main directions in dealing with these problems are exposed. Various issues that are related to the parameter estimation task, such as estimator efficiency, bias–variance dilemma, overfitting, and the curse of dimensionality, are introduced and discussed. The chapter can also be considered as a road map to the rest of the book. However, instead of just presenting the main ideas and directions in a rather “dry” way, we chose to deal and work with the involved tasks by adopting simple models and techniques, so that the reader gets a better feeling of the topic. An effort was made to pay more attention to the scientific notions than to algebraic manipulations and mathematical details, which will, unavoidably, be used to a larger extent while “embroidering” the chapters to follow.

The least-squares (LS), maximum likelihood (ML), regularization, and Bayesian inference techniques are presented and discussed. An effort has been made to assist the reader to grasp an informative view of the big picture conveyed by the book. Thus, this chapter could also be used as an overview introduction to the parametric modeling task in the realm of machine learning.

3.2 PARAMETER ESTIMATION: THE DETERMINISTIC POINT OF VIEW

The task of estimating the value of an unknown parameter vector, θ , has been at the center of interest in a number of application areas. For example, in the early years at university, one of the very first subjects any student has to study is the so-called curve fitting problem. Given a set of data points, one must find a curve or a surface that “fits” the data. The usual path to follow is to *adopt* a functional form,

such as a linear function or a quadratic one, and try to estimate the associated unknown parameters so that the graph of the function “passes through” the data and follows their deployment in space as close as possible. Figs. 3.1A and B are two such examples. The data lie in the \mathbb{R}^2 space and are given to us as a set of points (y_n, x_n) , $n = 1, 2, \dots, N$. The adopted functional form for the curve corresponding to Fig. 3.1A is

$$y = f_{\theta}(x) = \theta_0 + \theta_1 x, \quad (3.1)$$

and for the case of Fig. 3.1B it is

$$y = f_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2. \quad (3.2)$$

The unknown parameter vectors are $\theta = [\theta_0, \theta_1]^T$ and $\theta = [\theta_0, \theta_1, \theta_2]^T$, respectively. In both cases, the parameter values, which define the curves drawn by the red lines, provide a much better fit compared to the values associated with the black ones. In both cases, the task comprises two steps: (a) first adopt a specific *parametric functional form* which we reckon to be more appropriate for the data at hand and (b) estimate the values of the unknown parameters in order to obtain a “good” fit.

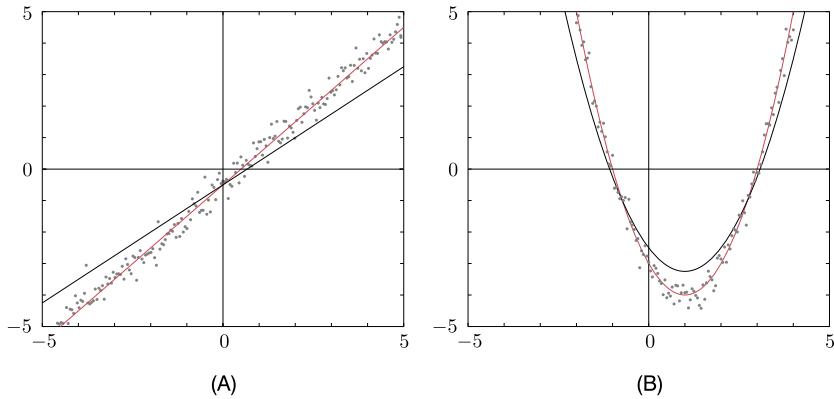


FIGURE 3.1

(A) Fitting a linear function. (B) Fitting a quadratic function. The red lines are the optimized ones.

In the more general and formal setting, the task can be defined as follows. Given a set of data points, (y_n, \mathbf{x}_n) , $y_n \in \mathbb{R}$, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, and a parametric¹ set of functions,

$$\mathcal{F} := \left\{ f_{\theta}(\cdot) : \theta \in \mathcal{A} \subseteq \mathbb{R}^K \right\}, \quad (3.3)$$

find a function in \mathcal{F} , which will be denoted as $f(\cdot) := f_{\theta_*}(\cdot)$, such that given a value of $\mathbf{x} \in \mathbb{R}^l$, $f(\mathbf{x})$ best approximates the corresponding value $y \in \mathbb{R}$. The set \mathcal{A} is a constraint set in case we wish to constrain the unknown K parameters to lie within a specific region in \mathbb{R}^K . Constraining the parameters to

¹ Recall from the Notations section that we use the symbol $f(\cdot)$ to denote a function of a single argument; $f(\mathbf{x})$ denotes the value that the function $f(\cdot)$ has at \mathbf{x} .

be within a subset of the parameter space \mathbb{R}^K is almost omnipresent in machine learning. We will deal with constrained optimization later on in this chapter (Section 3.8). We start our discussion by considering y to be a real variable, $y \in \mathbb{R}$, and as we move on and understand better the various “secrets,” we will allow it to move to higher-dimensional spaces. The value θ_* is the value that results from an estimation procedure. The values of θ_* that define the red curves in Figs. 3.1A and B are

$$\theta_* = [-0.5, 1]^T, \quad \theta_* = [-3, -2, 1]^T, \quad (3.4)$$

respectively.

To reach a decision with respect to the choice of \mathcal{F} is not an easy task. For the case of the data in Fig. 3.1, we were a bit “lucky.” First, the data live in the two-dimensional space, where we have the luxury of visualization. Second, the data were scattered along curves whose shape is pretty familiar to us; hence, a simple inspection suggested the proper family of functions, for each one of the two cases. Obviously, real life is hardly as generous as that and in the majority of practical applications, the data reside in high-dimensional spaces and/or the shape of the surface (hypersurface, for spaces of dimensionality higher than three) can be quite complex. Hence, the choice of \mathcal{F} , which dictates the functional form (e.g., linear, quadratic, etc.), is not easy. In practice, one has to use as much a priori information as possible concerning the physical mechanism that underlies the generation of the data, and most often use different families of functions and finally keep the one that results in the best performance, according to a chosen criterion.

Having adopted a parametric family of functions, \mathcal{F} , one has to get estimates for the set of the unknown parameters. To this end, a measure of fitness has to be adopted. The more classical approach is to adopt a *loss* function, which quantifies the deviation/error between the measured value of y and the predicted one using the corresponding measurements x , as in $f_\theta(x)$. In a more formal way, we adopt a *nonnegative* (loss) function,

$$\mathcal{L}(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \mapsto [0, \infty),$$

and compute θ_* so as to minimize the total loss, or as we say the *cost*, over all the data points, or

$$f(\cdot) := f_{\theta_*}(\cdot) : \theta_* = \arg \min_{\theta \in \mathcal{A}} J(\theta), \quad (3.5)$$

where

$$J(\theta) := \sum_{n=1}^N \mathcal{L}(y_n, f_\theta(x_n)), \quad (3.6)$$

assuming that a minimum exists. Note that, in general, there may be more than one optimal value θ_* , depending on the shape of $J(\theta)$.

As the book evolves, we are going to see different loss functions and different parametric families of functions. For the sake of simplicity, for the rest of this chapter we will adhere to the squared error loss function,

$$\mathcal{L}(y, f_\theta(x)) = (y - f_\theta(x))^2,$$

and to the linear class of functions.

The squared error loss function is credited to the great mathematician Carl Frederich Gauss, who proposed the fundamentals of the least-squares (LS) method in 1795 at the age of 18. However, it was Adrien-Marie Legendre who first published the method in 1805, working independently. Gauss published it in 1809. The strength of the method was demonstrated when it was used to predict the location of the asteroid Ceres. Since then, the squared error loss function has “haunted” all scientific fields, and even if it is not used directly, it is, most often, used as the standard against which the performance of more modern alternatives are compared. This success is due to some nice properties that this loss criterion has, which will be explored as we move on in this book.

The combined choice of linearity with the squared error loss function turns out to simplify the algebra and hence becomes very pedagogic for introducing the newcomer to the various “secrets” that underlie the area of parameter estimation. Moreover, understanding linearity is very important. Treating nonlinear tasks, most often, turns out to finally resort to a linear problem. Take, for example, the nonlinear model in Eq. (3.2) and consider the transformation

$$\mathbb{R} \ni x \longmapsto \phi(x) := \begin{bmatrix} x \\ x^2 \end{bmatrix} \in \mathbb{R}^2. \quad (3.7)$$

Then Eq. (3.2) becomes

$$y = \theta_0 + \theta_1 \phi_1(x) + \theta_2 \phi_2(x). \quad (3.8)$$

That is, the model is now linear with respect to the components $\phi_k(x)$, $k = 1, 2$, of the two-dimensional image $\phi(x)$ of x . As a matter of fact, this simple trick is at the heart of a number of nonlinear methods that will be treated later on in the book. No doubt, the procedure can be generalized to any number K of functions $\phi_k(x)$, $k = 1, 2, \dots, K$, and besides monomials, other types of nonlinear functions can be used, such as exponentials, splines, and wavelets. In spite of the nonlinear nature of the input–output dependence modeling, we still consider this model to be linear, because it retains its linearity with respect to the involved unknown parameters θ_k , $k = 1, 2, \dots, K$. Although for the rest of the chapter we will adhere to linear functions, in order to keep our discussion simple, everything that will be said applies to nonlinear ones as well. All that is needed is to replace x with $\phi(x) := [\phi_1(x), \dots, \phi_K(x)]^T \in \mathbb{R}^K$.

In the sequel, we will present two examples in order to demonstrate the use of parametric modeling. These examples are generic and can represent a wide class of problems.

3.3 LINEAR REGRESSION

In statistics, the term *regression* was coined to define the task of modeling the relationship of a *dependent* random variable y , which is considered to be the response of a system, when this is activated by a set of random variables x_1, x_2, \dots, x_l , which will be represented as the components of an equivalent random vector \mathbf{x} . The relationship is modeled via an additive disturbance or noise term η . The block diagram of the process, which relates the involved variables, is given in Fig. 3.2. The noise variable η is an *unobserved* random variable. The goal of the regression task is to estimate the parameter vector θ , given a set of measurements/observations (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, that we have at our disposal. This is also known as the *training data set*. The dependent variable is usually known as the *output* variable

and the vector \mathbf{x} as the *input* vector or the *regressor*. If we model the system as a linear combiner, the dependence relationship is written as

$$y = \theta_0 + \theta_1 x_1 + \cdots + \theta_l x_l + \eta = \theta_0 + \boldsymbol{\theta}^T \mathbf{x} + \eta. \quad (3.9)$$

The parameter θ_0 is known as the *bias* or the *intercept*. Usually, this term is absorbed by the parameter vector $\boldsymbol{\theta}$ with a simultaneous increase in the dimension of \mathbf{x} by adding the constant 1 as its last element. Indeed, we can write

$$\theta_0 + \boldsymbol{\theta}^T \mathbf{x} + \eta = [\boldsymbol{\theta}^T, \theta_0] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} + \eta.$$

From now on, the regression model will be written as

$$y = \boldsymbol{\theta}^T \mathbf{x} + \eta, \quad (3.10)$$

and, unless otherwise stated, this notation means that the bias term has been absorbed by $\boldsymbol{\theta}$ and \mathbf{x} has been extended by adding 1 as an extra component. Because the noise variable is unobserved, we need a model to be able to *predict* the output value of y , given an observed value, \mathbf{x} , of the random vector \mathbf{x} .

In linear regression, given an estimate $\hat{\boldsymbol{\theta}}$ of $\boldsymbol{\theta}$, we adopt the following prediction model:

$$\hat{y} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_l x_l := \hat{\boldsymbol{\theta}}^T \mathbf{x}. \quad (3.11)$$

Using the squared error loss function, the estimate $\hat{\boldsymbol{\theta}}$ is set equal to $\boldsymbol{\theta}_*$, which minimizes the square difference between \hat{y}_n and y_n over the set of the available observations; that is, one minimizes, with respect to $\boldsymbol{\theta}$, the cost function

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2. \quad (3.12)$$

We start our discussion by considering no constraints; hence, we set $\mathcal{A} = \mathbb{R}^K$, and we are going to search for solutions that lie anywhere in \mathbb{R}^K . Taking the derivative (gradient) with respect to $\boldsymbol{\theta}$ (see Appendix A) and equating to the zero vector, $\mathbf{0}$, we obtain (Problem 3.1)

$$\left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \right) \hat{\boldsymbol{\theta}} = \sum_{n=1}^N y_n \mathbf{x}_n. \quad (3.13)$$

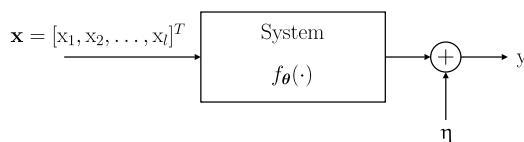


FIGURE 3.2

Block diagram showing the input–output relation in a regression model.

Note that the sum on the left-hand side is an $(l + 1) \times (l + 1)$ matrix, being the sum of N outer vector products, i.e., $\mathbf{x}_n \mathbf{x}_n^T$. As we know from linear algebra, we need at least $N = l + 1$ observation vectors to guarantee that the matrix is invertible, assuming of course linear independence among the vectors (see, e.g., [35]).

For those who are not (yet) very familiar with working with vectors, note that Eq. (3.13) is just a generalization of the scalar case. For example, in a “scalar” world, the input–output pairs would comprise scalars (y_n, x_n) and the unknown parameter would also be a scalar, θ . The cost function would be $\sum_{n=1}^N (y_n - \theta x_n)^2$. Taking the derivative and setting it equal to zero leads to

$$\left(\sum_{n=1}^N x_n^2 \right) \hat{\theta} = \sum_{n=1}^N y_n x_n, \text{ or } \hat{\theta} = \frac{\sum_{n=1}^N y_n x_n}{\sum_{n=1}^N x_n^2}.$$

A more popular way to write Eq. (3.13) is via the so-called input matrix X , defined as the $N \times (l + 1)$ matrix which has as rows the (extended) regressor vectors \mathbf{x}_n^T , $n = 1, 2, \dots, N$, expressed as

$$X := \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1l} & 1 \\ x_{21} & \dots & x_{2l} & 1 \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{Nl} & 1 \end{bmatrix}. \quad (3.14)$$

Then it is straightforward to see that Eq. (3.13) can be written as

$$(X^T X) \hat{\theta} = X^T \mathbf{y}, \quad (3.15)$$

where

$$\mathbf{y} := [y_1, y_2, \dots, y_N]^T. \quad (3.16)$$

Indeed,

$$X^T X = [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T,$$

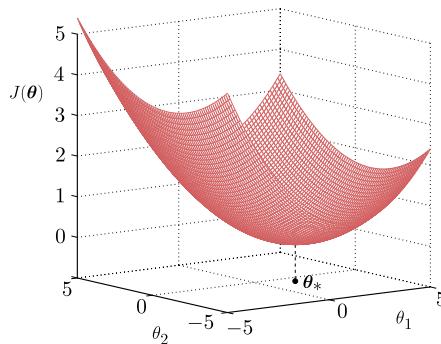
and similarly,

$$X^T \mathbf{y} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \sum_{n=1}^N y_n \mathbf{x}_n.$$

Thus, finally, the LS estimate is given by

$\hat{\theta} = (X^T X)^{-1} X^T \mathbf{y} :$ the LS estimate,

(3.17)

**FIGURE 3.3**

The squared error loss function has a unique minimum at the point θ_* .

assuming, of course, that $(X^T X)^{-1}$ exists.

In other words, the obtained estimate of the parameter vector is given by a *linear set of equations*. This is a major advantage of the squared error loss function, when applied to a linear model. Moreover, this solution is *unique*, provided that the $(l+1) \times (l+1)$ matrix $X^T X$ is invertible. The uniqueness is due to the parabolic shape of the graph of the sum of squared errors cost function. This is illustrated in Fig. 3.3 for the two-dimensional space. It is readily observed that the graph has a unique minimum. This is a consequence of the fact the sum of squared errors cost function is a strictly convex one. Issues related to convexity of loss functions will be treated in more detail in Chapter 8.

Example 3.1. Consider the system that is described by the following model:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \eta := [0.25, -0.25, 0.25] \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} + \eta, \quad (3.18)$$

where η is a Gaussian random variable of zero mean and variance $\sigma^2 = 1$. Observe that the generated data are spread, due to the noise, around the plane that is defined as

$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2, \quad (3.19)$$

in the two-dimensional space (see Fig. 3.4C).

The random variables x_1 and x_2 are assumed to be mutually independent and uniformly distributed over the interval $[0, 10]$. Furthermore, both variables are independent of the noise variable η . We generate $N = 50$ i.i.d. points² for each of the three random variables, i.e., x_1, x_2, η . For each triplet, we use Eq. (3.18) to generate the corresponding value, y , of y . In this way, the points (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, 50$, are generated, where each observation \mathbf{x}_n lies in \mathbb{R}^2 . These are used as the training points to obtain the

² Independent and identically distributed samples (see also Section 2.3.2, following Eq. (2.84)).

LS estimates of the parameters of the linear model

$$\hat{y} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2. \quad (3.20)$$

Then we repeat the experiments with $\sigma^2 = 10$. Note that, in general, Eq. (3.20) defines a different plane than the original (3.19).

The values of the LS optimal estimates are obtained by solving a 3×3 linear system of equations. They are

- (a) $\hat{\theta}_0 = 0.028$, $\hat{\theta}_1 = 0.226$, $\hat{\theta}_2 = -0.224$,
- (b) $\hat{\theta}_0 = 0.914$, $\hat{\theta}_1 = 0.325$, $\hat{\theta}_2 = -0.477$,

for the two cases, respectively. Figs. 3.4A and B show the recovered planes. Observe that in the case of Fig. 3.4A, corresponding to a noise variable of small variance, the obtained plane follows the data points much closer than that of Fig. 3.4B.

Remarks 3.1.

- The set of points $(\hat{y}_n, x_{n1}, \dots, x_{nl}), n = 1, 2, \dots, N$, lie on a *hyperplane* in the \mathbb{R}^{l+1} space. Equivalently, they lie on a hyperplane that crosses the origin and, thus, it is a linear subspace in the extended space \mathbb{R}^{l+2} when one absorbs θ_0 in $\boldsymbol{\theta}$, as explained previously.
- Note that the prediction model in Eq. (3.11) could still be used, even if the true system structure does not obey the linear model in Eq. (3.9). For example, the true dependence between y and \mathbf{x} may be a nonlinear one. In such a case, the predictions of y based on the model in Eq. (3.11) may, however, not be satisfactory. It all depends on the deviation of our adopted model from the true structure of the system that generates the data.
- The prediction performance of the model also depends on the statistical properties of the noise variable. This is an important issue. We will see later on that, depending on the statistical properties of the noise variable, some loss functions and methods may be more suitable than others.
- The two previous remarks suggest that in order to quantify the performance of an estimator some related criteria are necessary. In Section 3.9, we will present some theoretical touches that shed light on certain aspects related to the performance of an estimator.

3.4 CLASSIFICATION

Classification is the task of predicting the class to which an object, known as *pattern*, belongs. The pattern is assumed to belong to one and only one among a number of a priori known classes. Each pattern is *uniquely* represented by a set of values, known as *features*. One of the early stages in designing a classification system is to select an appropriate set of feature variables. These should “encode” as much class-discriminatory information so that, by measuring their value for a given pattern, we are able to predict, with high enough probability, the class of the pattern. Selecting the appropriate set of features for each problem is not an easy task, and it comprises one of the most important areas within the field of *pattern recognition* (see, e.g., [12,37]). Having selected, say, l feature (random) variables, x_1, x_2, \dots, x_l , we stack them as the components of the so-called *feature vector* $\mathbf{x} \in \mathbb{R}^l$. The goal is to

where $P(y)$ is the probability concerning the classes and $p(\mathbf{x}|y)$ is the distribution of the input given the class label. For such an approach, we end up with one distribution per class, which has to be learned. In parametric modeling, a set of parameters is associated with each one of these conditional distributions. Once the input–output joint distribution has been learned, the prediction of the class label of an unknown pattern, \mathbf{x} , is performed based on the a posteriori probability, i.e.,

$$P(y|\mathbf{x}) = \frac{p(y, \mathbf{x})}{p(\mathbf{x})} = \frac{p(y, \mathbf{x})}{\sum_y p(y, \mathbf{x})}.$$

We will return to these issues in more detail in Chapter 7.

3.5 BIASED VERSUS UNBIASED ESTIMATION

In supervised learning, we are given a set of training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, and we return an estimate of the unknown parameter vector, say, $\hat{\theta}$. However, the training points themselves are random variables. If we are given another set of N observations of the *same* random variables, these are going to be different, and obviously the resulting estimate will also be different. In other words, by changing our training data, different estimates will result. Hence, we can assume that the resulting estimate, of a fixed yet unknown parameter, is itself a random variable. This, in turn, poses questions on how good an estimator is. Undoubtedly, each time the obtained estimate is optimal with respect to the adopted loss function *and* the specific training set used. However, who guarantees that the resulting estimates are “close” to the true value, assuming that there is one? In this section, we will try to address this task and to illuminate some related theoretical aspects. Note that we have already used the term *estimator* in place of the term *estimate*. Let us elaborate a bit on their difference, before presenting more details.

An estimate, such as $\hat{\theta}$, has a specific value, which is the result of a function acting on a set of observations on which our chosen estimate depends (see Eq. (3.17)). In general, we could generalize Eq. (3.17) and write

$$\hat{\theta} = g(\mathbf{y}, \mathbf{X}).$$

However, once we allow the set of observations to change randomly, and the estimate becomes itself a random variable, we write the previous equation in terms of the corresponding random variables, i.e.,

$$\hat{\theta} = g(\mathbf{y}, \mathbf{X}),$$

and we refer to this functional dependence as the *estimator* of the unknown vector θ .

In order to simplify the analysis and focus on the insight behind the methods, we will assume that our parameter space is that of real numbers, \mathbb{R} . We will also assume that the model (i.e., the set of functions \mathcal{F}) which we have adopted for modeling our data is the correct one and that the (unknown to us) value of the associated true parameter is equal to⁴ θ_o . Let $\hat{\theta}$ denote the random variable of the

⁴ Not to be confused with the intercept; the subscript here is “o” and not “0.”

associated estimator. Adopting the squared error loss function to quantify deviations, a reasonable criterion to measure the performance of an estimator is the *mean-square error* (MSE),

$$\text{MSE} = \mathbb{E}[(\hat{\theta} - \theta_o)^2], \quad (3.21)$$

where the mean \mathbb{E} is taken over *all* possible training data sets of size N . If the MSE is small, then we expect that, on average, the resulting estimates are close to the true value. Note that although θ_o is not known, studying the way in which the MSE depends on various terms will still help us to learn how to proceed in practice and unravel possible paths that one can follow to obtain good estimators.

Indeed, this simple and “natural” looking criterion hides some interesting surprises for us. Let us insert the mean value $\mathbb{E}[\hat{\theta}]$ of $\hat{\theta}$ in Eq. (3.21). We get

$$\begin{aligned} \text{MSE} &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right) + \left(\mathbb{E}[\hat{\theta}] - \theta_o\right)\right]^2 \\ &= \underbrace{\mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right]}_{\text{Variance}} + \underbrace{\left(\mathbb{E}[\hat{\theta}] - \theta_o\right)^2}_{\text{Bias}^2}, \end{aligned} \quad (3.22)$$

where, for the second equality, we have taken into account that the mean value of the product of the two involved terms turns out to be zero, as can be readily seen. What Eq. (3.22) suggests is that the MSE consists of two terms. The first one is the variance around the mean value and the second one is due to the bias, that is, the deviation of the mean value of the estimator from the true one.

3.5.1 BIASED OR UNBIASED ESTIMATION?

One may naively think that choosing an estimator that is *unbiased*, as is $\mathbb{E}[\hat{\theta}] = \theta_o$, such that the second term in Eq. (3.22) becomes zero, is a reasonable choice. Adopting an unbiased estimator may also be appealing from the following point of view. Assume that we have L different training sets, each comprising N points. Let us denote each data set by \mathcal{D}_i , $i = 1, 2, \dots, L$. For each one, an estimate $\hat{\theta}_i$, $i = 1, 2, \dots, L$, will result. Then, we form a new estimator by taking the average value,

$$\hat{\theta}^{(L)} := \frac{1}{L} \sum_{i=1}^L \hat{\theta}_i.$$

This is also an unbiased estimator, because

$$\mathbb{E}[\hat{\theta}^{(L)}] = \frac{1}{L} \sum_{i=1}^L \mathbb{E}[\hat{\theta}_i] = \theta_o.$$

Moreover, assuming that the involved estimators are mutually uncorrelated, i.e.,

$$\mathbb{E}\left[(\hat{\theta}_i - \theta_o)(\hat{\theta}_j - \theta_o)\right] = 0,$$

and of the same variance, σ^2 , the variance of the new estimator is now much smaller (Problem 3.2), i.e.,

$$\sigma_{\hat{\theta}^{(L)}}^2 = \mathbb{E} \left[(\hat{\theta}^{(L)} - \theta_o)^2 \right] = \frac{\sigma^2}{L}.$$

Hence, by averaging a large number of such unbiased estimators, we expect to get an estimate close to the true value. However, in practice, data are a commodity that is not always abundant. As a matter of fact, very often the opposite is true and one has to be very careful about how to exploit them. In such cases, where one cannot afford to obtain and average a large number of estimators, an unbiased estimator may not necessarily be the best choice. Going back to Eq. (3.22), there is no reason to suggest that by making the second term equal to zero, the MSE (which, after all, is the quantity of interest to us) becomes minimal. Indeed, let us look at Eq. (3.22) from a slightly different point of view. Instead of computing the MSE for a given estimator, let us replace $\hat{\theta}$ with θ in Eq. (3.22) and compute an estimator that will minimize the MSE with respect to θ directly. In this case, focusing on unbiased estimators, or $\mathbb{E}[\theta] = \theta_o$, introduces a constraint to the task of minimizing the MSE, and it is well known that an unconstrained minimization problem always results in loss function values that are less than or equal to any value generated by a constrained counterpart, i.e.,

$$\min_{\theta} \text{MSE}(\theta) \leq \min_{\theta: \mathbb{E}[\theta] = \theta_o} \text{MSE}(\theta), \quad (3.23)$$

where the dependence of the MSE on the estimator θ in Eq. (3.23) is explicitly denoted.

Let us denote by $\hat{\theta}_{\text{MVU}}$ a solution of the task $\min_{\theta: \mathbb{E}[\theta] = \theta_o} \text{MSE}(\theta)$. It can be readily verified by Eq. (3.22) that $\hat{\theta}_{\text{MVU}}$ is an unbiased estimator of minimum variance. Such an estimator is known as the *minimum variance unbiased* (MVU) estimator and we assume that such an estimator exists. An MVU does not always exist (see [20], Problem 3.3). Moreover, if it exists it is unique (Problem 3.4). Motivated by Eq. (3.23), our next goal is to search for a *biased* estimator, which results, hopefully, in a smaller MSE. Let us denote this estimator as $\hat{\theta}_b$. For the sake of illustration, and in order to limit our search for $\hat{\theta}_b$, we consider here only $\hat{\theta}_b$ s that are scalar multiples of $\hat{\theta}_{\text{MVU}}$, so that

$$\hat{\theta}_b = (1 + \alpha)\hat{\theta}_{\text{MVU}}, \quad (3.24)$$

where $\alpha \in \mathbb{R}$ is a free parameter. Note that $\mathbb{E}[\hat{\theta}_b] = (1 + \alpha)\theta_o$. By substituting Eq. (3.24) into Eq. (3.22) and after some simple algebra we obtain

$$\text{MSE}(\hat{\theta}_b) = (1 + \alpha)^2 \text{MSE}(\hat{\theta}_{\text{MVU}}) + \alpha^2 \theta_o^2. \quad (3.25)$$

In order to get $\text{MSE}(\hat{\theta}_b) < \text{MSE}(\hat{\theta}_{\text{MVU}})$, α must be in the range (Problem 3.5) of

$$-\frac{2\text{MSE}(\hat{\theta}_{\text{MVU}})}{\text{MSE}(\hat{\theta}_{\text{MVU}}) + \theta_o^2} < \alpha < 0. \quad (3.26)$$

It is easy to verify that this range implies that $|1 + \alpha| < 1$. Hence, $|\hat{\theta}_b| = |(1 + \alpha)\hat{\theta}_{\text{MVU}}| < |\hat{\theta}_{\text{MVU}}|$. We can go a step further and try to compute the optimum value of α , which corresponds to the minimum

MSE. By taking the derivative of $\text{MSE}(\hat{\theta}_b)$ in Eq. (3.25) with respect to α , it turns out (Problem 3.6) that this occurs for

$$\alpha_* = -\frac{\text{MSE}(\hat{\theta}_{\text{MVU}})}{\text{MSE}(\hat{\theta}_{\text{MVU}}) + \theta_o^2} = -\frac{1}{1 + \frac{\theta_o^2}{\text{MSE}(\hat{\theta}_{\text{MVU}})}}. \quad (3.27)$$

Therefore, we have found a way to obtain the optimum estimator among those in the set $\{\hat{\theta}_b = (1 + \alpha)\hat{\theta}_{\text{MVU}} : \alpha \in \mathbb{R}\}$, which results in the minimum MSE. This is true, but as many nice things in life, this is not, in general, realizable. The optimal value for α is given in terms of the unknown θ_o ! However, Eq. (3.27) is useful in a number of other ways. First, there are cases where the MSE is proportional to θ_o^2 ; hence, this formula can be used. Also, for certain cases, it can be used to provide useful bounds [19]. Moreover, as far as we are concerned in this book, it says something very important. If we want to do better than the MVU, then, looking at the text following Eq. (3.26), a possible way is to *shrink* the norm of the MVU estimator. Shrinking the norm is a way of introducing bias into an estimator. We will discuss ways to achieve this in Section 3.8 and later on in Chapters 6 and 9.

Note that what we have said so far is readily generalized to parameter vectors. An unbiased parameter vector estimator satisfies

$$\mathbb{E}[\hat{\theta}] = \theta_o,$$

and the MSE around the true value θ_o is defined as

$$\text{MSE} = \mathbb{E}[(\hat{\theta} - \theta_o)^T (\hat{\theta} - \theta_o)] = \sum_{i=1}^l \mathbb{E}[(\hat{\theta}_i - \theta_{oi})^2].$$

Looking carefully at the previous definition reveals that the MSE for a parameter vector is the sum of the MSEs of the components $\hat{\theta}_i$, $i = 1, 2, \dots, l$, around the corresponding true values θ_{oi} .

3.6 THE CRAMÉR–RAO LOWER BOUND

In the previous sections, we saw how one can improve the performance of the MVU estimator, provided that this exists and it is known. However, how can one know that an unbiased estimator that has been obtained is also of minimum variance? The goal of this section is to introduce a criterion that can provide such information.

The *Cramér–Rao lower bound* [9,31] is an elegant theorem and one of the most well-known techniques used in statistics. It provides a lower bound on the variance of *any* unbiased estimator. This is very important because (a) it offers the means to assert whether an unbiased estimator has minimum variance, which, of course, in this case coincides with the corresponding MSE in Eq. (3.22), (b) if this is not the case, it can be used to indicate how far away the performance of an unbiased estimator is from the optimal one, and finally (c) it provides the designer with a tool to know the best possible performance that can be achieved by an unbiased estimator. Because our main purpose here is to focus on the insight and physical interpretation of the method, we will deal with the simple case where our unknown parameter is a real number. The general form of the theorem, involving vectors, is given in Appendix B.

3.8 REGULARIZATION

We have already seen that the LS estimator is an MVU estimator, under the assumptions of linearity of the regression model and in the presence of a Gaussian white noise source. We also know that one can improve the performance by shrinking the norm of the MVU estimator. There are various ways to achieve this goal, and they will be discussed later on in this book. In this section, we focus on one possibility. Moreover, we will see that trying to keep the norm of the solution small serves important needs in the context of machine learning.

Regularization is a mathematical tool to impose a priori information on the structure of the solution, which comes as the outcome of an optimization task. Regularization was first suggested by the great Russian mathematician Andrey Nikolayevich Tychonoff (sometimes spelled Tikhonov) for the solution of integral equations. Sometimes it is also referred as Tychonoff–Phillips regularization, to honor Thomas Phillips as well, who developed the method independently [29,39].

In the context of our task and in order to shrink the norm of the parameter vector estimate, we can reformulate the sum of squared errors minimization task, given in Eq. (3.12), as

$$\text{minimize} \quad J(\boldsymbol{\theta}) = \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2, \quad (3.39)$$

$$\text{subject to} \quad \|\boldsymbol{\theta}\|^2 \leq \rho, \quad (3.40)$$

where $\|\cdot\|$ stands for the Euclidean norm of a vector. In this way, we do not allow the LS criterion to be completely “free” to reach a solution, but we *limit* the space in which we search for it. Obviously, using different values of ρ , we can achieve different levels of shrinkage. As we have already discussed, the optimal value of ρ cannot be analytically obtained, and one has to experiment in order to select an estimator that results in a good performance. For the squared error loss function and the constraint used before, the optimization task can equivalently be written as [6,8]

$$\text{minimize} \quad L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2 + \lambda \|\boldsymbol{\theta}\|^2: \quad \text{ridge regression.} \quad (3.41)$$

It turns out that for specific choices of $\lambda \geq 0$ and ρ , the two tasks are equivalent. Note that this new cost function, $L(\boldsymbol{\theta}, \lambda)$, involves one term that measures the model *misfit* and a second one that quantifies the *size* of the norm of the parameter vector. It is straightforward to see that taking the gradient of L in Eq. (3.41) with respect to $\boldsymbol{\theta}$ and equating to zero, we obtain the *regularized version* of the LS solution for the linear regression task of Eq. (3.13),

$$\left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T + \lambda I \right) \hat{\boldsymbol{\theta}} = \sum_{n=1}^N y_n \mathbf{x}_n, \quad (3.42)$$

where I is the identity matrix of appropriate dimensions. The presence of λ biases the new solution away from that which would have been obtained from the unregularized LS formulation. The task is also known as *ridge regression*. Ridge regression attempts to reduce the norm of the estimated vector and *at the same time* tries to keep the sum of squared errors small; in order to achieve this *combined*

goal, the vector components, θ_i , are modified in such a way so that the contribution in the misfit measuring term, from the less informative directions in the input space, is minimized. In other words, those of the components that are associated with less informative directions will be pushed to smaller values to keep the norm small and at the same time have minimum influence on the misfit measuring term. We will return to this in more detail in Chapter 6. Ridge regression was first introduced in [18].

It has to be emphasized that in practice, the bias parameter θ_0 is left out from the norm in the regularization term; penalization of the bias would make the procedure dependent on the origin chosen for y . Indeed, it is easily checked that adding a constant term to each one of the output values, y_n , in the cost function would not result in just a shift of the predictions by the same constant, if the bias term is included in the norm. Hence, usually, ridge regression is formulated as

$$\text{minimize } L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^N \left(y_n - \theta_0 - \sum_{i=1}^l \theta_i x_{ni} \right)^2 + \lambda \sum_{i=1}^l |\theta_i|^2. \quad (3.43)$$

It turns out (Problem 3.11) that minimizing Eq. (3.43) with respect to θ_i , $i = 0, 1, \dots, l$, is equivalent to minimizing Eq. (3.41) using *centered* data and neglecting the intercept. That is, one solves the task

$$\text{minimize } L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^N \left((y_n - \bar{y}) - \sum_{i=1}^l \theta_i (x_{ni} - \bar{x}_i) \right)^2 + \lambda \sum_{i=1}^l |\theta_i|^2, \quad (3.44)$$

and the estimate of θ_0 in Eq. (3.43) is given in terms of the obtained estimates $\hat{\theta}_i$, i.e.,

$$\hat{\theta}_0 = \bar{y} - \sum_{i=1}^l \hat{\theta}_i \bar{x}_i,$$

where

$$\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n \quad \text{and} \quad \bar{x}_i = \frac{1}{N} \sum_{n=1}^N x_{ni}, \quad i = 1, 2, \dots, l.$$

In other words, $\hat{\theta}_0$ compensates for the differences between the sample means of the output and input variables. Note that similar arguments hold true if the Euclidean norm, used in Eq. (3.42) as a regularizer, is replaced by other norms, such as ℓ_1 or in general ℓ_p , $p > 1$, norms (Chapter 9).

From a different viewpoint, reducing the norm can be considered as an attempt to “simplify” the structure of the estimator, because a smaller number of components of the regressor now have an important say. This viewpoint becomes more clear if one considers nonlinear models, as discussed in Section 3.2. In this case, the existence of the norm of the respective parameter vector in Eq. (3.41) forces the model to get rid of the less important terms in the nonlinear expansion, $\sum_{k=1}^K \theta_k \phi_k(\mathbf{x})$, and effectively pushes K to lower values.

Although in the current context the complexity issue emerges in a rather disguised form, one can make it a major player in the game by choosing to use different functions and norms for the regularization term; as we will see next, there are many reasons that justify such choices.

INVERSE PROBLEMS: ILL-CONDITIONING AND OVERFITTING

Most tasks in machine learning belong to the so-called *inverse problems*. The latter term encompasses all the problems where one has to infer/predict/estimate the values of a model based on a set of available output/input observations (training data). In a less mathematical terminology, in inverse problems one has to unravel unknown causes from known effects, or in other words, to reverse the cause–effect relations. Inverse problems are typically *ill-posed*, as opposed to the *well-posed* ones. Well-posed problems are characterized by (a) the existence of a solution, (b) the uniqueness of the solution, and (c) the stability of the solution. The latter condition is usually violated in machine learning problems. This means that the obtained solution may be very sensitive to changes of the training set. *Ill-conditioning* is another term used to describe this sensitivity. The reason for this behavior is that the model used to describe the data can be complex, in the sense that the number of the unknown free parameters is large with respect to the number of data points. The “face” with which this problem manifests itself in machine learning is known as *overfitting*. This means that during training, the estimated parameters of the unknown model learn too much about the idiosyncrasies of the specific training data set, and the model performs badly when it deals with data sets other than that used for training. As a matter of fact, the MSE criterion discussed in Section 3.5 attempts to quantify exactly this data dependence of the task, that is, the mean deviation of the obtained estimates from the true value, by changing the training sets.

When the number of training samples is small with respect to the number of unknown parameters, the available information is not enough to “reveal” a sufficiently good model that fits the data, and it can be misleading due to the presence of the noise and possible outliers. Regularization is an elegant and efficient tool to cope with the complexity of the model, that is, to make it less complex, more smooth. There are different ways to achieve this. One way is by constraining the norm of the unknown vector, as ridge regression does. When dealing with more complex, compared to linear, models, one can use constraints on the smoothness of the involved nonlinear function, for example by involving derivatives of the model function in the regularization term. Also, regularization can help when the adopted model and the number of training points are such that no solution is possible. For example, in the LS linear regression task of Eq. (3.13), if the number N of training points is less than the dimension of the regressors \mathbf{x}_n , then the $(l+1) \times (l+1)$ matrix, $\tilde{\Sigma} = \sum_n \mathbf{x}_n \mathbf{x}_n^T$, is not invertible. Indeed, each term in the summation is the outer product of a vector with itself and hence it is a matrix of rank one. Thus, as we know from linear algebra, we need at least $l+1$ linearly independent terms of such matrices to guarantee that the sum is of full rank, and hence invertible. However, in ridge regression, this can be bypassed, because the presence of λI in Eq. (3.42) guarantees that the left-hand matrix is invertible. Furthermore, the presence of λI can also help when Σ is invertible but it is ill-conditioned. Usually in such cases, the resulting LS solution has a very large norm and, thus, it is meaningless. Regularization helps to replace the original ill-conditioned problem with a “nearby” one, which is well-conditioned and whose solution approximates the target one.

Another example where regularization can help to obtain a solution, or even a unique solution to an otherwise unsolvable problem, is when the model’s order is large compared to the number of data, although we know that it is sparse. That is, only a very small percentage of the model’s parameters are nonzero. For such a task, a standard LS linear regression approach has no solution. However, regularizing the sum of squared errors cost function using the ℓ_1 norm of the parameter vector can lead to a unique solution; the ℓ_1 norm of a vector comprises the sum of the absolute values of its components. This problem will be considered in Chapters 9 and 10.

Regularization is closely related to the task of using priors in Bayesian learning, as we will discuss in Section 3.11. Finally, note that regularization is not a panacea for facing the problem of overfitting. As a matter of fact, selecting the right set of functions \mathcal{F} in Eq. (3.3) is the first crucial step. The issue of the complexity of an estimator and the consequences on its “average” performance, as measured over all possible data sets, is discussed in Section 3.9.

Example 3.4. The goal of this example is to demonstrate that the estimator obtained via ridge regression can score a better MSE performance compared to the unconstrained LS solution. Let us consider, once again, the scalar model exposed in Example 3.2, and assume that the data are generated according to

$$y_n = \theta_o + \eta_n, \quad n = 1, 2, \dots, N,$$

where, for simplicity, we have assumed that the regressors $x_n \equiv 1$ and η_n , $n = 1, 2, \dots, N$, are i.i.d. zero mean Gaussian noise samples of variance σ_η^2 .

We have already seen in Example 3.2 that the solution to the LS parameter estimation task is the sample mean $\hat{\theta}_{\text{MVU}} = \frac{1}{N} \sum_{n=1}^N y_n$. We have also shown that this solution scores an MSE of σ_η^2/N and under the Gaussian assumption for the noise it achieves the Cramér–Rao bound. The question now is whether a biased estimator, $\hat{\theta}_b$, which corresponds to the solution of the associated ridge regression task, can achieve an MSE lower than $\text{MSE}(\hat{\theta}_{\text{MVU}})$.

It can be readily verified that Eq. (3.42), adapted to the needs of the current linear regression scenario, results in

$$\hat{\theta}_b(\lambda) = \frac{1}{N + \lambda} \sum_{n=1}^N y_n = \frac{N}{N + \lambda} \hat{\theta}_{\text{MVU}},$$

where we have explicitly expressed the dependence of the estimate $\hat{\theta}_b$ on the regularization parameter λ . Note that for the associated estimator we have $\mathbb{E}[\hat{\theta}_b(\lambda)] = \frac{N}{N + \lambda} \theta_o$.

A simple inspection of the previous relation takes us back to the discussion related to Eq. (3.24). Indeed, by following a sequence of steps similar to those in Section 3.5.1, one can verify (see Problem 3.12) that the minimum value of $\text{MSE}(\hat{\theta}_b)$ is

$$\text{MSE}(\hat{\theta}_b(\lambda_*)) = \frac{\frac{\sigma_\eta^2}{N}}{1 + \frac{\sigma_\eta^2}{N\theta_o^2}} < \frac{\sigma_\eta^2}{N} = \text{MSE}(\hat{\theta}_{\text{MVU}}), \quad (3.45)$$

attained at $\lambda_* = \sigma_\eta^2/\theta_o^2$. The answer to the question whether the ridge regression estimate offers an improvement to the MSE performance is therefore positive in the current context. As a matter of fact, there *always* exists a $\lambda > 0$ such that the ridge regression estimate, which solves the general task of Eq. (3.41), achieves an MSE lower than the one corresponding to the MVU estimate [5, Section 8.4].

We will now demonstrate the previous theoretical findings via some simulations. To this end, the true value of the model was chosen to be $\theta_o = 10^{-2}$. The noise was Gaussian of zero mean value and variance $\sigma_\eta^2 = 0.1$. The number of i.i.d. generated samples was $N = 100$. Note that this is quite large, compared to the single parameter we have to estimate. The previous values imply that $\theta_o^2 < \sigma_\eta^2/N$.

Table 3.1 Attained values of the MSE for ridge regression and different values of the regularization parameter.

λ	$MSE(\hat{\theta}_b(\lambda))$
0.1	9.99082×10^{-4}
1.0	9.79790×10^{-4}
100.0	2.74811×10^{-4}
$\lambda_* = 10^3$	9.09671×10^{-5}

The attained MSE for the unconstrained LS estimate was $MSE(\hat{\theta}_{MVU}) = 1.00108 \times 10^{-3}$.

Then it can be shown that for any value of $\lambda > 0$, we can obtain a value for $MSE(\hat{\theta}_b(\lambda))$ which is smaller than that of $MSE(\hat{\theta}_{MVU})$ (see Problem 3.12). This is verified by the values shown in Table 3.1. To compute the MSE values in the table, the expectation operation in the definition in Eq. (3.21) was approximated by the respective sample mean. To this end, the experiment was repeated L times and the MSE was computed as

$$MSE \approx \frac{1}{L} \sum_{i=1}^L (\hat{\theta}_i - \theta_o)^2.$$

To get accurate results, we perform $L = 10^6$ trials. The corresponding MSE value for the unconstrained LS task is equal to $MSE(\hat{\theta}_{MVU}) = 1.00108 \times 10^{-3}$. Observe that substantial improvements can be attained when using regularization, in spite of the relatively large number of training data.

However, the percentage of performance improvement depends heavily on the specific values that define the model, as Eq. (3.45) suggests. For example, if $\theta_o = 0.1$, the obtained values from the experiments were $MSE(\hat{\theta}_{MVU}) = 1.00061 \times 10^{-3}$ and $MSE(\hat{\theta}_b(\lambda_*)) = 9.99578 \times 10^{-4}$. The theoretical ones, as computed from Eq. (3.45), are 1×10^{-3} and 9.99001×10^{-4} , respectively. The improvement obtained by using the ridge regression is now rather insignificant.

3.9 THE BIAS–VARIANCE DILEMMA

This section goes one step beyond Section 3.5. There, the MSE criterion was used to quantify the performance with respect to the unknown parameter. Such a setting was useful to help us understand some trends and also better digest the notions of “biased” versus “unbiased” estimation. Here, although the criterion will be the same, it will be used in a more general setting. To this end, we shift our interest from the unknown parameter to the dependent variable and our goal becomes obtaining an estimator of the value y , given a measurement of the regressor vector, $\mathbf{x} = \mathbf{x}$. Let us first consider the more general form of regression,

$$y = g(\mathbf{x}) + \eta, \quad (3.46)$$

where, for simplicity and without loss of generality, once more we have assumed that the dependent variable takes values in the real axis, $y \in \mathbb{R}$. The first question to be addressed is whether there exists an estimator that guarantees minimum MSE performance.

3.9.1 MEAN-SQUARE ERROR ESTIMATION

Our goal is estimating an unknown (nonlinear in general) function $g(\mathbf{x})$. This problem can be cast in the context of the more general estimation task setting.

Consider the *jointly distributed* random variables y, \mathbf{x} . Then, given any observation, $\mathbf{x} = \mathbf{x} \in \mathbb{R}^l$, the task is to obtain a function $\hat{y} := \hat{g}(\mathbf{x}) \in \mathbb{R}$, such that

$$\hat{g}(\mathbf{x}) = \arg \min_{f: \mathbb{R}^l \rightarrow \mathbb{R}} \mathbb{E}[(y - f(\mathbf{x}))^2], \quad (3.47)$$

where the expectation is taken with respect to the conditional probability of y given the value of \mathbf{x} , or in other words, $p(y|\mathbf{x})$.

We will show that the optimal estimator is the mean value of y , or

$$\hat{g}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] := \int_{-\infty}^{+\infty} y p(y|\mathbf{x}) dy : \text{optimal MSE estimator.} \quad (3.48)$$

Proof. We have

$$\begin{aligned} \mathbb{E}[(y - f(\mathbf{x}))^2] &= \mathbb{E}\left[\left(y - \mathbb{E}[y|\mathbf{x}] + \mathbb{E}[y|\mathbf{x}] - f(\mathbf{x})\right)^2\right] \\ &= \mathbb{E}\left[\left(y - \mathbb{E}[y|\mathbf{x}]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x})\right)^2\right] \\ &\quad + 2\mathbb{E}\left[\left(y - \mathbb{E}[y|\mathbf{x}]\right)\left(\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x})\right)\right], \end{aligned}$$

where the dependence of the expectation on \mathbf{x} has been omitted for notational convenience. It is readily seen that the last (product) term on the right-hand side is zero, hence, we are left with the following:

$$\mathbb{E}[(y - f(\mathbf{x}))^2] = \mathbb{E}\left[\left(y - \mathbb{E}[y|\mathbf{x}]\right)^2\right] + (\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))^2, \quad (3.49)$$

where we have taken into account that, for fixed \mathbf{x} , the terms $\mathbb{E}[y|\mathbf{x}]$ and $f(\mathbf{x})$ are not random variables. From Eq. (3.49) we finally obtain our claim,

$$\mathbb{E}[(y - f(\mathbf{x}))^2] \geq \mathbb{E}\left[\left(y - \mathbb{E}[y|\mathbf{x}]\right)^2\right]. \quad (3.50)$$

□

Note that this is a very elegant result. The optimal estimate, in the MSE sense, of the value of the unknown function at a point \mathbf{x} is given as $\hat{g}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}]$. Sometimes, the latter is also known as the *regression of y conditioned on $\mathbf{x} = \mathbf{x}$* . This is, in general, a nonlinear function. It can be shown that if (y, \mathbf{x}) take values in $\mathbb{R} \times \mathbb{R}^l$ and are jointly Gaussian, then the optimal MSE estimator $\mathbb{E}[y|\mathbf{x}]$ is a linear (affine) function of \mathbf{x} .

The previous results generalize to the case where \mathbf{y} is a random vector that takes values in \mathbb{R}^k . The optimal MSE estimate, given the values of $\mathbf{x} = \mathbf{x}$, is equal to

$$\hat{g}(\mathbf{x}) = \mathbb{E}[\mathbf{y}|\mathbf{x}],$$

where now $\hat{g}(\mathbf{x}) \in \mathbb{R}^k$ (Problem 3.15). Moreover, if (\mathbf{y}, \mathbf{x}) are jointly Gaussian random vectors, the MSE optimal estimator is also an affine function of \mathbf{x} (Problem 3.16).

The findings of this subsection can be fully justified by physical reasoning. Assume, for simplicity, that the noise source in Eq. (3.46) is of zero mean. Then, for a fixed value $\mathbf{x} = \mathbf{x}$, we have $\mathbb{E}[\mathbf{y}|\mathbf{x}] = g(\mathbf{x})$ and the respective MSE is equal to

$$\text{MSE} = \mathbb{E}[(\mathbf{y} - \mathbb{E}[\mathbf{y}|\mathbf{x}])^2] = \sigma_\eta^2. \quad (3.51)$$

No other function of \mathbf{x} can do better, because the optimal one achieves an MSE equal to the noise variance, which is irreducible; it represents the intrinsic uncertainty of the system. As Eq. (3.49) suggests, any other function $f(\mathbf{x})$ will result in an MSE larger by the factor $(\mathbb{E}[\mathbf{y}|\mathbf{x}] - f(\mathbf{x}))^2$, which corresponds to the deviation from the optimal one.

3.9.2 BIAS–VARIANCE TRADEOFF

We have just seen that the optimal estimate, in the MSE sense, of the dependent variable in a regression task is given by the conditional expectation $\mathbb{E}[\mathbf{y}|\mathbf{x}]$. In practice, any estimator is computed based on a specific training data set, say, \mathcal{D} . Let us make the dependence on the training set explicit and express the estimate as a function of \mathbf{x} parameterized on \mathcal{D} , or $f(\mathbf{x}; \mathcal{D})$. A reasonable measure to quantify the performance of an estimator is its mean-square deviation from the optimal one, expressed by $\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}[\mathbf{y}|\mathbf{x}])^2]$, where the mean is taken with respect to all possible training sets, because each one results in a different estimate. Following a similar path as for Eq. (3.22), we obtain

$$\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}[\mathbf{y}|\mathbf{x}])^2] = \underbrace{\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})])^2]}_{\text{Variance}} + \underbrace{(\mathbb{E}_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})] - \mathbb{E}[\mathbf{y}|\mathbf{x}])^2}_{\text{Bias}^2}. \quad (3.52)$$

As was the case for the MSE parameter estimation task when changing from one training set to another, the mean-square deviation from the optimal estimate comprises two terms. The first one is the variance of the estimator around its own mean value and the second one is the squared difference of the mean from the optimal estimate, that is, the bias. It turns out that one *cannot* make *both* terms small simultaneously. For a fixed number of training points N in the data sets \mathcal{D} , trying to minimize the variance term results in an increase of the bias term and vice versa. This is because in order to reduce the bias term, one has to increase the complexity (more free parameters) of the adopted estimator $f(\cdot; \mathcal{D})$. This, in turn, results in higher variance as we change the training sets. This is a manifestation of the overfitting issue that we have already discussed. The only way to reduce both terms simultaneously is to increase

the number of the training data points N , and at the same time increase the complexity of the model *carefully*, so as to achieve the aforementioned goal. If one increases the number of training points and at the same time increases the model complexity excessively, the overall MSE may increase. This is known as the *bias–variance dilemma or tradeoff*. This is an issue that is omnipresent in any estimation task. Usually, we refer to it as *Occam’s razor* rule.

Occam was a logician and a nominalist scholastic medieval philosopher who expressed the following law of parsimony: “Plurality must never be posited without necessity.” The great physicist Paul Dirac expressed the same statement from an esthetics point of view, which underlies mathematical theories: “A theory with a mathematical beauty is more likely to be correct than an ugly one that fits the data.” In our context of model selection, it is understood that one has to select the simplest model that can “explain” the data. Although this is not a scientifically proven result, it underlies the rationale behind a number of developed model selection techniques [1,32,33,40] and [37, Chapter 5], which trade off complexity with accuracy.

Let us now try to find the MSE, given \mathbf{x} , by considering all possible sets \mathcal{D} . To this end, note that the left-hand side of Eq. (3.52) is the mean with respect to \mathcal{D} of the second term in Eq. (3.49), if \mathcal{D} is brought explicitly in the notation. It is easy to see that, by reconsidering Eq. (3.49) and taking the expectation on both y and \mathcal{D} , given the value of $\mathbf{x} = \mathbf{x}$, the resulting MSE becomes (try it, following similar arguments as for Eq. (3.52))

$$\begin{aligned}\text{MSE}(\mathbf{x}) &= \mathbb{E}_{y|\mathbf{x}} \mathbb{E}_{\mathcal{D}} \left[(y - f(\mathbf{x}; \mathcal{D}))^2 \right] \\ &= \sigma_\eta^2 + \mathbb{E}_{\mathcal{D}} \left[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}} [f(\mathbf{x}; \mathcal{D})])^2 \right] \\ &\quad + \left(\mathbb{E}_{\mathcal{D}} [f(\mathbf{x}; \mathcal{D})] - \mathbb{E}[y|\mathbf{x}] \right)^2,\end{aligned}\tag{3.53}$$

where Eq. (3.51) has been used and the product rule, as stated in Chapter 2, has been exploited. In the sequel, one can take the mean over \mathbf{x} . In other words, this is the prediction MSE over all possible inputs, averaged over all possible training sets. The resulting MSE is also known as the *test* or *generalization error* and it is a measure of the performance of the respective adopted model. Note that the generalization error in Eq. (3.53) involves averaging over (theoretically) all possible training data sets of certain size N . In contrast, the so-called *training error* is computed over a single data set, the one used for the training, and this results in an overoptimistic estimate of the error. We will come back to this important issue in Section 3.13.

Example 3.5. Let us consider a simplistic, yet pedagogic, example that demonstrates this tradeoff between bias and variance. We are given a set of training points that are generated according to a regression model of the form

$$y = g(x) + \eta.\tag{3.54}$$

The graph of $g(x)$ is shown in Fig. 3.7. The function $g(x)$ is a fifth-order polynomial. Training sets are generated as follows. For each set, the x -axis is sampled at N equidistant points x_n , $n = 1, \dots, N$, in the interval $[-1, 1]$. Then, each training set \mathcal{D}_i is created as

$$\mathcal{D}_i = \{(g(x_n) + \eta_{n,i}, x_n) : n = 1, 2, \dots, N\}, \quad i = 1, 2, \dots,$$

where $\eta_{n,i}$ denotes different noise samples, drawn i.i.d. from a white noise process. In other words, all training points have the same x -coordinate but different y -coordinates due to the different values of the noise. The gray dots in the (x, y) -plane in Fig. 3.7 correspond to one realization of a training set, for the case of $N = 10$. For comparison, the set of noiseless points, $(g(x_n), x_n)$, $n = 1, 2, \dots, 10$, is shown as red dots on the graph of $g(x)$.

First, we are going to be very naive, and choose a fixed linear model to fit the data,

$$\hat{y} = f_1(x) = \theta_0 + \theta_1 x,$$

where the values θ_1 and θ_0 have been chosen arbitrarily, irrespective of the training data. The graph of this straight line is shown in Fig. 3.7. Because no training was involved and the model parameters are fixed, there is no variation as one changes the training sets and $\mathbb{E}_{\mathcal{D}}[f_1(x)] = f_1(x)$, with the variance term being equal to zero. On the other hand, the square of the bias, which is equal to $(f_1(x) - \mathbb{E}[y|x])^2$, is expected to be large because the choice of the model was arbitrary, without paying attention to the training data.

In the sequel, we go to the other “extreme.” We choose a relatively complex class of functions, such as a high-order (10th-order) polynomial f_2 . The dependence of the resulting estimates on the respective set \mathcal{D} , which is used for training, is explicitly denoted as $f_2(\cdot; \mathcal{D})$. Note that for each one of the sets, the corresponding graph of the resulting optimal model is expected to go through the training points, i.e.,

$$f_2(x_n; \mathcal{D}_i) = g(x_n) + \eta_{n,i}, \quad n = 1, 2, \dots, N.$$

This is shown in Fig. 3.7 for one curve. Note that, in general, this is always the case if the order of the polynomial (model) is large and the number of parameters to be estimated is larger than the number of

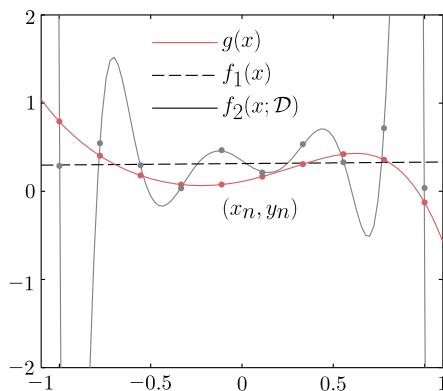


FIGURE 3.7

The observed data are denoted as gray dots. These are the result of adding noise to the red points, which lie on the red curve associated with the unknown $g(\cdot)$. Fitting the data by the fixed polynomial $f_1(x)$ results in high bias; observe that most of the data points lie outside the straight line. On the other hand, the variance of the “estimator” will be zero. In contrast, fitting a high-degree polynomial $f_2(x; \mathcal{D})$ results in low bias, because the corresponding curve goes through all the data points; however, the respective variance will be high.

training points. For the example shown in the figure, we are given 10 training points and 11 parameters have to be estimated, for a 10th-order polynomial (including the bias). One can obtain a perfect fit on the training points by solving a linear system comprising 10 equations ($N = 10$) with 11 unknowns. This is the reason that when the order of the model is large, a *zero* (or very small, in practice) error can be achieved on the training data set. We will come back to this issue in Section 3.13.

For this high-order polynomial fitting setup of the experiment, the following reasoning holds true. The bias term at each point x_n , $n = 1, 2, \dots, N$, is zero, because

$$\mathbb{E}_{\mathcal{D}}[f_2(x_n; \mathcal{D})] = \mathbb{E}_{\mathcal{D}}[g(x_n) + \eta] = g(x_n) = \mathbb{E}_{\mathcal{D}}[y|x_n].$$

On the other hand, the variance term at the points x_n , $n = 1, 2, \dots, N$, is expected to be large, because

$$\mathbb{E}_{\mathcal{D}}[(f_2(x_n; \mathcal{D}) - g(x_n))^2] = \mathbb{E}_{\mathcal{D}}[(g(x_n) + \eta - g(x_n))^2] = \sigma_{\eta}^2.$$

Assuming that the functions f_2 and g are continuous and smooth enough and the points x_n are sampled densely enough to cover the interval of interest in the real axis, we expect similar behavior at all points $x \neq x_n$.

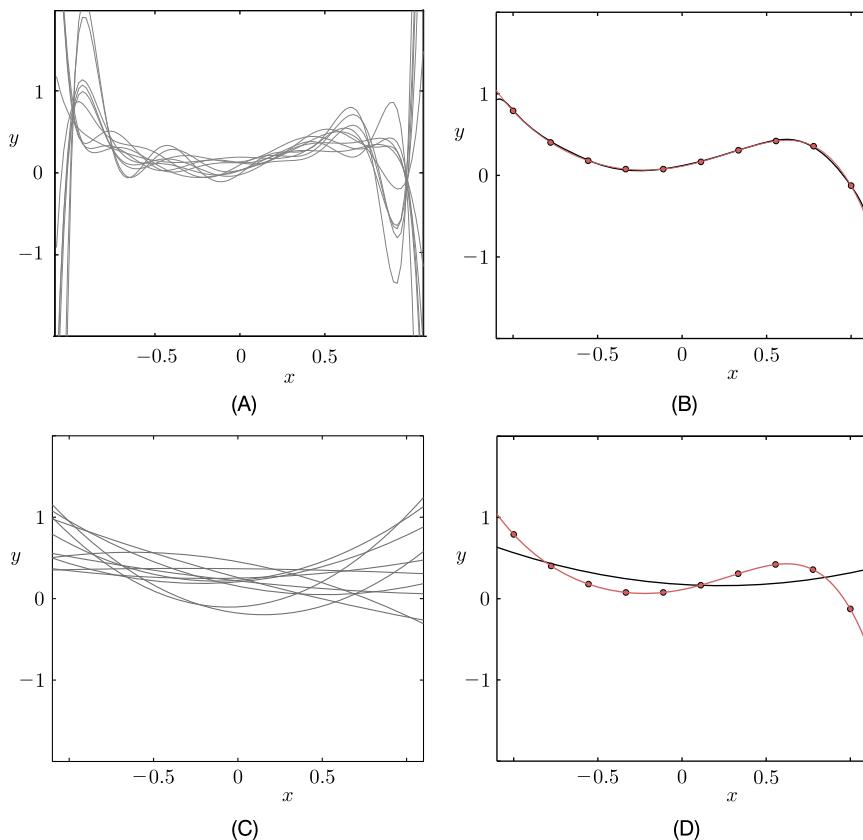
Example 3.6. This is a more realistic example that builds upon the previous one. The data are generated as before, via the regression model in Eq. (3.54) using the fifth-order polynomial for g . The number of training points is equal to $N = 10$ and 1000 training sets \mathcal{D}_i , $i = 1, 2, \dots, 1000$, are generated. Two sets of experiments have been run.

The first one attempts to fit in the noisy data a high-order polynomial of degree equal to 10 (as in the previous example) and the second one adopts a second-order polynomial. For each one of the two setups, the experiment is repeated 1000 times, each time with a different data set \mathcal{D}_i . Figs. 3.8A and C show 10 (for visibility reasons) out of the 1000 resulting curves for the high- and low-order polynomials, respectively. The substantially higher variance for the case of the high-order polynomial is readily noticed. Figs. 3.8B and D show the corresponding curves which result from averaging over the 1000 performed experiments, together with the graph of the “unknown” (original g) function. The high-order polynomial results in an excellent fit of very low bias. The opposite is true for the case of the second-order polynomial.

Thus, in summary, for a fixed number of training points, the more complex the prediction model is (larger number of parameters), the larger the variance becomes, as we change from one training set to another. On the other hand, the more complex the model is, the smaller the bias gets; that is, the average model by training over different data sets gets closer to the optimal MSE one. The reader may find more information on the bias–variance dilemma task in [16].

3.10 MAXIMUM LIKELIHOOD METHOD

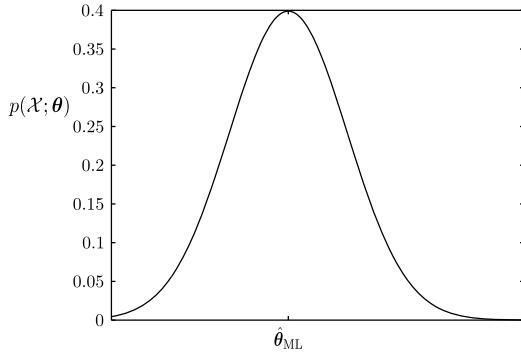
So far, we have approached the estimation problem as an optimization task around a set of training examples, without paying any attention to the underlying statistics that generates these points. We only used statistics in order to check under which conditions the estimators were efficient. However, the optimization step did not involve any statistical information. For the rest of the chapter, we are going to

**FIGURE 3.8**

(A) Ten of the resulting curves from fitting a 10th-order polynomial. (B) The corresponding average over 1000 different experiments. The red curve represents the unknown polynomial. The dots indicate the points that give birth to the training data, as described in the text. (C), (D) The results from fitting a second-order polynomial. Observe the bias–variance tradeoff as a function of the complexity of the fitted model.

involve statistics more and more. In this section, the ML method is introduced. It is not an exaggeration to say that ML and LS are two of the major pillars on which parameter estimation is based and new methods are inspired from. The ML method was suggested by Sir Ronald Aylmer Fisher.

Once more, we will first formulate the method in a general setting, independent of the regression/classification tasks. We are given a set of, say, N observations, $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, drawn from a probability distribution. We assume that the joint PDF of these N observations is of a known parametric functional type, denoted as $p(\mathcal{X}; \boldsymbol{\theta})$, where the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^K$ is unknown and the task is to estimate its value. This joint PDF is known as the *likelihood function* of $\boldsymbol{\theta}$ with respect to the given set of observations \mathcal{X} . According to the ML method, the estimate is provided by

**FIGURE 3.9**

According to the maximum likelihood method, we assume that, given the set of observations, the estimate of the unknown parameter is the value that maximizes the corresponding likelihood function.

$$\hat{\boldsymbol{\theta}}_{\text{ML}} := \arg \max_{\boldsymbol{\theta} \in \mathcal{A} \subset \mathbb{R}^K} p(\mathcal{X}; \boldsymbol{\theta}) : \quad \text{maximum likelihood estimate.} \quad (3.55)$$

For simplicity, we will assume that the constraint set, \mathcal{A} , coincides with \mathbb{R}^K , i.e., $\mathcal{A} = \mathbb{R}^K$, and that the parameterized family $\{p(\mathcal{X}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \mathbb{R}^K\}$ enjoys a unique minimizer with respect to the parameter $\boldsymbol{\theta}$. This is illustrated in Fig. 3.9. In other words, given the set of observations $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, one selects the unknown parameter vector so as to make this joint event the most likely one to happen.

Because the logarithmic function $\ln(\cdot)$ is monotone and increasing, one can instead search for the maximum of the *log-likelihood function*,

$$\frac{\partial \ln p(\mathcal{X}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{\text{ML}}} = \mathbf{0}. \quad (3.56)$$

Assuming the observations to be i.i.d., the ML estimator has some very attractive properties, namely:

- The ML estimator is asymptotically unbiased; that is, assuming that the model of the PDF which we have adopted is correct and there exists a true parameter $\boldsymbol{\theta}_o$, we have

$$\lim_{N \rightarrow \infty} \mathbb{E}[\hat{\boldsymbol{\theta}}_{\text{ML}}] = \boldsymbol{\theta}_o. \quad (3.57)$$

- The ML estimate is asymptotically *consistent* so that given any value of $\epsilon > 0$,

$$\lim_{N \rightarrow \infty} \text{Prob} \left\{ \left| \hat{\boldsymbol{\theta}}_{\text{ML}} - \boldsymbol{\theta}_o \right| > \epsilon \right\} = 0, \quad (3.58)$$

that is, for large values of N , we expect the ML estimate to be very close to the true value with high probability.

- The ML estimator is asymptotically efficient; that is, it achieves the Cramér–Rao lower bound.
- If there exists a sufficient statistic $T(\mathcal{X})$ for an unknown parameter, then only $T(\mathcal{X})$ suffices to express the respective ML estimate (Problem 3.20).

- Moreover, assuming that an efficient estimator does exist, this estimator is optimal in the ML sense (Problem 3.21).

Example 3.7. Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ be the observation vectors stemming from a multivariate normal distribution with known covariance matrix and unknown mean (Chapter 2), that is,

$$p(\mathbf{x}_n; \boldsymbol{\mu}) = \frac{1}{(2\pi)^{l/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu})\right).$$

Assume that the observations are mutually independent. Obtain the ML estimate of the unknown mean vector.

For the N statistically independent observations, the joint log-likelihood function is given by

$$L(\boldsymbol{\mu}) = \ln \prod_{n=1}^N p(\mathbf{x}_n; \boldsymbol{\mu}) = -\frac{N}{2} \ln((2\pi)^l |\Sigma|) - \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}).$$

Taking the gradient with respect to $\boldsymbol{\mu}$, we obtain⁷

$$\frac{\partial L(\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} := \begin{bmatrix} \frac{\partial L}{\partial \mu_1} \\ \frac{\partial L}{\partial \mu_2} \\ \vdots \\ \frac{\partial L}{\partial \mu_l} \end{bmatrix} = \sum_{n=1}^N \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}),$$

and equating to $\mathbf{0}$ leads to

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

In other words, for Gaussian distributed data, the ML estimate of the mean is the sample mean. Moreover, note that the ML estimate is expressed in terms of its sufficient statistic (see Section 3.7).

3.10.1 LINEAR REGRESSION: THE NONWHITE GAUSSIAN NOISE CASE

Consider the linear regression model

$$\mathbf{y} = \boldsymbol{\theta}^T \mathbf{x} + \eta.$$

We are given N training data points (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$. The corresponding (unobserved) noise samples η_n , $n = 1, \dots, N$, are assumed to follow a jointly Gaussian distribution with zero mean and

⁷ Recall from matrix algebra that $\frac{\partial(\mathbf{x}^T \mathbf{b})}{\partial \mathbf{x}} = \mathbf{b}$ and $\frac{\partial(\mathbf{x}^T A \mathbf{x})}{\partial \mathbf{x}} = 2A\mathbf{x}$ if A is symmetric (Appendix A).

covariance matrix equal to Σ_η . That is, the corresponding random vector of all the noise samples stacked together, $\boldsymbol{\eta} = [\eta_1, \dots, \eta_N]^T$, follows the multivariate Gaussian distribution

$$p(\boldsymbol{\eta}) = \frac{1}{(2\pi)^{1/N} |\Sigma_\eta|^{1/2}} \exp\left(-\frac{1}{2} \boldsymbol{\eta}^T \Sigma_\eta^{-1} \boldsymbol{\eta}\right).$$

Our goal is to obtain the ML estimate of the parameters, $\boldsymbol{\theta}$.

Replacing $\boldsymbol{\eta}$ with $\mathbf{y} - X\boldsymbol{\theta}$, and taking the logarithm, the joint log-likelihood function of $\boldsymbol{\theta}$ with respect to the training set is given by

$$L(\boldsymbol{\theta}) = -\frac{N}{2} \ln(2\pi) - \frac{1}{2} \ln |\Sigma_\eta| - \frac{1}{2} (\mathbf{y} - X\boldsymbol{\theta})^T \Sigma_\eta^{-1} (\mathbf{y} - X\boldsymbol{\theta}), \quad (3.59)$$

where $\mathbf{y} := [y_1, y_2, \dots, y_N]^T$, and $X := [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$ stands for the input matrix. Taking the gradient with respect to $\boldsymbol{\theta}$, we get

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = X^T \Sigma_\eta^{-1} (\mathbf{y} - X\boldsymbol{\theta}), \quad (3.60)$$

and equating to the zero vector, we obtain

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = (X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1} \mathbf{y}. \quad (3.61)$$

Remarks 3.3.

- Compare Eq. (3.61) with the LS solution given in Eq. (3.17). They are different, unless the covariance matrix of the successive noise samples, Σ_η , is diagonal and of the form $\sigma_\eta^2 I$, that is, if the noise is Gaussian as well as white. In this case, the LS and ML solutions coincide. However, if the noise sequence is nonwhite, the two estimates differ. Moreover, it can be shown (Problem 3.9) that *in this case of colored Gaussian noise, the ML estimate is an efficient one and it attains the Cramér–Rao bound, even if N is finite.*

3.11 BAYESIAN INFERENCE

In our discussion so far, we have assumed that the parameters associated with the functional form of the adopted model are deterministic constants whose values are unknown to us. In this section, we will follow a different rationale. The unknown parameters will be treated as random variables. Hence, whenever our goal is to estimate their values, this is conceived as an effort to estimate the values of a *specific* realization that corresponds to the observed data. A more detailed discussion concerning the Bayesian inference rationale is provided in Chapter 12. As the name Bayesian suggests, the heart of the method beats around the celebrated Bayes theorem. Given two jointly distributed random vectors, say, $\mathbf{x}, \boldsymbol{\theta}$, the Bayes theorem states that

$$p(\mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathbf{x})p(\mathbf{x}). \quad (3.62)$$

David Bayes (1702–1761) was an English mathematician and a Presbyterian minister who first developed the basics of the theory. However, it was Pierre-Simon Laplace (1749–1827), the great French mathematician, who further developed and popularized it.

Assume that $\mathbf{x}, \boldsymbol{\theta}$ are two statistically dependent random vectors. Let $\mathcal{X} = \{\mathbf{x}_n \in \mathbb{R}^l, n = 1, 2, \dots, N\}$ be the set of observations resulting from N successive experiments. Then the Bayes theorem gives

$$p(\boldsymbol{\theta}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) d\boldsymbol{\theta}}. \quad (3.63)$$

Obviously, if the observations are i.i.d., then we can write

$$p(\mathcal{X}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}).$$

In the previous formulas, $p(\boldsymbol{\theta})$ is the a priori or prior PDF concerning the statistical distribution of $\boldsymbol{\theta}$, and $p(\boldsymbol{\theta}|\mathcal{X})$ is the conditional or a posteriori or posterior PDF, formed after the set of N observations has been obtained. The prior probability density, $p(\boldsymbol{\theta})$, can be considered as a constraint that *encapsulates our prior knowledge* about $\boldsymbol{\theta}$. Undoubtedly, our uncertainty about $\boldsymbol{\theta}$ is modified after the observations have been received, because more information is now disclosed to us. If the adopted assumptions about the underlying models are sensible, we expect the posterior PDF to be a more accurate one to describe the statistical nature of $\boldsymbol{\theta}$. We will refer to the process of approximating the PDF of a random quantity based on a set of training data as *inference*, to differentiate it from the process of estimation, which returns a single value for each parameter/variable. So, according to the inference approach, one attempts to draw conclusions about the nature of the randomness that underlies the variables of interest. This information can in turn be used to make predictions and to take decisions.

We will exploit Eq. (3.63) in two ways. The first refers to our familiar goal of obtaining an estimate of the parameter vector $\boldsymbol{\theta}$, which “controls” the model that describes the generation mechanism of our observations, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$. Because \mathbf{x} and $\boldsymbol{\theta}$ are two statistically dependent random vectors, we know from Section 3.9 that the MSE optimal estimate of the value of $\boldsymbol{\theta}$, given \mathcal{X} , is

$$\hat{\boldsymbol{\theta}} = \mathbb{E}[\boldsymbol{\theta}|\mathcal{X}] = \int \boldsymbol{\theta} p(\boldsymbol{\theta}|\mathcal{X}) d\boldsymbol{\theta}. \quad (3.64)$$

Another direction along which one can exploit the Bayes theorem, in the context of statistical inference, is to obtain an estimate of the PDF of \mathbf{x} given the observations \mathcal{X} . This can be done by *marginalizing* over a distribution, using the equation

$$p(\mathbf{x}|\mathcal{X}) = \int p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{X}) d\boldsymbol{\theta}, \quad (3.65)$$

where the conditional independence of \mathbf{x} on \mathcal{X} , given the value $\boldsymbol{\theta} = \boldsymbol{\theta}$, expressed as $p(\mathbf{x}|\mathcal{X}, \boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta})$, has been used. Indeed, if the value $\boldsymbol{\theta}$ is given, then the conditional $p(\mathbf{x}|\boldsymbol{\theta})$ is fully defined and does not depend on \mathcal{X} . The dependence of \mathbf{x} on \mathcal{X} is through $\boldsymbol{\theta}$, if the latter is unknown. Eq. (3.65) provides an estimate of the unknown PDF, by exploiting the information that resides in the obtained

observations and in the adopted functional dependence on the parameters θ . Note that, in contrast to what we did in the case of the ML method, where we used the observations to obtain an estimate of the parameter vector, here we assume the parameters to be random variables, provide our prior knowledge about θ via $p(\theta)$, and integrate the joint PDF, $p(\mathbf{x}, \theta | \mathcal{X})$, over θ .

Once $p(\mathbf{x} | \mathcal{X})$ is available, it can be used for prediction. Assuming that we have obtained the observations $\mathbf{x}_1, \dots, \mathbf{x}_N$, our estimate about the next value, \mathbf{x}_{N+1} , can be determined via $p(\mathbf{x}_{N+1} | \mathcal{X})$. Obviously, the form of $p(\mathbf{x} | \mathcal{X})$ is, in general, changing as new observations are obtained, because each time an observation becomes available, part of our uncertainty about the underlying randomness is removed.

Example 3.8. Consider the simplified linear regression task of Eq. (3.33) and assume $x = 1$. As we have already said, this problem is that of estimating the value of a constant buried in noise. Our methodology will follow the Bayesian philosophy. Assume that the noise samples are i.i.d. drawn from a Gaussian PDF of zero mean and variance σ_η^2 . However, we impose our a priori knowledge concerning the unknown θ via the prior distribution

$$p(\theta) = \mathcal{N}(\theta_0, \sigma_0^2). \quad (3.66)$$

That is, we assume that we know that the values of θ lie around θ_0 , and σ_0^2 quantifies our degree of uncertainty about this prior knowledge. Our goals are first to obtain the posterior PDF, given the set of observations $\mathbf{y} = [y_1, \dots, y_N]^T$, and then to obtain $\mathbb{E}[\theta | \mathbf{y}]$, according to Eqs. (3.63) and (3.64) after adapting them to our current notational needs. We have

$$\begin{aligned} p(\theta | \mathbf{y}) &= \frac{p(\mathbf{y} | \theta) p(\theta)}{p(\mathbf{y})} = \frac{1}{p(\mathbf{y})} \left(\prod_{n=1}^N p(y_n | \theta) \right) p(\theta) \\ &= \frac{1}{p(\mathbf{y})} \left(\prod_{n=1}^N \frac{1}{\sqrt{2\pi}\sigma_\eta} \exp\left(-\frac{(y_n - \theta)^2}{2\sigma_\eta^2}\right) \right) \\ &\quad \times \frac{1}{\sqrt{2\pi}\sigma_0} \exp\left(-\frac{(\theta - \theta_0)^2}{2\sigma_0^2}\right). \end{aligned} \quad (3.67)$$

After some algebraic manipulations of Eq. (3.67) (Problem 3.25), one ends up with the following:

$$p(\theta | \mathbf{y}) = \frac{1}{\sqrt{2\pi}\sigma_N} \exp\left(-\frac{(\theta - \bar{\theta}_N)^2}{2\sigma_N^2}\right), \quad (3.68)$$

where

$$\bar{\theta}_N = \frac{N\sigma_0^2 \bar{y}_N + \sigma_\eta^2 \theta_0}{N\sigma_0^2 + \sigma_\eta^2}, \quad (3.69)$$

with $\bar{y}_N = \frac{1}{N} \sum_{n=1}^N y_n$ being the sample mean of the observations and

$$\sigma_N^2 = \frac{\sigma_\eta^2 \sigma_0^2}{N \sigma_0^2 + \sigma_\eta^2}. \quad (3.70)$$

In other words, if the prior and conditional PDFs are Gaussians, then the posterior is also Gaussian. Moreover, the mean and the variance of the posterior are given by Eqs. (3.69) and (3.70), respectively.

Observe that as the number of observations increases, $\bar{\theta}_N$ tends to the sample mean of the observations; recall that the latter is the estimate that results from the ML method. Also, note that the variance keeps decreasing as the number of observations increases, which is in line with common sense, because more observations mean less uncertainty. Fig. 3.10 illustrates the previous discussion. Data samples, y_n , were generated using a Gaussian pseudorandom generator with the mean equal to $\theta = 1$ and variance equal to $\sigma_\eta^2 = 0.1$. So the true value of our constant is equal to 1. We used a Gaussian prior PDF with mean value equal to $\theta_0 = 2$ and variance $\sigma_0^2 = 6$. We observe that as N increases, the posterior PDF gets narrower and its mean tends to the true value of 1.

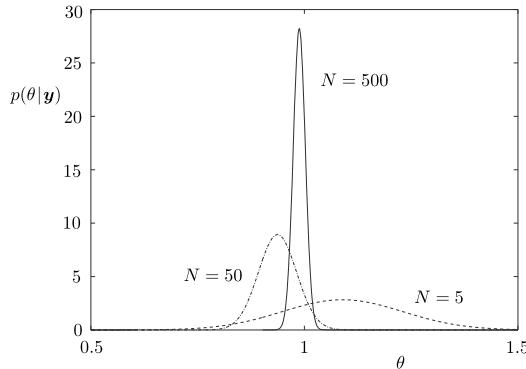


FIGURE 3.10

In the Bayesian inference approach, note that as the number of observations increases, our uncertainty about the true value of the unknown parameter is reduced and the mean of the posterior PDF tends to the true value and the variance tends to zero.

It should be pointed out that in this example case, both the ML and LS estimates become identical, or

$$\hat{\theta} = \frac{1}{N} \sum_{n=1}^N y_n = \bar{y}_N.$$

This will also be the case for the mean value in Eq. (3.69) if we set σ_0^2 very large, as might happen if we have no confidence in our initial estimate of θ_0 and we assign a very large value to σ_0^2 . In effect, this is equivalent to using no prior information.

Let us now investigate what happens if our prior knowledge about θ_0 is “embedded” in the LS criterion in the form of a constraint. This can be done by modifying the constraint in Eq. (3.40) such that

$$(\theta - \theta_0)^2 \leq \rho, \quad (3.71)$$

which leads to the equivalent minimization of the following Lagrangian:

$$\text{minimize } L(\theta, \lambda) = \sum_{n=1}^N (y_n - \theta)^2 + \lambda ((\theta - \theta_0)^2 - \rho). \quad (3.72)$$

Taking the derivative with respect to θ and equating to zero, we obtain

$$\hat{\theta} = \frac{N\bar{y}_N + \lambda\theta_0}{N + \lambda},$$

which, for $\lambda = \sigma_\eta^2/\sigma_0^2$, becomes identical to Eq. (3.69). The world is small after all! This has happened only because we used Gaussians both for the conditional and for the prior PDFs. For different forms of PDFs, this would not be the case. However, this example shows that a close relationship ties priors and constraints. They both attempt to impose prior information. Each method, in its own unique way, is associated with the respective pros and cons. In Chapters 12 and 13, where a more extended treatment of the Bayesian inference task is provided, we will see that the very essence of regularization, which is a means against overfitting, lies at the heart of the Bayesian approach.

Finally, one may wonder if the Bayesian inference has offered us any more information, compared to the deterministic parameter estimation path. After all, when the aim is to obtain a specific value for the unknown parameter, taking the mean of the Gaussian posterior comes to the same solution which results from the regularized LS approach. Well, even for this simple case, the Bayesian inference readily provides a piece of extra information; this is an estimate of the variance around the mean, which is very valuable in order to assess our trust of the recovered estimate. Of course, all these are valid provided that the adopted PDFs offer a good description of the statistical nature of the process at hand [24].

Finally, it can be shown (Problem 3.26) that the previously obtained results can be generalized for the more general linear regression model, of nonwhite Gaussian noise, considered in Section 3.10, which is modeled as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}.$$

It turns out that the posterior PDF is also Gaussian with mean value equal to

$$\mathbb{E}[\boldsymbol{\theta}|\mathbf{y}] = \boldsymbol{\theta}_0 + \left(\boldsymbol{\Sigma}_0^{-1} + \mathbf{X}^T \boldsymbol{\Sigma}_\eta^{-1} \mathbf{X} \right)^{-1} \mathbf{X}^T \boldsymbol{\Sigma}_\eta^{-1} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}_0) \quad (3.73)$$

and covariance matrix

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}|\mathbf{y}} = \left(\boldsymbol{\Sigma}_0^{-1} + \mathbf{X}^T \boldsymbol{\Sigma}_\eta^{-1} \mathbf{X} \right)^{-1}. \quad (3.74)$$

3.11.1 THE MAXIMUM A POSTERIORI PROBABILITY ESTIMATION METHOD

The maximum a posteriori probability estimation technique, usually denoted as MAP, is based on the Bayesian theorem, but it does not go as far as the Bayesian philosophy allows. The goal becomes that of obtaining an estimate which maximizes Eq. (3.63); in other words,

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg \max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} | \mathcal{X}) : \quad \text{MAP estimate,} \quad (3.75)$$

and because $p(\mathcal{X})$ is independent of $\boldsymbol{\theta}$, this leads to

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\text{MAP}} &= \arg \max_{\boldsymbol{\theta}} p(\mathcal{X} | \boldsymbol{\theta}) p(\boldsymbol{\theta}) \\ &= \arg \max_{\boldsymbol{\theta}} \{ \ln p(\mathcal{X} | \boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}) \}. \end{aligned} \quad (3.76)$$

If we consider Example 3.8, it is a matter of simple exercise to obtain the MAP estimate and show that

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \frac{N \bar{y}_N + \frac{\sigma_\eta^2}{\sigma_0^2} \theta_0}{N + \frac{\sigma_\eta^2}{\sigma_0^2}} = \bar{\theta}_N. \quad (3.77)$$

Note that for this case, the MAP estimate coincides with the regularized LS solution for $\lambda = \sigma_\eta^2 / \sigma_0^2$. Once more, we verify that adopting a prior PDF for the unknown parameter acts as a regularizer which embeds into the problem the available prior information.

Remarks 3.4.

- Observe that for the case of Example 3.8, all three estimators, namely, ML, MAP, and the Bayesian (taking the mean), result *asymptotically*, as N increases, in the same estimate. This is a more general result and it is true for other PDFs as well as for the case of parameter vectors. As the number of observations increases, our uncertainty is reduced and $p(\mathcal{X} | \boldsymbol{\theta})$, $p(\boldsymbol{\theta} | \mathcal{X})$ peak sharply around a value of $\boldsymbol{\theta}$. This forces all the methods to result in similar estimates. However, the obtained estimates are different for finite values of N . More recently, as we will see in Chapters 12 and 13, Bayesian methods have become very popular, and they seem to be the preferred choice for a number of practical problems.
- Choosing the prior PDF in the Bayesian methods is not an innocent task. In Example 3.8, we chose the conditional PDF (likelihood function) as well as the prior PDF to be Gaussians. We saw that the posterior PDF was also Gaussian. The advantage of such a choice was that we could come to closed form solutions. This is not always the case, and then the computation of the posterior PDF needs sampling methods or other approximate techniques. We will come to that in Chapters 12 and 14. However, the family of Gaussians is not the only one with this nice property of leading to closed form solutions. In probability theory, if the posterior is of the same form as the prior, we say that $p(\boldsymbol{\theta})$ is a *conjugate prior* of the likelihood function $p(\mathcal{X} | \boldsymbol{\theta})$ and then the involved integrations can be carried out in closed form (see, e.g., [15,30] and Chapter 12). Hence, the Gaussian PDF is a conjugate of itself.

- Just for the sake of pedagogical purposes, it is useful to recapitulate some of the nice properties that the Gaussian PDF possesses. We have met the following properties in various sections and problems in the book so far: (a) it is a conjugate of itself; (b) if two random variables (vectors) are jointly Gaussian, then their marginal PDFs are also Gaussian and the posterior PDF of one with respect to the other is also Gaussian; (c) moreover, the linear combination of jointly Gaussian variables turns out to be Gaussian; (d) as a by-product, it turns out that the sum of statistically independent Gaussian random variables is also a Gaussian one; and finally (e) the central limit theorem states that the sum of a large number of independent random variables tends to be Gaussian, as the number of the summands increases.

3.12 CURSE OF DIMENSIONALITY

In a number of places in this chapter, we mentioned the need of having a large number of training points. In Section 3.9.2, while discussing the bias–variance tradeoff, it was stated that in order to end up with a low overall MSE, the complexity (number of parameters) of the model should be small enough with respect to the number of training points. In Section 3.8, overfitting was discussed and it was pointed out that if the number of training points is small with respect to the number of parameters, overfitting occurs.

The question that is now raised is how big a data set should be, in order to be more relaxed concerning the performance of the designed predictor. The answer to the previous question depends largely on the dimensionality of the input space. It turns out that the larger the dimension of the input space is, the more data points are needed. This is related to the so-called *curse of dimensionality*, a term coined for the first time in [4].

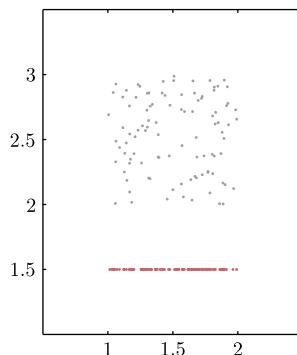


FIGURE 3.11

A simple experiment which demonstrates the curse of dimensionality. A number of 100 points are generated randomly, drawn from a uniform distribution, in order to fill the one-dimensional segment of length equal to one ($[1, 2] \times \{1.5\}$) (red points), and the two-dimensional rectangular region of unit area $[1, 2] \times [2, 3]$ (gray points). Observe that although the number of points in both cases is the same, the rectangular region is more sparsely populated than the densely populated line segment.

Let us assume that we are given the same number of points, N , thrown randomly in a unit cube (hypercube) in two different spaces, one being of low and the other of very high dimension. Then, the average distance of the points in the latter case will be much larger than that in the low-dimensional space case. As a matter of fact, the average distance shows a dependence that is analogous to the exponential term ($N^{-1/l}$), where l is the dimensionality of the space [14,37]. For example, the average distance between two out of 10^{10} points in the two-dimensional space is 10^{-5} , and in the 40-dimensional space it is equal to 1.83. Fig. 3.11 shows two cases, each one consisting of 100 points. The red points lie on a (one-dimensional) line segment of length equal to one and were generated according to the uniform distribution. Gray points cover a (two-dimensional) square region of unit area, which were also generated by a two-dimensional uniform distribution. Observe that the square area is more sparsely populated compared to the line segment. This is the general trend and high-dimensional spaces are sparsely populated; thus, many more data points are needed in order to fill in the space with enough data. Fitting a model in a parameter space, one must have enough data covering sufficiently well all regions in the space, in order to be able to learn well enough the input–output functional dependence (Problem 3.13).

There are various ways to cope with the curse of dimensionality and try to exploit the available data set in the best possible way. A popular direction is to resort to suboptimal solutions by projecting the input/feature vectors in a lower-dimensional subspace or manifold. Very often, such an approach leads to small performance losses, because the original training data, although they are generated in a high-dimensional space, may in fact “live” in a lower-dimensional subspace or manifold, due to physical dependencies that restrict the number of free parameters. Take as an example a case where the data are three-dimensional vectors, but they lie around a straight line, which is a one-dimensional linear manifold (affine set or subspace if it crosses the origin) or around a circle (one-dimensional nonlinear manifold) embedded in the three-dimensional space. In this case, the true number of free parameters is equal to one; this is because one free parameter suffices to describe the location of a point on a circle or on a straight line. The true number of free parameters is also known as the *intrinsic dimensionality* of the problem. The challenge, now, becomes that of learning the subspace/manifold onto which to project. These issues will be considered in more detail in Chapter 19.

Finally, it has to be noted that the dimensionality of the input space is not always the crucial issue. In pattern recognition, it has been shown that the critical factor is the so-called *VC dimension* of a classifier. In a number of classifiers, such as (generalized) linear classifiers or neural networks (to be considered in Chapter 18), the VC dimension is directly related to the dimensionality of the input space. However, one can design classifiers, such as the support vector machines (Chapter 11), whose performance is not directly related to the input space and they can be efficiently designed in spaces of very high (or even infinite) dimensionality [37,40].

3.13 VALIDATION

From previous sections, we already know that what is a “good” estimate according to one set of training points is not necessarily a good one for other data sets. This is an important aspect in any machine learning task; the performance of a method may vary with the random choice of the training set. A major phase, in any machine learning task, is to quantify/predict the performance that the designed (prediction) model is expected to exhibit in practice. It will not come as a surprise that “measuring” the

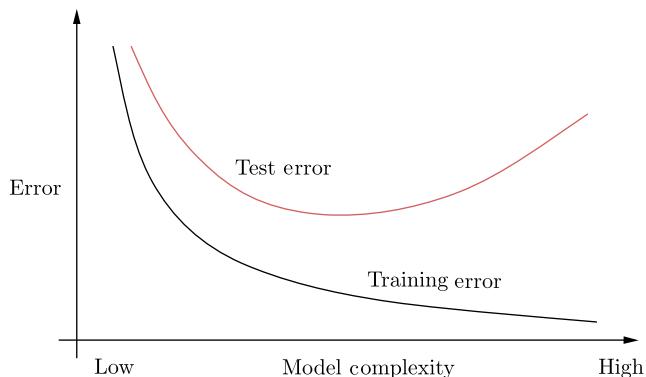


FIGURE 3.12

The training error tends to zero as the model complexity increases; for complex enough models with a large number of free parameters, a perfect fit to the training data is possible. However, the test error initially decreases, because more complex models “learn” the data better, up to a certain point. After that point of complexity, the test error increases.

performance against the training data set would lead to an “optimistic” value of the performance index, because this is computed on the same set on which the estimate was optimized; this trend has been known since the early 1930s [22]. For example, if the model is complex enough, with a large number of free parameters, the training error may even become zero, since a perfect fit to the data can be achieved. What is more meaningful and fair is to look for the so-called *generalization* performance of an estimator, that is, its average performance computed over *different* data sets which *did not* participate in the training (see the last paragraph of Section 3.9.2). The error associated with this average performance is known as the test error or the generalization error.⁸

Fig. 3.12 shows a typical performance that is expected to result in practice. The error measured on the (single) training data set is shown together with the (average) test error as the model complexity varies. If one tries to fit a complex model, with respect to the size of the available training set, then the error measured on the training set will be overoptimistic. On the contrary, the true error, as this is represented by the test error, takes large values; in the case where the performance index is the MSE, this is mainly contributed by the variance term (Section 3.9.2). On the other hand, if the model is too simple the test error will also attain large values; for the MSE case, this time the contribution is mainly due to the bias term. The idea is to have a model complexity that corresponds to the minimum of the respective curve. As a matter of fact, this is the point that various model selection techniques try to predict.

For some simple cases and under certain assumptions concerning the underlying models, we are able to have analytical formulas that quantify the average performance as we change data sets. However, in practice, this is hardly ever the case, and one must have a way to test the performance of an

⁸ Note that some authors use the term generalization error to denote the difference between the test and the training errors. Another term for this difference is generalization gap.

obtained classifier/predictor using different data sets. This process is known as *validation*, and there are a number of alternatives that one can resort to.

Assuming that enough data are at the designer's disposal, one can split the data into one part, to be used for training, and another part for testing the performance. For example, the probability of error is computed over the test data set for the case of a classifier, or the MSE for the case of a regression task; other measures of fit can also be used. If this path is taken, one has to make sure that both the size of the training set and the size of the test set are large enough with respect to the model complexity; a large test data set is required in order to provide a statistically sound result on the test error. Especially if different methods are compared, the smaller the difference in their comparative performance is expected to be, the larger the size of the test set must be, in order to guarantee reliable conclusions [37, Chapter 10].

CROSS-VALIDATION

In practice, very often the size of the available data set is not sufficient and one cannot afford to “lose” part of it from the training set for the sake of testing. *Cross-validation* is a very common technique that is usually employed. Cross-validation has been rediscovered a number of times; however, to our knowledge, the first published description can be traced back to [25]. According to this method, the data set is split into K roughly equal-sized parts. We repeat training K times, each time selecting one (different each time) part of the data for testing and the remaining $K - 1$ parts for training. This gives us the advantage of testing with a part of the data that has not been involved in the training, so it can be considered independent, and at the same time using, eventually, all the data, both for training and testing. Once we finish, we can (a) combine the obtained K estimates by averaging or via another more advanced way and (b) combine the errors from the test sets to get a better estimate of the test error that our estimator is expected to exhibit in real-life applications. This method is known as K -fold cross-validation. An extreme case is when we use $K = N$, so that each time one sample is left for testing. This is sometimes referred to as the *leave-one-out* (LOO) cross-validation method. The price one pays for K -fold cross-validation is the complexity of training K times. In practice, the value of K depends very much on the application, but typical values are of the order of 5 to 10.

The cross-validation estimator of the test error is very nearly unbiased. The reason for the slight bias is that the training set in cross-validation is slightly smaller than the actual data set. The effect of this bias will be conservative in the sense that the estimated fit will be slightly biased in the direction suggesting a poorer fit. In practice, this bias is rarely a concern, especially in the LOO case, where each time only one sample is left out. The variance, however, of the cross-validation estimator can be large, and this has to be taken into account when comparing different methods. In [13], the use of *bootstrap* techniques is suggested in order to reduce the variance of the obtained error predictions by the cross-validation method.

Moreover, besides complexity and high variance, cross-validation schemes are not beyond criticisms. Unfortunately, the overlap among the training sets introduces unknowable dependencies between runs, making the use of formal statistical tests difficult [11]. All this discussion reveals that the validation task is far from innocent. Ideally, one should have at her/his disposal large data sets and divide them in several *nonoverlapping* training sets, of whatever size is appropriate, along with separate test sets (or a single one) that are (is) large enough. More on different validation schemes and their properties can be found in, e.g., [3,12,17,37] and an insightful related discussion provided in [26].

MEAN-SQUARE ERROR LINEAR ESTIMATION

4

CONTENTS

4.1	Introduction	121
4.2	Mean-Square Error Linear Estimation: the Normal Equations	122
4.2.1	The Cost Function Surface	123
4.3	A Geometric Viewpoint: Orthogonality Condition	124
4.4	Extension to Complex-Valued Variables	127
4.4.1	Widely Linear Complex-Valued Estimation	129
	<i>Circularity Conditions</i>	130
4.4.2	Optimizing With Respect to Complex-Valued Variables: Wirtinger Calculus	132
4.5	Linear Filtering	134
4.6	MSE Linear Filtering: a Frequency Domain Point of View	136
	Deconvolution: Image Deblurring	137
4.7	Some Typical Applications	140
4.7.1	Interference Cancelation	140
4.7.2	System Identification	141
4.7.3	Deconvolution: Channel Equalization	143
4.8	Algorithmic Aspects: the Levinson and Lattice-Ladder Algorithms	149
	Forward and Backward MSE Optimal Predictors	151
4.8.1	The Lattice-Ladder Scheme	154
	<i>Orthogonality of the Optimal Backward Errors</i>	155
4.9	Mean-Square Error Estimation of Linear Models	158
4.9.1	The Gauss–Markov Theorem	160
4.9.2	Constrained Linear Estimation: the Beamforming Case	162
4.10	Time-Varying Statistics: Kalman Filtering	166
Problems	172	
MATLAB® Exercises	174	
References	176	

4.1 INTRODUCTION

Mean-square error (MSE) linear estimation is a topic of fundamental importance for parameter estimation in statistical learning. Besides historical reasons, which take us back to the pioneering works of Kolmogorov, Wiener, and Kalman, who laid the foundations of the optimal estimation field, understanding MSE estimation is a must, prior to studying more recent techniques. One always has to grasp the basics and learn the classics prior to getting involved with new “adventures.” Many of the concepts to be discussed in this chapter are also used in the next chapters.

Optimizing via a loss function that builds around the square of the error has a number of advantages, such as a single optimal value, which can be obtained via the solution of a linear set of equations; this is a very attractive feature in practice. Moreover, due to the relative simplicity of the resulting equations, the newcomer in the field can get a better feeling of the various notions associated with optimal parameter estimation. The elegant geometric interpretation of the MSE solution, via the orthogonality theorem, is presented and discussed. In the chapter, emphasis is also given to computational complexity issues while solving for the optimal solution. The essence behind these techniques remains exactly the same as that inspiring a number of computationally efficient schemes for online learning, to be discussed later in this book.

The development of the chapter is around real-valued variables, something that will be true for most of the book. However, complex-valued signals are particularly useful in a number of areas, with communications being a typical example, and the generalization from the real to the complex domain may not always be trivial. Although in most of the cases the difference lies in changing matrix transpositions by Hermitian ones, this is not the whole story. This is the reason that we chose to deal with complex-valued data in separate sections, whenever the differences from the real data are not trivial and some subtle issues are involved.

4.2 MEAN-SQUARE ERROR LINEAR ESTIMATION: THE NORMAL EQUATIONS

The general estimation task has been introduced in Chapter 3. There, it was stated that given two dependent random vectors, \mathbf{y} and \mathbf{x} , the goal of the estimation task is to obtain a function, g , so as, given a value \mathbf{x} of \mathbf{x} , to be able to predict (estimate), in some optimal sense, the corresponding value \mathbf{y} of \mathbf{y} , or $\hat{\mathbf{y}} = g(\mathbf{x})$. The MSE estimation was also presented in Chapter 3 and it was shown that the optimal MSE estimate of \mathbf{y} given the value $\mathbf{x} = \mathbf{x}$ is

$$\hat{\mathbf{y}} = \mathbb{E}[\mathbf{y}|\mathbf{x}].$$

In general, this is a nonlinear function. We now turn our attention to the case where g is *constrained* to be a linear function. For simplicity and in order to pay more attention to the concepts, we will restrict our discussion to the case of scalar dependent (output) variables. The more general case will be discussed later on.

Let $(\mathbf{y}, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$ be two jointly distributed random entities of *zero mean values*. In case the mean values are not zero, they are subtracted. Our goal is to obtain an estimate of $\boldsymbol{\theta} \in \mathbb{R}^l$ in the linear estimator model,

$$\hat{\mathbf{y}} = \boldsymbol{\theta}^T \mathbf{x}, \quad (4.1)$$

so that

$$J(\boldsymbol{\theta}) = \mathbb{E}[(\mathbf{y} - \hat{\mathbf{y}})^2], \quad (4.2)$$

is minimum, or

$$\boldsymbol{\theta}_* := \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \quad (4.3)$$

In other words, the optimal estimator is chosen so as to minimize the variance of the error random variable

$$e = y - \hat{y}. \quad (4.4)$$

Minimizing the cost function $J(\theta)$ is equivalent to setting its gradient with respect to θ equal to zero (see Appendix A),

$$\begin{aligned} \nabla J(\theta) &= \nabla \mathbb{E} \left[(y - \theta^T \mathbf{x}) (y - \mathbf{x}^T \theta) \right] \\ &= \nabla \left\{ \mathbb{E}[y^2] - 2\theta^T \mathbb{E}[\mathbf{x}y] + \theta^T \mathbb{E}[\mathbf{x}\mathbf{x}^T]\theta \right\} \\ &= -2\mathbf{p} + 2\Sigma_x \theta = \mathbf{0} \end{aligned}$$

or

$$\Sigma_x \theta_* = \mathbf{p} : \text{ normal equations,} \quad (4.5)$$

where the input–output cross-correlation vector \mathbf{p} is given by¹

$$\mathbf{p} = [\mathbb{E}[x_1 y], \dots, \mathbb{E}[x_l y]]^T = \mathbb{E}[\mathbf{x}y], \quad (4.6)$$

and the respective covariance matrix is given by

$$\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^T].$$

Thus, the weights of the optimal linear estimator are obtained via a linear system of equations, provided that the covariance matrix is *positive definite* and hence it can be inverted (Appendix A). Moreover, in this case, the solution is *unique*. On the contrary, if Σ_x is singular and hence cannot be inverted, there are infinitely many solutions (Problem 4.1).

4.2.1 THE COST FUNCTION SURFACE

Elaborating on the cost function $J(\theta)$, as defined in (4.2), we get

$$J(\theta) = \sigma_y^2 - 2\theta^T \mathbf{p} + \theta^T \Sigma_x \theta. \quad (4.7)$$

Adding and subtracting the term $\theta_*^T \Sigma_x \theta_*$ and taking into account the definition of θ_* from (4.5), it is readily seen that

$$J(\theta) = J(\theta_*) + (\theta - \theta_*)^T \Sigma_x (\theta - \theta_*), \quad (4.8)$$

where

$$J(\theta_*) = \sigma_y^2 - \mathbf{p}^T \Sigma_x^{-1} \mathbf{p} = \sigma_y^2 - \theta_*^T \Sigma_x \theta_* = \sigma_y^2 - \mathbf{p}^T \theta_*. \quad (4.9)$$

is the minimum achieved at the optimal solution. From (4.8) and (4.9), the following remarks can be made.

¹ The cross-correlation vector is often denoted as \mathbf{r}_{xy} . Here we will use \mathbf{p} , in order to simplify the notation.

Remarks 4.1.

- The cost at the optimal value θ_* is always less than the variance $\mathbb{E}[y^2]$ of the output variable. This is guaranteed by the positive definite nature of Σ_x or Σ_x^{-1} , which makes the second term on the right-hand side in (4.9) always positive, unless $p = \mathbf{0}$; however, the cross-correlation vector will only be zero if x and y are uncorrelated. Well, in this case, one cannot say anything (make any prediction) about y by observing samples of x , at least as far as the MSE criterion is concerned, which turns out to involve information residing up to the second-order statistics. In this case, the variance of the error, which coincides with $J(\theta_*)$, will be equal to the variance σ_y^2 ; the latter is a measure of the “intrinsic” uncertainty of y around its (zero) mean value. On the contrary, if the input–output variables are correlated, then observing x removes part of the uncertainty associated with y .
- For any value θ other than the optimal θ_* , the error variance increases as (4.8) suggests, due to the positive definite nature of Σ_x . Fig. 4.1 shows the cost function (MSE) surface defined by $J(\theta)$ in (4.8). The corresponding isovalue contours are shown in Fig. 4.2. In general, they are ellipses, whose axes are determined by the eigenstructure of Σ_x . For $\Sigma_x = \sigma^2 I$, where all eigenvalues are equal to σ^2 , the contours are circles (Problem 4.3).

4.3 A GEOMETRIC VIEWPOINT: ORTHOGONALITY CONDITION

A very intuitive view of what we have said so far comes from the geometric interpretation of the random variables. The reader can easily check out that the set of random variables is a *linear space* over the field of real (and complex) numbers. Indeed, if x and y are any two random variables then $x + y$, as well as αx , are also random variables for every $\alpha \in \mathbb{R}$.² We can now equip this linear space with an inner product operation, which also implies a norm and makes it an *inner product space*. The

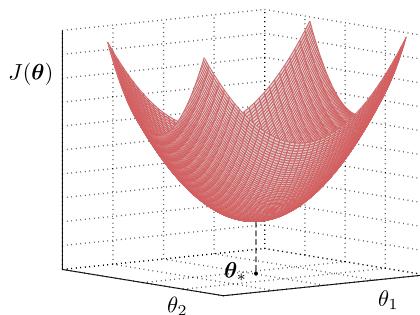
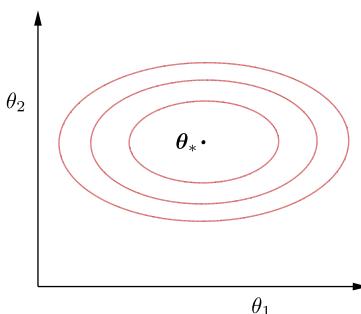


FIGURE 4.1

The MSE cost function has the form of a (hyper)paraboloid.

² These operations also satisfy all the properties required for a set to be a linear space, including associativity, commutativity, and so on (see [47] and the appendix associated with Chapter 8).

**FIGURE 4.2**

The isovalue contours for the cost function surface corresponding to Fig. 4.1. They are ellipses; the major and the minor axes of each ellipse are determined by the maximum and minimum eigenvalues, λ_{\max} and λ_{\min} , of the covariance matrix, Σ , of the input random variables. The largest the ratio $\frac{\lambda_{\max}}{\lambda_{\min}}$ is, the more elongated the ellipses are. The ellipses become circles if the covariance matrix has the special form of $\sigma^2 I$. That is, all variables are mutually uncorrelated and they have the same variance. By varying Σ , different shapes of the ellipses and different orientations result.

reader can easily check that the mean value operation has all the properties required for an operation to be called an inner product. Indeed, for any subset of random variables,

- $\mathbb{E}[xy] = \mathbb{E}[yx]$,
- $\mathbb{E}[(\alpha_1 x_1 + \alpha_2 x_2)y] = \alpha_1 \mathbb{E}[x_1 y] + \alpha_2 \mathbb{E}[x_2 y]$,
- $\mathbb{E}[x^2] \geq 0$, with equality if and only if $x = 0$.

Thus, the norm induced by this inner product,

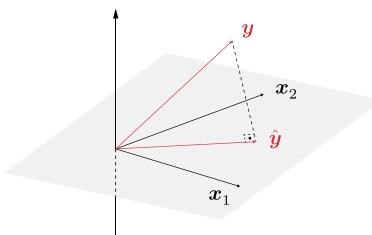
$$\|x\| := \sqrt{\mathbb{E}[x^2]},$$

coincides with the respective standard deviation (assuming $\mathbb{E}[x] = 0$). From now on, given two uncorrelated random variables x, y , or $\mathbb{E}[xy] = 0$, we can call them *orthogonal*, because their inner product is zero. We are now free to apply to our task of interest the orthogonality theorem, which is known to us from our familiar finite-dimensional (Euclidean) linear (vector) spaces.

Let us now rewrite (4.1) as

$$\hat{y} = \theta_1 x_1 + \cdots + \theta_l x_l.$$

Thus, the random variable, \hat{y} , which is now interpreted as a point in a vector space, results as a linear combination of l elements in this space. Thus, the variable \hat{y} will necessarily lie in the subspace spanned by these points. In contrast, the true variable, y , will not lie, in general, in this subspace. Because our goal is to obtain a \hat{y} that is a good approximation of y , we have to seek the specific linear combination that makes the norm of the error, $e = y - \hat{y}$, minimum. This specific linear combination corresponds to the *orthogonal* projection of y onto the subspace spanned by the points x_1, x_2, \dots, x_l . This is equivalent

**FIGURE 4.3**

Projecting y on the subspace spanned by x_1, x_2 (shaded plane) guarantees that the deviation between y and \hat{y} corresponds to the minimum MSE.

to requiring

$$\mathbb{E}[ex_k] = 0, \quad k = 1, \dots, l : \quad \text{orthogonality condition.} \quad (4.10)$$

The error variable being orthogonal to every point x_k , $k = 1, 2, \dots, l$, will be orthogonal to the respective subspace. This is illustrated in Fig. 4.3. Such a choice guarantees that the resulting error will have the minimum norm; by the definition of the norm, this corresponds to the minimum MSE, i.e., to the minimum $\mathbb{E}[e^2]$.

The set of equations in (4.10) can now be written as

$$\mathbb{E}\left[\left(y - \sum_{i=1}^l \theta_i x_i\right)x_k\right] = 0, \quad k = 1, 2, \dots, l,$$

or

$$\sum_{i=1}^l \mathbb{E}[x_i x_k] \theta_i = \mathbb{E}[x_k y], \quad k = 1, 2, \dots, l, \quad (4.11)$$

which leads to the linear set of equations in (4.5).

This is the reason that this elegant set of equations is known as *normal equations*. Another name is *Wiener–Hopf equations*. Strictly speaking, the Wiener–Hopf equations were first derived for continuous-time processes in the context of the causal estimation task [49,50]; for a discussion see [16,44].

Nobert Wiener was a mathematician and philosopher. He was awarded a PhD at Harvard at the age of 17 in mathematical logic. During the Second World War, he laid the foundations of linear estimation theory in a classified work, independently of Kolmogorov. Later on, Wiener was involved in pioneering work embracing automation, artificial intelligence, and cognitive science. Being a pacifist, he was regarded with suspicion during the Cold War years.

The other pillar on which linear estimation theory is based is the pioneering work of Andrey Nikolaevich Kolmogorov (1903–1987) [24], who developed his theory independent of Wiener. Kolmogorov's contributions cover a wide range of topics in mathematics, including probability, computa-

tional complexity, and topology. He is the father of the modern axiomatic foundation of the notion of probability (see Chapter 2).

Remarks 4.2.

- So far, in our theoretical findings, we have assumed that \mathbf{x} and y are jointly distributed (correlated) variables. If, in addition, we assume that they are linearly related according to the linear regression model,

$$y = \boldsymbol{\theta}_o^T \mathbf{x} + \eta, \quad \boldsymbol{\theta}_o \in \mathbb{R}^k, \quad (4.12)$$

where η is a zero mean noise variable independent of \mathbf{x} , then, if the dimension k of the true system $\boldsymbol{\theta}_o$ is equal to the number of parameters, l , adopted for the model, so that $k = l$, then it turns out that (Problem 4.4)

$$\boldsymbol{\theta}_* = \boldsymbol{\theta}_o,$$

and the optimal MSE is equal to the variance of the noise, σ_η^2 .

- *Undermodeling.* If $k > l$, then the order of the model is less than that of the true system, which relates y and \mathbf{x} in (4.12); this is known as *undermodeling*. It is easy to show that if the variables comprising \mathbf{x} are uncorrelated, then (Problem 4.5)

$$\boldsymbol{\theta}_* = \boldsymbol{\theta}_o^1,$$

where

$$\boldsymbol{\theta}_o := \begin{bmatrix} \boldsymbol{\theta}_o^1 \\ \boldsymbol{\theta}_o^2 \end{bmatrix}, \quad \boldsymbol{\theta}_o^1 \in \mathbb{R}^l, \quad \boldsymbol{\theta}_o^2 \in \mathbb{R}^{k-l}.$$

In other words, the MSE optimal estimator identifies the first l components of $\boldsymbol{\theta}_o$.

4.4 EXTENSION TO COMPLEX-VALUED VARIABLES

Everything that has been said so far can be extended to complex-valued signals. However, there are a few subtle points involved and this is the reason that we chose to treat this case separately. Complex-valued variables are very common in a number of applications, as for example in communications, e.g., [41].

Given two real-valued variables, (x, y) , one can either consider them as a vector quantity in the two-dimensional space, $[x, y]^T$, or describe them as a complex variable, $z = x + jy$, where $j^2 := -1$. Adopting the latter approach offers the luxury of exploiting the operations available in the field \mathbb{C} of complex numbers, i.e., multiplication and division. The existence of such operations greatly facilitates the algebraic manipulations. Recall that such operations are not defined in vector spaces.³

³ Multiplication and division can also be defined for groups of four variables, (x, ϕ, z, y) , known as quaternions; the related algebra was introduced by Hamilton in 1843. The real and complex numbers as well as quaternions are all special cases of the so-called Clifford algebras [39].

Thus, treating θ as a constant, the optimal occurs at

$$\nabla_{\theta^*} J = \mathbb{E}[\mathbf{x}\mathbf{e}^*] = \mathbf{0},$$

which is the orthogonality condition leading to the normal equations (4.16)–(4.18).

Application in widely linear estimation. The cost function is now (see notation in (4.35))

$$J(\boldsymbol{\varphi}, \boldsymbol{\varphi}^*) = \mathbb{E}[(\mathbf{y} - \boldsymbol{\varphi}^H \tilde{\mathbf{x}})(\mathbf{y}^* - \boldsymbol{\varphi}^T \tilde{\mathbf{x}}^*)],$$

and treating $\boldsymbol{\varphi}$ as a constant,

$$\nabla_{\boldsymbol{\varphi}^*} J = \mathbb{E}[\tilde{\mathbf{x}}\mathbf{e}^*] = \mathbb{E}\begin{bmatrix} \mathbf{x}\mathbf{e}^* \\ \mathbf{x}^*\mathbf{e}^* \end{bmatrix} = \mathbf{0},$$

which leads to the set derived in (4.36).

Wirtinger's calculus will prove very useful in subsequent chapters for deriving gradient operations in the context of online/adaptive estimation in Euclidean as well as in reproducing kernel Hilbert spaces.

4.5 LINEAR FILTERING

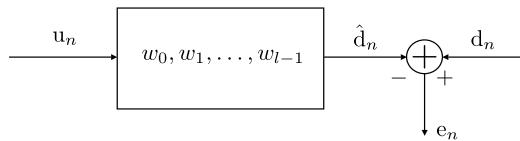
Linear statistical filtering is an instance of the general estimation task, when the notion of time evolution needs to be taken into consideration and estimates are obtained at each time instant. There are three major types of problems that emerge:

- *Filtering*, where the estimate at time instant n is based on all previously received (measured) input information *up to and including* the current time index, n .
- *Smoothing*, where data over a time interval $[0, N]$ are first collected and an estimate is obtained at each time instant $n \leq N$, using *all* the available information in the interval $[0, N]$.
- *Prediction*, where estimates at times $n + \tau$, $\tau > 0$, are to be obtained based on the information up to and including time instant n .

To fit in the above definitions more with what has been said so far in the chapter, take for example a time-varying case, where the output variable at time instant n is y_n and its value depends on observations included in the corresponding input vector \mathbf{x}_n . In filtering, the latter can include measurements received only at time instants $n, n-1, \dots, 0$. This restriction in the index set is directly related to *causality*. In contrast, in smoothing, future time instants are included in addition to the past, i.e., $\dots, n+2, n+1, n, n-1, \dots$

Most of the effort in this book will be spent on filtering whenever time information enters into the picture. The reason is that this is the most commonly encountered task and, also, the techniques used for smoothing and prediction are similar in nature to that of filtering, with usually minor modifications.

In signal processing, the term filtering is usually used in a more specific context, and it refers to the operation of a *filter*, which acts on an input random process/signal (\mathbf{u}_n), to transform it into another one (\mathbf{d}_n); see Section 2.4.3. Note that we have switched into the notation, introduced in Chapter 2, used to denote random processes. We prefer to keep different notation for processes and random variables,

**FIGURE 4.5**

In statistical filtering, the impulse response coefficients are estimated so as to minimize the error between the output and the desired response processes. In MSE linear filtering, the cost function is $\mathbb{E}[e_n^2]$.

because in the case of random processes, the filtering task obtains a special structure and properties, as we will soon see. Moreover, although the mathematical formulation of the involved equations, for both cases, may end up to be the same, we feel that it is good for the reader to keep in mind that there is a different underlying mechanism for generating the data.

The task in statistical linear filtering is to compute the coefficients (impulse response) of the filter so that the output process of the filter, \hat{d}_n , when the filter is excited by the input random process, u_n , is as close as possible to a *desired* response process, d_n . In other words, the goal is to minimize, in some sense, the corresponding error process (see Fig. 4.5). Assuming that the unknown filter is of a finite impulse response (FIR) (see Section 2.4.3 for related definitions), denoted as w_0, w_1, \dots, w_{l-1} , the output \hat{d}_n of the filter is given as

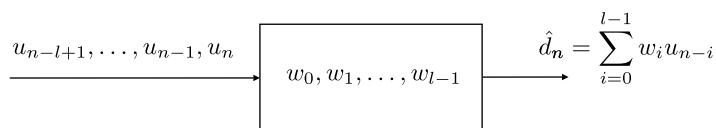
$$\hat{d}_n = \sum_{i=0}^{l-1} w_i u_{n-i} = \mathbf{w}^T \mathbf{u}_n : \text{ convolution sum,} \quad (4.41)$$

where

$$\mathbf{w} = [w_0, w_1, \dots, w_{l-1}]^T \quad \text{and} \quad \mathbf{u}_n = [u_n, u_{n-1}, \dots, u_{n-l+1}]^T. \quad (4.42)$$

Fig. 4.6 illustrates the convolution operation of the linear filter, when its input is excited by a realization u_n of the input processes to provide in the output the signal/sequence \hat{d}_n .

Alternatively, (4.41) can be viewed as a *linear estimator* function; given the jointly distributed variables, at time instant n , (d_n, \mathbf{u}_n) , (4.41) provides the estimator, \hat{d}_n , given the values of \mathbf{u}_n . In order to obtain the coefficients, \mathbf{w} , the MSE criterion will be adopted. Furthermore, we will assume that:

**FIGURE 4.6**

The linear filter is excited by a realization of an input process. The output signal is the convolution between the input sequence and the filter's impulse response.

- The processes $\mathbf{u}_n, \mathbf{d}_n$ are *wide-sense stationary* real random processes.
- Their mean values are equal to zero, in other words, $\mathbb{E}[\mathbf{u}_n] = \mathbb{E}[\mathbf{d}_n] = 0, \forall n$. If this is not the case, we can subtract the respective mean values from the processes, \mathbf{u}_n and \mathbf{d}_n , during a preprocessing stage. Due to this assumption, the autocorrelation and covariance matrices of \mathbf{u}_n coincide, so that

$$\mathbf{R}_u = \Sigma_u.$$

The normal equations in (4.5) now take the form

$$\Sigma_u \mathbf{w} = \mathbf{p},$$

where

$$\mathbf{p} = [\mathbb{E}[\mathbf{u}_n \mathbf{d}_n], \dots, \mathbb{E}[\mathbf{u}_{n-l+1} \mathbf{d}_n]]^T,$$

and the respective covariance/autocorrelation matrix, of order l , of the input process is given by

$$\Sigma_u := \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^T] = \begin{bmatrix} r(0) & r(1) & \dots & r(l-1) \\ r(1) & r(0) & \dots & r(l-2) \\ \vdots & & & \ddots \\ r(l-1) & r(l-2) & \dots & r(0) \end{bmatrix}, \quad (4.43)$$

where $r(k)$ is the autocorrelation sequence of the input process. Because we have assumed that the involved processes are wide-sense stationary, we have

$$r(n, n-k) := \mathbb{E}[\mathbf{u}_n \mathbf{u}_{n-k}] = r(k).$$

Also, recall that, for real wide-sense stationary processes, the autocorrelation sequence is symmetric, or $r(k) = r(-k)$ (Section 2.4.3). Observe that in this case, where the input vector results from a random process, the covariance matrix has a special structure, which will be exploited later on to derive efficient schemes for the solution of the normal equations.

For the complex linear filtering case, the only differences are:

- the output is given as $\hat{\mathbf{d}}_n = \mathbf{w}^H \mathbf{u}_n$,
- $\mathbf{p} = \mathbb{E}[\mathbf{u}_n \mathbf{d}_n^*]$,
- $\Sigma_u = \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^H]$,
- $r(-k) = r^*(k)$.

4.6 MSE LINEAR FILTERING: A FREQUENCY DOMAIN POINT OF VIEW

Let us now turn our attention to the more general case, and assume that our filter is of *infinite impulse response* (IIR). Then (4.41) becomes

$$\hat{\mathbf{d}}_n = \sum_{i=-\infty}^{+\infty} w_i \mathbf{u}_{n-i}. \quad (4.44)$$

Moreover, we have allowed the filter to be *noncausal*.⁴ Following similar arguments as those used to prove the MSE optimality of $\mathbb{E}[y|\mathbf{x}]$ in Section 3.9.1, it turns out that the optimal filter coefficients must satisfy the following condition (Problem 4.12):

$$\mathbb{E} \left[\left(d_n - \sum_{i=-\infty}^{+\infty} w_i u_{n-i} \right) u_{n-j} \right] = 0, \quad j \in \mathbb{Z}. \quad (4.45)$$

Observe that this is a generalization (involving an infinite number of terms) of the orthogonality condition stated in (4.10). A rearrangement of the terms in (4.45) results in

$$\sum_{i=-\infty}^{+\infty} w_i \mathbb{E}[u_{n-i} u_{n-j}] = \mathbb{E}[d_n u_{n-j}], \quad j \in \mathbb{Z}, \quad (4.46)$$

and finally to

$$\sum_{i=-\infty}^{+\infty} w_i r(j-i) = r_{du}(j), \quad j \in \mathbb{Z}, \quad (4.47)$$

where $r_{du}(j)$ denotes the cross-correlation sequence between the processes d_n and u_n . Eq. (4.47) can be considered as the generalization of (4.5) to the case of random processes. The problem now is how one can solve (4.47) that involves infinite many parameters. The way out is to cross into the frequency domain. Eq. (4.47) can be seen as the convolution of the unknown sequence with the autocorrelation sequence of the input process, which gives rise to the cross-correlation sequence. However, we know that convolution of two sequences corresponds to the product of the respective Fourier transforms (e.g., [42] and Section 2.4.2). Thus, we can now write that

$$W(\omega) S_u(\omega) = S_{du}(\omega), \quad (4.48)$$

where $W(\omega)$ is the Fourier transform of the sequence of the unknown parameters and $S_u(\omega)$ is the *power spectral density* of the input process, defined in Section 2.4.3. In analogy, the Fourier transform $S_{du}(\omega)$ of the cross-correlation sequence is known as the *cross-spectral density*. If the latter two quantities are available, then once $W(\omega)$ has been computed, the unknown parameters can be obtained via the inverse Fourier transform.

DECONVOLUTION: IMAGE DEBLURRING

We will now consider an important application in order to demonstrate the power of MSE linear estimation. Image deblurring is a typical *deconvolution* task. An image is degraded due to its transmission via a nonideal system; the task of deconvolution is to optimally recover (in the MSE sense in our case)

⁴ A system is called *causal* if the output \hat{d}_n depends *only* on input values u_m , $m \leq n$. A necessary and sufficient condition for causality is that the impulse response is zero for negative time instants, meaning that $w_n = 0$, $n < 0$. This can easily be checked out; try it.

**FIGURE 4.7**

(A) The original image and (B) its blurred and noisy version.

the original undegraded image. Fig. 4.7A shows the original image and 4.7B a blurred version (e.g., taken by a nonsteady camera) with some small additive noise.

At this point, it is interesting to recall that deconvolution is a process that our human brain performs all the time. The human (and not only) vision system is one of the most complex and highly developed biological systems that has been formed over millions years of a continuous evolution process. Any raw image that falls on the retina of the eye is *severely blurred*. Thus, one of the main early processing activities of our visual system is to deblur it (see, e.g., [29] and the references therein for a related discussion).

Before we proceed any further, the following assumptions are adopted:

- The image is a *wide-sense stationary* two-dimensional random process. Two-dimensional random processes are also known as *random fields* (see Chapter 15).
- The image is of an infinite extent; this can be justified for the case of large images. This assumption will grant us the “permission” to use (4.48). The fact that an image is a two-dimensional process does not change anything in the theoretical analysis; the only difference is that now the Fourier transforms involve two frequency variables, ω_1, ω_2 , one for each of the two dimensions.

A gray image is represented as a two-dimensional array. To stay close to the notation used so far, let $d(n, m)$, $n, m \in \mathbb{Z}$, be the original undegraded image (which for us is now the desired response), and $u(n, m)$, $n, m \in \mathbb{Z}$, be the degraded one, obtained as

$$u(n, m) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} h(i, j)d(n - i, m - j) + \eta(n, m), \quad (4.49)$$

where $\eta(n, m)$ is the realization of a noise field, which is assumed to be zero mean and independent of the input (undegraded) image. The sequence $h(i, j)$ is the *point spread sequence* (impulse

response) of the system (e.g., camera). We will assume that this is known and it has, somehow, been measured.⁵

Our task now is to estimate a two-dimensional filter, $w(n, m)$, which is applied to the degraded image to optimally reconstruct (in the MSE sense) the original undegraded image. In the current context, Eq. (4.48) is written as

$$W(\omega_1, \omega_2)S_u(\omega_1, \omega_2) = S_{du}(\omega_1, \omega_2).$$

Following similar arguments as those used to derive Eq. (2.130) of Chapter 2, it is shown that (Problem 4.13)

$$S_{du}(\omega_1, \omega_2) = H^*(\omega_1, \omega_2)S_d(\omega_1, \omega_2) \quad (4.50)$$

and

$$S_u(\omega_1, \omega_2) = |H(\omega_1, \omega_2)|^2 S_d(\omega_1, \omega_2) + S_\eta(\omega_1, \omega_2), \quad (4.51)$$

where “*” denotes complex conjugation and S_η is the power spectral density of the noise field. Thus, we finally obtain

$$W(\omega_1, \omega_2) = \frac{1}{H(\omega_1, \omega_2)} \frac{|H(\omega_1, \omega_2)|^2}{|H(\omega_1, \omega_2)|^2 + \frac{S_\eta(\omega_1, \omega_2)}{S_d(\omega_1, \omega_2)}}.$$

(4.52)

Once $W(\omega_1, \omega_2)$ has been computed, the unknown parameters could be obtained via an inverse (two-dimensional) Fourier transform. The deblurred image then results as

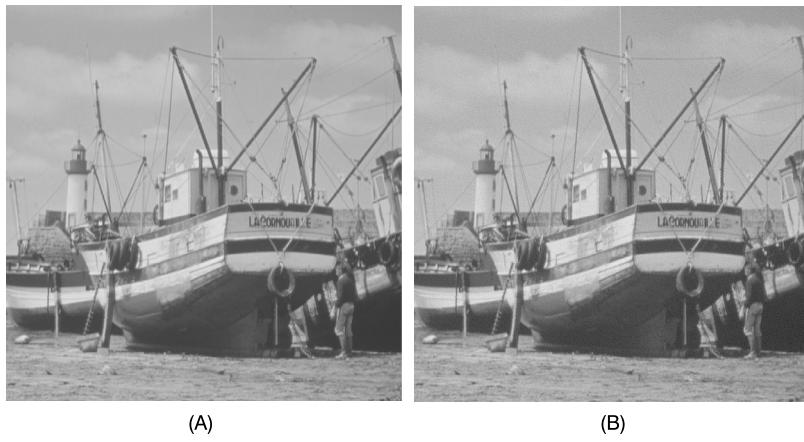


FIGURE 4.8

(A) The original image and (B) the deblurred one for $C = 2.3 \times 10^{-6}$. Observe that in spite of the simplicity of the method, the reconstruction is pretty good. The differences become more obvious to the eye when the images are enlarged.

⁵ Note that this is not always the case.

$$\hat{d}(n, m) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} w(i, j) u(n-i, m-j). \quad (4.53)$$

In practice, because we are not really interested in obtaining the weights of the deconvolution filter, we implement (4.53) in the frequency domain

$$\hat{D}(\omega_1, \omega_2) = W(\omega_1, \omega_2) U(\omega_1, \omega_2),$$

and then obtain the inverse Fourier transform. Thus all processing is efficiently performed in the frequency domain. Software packages to perform Fourier transforms (via the fast Fourier transform [FFT]) of an image array are “omnipresent” on the internet.

Another important issue is that in practice we do not know $S_d(\omega_1, \omega_2)$. An approximation which is usually adopted and renders sensible results can be made by assuming that $\frac{S_\eta(\omega_1, \omega_2)}{S_d(\omega_1, \omega_2)}$ is a constant, C , and trying different values of it. Fig. 4.8 shows the deblurred image for $C = 2.3 \times 10^{-6}$. The quality of the end result depends a lot on the choice of this value (MATLAB® exercise 4.25). Other, more advanced, techniques have also been proposed. For example, one can get a better estimate of $S_d(\omega_1, \omega_2)$ by using information from $S_\eta(\omega_1, \omega_2)$ and $S_u(\omega_1, \omega_2)$. The interested reader can obtain more on the image deconvolution/restoration task from, e.g., [14,34].

4.7 SOME TYPICAL APPLICATIONS

Optimal linear estimation/filtering has been applied in a wide range of diverse applications of statistical learning, such as regression modeling, communications, control, biomedical signal processing, seismic signal processing, and image processing. In the sequel, we present some typical applications in order for the reader to grasp the main rationale of how the previously stated theory can find its way in solving practical problems. In all cases, wide-sense stationarity of the involved random processes is assumed.

4.7.1 INTERFERENCE CANCELATION

In interference cancelation, we have access to a mixture of two signals expressed as $d_n = y_n + s_n$. Ideally, we would like to remove one of them, say, y_n . We will consider them as realizations of respective

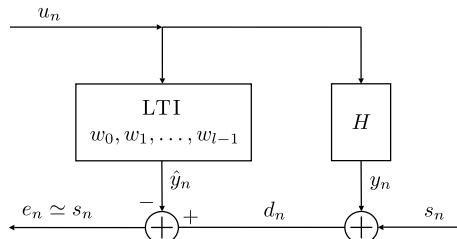


FIGURE 4.9

A basic block diagram illustrating the interference cancelation task.

random processes/signals, or d_n , y_n , and s_n . To achieve this goal, the only available information is another signal, say, u_n , that is statistically related to the unwanted signal, y_n . For example, y_n may be a filtered version of u_n . This is illustrated in Fig. 4.9, where the corresponding realizations of the involved random processes are shown.

Process y_n is the output of an unknown system H , whose input is excited by u_n . The task is to model H by obtaining estimates of its impulse response (assuming that it is linear time-invariant and of known order). Then the output of the model will be an approximation of y_n when this is activated by the same input, u_n . We will use d_n as the desired response process. The optimal estimates of w_0, \dots, w_{l-1} (assuming the order of the unknown system H to be l) are provided by the normal equations

$$\Sigma_u \mathbf{w}_* = \mathbf{p}.$$

However,

$$\begin{aligned} \mathbf{p} &= \mathbb{E}[\mathbf{u}_n d_n] = \mathbb{E}[\mathbf{u}_n (y_n + s_n)] \\ &= \mathbb{E}[\mathbf{u}_n y_n], \end{aligned} \quad (4.54)$$

because the respective input vector \mathbf{u}_n and s_n are considered statistically independent. That is, the previous formulation of the problem leads to the same normal equations as when the desired response was the signal y_n , which we want to remove! Hence, the output of our model will be an approximation (in the MSE sense), \hat{y}_n , of y_n , and if subtracted from d_n the resulting (error) signal e_n will be an approximation to s_n . How good this approximation is depends on whether l is a good “estimate” of the true order of H . The cross-correlation in the right-hand side of (4.54) can be approximated by computing the respective sample mean values, in particular over periods where s_n is absent. In practical systems, online/adaptive versions of this implementation are usually employed, as we will see in Chapter 5.

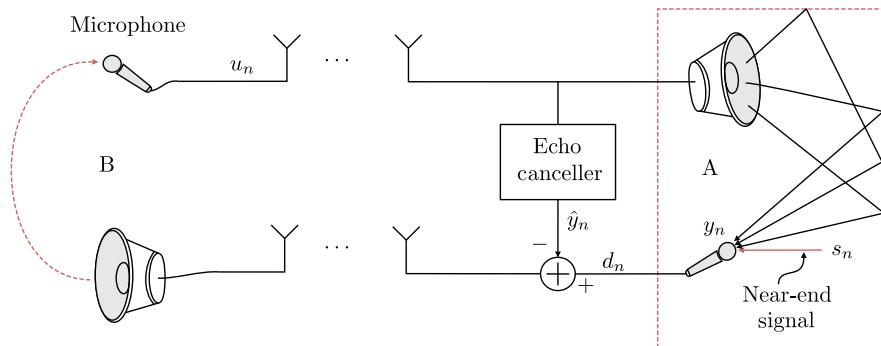
Interference cancelation schemes have been widely used in many systems such as noise cancelation, echo cancelation in telephone networks, and video conferencing, and in biomedical applications, for example, in order to cancel the maternal interference in a fetal electrocardiograph.

Fig. 4.10 illustrates the echo cancelation task in a video conference application. The same setup applies to the hands-free telephone service in a car. The *far-end* speech signal is considered to be a realization u_n of a random process u_n ; through the loudspeakers, it is broadcasted in room A (car) and it is reflected in the interior of the room. Part of it is absorbed and part of it enters the microphone; this is denoted as y_n . The equivalent response of the room (reflections) on u_n can be represented by a filter, H , as in Fig. 4.9. Signal y_n returns back and the speaker in location B listens to her or his own voice, together with the *near-end* speech signal, s_n , of the speaker in A. In certain cases, this feedback path from the loudspeakers to the microphone can cause instabilities, giving rise to a “howling” sound effect. The goal of the echo canceler is to optimally remove y_n .

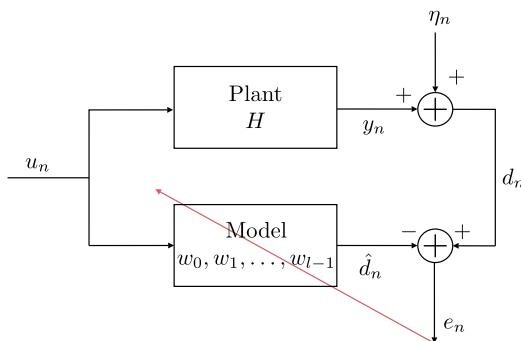
4.7.2 SYSTEM IDENTIFICATION

System identification is similar in nature to the interference cancelation task. Note that in Fig. 4.9, one basically models the unknown system. However, the focus there was on replicating the output y_n and not on the system’s impulse response.

In system identification, the aim is to model the impulse response of an unknown plant. To this end, we have access to its input signal as well as to a *noisy* version of its output. The task is to design a

**FIGURE 4.10**

The echo canceler is optimally designed to remove the part of the far-end signal, u_n , that interferes with the near-end signal, s_n .

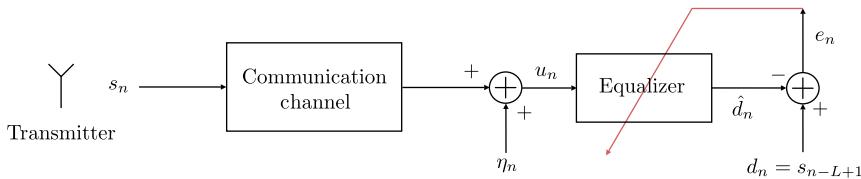
**FIGURE 4.11**

In system identification, the impulse response of the model is optimally estimated so that the output is close, in the MSE sense, to that of the unknown plant. The red line indicates that the error is used for the optimal estimation of the unknown parameters of the filter.

model whose impulse response approximates that of the unknown plant. To achieve this, we optimally design a linear filter whose input is the same signal as the one that activates the plant and its desired response is the noisy output of the plant (see Fig. 4.11). The associated normal equations are

$$\Sigma_u \mathbf{w}_* = \mathbb{E}[\mathbf{u}_n \mathbf{d}_n] = \mathbb{E}[\mathbf{u}_n y_n] + 0,$$

assuming the noise η_n is statistically independent of \mathbf{u}_n . Thus, once more, the resulting normal equations are as if we had provided the model with a desired response equal to the noiseless output of the unknown plant, expressed as $\mathbf{d}_n = y_n$. Hence, the impulse response of the model is estimated so that its output is close, in the MSE sense, to the true (noiseless) output of the unknown plant. System identification is of major importance in a number of applications. In control, it is used for driving the as-

**FIGURE 4.12**

The task of an equalizer is to optimally recover the originally transmitted information sequence s_n , delayed by L time lags.

sociated controllers. In data communications, it is used for estimating the transmission channel in order to build up maximum likelihood estimators of the transmitted data. In many practical systems, adaptive versions of the system identification scheme are implemented, as we will discuss in following chapters.

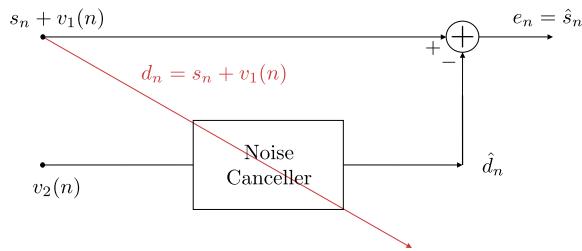
4.7.3 DECONVOLUTION: CHANNEL EQUALIZATION

Note that in the cancelation task the goal was to “remove” the (filtered version of the) input signal (u_n) to the unknown system H . In system identification, the focus was on the (unknown) system itself. In *deconvolution*, the emphasis is on the input of the unknown system. That is, our goal now is to recover, in the MSE optimal sense, a (delayed) input signal, $d_n = s_{n-L+1}$, where L is the delay in units of the sampling period T . The task is also called *inverse system identification*. The term *equalization* or *channel equalization* is used in communications. The deconvolution task was introduced in the context of image deblurring in Section 4.6. There, the required information about the *unknown* input process was obtained via an approximation. In the current framework, this can be approached via the transmission of a training sequence.

The goal of an *equalizer* is to recover the transmitted information symbols, by mitigating the so-called *intersymbol interference* (ISI) that any (imperfect) dispersive communication channel imposes on the transmitted signal; besides ISI, additive noise is also present in the transmitted information bits (see Example 4.2). Equalizers are “omnipresent” in these days; in our mobile phones, in our modems, etc. Fig. 4.12 presents the basic scheme for an equalizer. The equalizer is trained so that its output is as close as possible to the transmitted data bits delayed by some time lag L ; the delay is used in order to account for the overall delay imposed by the channel equalizer system. Deconvolution/channel equalization is at the heart of a number of applications besides communications, such as acoustics, optics, seismic signal processing, and control. The channel equalization task will also be discussed in the next chapter in the context of online learning via the decision feedback equalization mode of operation.

Example 4.1 (Noise cancelation). The noise cancelation application is illustrated in Fig. 4.13. The signal of interest is a realization of a process s_n , which is contaminated by the noise process $v_1(n)$. For example, s_n may be the speech signal of the pilot in the cockpit and $v_1(n)$ the aircraft noise at the location of the microphone. We assume that $v_1(n)$ is an AR process of order one, expressed as

$$v_1(n) = a_1 v_1(n - 1) + \eta_n.$$

**FIGURE 4.13**

A block diagram for a noise canceller. The signals shown are realizations of the corresponding random variables that are used in the text. Using as desired response the contaminated signal, the output of the optimal filter is an estimate of the noise component.

The random signal $v_2(n)$ is a noise sequence⁶ which is related to $v_1(n)$, but it is statistically independent of s_n . For example, it may be the noise picked up from another microphone positioned at a nearby location. This is also assumed to be an AR process of the first order,

$$v_2(n) = a_2 v_2(n-1) + \eta_n.$$

Note that both $v_1(n)$ and $v_2(n)$ are generated by the same noise source, η_n , which is assumed to be white of variance σ_η^2 . For example, in an aircraft it can be assumed that the noise at different points is due to a “common” source, especially for nearby locations.

The goal of the example is to compute estimates of the weights of the noise canceler, in order to optimally remove (in the MSE sense) the noise $v_1(n)$ from the mixture $s_n + v_1(n)$. Assume the canceler to be of order two.

The input to the canceler is $v_2(n)$ and as desired response the mixture signal, $d_n = s_n + v_1(n)$, will be used. To establish the normal equations, we need to compute the covariance matrix, Σ_2 , of $v_2(n)$ and the cross-correlation vector, p_2 , between the input random vector, $v_2(n)$, and d_n .

Because $v_2(n)$ is an AR process of the first order, recall from Section 2.4.4 that the autocorrelation sequence is given by

$$r_2(k) = \frac{a_2^k \sigma_\eta^2}{1 - a_2^2}, \quad k = 0, 1, \dots \quad (4.55)$$

Hence,

$$\Sigma_2 = \begin{bmatrix} r_2(0) & r_2(1) \\ r_2(1) & r_2(0) \end{bmatrix} = \begin{bmatrix} \frac{\sigma_\eta^2}{1 - a_2^2} & \frac{a_2 \sigma_\eta^2}{1 - a_2^2} \\ \frac{a_2 \sigma_\eta^2}{1 - a_2^2} & \frac{\sigma_\eta^2}{1 - a_2^2} \end{bmatrix}.$$

⁶ We use the index n in parenthesis to unclutter notation due to the presence of a second subscript.

Next, we are going to compute the cross-correlation vector. We have

$$\begin{aligned} p_2(0) &:= \mathbb{E}[v_2(n)d_n] = \mathbb{E}[v_2(n)(s_n + v_1(n))] \\ &= \mathbb{E}[v_2(n)v_1(n)] + 0 = \mathbb{E}[(a_2 v_2(n-1) + \eta_n)(a_1 v_1(n-1) + \eta_n)] \\ &= a_2 a_1 p_2(0) + \sigma_\eta^2, \end{aligned}$$

or

$$p_2(0) = \frac{\sigma_\eta^2}{1 - a_2 a_1}. \quad (4.56)$$

We used the fact that $\mathbb{E}[v_2(n-1)\eta_n] = \mathbb{E}[v_1(n-1)\eta_n] = 0$, because $v_2(n-1)$ and $v_1(n-1)$ depend recursively on previous values, i.e., $\eta(n-1), \eta(n-2), \dots$, and also that η_n is a white noise sequence, hence the respective correlation values are zero. Also, due to stationarity, $\mathbb{E}[v_2(n)v_1(n)] = \mathbb{E}[v_2(n-1)v_1(n-1)]$.

For the other value of the cross-correlation vector we have

$$\begin{aligned} p_2(1) &= \mathbb{E}[v_2(n-1)d_n] = \mathbb{E}[v_2(n-1)(s_n + v_1(n))] \\ &= \mathbb{E}[v_2(n-1)v_1(n)] + 0 = \mathbb{E}[v_2(n-1)(a_1 v_1(n-1) + \eta_n)] \\ &= a_1 p_2(0) = \frac{a_1 \sigma_\eta^2}{1 - a_1 a_2}. \end{aligned}$$

In general, it is easy to show that

$$p_2(k) = \frac{a_1^k \sigma_\eta^2}{1 - a_2 a_1}, \quad k = 0, 1, \dots \quad (4.57)$$

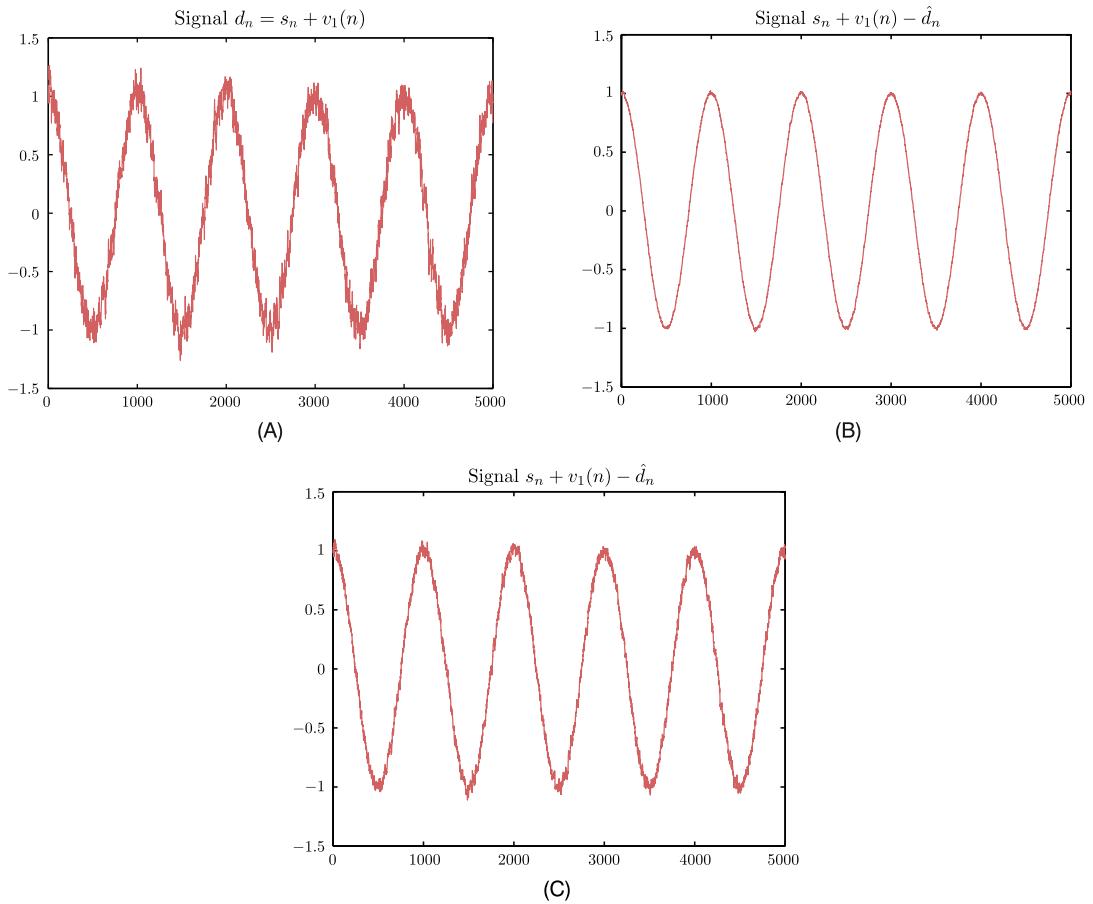
Recall that because the processes are real-valued, the covariance matrix is symmetric, meaning $r_2(k) = r_2(-k)$. Also, for (4.55) to make sense (recall that $r_2(0) > 0$), $|a_2| < 1$. The same holds true for $|a_1|$, following similar arguments for the autocorrelation process of $v_1(n)$.

Thus, the optimal weights of the noise canceler are given by the following set of normal equations:

$$\begin{bmatrix} \frac{\sigma_\eta^2}{1 - a_2^2} & \frac{a_2 \sigma_\eta^2}{1 - a_2^2} \\ \frac{a_2 \sigma_\eta^2}{1 - a_2^2} & \frac{\sigma_\eta^2}{1 - a_2^2} \end{bmatrix} \mathbf{w} = \begin{bmatrix} \frac{\sigma_\eta^2}{1 - a_1 a_2} \\ \frac{a_1 \sigma_\eta^2}{1 - a_1 a_2} \end{bmatrix}.$$

Note that the canceler optimally “removes” from the mixture, $s_n + v_1(n)$, the component that is correlated to the input, $v_2(n)$; observe that $v_1(n)$ basically acts as the desired response.

Fig. 4.14A shows a realization of the signal $d_n = s_n + v_1(n)$, where $s_n = \cos(\omega_0 n)$ with $\omega_0 = 2 * 10^{-3} * \pi$, $a_1 = 0.8$, and $\sigma_\eta^2 = 0.05$. Fig. 4.14B is the respective realization of the signal $s_n + v_1(n) -$

**FIGURE 4.14**

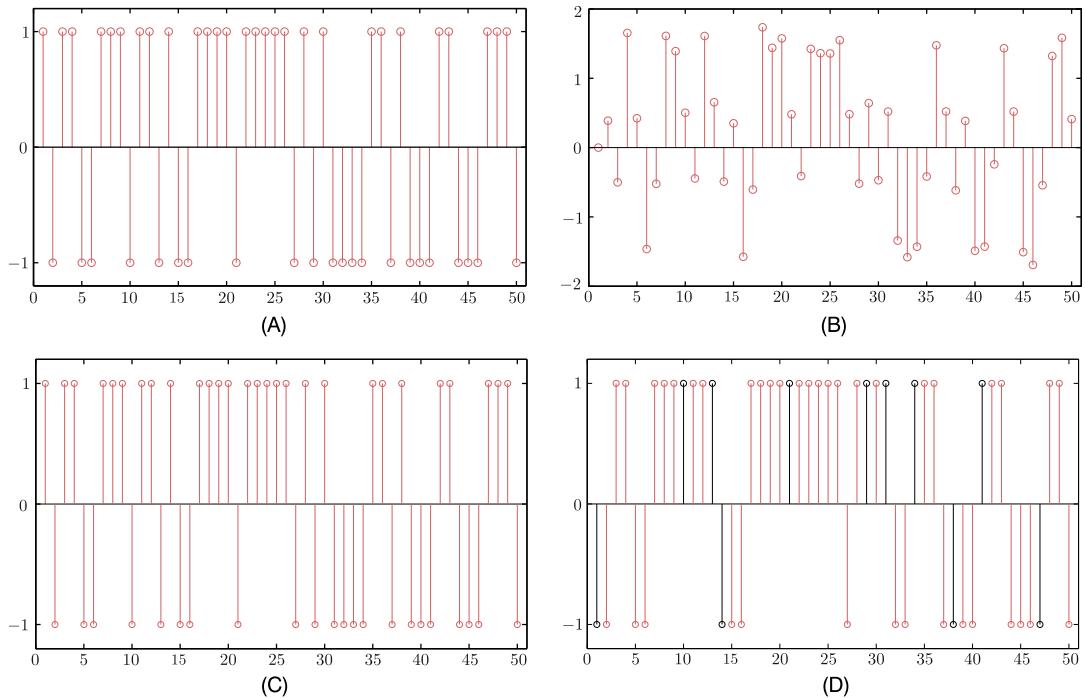
(A) The noisy sinusoid signal of Example 4.1. (B) The denoised signal for strongly correlated noise sources, v_1 and v_2 . (C) The obtained denoised signal for less correlated noise sources.

$\hat{d}(n)$ for $a_2 = 0.75$. The corresponding weights for the canceler are $\mathbf{w}_* = [1, 0.125]^T$. Fig. 4.14C corresponds to $a_2 = 0.5$. Observe that the higher the cross-correlation between $v_1(n)$ and $v_2(n)$, the better the obtained result becomes.

Example 4.2 (Channel equalization). Consider the channel equalization setup in Fig. 4.12, where the output of the channel, which is sensed by the receiver, is given by

$$\mathbf{u}_n = 0.5\mathbf{s}_n + \mathbf{s}_{n-1} + \boldsymbol{\eta}_n. \quad (4.58)$$

The goal is to design an equalizer comprising three taps, $\mathbf{w} = [w_0, w_1, w_2]^T$, so that

**FIGURE 4.15**

(A) A realization of the information sequence comprising equiprobable, randomly generated, ± 1 samples of Example 4.2. (B) The received at the receiver-end corresponding sequence. (C) The sequence at the output of the equalizer for a low channel noise case. The original sequence is fully recovered with no errors. (D) The output of the equalizer for high channel noise. The samples in gray are in error and of opposite polarity compared to the originally transmitted samples.

$$\hat{d}_n = \mathbf{w}^T \mathbf{u}_n,$$

and estimate the unknown taps using as a desired response sequence $d_n = s_{n-1}$. We are given that $\mathbb{E}[s_n] = \mathbb{E}[\eta_n] = 0$ and

$$\Sigma_s = \sigma_s^2 I, \quad \Sigma_\eta = \sigma_\eta^2 I.$$

Note that for the desired response we have used a delay $L = 1$. In order to better understand the reason that a delay is used and without going into many details (for the more experienced reader, note that the channel is nonminimum phase, e.g., [41]), observe that at time n , most of the contribution to u_n in (4.58) comes from the symbol s_{n-1} , which is weighted by one, while the sample s_n is weighted by 0.5; hence, it is most natural from an intuitive point of view, at time n , having received u_n , to try to obtain an estimate for s_{n-1} . This justifies the use of the delay.

Fig. 4.15A shows a realization of the input information sequence s_n . It consists of randomly generated, equiprobable ± 1 samples. The effect of the channel is (a) to combine successive information

samples together (ISI) and (b) to add noise; the purpose of the equalizer is to optimally remove both of them. Fig. 4.15B shows the respective realization sequence of \mathbf{u}_n , which is received at the receiver's front end. Observe that, by looking at it, one cannot recognize in it the original sequence; the noise together with the ISI has really changed its "look."

Following a similar procedure as in the previous example, we obtain (Problem 4.14)

$$\Sigma_u = \begin{bmatrix} 1.25\sigma_s^2 + \sigma_\eta^2 & 0.5\sigma_s^2 & 0 \\ 0.5\sigma_s^2 & 1.25\sigma_s^2 + \sigma_\eta^2 & 0.5\sigma_s^2 \\ 0 & 0.5\sigma_s^2 & 1.25\sigma_s^2 + \sigma_\eta^2 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} \sigma_s^2 \\ 0.5\sigma_s^2 \\ 0 \end{bmatrix}.$$

Solving the normal equations,

$$\Sigma_u \mathbf{w}_* = \mathbf{p},$$

for $\sigma_s^2 = 1$ and $\sigma_\eta^2 = 0.01$, results in

$$\mathbf{w}_* = [0.7462, 0.1195, -0.0474]^T.$$

Fig. 4.15C shows the recovered sequence by the equalizer ($\mathbf{w}_*^T \mathbf{u}_n$), after appropriate thresholding. It is exactly the same as the transmitted one: no errors. Fig. 4.15D shows the recovered sequence for the case where the variance of the noise was increased to $\sigma_\eta^2 = 1$. The corresponding MSE optimal equalizer is equal to

$$\mathbf{w}_* = [0.4132, 0.1369, -0.0304]^T.$$

This time, the sequence reconstructed by the equalizer has errors with respect to the transmitted one (gray lines).

A slightly alternative formulation for obtaining Σ_u , instead of computing each one of its elements individually, is the following. Verify that the input vector to the equalizer (with tree taps) at time n is given by

$$\mathbf{u}_n = \begin{bmatrix} 0.5 & 1 & 0 & 0 \\ 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} s_n \\ s_{n-1} \\ s_{n-2} \\ s_{n-3} \end{bmatrix} + \begin{bmatrix} \eta_n \\ \eta_{n-1} \\ \eta_{n-2} \\ \eta_{n-3} \end{bmatrix} := H \mathbf{s}_n + \boldsymbol{\eta}_n, \quad (4.59)$$

which results in

$$\begin{aligned} \Sigma_u &= \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^T] = H \sigma_s^2 H^T + \Sigma_\eta \\ &= \sigma_s^2 H H^T + \sigma_\eta^2 I. \end{aligned}$$

The reader can easily verify that this is the same as before. Note, however, that (4.59) reminds us of the linear regression model. Moreover, note the special structure of the matrix H . Such matrices are also known as *convolution* matrices. This structure is imposed by the fact that the elements of \mathbf{u}_n are

time-shifted versions of the first element, because the input vector corresponds to a random process. This is exactly the property that will be exploited next to derive efficient schemes for the solution of the normal equations.

4.8 ALGORITHMIC ASPECTS: THE LEVISON AND LATTICE-LADDER ALGORITHMS

The goal of this section is to present algorithmic schemes for the efficient solution of the normal equations in (4.16). The filtering case where the input and output entities are random processes will be considered. In this case, we have already pointed out that the input covariance matrix has a special structure. The main concepts to be presented here have a generality that goes beyond the specific form of the normal equations. A vast literature concerning efficient (fast) algorithms for the least-squares task as well as a number of its online/adaptive versions have their roots to the schemes to be presented here. At the heart of all these schemes lies the specific structure of the input vector, whose elements are *time-shifted versions* of its first element, u_n .

Recall from linear algebra that in order to solve a general linear system of l equations with l unknowns, one requires $O(l^3)$ operations (multiplications and additions (MADS)). Exploiting the rich structure of the autocorrelation/covariance matrix, associated with random processes, an algorithm with $O(l^2)$ operations will be derived. The more general complex-valued case will be considered.

The autocorrelation/covariance matrix (for zero mean processes) of the input random vector has been defined in (4.17). That is, it is Hermitian as well as semipositive definite. From now on, we will assume that it is positive definite. The covariance matrix in $\mathbb{C}^{m \times m}$, associated with a complex wide-sense stationary process, is given by

$$\begin{aligned}\Sigma_m &= \begin{bmatrix} r(0) & r(1) & \cdots & r(m-1) \\ r(-1) & r(0) & \cdots & r(m-2) \\ \vdots & \vdots & \ddots & \vdots \\ r(-m+1) & r(-m+2) & \cdots & r(0) \end{bmatrix} \\ &= \begin{bmatrix} r(0) & r(1) & \cdots & r(m-1) \\ r^*(1) & r(0) & \cdots & r(m-2) \\ \vdots & \vdots & \ddots & \vdots \\ r^*(m-1) & r^*(m-2) & \cdots & r(0) \end{bmatrix},\end{aligned}$$

where the property

$$r(i) := \mathbb{E}[u_n u_{n-i}^*] = \mathbb{E}[(u_{n-i} u_n^*)^*] := r^*(-i)$$

has been used. We have relaxed the notational dependence of Σ on u and we have instead explicitly indicated the order of the matrix, because this will be a very useful index from now on.

We will follow a recursive approach, and our aim will be to express the optimal filter solution of order m , denoted from now on as w_m , in terms of the optimal one, w_{m-1} , of order $m-1$.

4.9 MEAN-SQUARE ERROR ESTIMATION OF LINEAR MODELS

We now turn our attention to the case where the underlying model that relates the input–output variables is a linear one. To prevent confusion with what was treated in the previous sections, it must be stressed that, so far, we have been concerned with the linear estimation task. At no point in this stage of our discussion has the generation model of the data been brought in (with the exception in the comment of Remarks 4.2). We just adopted a linear estimator and obtained the MSE solution for it. The focus was on the solution and its properties. The emphasis here is on cases where the input–output variables are related via a linear data generation model.

Let us assume that we are given two jointly distributed random vectors, \mathbf{y} and $\boldsymbol{\theta}$, which are related according to the following linear model,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}, \quad (4.82)$$

where $\boldsymbol{\eta}$ denotes the set of the involved noise variables. Note that such a model covers the case of our familiar regression task, where the unknown parameters $\boldsymbol{\theta}$ are considered random, which is in line with the Bayesian philosophy, as discussed in Chapter 3. Once more, we assume zero mean vectors; otherwise, the respective mean values are subtracted. The dimensions of \mathbf{y} ($\boldsymbol{\eta}$) and $\boldsymbol{\theta}$ may not necessarily be the same; to be in line with the notation used in Chapter 3, let $\mathbf{y}, \boldsymbol{\eta} \in \mathbb{R}^N$ and $\boldsymbol{\theta} \in \mathbb{R}^l$. Hence, \mathbf{X} is an $N \times l$ matrix. Note that the matrix \mathbf{X} is considered to be deterministic and not random.

Assume the covariance matrices of our zero mean variables,

$$\Sigma_{\boldsymbol{\theta}} = \mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^T], \quad \Sigma_{\boldsymbol{\eta}} = \mathbb{E}[\boldsymbol{\eta}\boldsymbol{\eta}^T],$$

are known. The goal is to compute a matrix H of dimension $l \times N$ so that the linear estimator

$$\hat{\boldsymbol{\theta}} = H\mathbf{y} \quad (4.83)$$

minimizes the MSE cost

$$J(H) := \mathbb{E}[(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})] = \sum_{i=1}^l \mathbb{E}[(\theta_i - \hat{\theta}_i)^2]. \quad (4.84)$$

Note that this is a *multichannel* estimation task and it is equivalent to solving l optimization tasks, one for each component, θ_i , of $\boldsymbol{\theta}$. Defining the error vector as

$$\boldsymbol{\epsilon} := \boldsymbol{\theta} - \hat{\boldsymbol{\theta}},$$

the cost function is equal to the trace of the corresponding *error covariance matrix*, so that

$$J(H) := \text{trace} \left\{ \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T] \right\}.$$

Focusing on the i th component in (4.83), we write

$$\hat{\theta}_i = \mathbf{h}_i^T \mathbf{y}, \quad i = 1, 2, \dots, l, \quad (4.85)$$

where \mathbf{h}_i^T is the i th row of H and its optimal estimate is given by

$$\mathbf{h}_{*,i} := \arg \min_{\mathbf{h}_i} \mathbb{E}[(\theta_i - \hat{\theta}_i)^2] = \mathbb{E}[(\theta_i - \mathbf{h}_i^T \mathbf{y})^2]. \quad (4.86)$$

Minimizing (4.86) is exactly the same task as that of the linear estimation considered in the previous section (with \mathbf{y} in place of \mathbf{x} and θ_i in place of y); hence,

$$\Sigma_y \mathbf{h}_{*,i} = \mathbf{p}_i, \quad i = 1, 2, \dots, l,$$

where

$$\Sigma_y = \mathbb{E}[\mathbf{y}\mathbf{y}^T] \quad \text{and} \quad \mathbf{p}_i = \mathbb{E}[\mathbf{y}\theta_i], \quad i = 1, 2, \dots, l,$$

or

$$\mathbf{h}_{*,i}^T = \mathbf{p}_i^T \Sigma_y^{-1}, \quad i = 1, 2, \dots, l,$$

and finally,

$$H_* = \Sigma_{y\theta} \Sigma_y^{-1}, \quad \hat{\boldsymbol{\theta}} = \Sigma_{y\theta} \Sigma_y^{-1} \mathbf{y}, \quad (4.87)$$

where

$$\Sigma_{y\theta} := \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_l^T \end{bmatrix} = \mathbb{E}[\boldsymbol{\theta} \mathbf{y}^T] \quad (4.88)$$

is an $l \times N$ cross-correlation matrix. All that is now required is to compute Σ_y and $\Sigma_{y\theta}$. To this end,

$$\begin{aligned} \Sigma_y &= \mathbb{E}[\mathbf{y}\mathbf{y}^T] = \mathbb{E}[(X\boldsymbol{\theta} + \boldsymbol{\eta})(\boldsymbol{\theta}^T X^T + \boldsymbol{\eta}^T)] \\ &= X \Sigma_\theta X^T + \Sigma_\eta, \end{aligned} \quad (4.89)$$

where the independence of the zero mean vectors $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ has been used. Similarly,

$$\Sigma_{y\theta} = \mathbb{E}[\boldsymbol{\theta} \mathbf{y}^T] = \mathbb{E}[\boldsymbol{\theta} (\boldsymbol{\theta}^T X^T + \boldsymbol{\eta}^T)] = \Sigma_\theta X^T, \quad (4.90)$$

and combining (4.87), (4.89), and (4.90), we obtain

$$\hat{\boldsymbol{\theta}} = \Sigma_\theta X^T (\Sigma_\eta + X \Sigma_\theta X^T)^{-1} \mathbf{y}. \quad (4.91)$$

Employing from Appendix A.1 the matrix identity

$$(A^{-1} + B^T C^{-1} B)^{-1} B^T C^{-1} = A B^T (B A B^T + C)^{-1}$$

in (4.91) we obtain

$$\hat{\boldsymbol{\theta}} = (\Sigma_{\theta}^{-1} + X^T \Sigma_{\eta}^{-1} X)^{-1} X^T \Sigma_{\eta}^{-1} \mathbf{y} : \text{MSE linear estimator.} \quad (4.92)$$

In case of complex-valued variables, the only difference is that transposition is replaced by Hermitian transposition.

Remarks 4.5.

- Recall from Chapter 3 that the optimal MSE estimator of $\boldsymbol{\theta}$ given the values of \mathbf{y} is provided by

$$\mathbb{E}[\boldsymbol{\theta}|\mathbf{y}].$$

However, as shown in Problem 3.16, if $\boldsymbol{\theta}$ and \mathbf{y} are jointly Gaussian vectors, then the optimal estimator is linear (affine for nonzero mean variables), and it coincides with the MSE linear estimator of (4.92).

- If we allow nonzero mean values, then instead of (4.83) the affine model should be adopted,

$$\hat{\boldsymbol{\theta}} = H\mathbf{y} + \boldsymbol{\mu}.$$

Then

$$\mathbb{E}[\hat{\boldsymbol{\theta}}] = H\mathbb{E}[\mathbf{y}] + \boldsymbol{\mu} \Rightarrow \boldsymbol{\mu} = \mathbb{E}[\hat{\boldsymbol{\theta}}] - H\mathbb{E}[\mathbf{y}].$$

Hence,

$$\hat{\boldsymbol{\theta}} = \mathbb{E}[\hat{\boldsymbol{\theta}}] + H(\mathbf{y} - \mathbb{E}[\mathbf{y}]),$$

and finally,

$$\hat{\boldsymbol{\theta}} - \mathbb{E}[\hat{\boldsymbol{\theta}}] = H(\mathbf{y} - \mathbb{E}[\mathbf{y}]),$$

which justifies our approach to subtract the means and work with zero mean value variables. For nonzero mean values, the analogue of (4.92) is

$$\hat{\boldsymbol{\theta}} = \mathbb{E}[\hat{\boldsymbol{\theta}}] + \left(\Sigma_{\theta}^{-1} + X^T \Sigma_{\eta}^{-1} X \right)^{-1} X^T \Sigma_{\eta}^{-1} (\mathbf{y} - \mathbb{E}[\mathbf{y}]). \quad (4.93)$$

Note that for zero mean noise $\boldsymbol{\eta}$, $\mathbb{E}[\mathbf{y}] = X\mathbb{E}[\boldsymbol{\theta}]$.

- Compare (4.93) with (3.73) for the Bayesian inference approach. They are identical, provided that the covariance matrix of the prior (Gaussian) PDF is equal to Σ_{θ} and $\boldsymbol{\theta}_0 = \mathbb{E}[\hat{\boldsymbol{\theta}}]$ for a zero mean noise variable.

4.9.1 THE GAUSS–MARKOV THEOREM

We now turn our attention to the case where $\boldsymbol{\theta}$ in the regression model is considered to be an (unknown) constant, instead of a random vector. Thus, the linear model is now written as

$$\mathbf{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}, \quad (4.94)$$

and the randomness of \mathbf{y} is solely due to $\boldsymbol{\eta}$, which is assumed to be zero mean with covariance matrix Σ_η . The goal is to design an *unbiased linear estimator* of $\boldsymbol{\theta}$, which minimizes the MSE,

$$\hat{\boldsymbol{\theta}} = H\mathbf{y}, \quad (4.95)$$

and select H such that

$$\begin{aligned} & \text{minimize}_{H} \quad \text{trace} \left\{ \mathbb{E} [(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T] \right\} \\ & \text{subject to} \quad \mathbb{E}[\hat{\boldsymbol{\theta}}] = \boldsymbol{\theta}. \end{aligned} \quad (4.96)$$

From (4.94) and (4.95), we get

$$\mathbb{E}[\hat{\boldsymbol{\theta}}] = H\mathbb{E}[\mathbf{y}] = H\mathbb{E}[(X\boldsymbol{\theta} + \boldsymbol{\eta})] = HX\boldsymbol{\theta},$$

which implies that the unbiased constraint is equivalent to

$$HX = I. \quad (4.97)$$

Employing (4.95), the error vector becomes

$$\boldsymbol{\epsilon} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}} = \boldsymbol{\theta} - H\mathbf{y} = \boldsymbol{\theta} - H(X\boldsymbol{\theta} + \boldsymbol{\eta}) = -H\boldsymbol{\eta}. \quad (4.98)$$

Hence, the constrained minimization in (4.96) can now be written as

$$\begin{aligned} H_* &= \arg \min_H \text{trace}\{H\Sigma_\eta H^T\}, \\ \text{s.t.} \quad HX &= I. \end{aligned} \quad (4.99)$$

Solving (4.99) results in (Problem 4.18)

$$H_* = (X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1}, \quad (4.100)$$

and the associated minimum MSE is

$$J(H_*) := \text{MSE}(H_*) = \text{trace} \left\{ (X^T \Sigma_\eta^{-1} X)^{-1} \right\}. \quad (4.101)$$

The reader can verify that

$$J(H) \geq J(H_*)$$

for any other linear unbiased estimator (Problem 4.19).

The previous result is known as the *Gauss–Markov theorem*. The optimal MSE linear unbiased estimator is given by

$\hat{\boldsymbol{\theta}} = (X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1} \mathbf{y}; \quad \text{BLUE.}$

(4.102)

It is also known as the *best linear unbiased estimator* (BLUE), or the *minimum variance unbiased linear estimator*. For complex-valued variables, the transposition is simply replaced by the Hermitian one.

Remarks 4.6.

- For the BLUE to exist, $X^T \Sigma_{\eta}^{-1} X$ must be invertible. This is guaranteed if Σ_{η} is positive definite and the $N \times l$ matrix X , $N \geq l$, is full rank (Problem 4.20).
- Observe that the BLUE coincides with the maximum likelihood estimator (Chapter 3) if η follows a multivariate Gaussian distribution; recall that under this assumption, the Cramér–Rao bound is achieved. If this is not the case, there may be another unbiased estimator (nonlinear), which results in lower MSE. Recall also from Chapter 3 that there may be a biased estimator that results in lower MSE; see [13,38] and the references therein for a related discussion.

Example 4.3 (Channel identification). The task is illustrated in Fig. 4.11. Assume that we have access to a set of input–output observations, u_n and d_n , $n = 0, 1, 2, \dots, N - 1$. Moreover, we are given that the impulse response of the system comprises l taps, it is zero mean, and its covariance matrix is Σ_w . Also, the second-order statistics of the zero mean noise are known, and we are given its covariance matrix, Σ_{η} . Then, assuming that the plant starts from zero initial conditions, we can adopt the following model relating the involved random variables (in line with the model in (4.82)):

$$\mathbf{d} := \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{l-1} \\ \vdots \\ d_{N-1} \end{bmatrix} = U \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{l-1} \end{bmatrix} + \begin{bmatrix} \eta_0 \\ \eta_1 \\ \vdots \\ \eta_{l-1} \\ \vdots \\ \eta_{N-1} \end{bmatrix}, \quad (4.103)$$

where

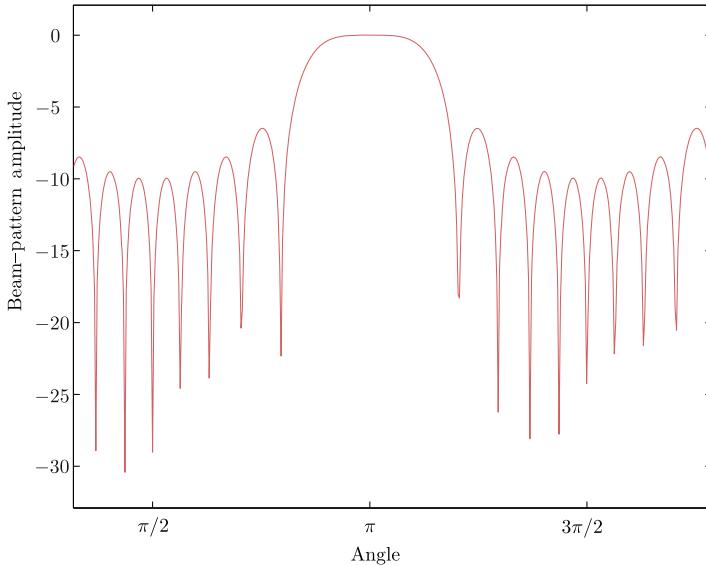
$$U := \begin{bmatrix} u_0 & 0 & 0 & \cdots & 0 \\ u_1 & u_0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ u_{l-1} & u_{l-2} & \cdots & \cdots & u_0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ u_{N-1} & \cdots & \cdots & \cdots & u_{N-l} \end{bmatrix}.$$

Note that U is treated deterministically. Then, recalling (4.92) and plugging in the set of obtained measurements, the following estimate results:

$$\hat{\mathbf{w}} = (\Sigma_w^{-1} + U^T \Sigma_{\eta}^{-1} U) U^T \Sigma_{\eta}^{-1} \mathbf{d}. \quad (4.104)$$

4.9.2 CONSTRAINED LINEAR ESTIMATION: THE BEAMFORMING CASE

We have already dealt with a constrained linear estimation task in Section 4.9.1 in our effort to obtain an unbiased estimator of a fixed-value parameter vector. In the current section, we will see that the pro-

**FIGURE 4.20**

The amplitude beam pattern, in dBs, as a function of the angle ϕ with respect to the planar array.

This time, the beamformer is pushed to reduce its output signal, which, due to the presence of the constraint, is equivalent to optimally minimizing the contributions originating from the noise as well as from all other interference sources impinging on the array from different, to ϕ , directions. The resulting solution of (4.111) is obviously the same as (4.109) and (4.110) if one replaces Σ_η with Σ_u .

This type of linearly constrained task is known as *linearly constrained minimum variance* (LMV) or *Capon beamforming* or *minimum variance distortionless response (MVDR) beamforming*. For a concise introduction to beamforming, see, e.g., [48].

Widely linear versions for the beamforming task have also been proposed (e.g., [10,32]) (Problem 4.21).

Fig. 4.20 shows the resulting beam pattern as a function of the angle ϕ . The desired angle for designing the optimal set of weights in (4.108) is $\phi = \pi$. The number of antenna elements is $l = 10$, the spacing has been chosen as $\frac{\Delta x}{\lambda} = 0.5$, and the noise covariance matrix is chosen as $0.1I$. The beam pattern amplitude is in dBs, meaning the vertical axis shows $20\log_{10}(|\mathbf{w}_*^H \mathbf{x}(\phi)|)$. Thus, any signal arriving from directions ϕ not close to $\phi = \pi$ will be absorbed. The main beam can become sharper if more elements are used.

4.10 TIME-VARYING STATISTICS: KALMAN FILTERING

So far, our discussion about the linear estimation task has been limited to stationary environments, where the statistical properties of the involved random variables are assumed to be invariant with time.

However, very often in practice this is not the case, and the statistical properties may be different at different time instants. As a matter of fact, a large effort in the subsequent chapters will be devoted to studying the estimation task under time-varying environments.

Rudolf Kalman is the third scientist, the other two being Wiener and Kolmogorov, whose significant contributions laid the foundations of estimation theory. Kalman was Hungarian-born and emigrated to the United States. He is the father of what is today known as system theory based on the state-space formulation, as opposed to the more limited input-output description of systems.

In 1960, in two seminal papers, Kalman proposed the celebrated Kalman filter, which exploits the state-space formulation in order to accommodate in an elegant way time-varying dynamics [18,19]. We will derive the basic recursions of the Kalman filter in the general context of two jointly distributed random vectors \mathbf{y} , \mathbf{x} . The task is to estimate the values of \mathbf{x} given observations of \mathbf{y} . Let \mathbf{y} and \mathbf{x} be linearly related via the following set of recursions:

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \quad n \geq 0 : \text{ state equation,} \quad (4.112)$$

$$\mathbf{y}_n = H_n \mathbf{x}_n + \mathbf{v}_n, \quad n \geq 0 : \text{ output equation,} \quad (4.113)$$

where $\boldsymbol{\eta}_n, \mathbf{x}_n \in \mathbb{R}^l$, $\mathbf{v}_n, \mathbf{y}_n \in \mathbb{R}^k$. The vector \mathbf{x}_n is known as the *state* of the system at time n and \mathbf{y}_n is the output, which is the vector that can be observed (measured); $\boldsymbol{\eta}_n$ and \mathbf{v}_n are the noise vectors, known as *process noise* and *measurement noise*, respectively. Matrices F_n and H_n are of appropriate dimensions and they are assumed to be known. Observe that the so-called *state equation* provides the information related to the time-varying dynamics of the corresponding system. It turns out that a large number of real-world tasks can be brought into the form of (4.112) and (4.113). The model is known as the *state-space* model for \mathbf{y}_n . In order to derive the time-varying estimator, $\hat{\mathbf{x}}_n$, given the measured values of \mathbf{y}_n , the following assumptions will be adopted:

- $\mathbb{E}[\boldsymbol{\eta}_n \boldsymbol{\eta}_n^T] = Q_n$, $\mathbb{E}[\boldsymbol{\eta}_n \boldsymbol{\eta}_m^T] = O$, $n \neq m$,
- $\mathbb{E}[\mathbf{v}_n \mathbf{v}_n^T] = R_n$, $\mathbb{E}[\mathbf{v}_n \mathbf{v}_m^T] = O$, $n \neq m$,
- $\mathbb{E}[\boldsymbol{\eta}_n \mathbf{v}_m^T] = O$, $\forall n, m$,
- $\mathbb{E}[\boldsymbol{\eta}_n] = \mathbb{E}[\mathbf{v}_n] = \mathbf{0}$, $\forall n$,

where O denotes a matrix with zero elements. That is, $\boldsymbol{\eta}_n, \mathbf{v}_n$ are uncorrelated; moreover, noise vectors at different time instants are also considered uncorrelated. Versions where some of these conditions are relaxed are also available. The respective covariance matrices, Q_n and R_n , are assumed to be known.

The development of the time-varying estimation task evolves around two types of estimators for the state variables:

- The first one is denoted as

$$\hat{\mathbf{x}}_{n|n-1},$$

and it is based on all information that has been received up to and including time instant $n - 1$; in other words, the obtained observations of $\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_{n-1}$. This is known as the *a priori* or *prior* estimator.

- The second estimator at time n is known as the *posterior* one, it is denoted as

$$\hat{\mathbf{x}}_{n|n},$$

and it is computed by updating $\hat{\mathbf{x}}_{n|n-1}$ after \mathbf{y}_n has been observed.

For the development of the algorithm, assume that at time $n - 1$ all required information is available; that is, the value of the posterior estimator as well the respective error covariance matrix

$$\hat{\mathbf{x}}_{n-1|n-1}, \quad P_{n-1|n-1} := \mathbb{E}[\mathbf{e}_{n-1|n-1}\mathbf{e}_{n-1|n-1}^T],$$

where

$$\mathbf{e}_{n-1|n-1} := \mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}.$$

Step 1: Using $\hat{\mathbf{x}}_{n-1|n-1}$, predict $\hat{\mathbf{x}}_{n|n-1}$ using the state equation; that is,

$$\hat{\mathbf{x}}_{n|n-1} = F_n \hat{\mathbf{x}}_{n-1|n-1}. \quad (4.114)$$

In other words, ignore the contribution from the noise. This is natural, because prediction cannot involve the unobserved variables.

Step 2: Obtain the respective error covariance matrix,

$$P_{n|n-1} = \mathbb{E}[(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^T]. \quad (4.115)$$

However,

$$\begin{aligned} \mathbf{e}_{n|n-1} &:= \mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} = F_n \mathbf{x}_{n-1} + \mathbf{\eta}_n - F_n \hat{\mathbf{x}}_{n-1|n-1} \\ &= F_n \mathbf{e}_{n-1|n-1} + \mathbf{\eta}_n. \end{aligned} \quad (4.116)$$

Combining (4.115) and (4.116), it is straightforward to see that

$$P_{n|n-1} = F_n P_{n-1|n-1} F_n^T + Q_n. \quad (4.117)$$

Step 3: Update $\hat{\mathbf{x}}_{n|n-1}$. To this end, adopt the following recursion:

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + K_n \mathbf{e}_n, \quad (4.118)$$

where

$$\mathbf{e}_n := \mathbf{y}_n - H_n \hat{\mathbf{x}}_{n|n-1}. \quad (4.119)$$

This time update recursion, once the observations for \mathbf{y}_n have been received, has a form that we will meet over and over again in this book. The “new” (posterior) estimate is equal to the “old” (prior) one, which is based on *the past history, plus a correction term*; the latter is proportional to the error \mathbf{e}_n in predicting the newly arrived observations vector and its prediction based on the “old” estimate. Matrix K_n , known as the *Kalman gain*, controls the amount of correction and its value is computed so as to minimize the MSE; in other words,

$$J(K_n) := \mathbb{E}[\mathbf{e}_{n|n}^T \mathbf{e}_{n|n}] = \text{trace}\{P_{n|n}\}, \quad (4.120)$$

where

$$P_{n|n} = \mathbb{E}[\mathbf{e}_{n|n} \mathbf{e}_{n|n}^T] \quad (4.121)$$

and

$$\mathbf{e}_{n|n} := \mathbf{x}_n - \hat{\mathbf{x}}_{n|n}.$$

It can be shown that the optimal Kalman gain is equal to (Problem 4.22)

$$K_n = P_{n|n-1} H_n^T S_n^{-1}, \quad (4.122)$$

where

$$S_n = R_n + H_n P_{n|n-1} H_n^T. \quad (4.123)$$

Step 4: The final recursion that is now needed in order to complete the scheme is that for the update of $P_{n|n}$. Combining the definitions in (4.119) and (4.121) with (4.118), we obtain the following result (Problem 4.23):

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}. \quad (4.124)$$

The algorithm has now been derived. All that is now needed is to select the initial conditions, which are chosen such that

$$\hat{\mathbf{x}}_{1|0} = \mathbb{E}[\mathbf{x}_1] \quad (4.125)$$

$$P_{1|0} = \mathbb{E}[(\mathbf{x}_1 - \hat{\mathbf{x}}_{1|0})(\mathbf{x}_1 - \hat{\mathbf{x}}_{1|0})^T] = \Pi_0, \quad (4.126)$$

for some initial guess Π_0 . The Kalman algorithm is summarized in Algorithm 4.2.

Algorithm 4.2 (Kalman filtering).

- Input: F_n , H_n , Q_n , R_n , y_n , $n = 1, 2, \dots$
- Initialization:
 - $\hat{\mathbf{x}}_{1|0} = \mathbb{E}[\mathbf{x}_1]$
 - $P_{1|0} = \Pi_0$
- For $n = 1, 2, \dots$, Do
 - $S_n = R_n + H_n P_{n|n-1} H_n^T$
 - $K_n = P_{n|n-1} H_n^T S_n^{-1}$
 - $\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + K_n (y_n - H_n \hat{\mathbf{x}}_{n|n-1})$
 - $P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}$
 - $\hat{\mathbf{x}}_{n+1|n} = F_{n+1} \hat{\mathbf{x}}_{n|n}$
 - $P_{n+1|n} = F_{n+1} P_{n|n} F_{n+1}^T + Q_{n+1}$
- End For

For complex-valued variables, transposition is replaced by the Hermitian operation.

Remarks 4.7.

- Besides the previously derived basic scheme, there are a number of variants. Although, in theory, they are all equivalent, their practical implementation may lead to different performance. Observe

that $P_{n|n}$ is computed as the difference of two positive definite matrices; this may lead to a $P_{n|n}$ that is not positive definite, due to numerical errors. This can cause the algorithm to diverge. A popular alternative is the so-called *information filtering* scheme, which propagates the inverse state-error covariance matrices, $P_{n|n}^{-1}$ and $P_{n|n-1}^{-1}$ [20]. In contrast, the scheme in Algorithm 4.2 is known as the *covariance Kalman algorithm* (Problem 4.24).

To cope with the numerical stability issues, a family of algorithms propagates the factors of $P_{n|n}$ (or $P_{n|n}^{-1}$), resulting from the respective Cholesky factorization [5,40].

- There are different approaches to arrive at the Kalman filtering recursions. An alternative derivation is based on the orthogonality principle applied to the so-called *innovations process* associated with the observation sequence, so that

$$\epsilon(n) = \mathbf{y}_n - \hat{\mathbf{y}}_{n|1:n-1},$$

where $\hat{\mathbf{y}}_{n|1:n-1}$ is the prediction based on the past observations history [17]. In Chapter 17, we are going to rederive the Kalman recursions looking at it as a Bayesian network.

- Kalman filtering is a generalization of the optimal MSE linear filtering. It can be shown that when the involved processes are stationary, Kalman filter converges in its steady state to our familiar normal equations [31].
- *Extended Kalman filters*. In (4.112) and (4.113), both the state and the output equations have a linear dependence on the state vector \mathbf{x}_n . Kalman filtering, in a more general formulation, can be cast as

$$\mathbf{x}_n = f_n(\mathbf{x}_{n-1}) + \eta_n,$$

$$\mathbf{y}_n = h_n(\mathbf{x}_n) + v_n,$$

where f_n and h_n are nonlinear vector functions. In the extended Kalman filtering (EKF), the idea is to *linearize* the functions h_n and f_n , at each time instant, via their Taylor series expansions, and keep the linear term only, so that

$$F_n = \frac{\partial f_n(\mathbf{x}_n)}{\partial \mathbf{x}_n} \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n-1|n-1}},$$

$$H_n = \frac{\partial h_n(\mathbf{x}_n)}{\partial \mathbf{x}_n} \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}},$$

and then proceed by using the updates derived for the linear case.

By its very definition, the EKF is suboptimal and often in practice one may face divergence of the algorithm; in general, it must be stated that its practical implementation needs to be carried out with care. Having said that, it must be pointed out that it is heavily used in a number of practical systems.

Unscented Kalman filters represent an alternative way to cope with the nonlinearity, and the main idea springs forth from probabilistic arguments. A set of points are deterministically selected from a Gaussian approximation of $p(\mathbf{x}_n | \mathbf{y}_1, \dots, \mathbf{y}_n)$; these points are propagated through the nonlinearities, and estimates of the mean values and covariances are obtained [15]. Particle filtering, to be discussed in Chapter 17, is another powerful and popular approach to deal with nonlinear state-space models via probabilistic arguments.

More recently, extensions of Kalman filtering in reproducing kernel Hilbert spaces offers an alternative approach to deal with nonlinearities [52].

A number of Kalman filtering versions for distributed learning (Chapter 5) have appeared in, e.g., [9, 23, 33, 43]. In the latter of the references, subspace learning methods are utilized in the prediction stage associated with the state variables.

- The literature on Kalman filtering is huge, especially when applications are concerned. The interested reader may consult more specialized texts, for example [4, 8, 17] and the references therein.

Example 4.4 (Autoregressive process estimation). Let us consider an AR process (Chapter 2) of order l , represented as

$$\mathbf{x}_n = - \sum_{i=1}^l a_i \mathbf{x}_{n-i} + \eta_n, \quad (4.127)$$

where η_n is a white noise sequence of variance σ_η^2 . Our task is to obtain an estimate $\hat{\mathbf{x}}_n$ of \mathbf{x}_n , having observed a noisy version of it, y_n . The corresponding random variables are related as

$$y_n = \mathbf{x}_n + v_n. \quad (4.128)$$

To this end, the Kalman filtering formulation will be used. Note that the MSE linear estimation, presented in Section 4.9, cannot be used here. As we have already discussed in Chapter 2, an AR process is asymptotically stationary; for finite-time samples, the initial conditions at time $n = 0$ are “remembered” by the process and the respective second-order statistics are time dependent, hence it is a nonstationary process. However, Kalman filtering is specially suited for such cases.

Let us rewrite (4.127) and (4.128) as

$$\begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_{n-1} \\ \mathbf{x}_{n-2} \\ \vdots \\ \mathbf{x}_{n-l+1} \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{l-1} & -a_l \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{n-1} \\ \mathbf{x}_{n-2} \\ \mathbf{x}_{n-3} \\ \vdots \\ \mathbf{x}_{n-l} \end{bmatrix} + \begin{bmatrix} \eta_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

$$y_n = [1 \ 0 \ \cdots \ 0] \begin{bmatrix} \mathbf{x}_n \\ \vdots \\ \mathbf{x}_{n-l+1} \end{bmatrix} + v_n$$

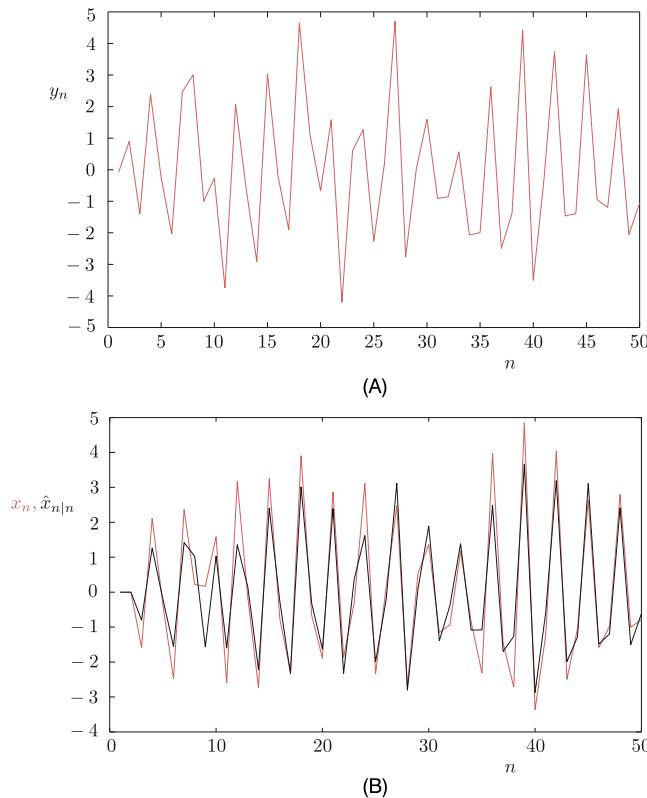
or

$$\mathbf{x}_n = F \mathbf{x}_{n-1} + \eta, \quad (4.129)$$

$$y_n = H \mathbf{x}_n + v_n, \quad (4.130)$$

where the definitions of $F_n := F$ and $H_n := H$ are obvious and

$$Q_n = \begin{bmatrix} \sigma_\eta^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \quad R_n = \sigma_v^2 \quad (\text{scalar}).$$

**FIGURE 4.21**

(A) A realization of the observation sequence, y_n , which is used by the Kalman filter to obtain the predictions of the state variable. (B) The AR process (state variable) in red together with the predicted by the Kalman filter sequence (gray), for Example 4.4. The Kalman filter has removed the effect of the noise v_n .

Fig. 4.21A shows the values of a specific realization y_n , and Fig. 4.21B shows the corresponding realization of the AR(2) (red) together with the predicted Kalman filter sequence \hat{x}_n . Observe that the match is very good. For the generation of the AR process we used $l = 2$, $\alpha_1 = 0.95$, $\alpha_2 = 0.9$, $\sigma_\eta^2 = 0.5$. For the Kalman filter output noise, $\sigma_v^2 = 1$.

PROBLEMS

4.1 Show that the set of equations

$$\Sigma\theta = p$$

has a unique solution if $\Sigma > 0$ and infinitely many if Σ is singular.

ONLINE LEARNING: THE STOCHASTIC GRADIENT DESCENT FAMILY OF ALGORITHMS

5

CONTENTS

5.1	Introduction	180
5.2	The Steepest Descent Method	181
5.3	Application to the Mean-Square Error Cost Function	184
	Time-Varying Step Sizes	190
5.3.1	The Complex-Valued Case	193
5.4	Stochastic Approximation	194
	Application to the MSE Linear Estimation	196
5.5	The Least-Mean-Squares Adaptive Algorithm	198
5.5.1	Convergence and Steady-State Performance of the LMS in Stationary Environments	199
	Convergence of the Parameter Error Vector	199
5.5.2	Cumulative Loss Bounds	204
5.6	The Affine Projection Algorithm	206
	Geometric Interpretation of APA	208
	Orthogonal Projections	208
5.6.1	The Normalized LMS	211
5.7	The Complex-Valued Case	213
	The Widely Linear LMS	213
	The Widely Linear APA	214
5.8	Relatives of the LMS	214
	The Sign-Error LMS	214
	The Least-Mean-Fourth (LMF) Algorithm	215
	Transform-Domain LMS	215
5.9	Simulation Examples	218
5.10	Adaptive Decision Feedback Equalization	221
5.11	The Linearly Constrained LMS	224
5.12	Tracking Performance of the LMS in Nonstationary Environments	225
5.13	Distributed Learning: the Distributed LMS	227
5.13.1	Cooperation Strategies	228
	Centralized Networks	229
	Decentralized Networks	229
5.13.2	The Diffusion LMS	231
5.13.3	Convergence and Steady-State Performance: Some Highlights	237
5.13.4	Consensus-Based Distributed Schemes	240
5.14	A Case Study: Target Localization	241
5.15	Some Concluding Remarks: Consensus Matrix	243

Problems	244
MATLAB® Exercises	246
References	247

5.1 INTRODUCTION

In Chapter 4, we introduced the notion of mean-square error (MSE) optimal linear estimation and stated the normal equations for computing the parameters/coefficients of the optimal estimator/filter. A prerequisite for the normal equations is the knowledge of the second-order statistics of the involved processes/variables, so that the covariance matrix of the input and the input–output cross-correlation vector can be obtained. However, most often in practice, all the designer has at her/his disposal is a set of training samples; thus, the covariance matrix and the cross-correlation vector have to be estimated somehow. More importantly, in a number of practical applications, the underlying statistics may be time-varying. We discussed this scenario while introducing Kalman filtering. The path taken there was to adopt a state-space representation and assume that the time dynamics of the model were known. However, although Kalman filtering is an elegant tool, it does not scale well in high-dimensional spaces due to the involved matrix operations and inversions.

The focus of this chapter is to introduce *online learning* techniques for estimating the unknown parameter vector. These are time-iterative schemes, which update the available estimate every time a measurement set (input–output pair of observations) is acquired. Thus, in contrast to the so-called *batch processing* methods, which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance. Online algorithmic schemes learn the underlying statistics from the data in a *time-iterative* fashion. Hence, one does not have to provide further statistical information. Another characteristic of the algorithmic family, to be developed and studied in this chapter, is its computational simplicity. The required complexity for updating the estimate of the unknown parameter vector is linear with respect to the number of the unknown parameters. This is one of the major reasons that have made such schemes very popular in a number of practical applications; besides complexity, we will discuss other reasons that have contributed to their popularity. A discussion concerning batch versus online algorithms is provided in Section 8.12.

The fact that such learning algorithms work in a time-iterative mode gives them the agility to learn and *track* slow time variations of the statistics of the involved processes/variables; this is the reason these algorithms are also called *time-adaptive* or simply *adaptive*, because they can adapt to the needs of a changing environment. Online/time-adaptive algorithms have been used extensively since the early 1960s in a wide range of applications, including signal processing, control, and communications. More recently, the philosophy behind such schemes has been gaining in popularity in the context of applications where data reside in large datasets, with a massive number of samples; for such tasks, storing all the data points in the memory may not be possible, and they have to be considered one at a time. Moreover, the complexity of batch processing techniques can amount to prohibitive levels, for today’s technology. The current trend is to refer to such applications as *big data* problems.

In this chapter, we focus on a very popular class of online/adaptive algorithms that springs from the classical gradient descent method for optimization. Although our emphasis will be on the squared error loss function, the same rationale can also be adopted for other (differentiable) loss functions. The case of nondifferentiable loss functions will be treated in Chapter 8. A number of variants of the stochastic

gradient rationale, in the context of deep neural networks, is given in Chapter 18. The online processing rationale will be a recurrent theme in this book.

5.2 THE STEEPEST DESCENT METHOD

Our starting point is the method of *gradient descent*, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \mathbb{R}^l$. As does any other iterative technique, the method starts from an initial estimate, $\boldsymbol{\theta}^{(0)}$, and generates a sequence, $\boldsymbol{\theta}^{(i)}$, $i = 1, 2, \dots$, such that

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}, \quad i > 0, \quad (5.1)$$

where $\mu_i > 0$. All the schemes for the iterative minimization of a cost function, which we will deal with in this book, have the general form of (5.1). Their differences are in the way that μ_i and $\Delta\boldsymbol{\theta}^{(i)}$ are chosen; the latter vector is known as the *update direction* or the *search direction*. The sequence μ_i is known as the *step size* or the *step length* at the i th iteration; note that the values of μ_i may either be constant or change at each iteration. In the gradient descent method, the choice of $\Delta\boldsymbol{\theta}^{(i)}$ is made to guarantee that

$$J(\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)}),$$

except at a minimizer, $\boldsymbol{\theta}_*$.

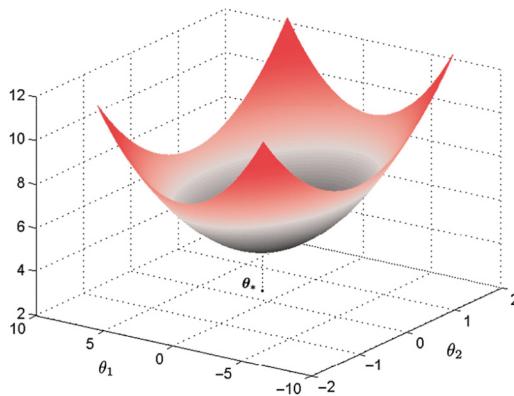
Assume that at the $(i - 1)$ th iteration step the value $\boldsymbol{\theta}^{(i-1)}$ has been obtained. Then, for sufficiently small μ_i and mobilizing a first-order Taylor expansion around $\boldsymbol{\theta}^{(i-1)}$, we can write

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}) \approx J(\boldsymbol{\theta}^{(i-1)}) + \mu_i \nabla J(\boldsymbol{\theta}^{(i-1)})^T \Delta\boldsymbol{\theta}^{(i)}.$$

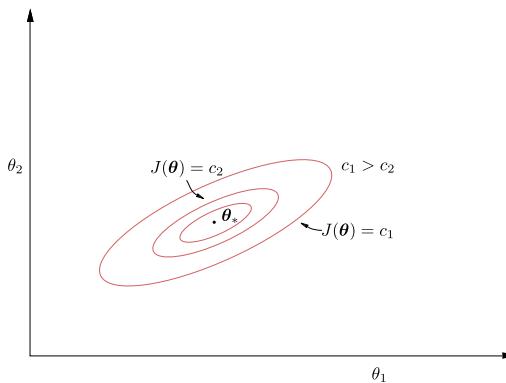
Looking carefully at the above approximation, what we basically do is to *linearize* locally the cost function with respect to $\Delta\boldsymbol{\theta}^{(i)}$. Selecting the search direction so that

$$\nabla J(\boldsymbol{\theta}^{(i-1)})^T \Delta\boldsymbol{\theta}^{(i)} < 0, \quad (5.2)$$

this guarantees that $J(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)})$. For such a choice, $\Delta\boldsymbol{\theta}^{(i)}$ and $\nabla J(\boldsymbol{\theta}^{(i-1)})$ must form an *obtuse angle*. Fig. 5.1 shows the graph of a cost function in the two-dimensional case, $\boldsymbol{\theta} \in \mathbb{R}^2$, and Fig. 5.2 shows the respective isovalue contours in the two-dimensional plane. Note that, in general, the contours can have any shape and are not necessarily ellipses; it all depends on the functional form of $J(\boldsymbol{\theta})$. However, because $J(\boldsymbol{\theta})$ has been assumed differentiable, the contours must be smooth and accept at any point a (unique) tangent plane, as this is defined by the respective gradient. Furthermore, recall from basic calculus that the gradient vector, $\nabla J(\boldsymbol{\theta})$, is perpendicular to the plane (line) tangent to the corresponding isovalue contour at the point $\boldsymbol{\theta}$ (Problem 5.1). The geometry is illustrated in Fig. 5.3; to facilitate the drawing and unclutter notation, we have removed the iteration index i . Note that by selecting the search direction which forms an obtuse angle with the gradient, it places $\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}$ at a point on a contour which corresponds to a lower value of $J(\boldsymbol{\theta})$. Two issues are now raised: (a) to choose the best search direction along which to move and (b) to compute how far along this direction one can go. Even without much mathematics, it is obvious from Fig. 5.3 that if $\mu_i \|\Delta\boldsymbol{\theta}^{(i)}\|$ is too large, then the new point can be placed on a contour corresponding to a larger value than that of the current

**FIGURE 5.1**

A cost function in the two-dimensional parameter space.

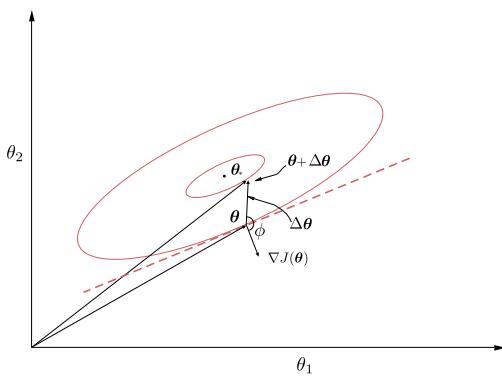
**FIGURE 5.2**

The corresponding isovalue curves of the cost function in Fig. 5.1, in the two-dimensional plane. All the points θ lying on the same (isovalue) ellipse score the same value for the cost $J(\theta)$. Note that as we move away from the optimal value, θ_* , the values of c increase.

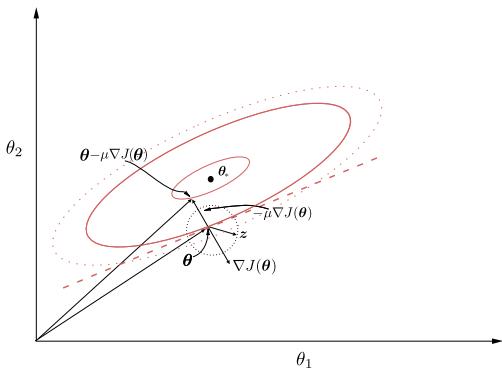
contour; after all, the first-order Taylor expansion holds approximately true for small deviations from $\theta^{(i-1)}$.

To address the first of the two issues, let us assume $\mu_i = 1$ and search for all vectors, z , with unit Euclidean norm and having their origin at $\theta^{(i-1)}$. Then it does not take long to see that for all possible directions, the one that gives the most negative value of the inner product, $\nabla J(\theta^{(i-1)})^T z$, is that of the negative gradient

$$z = -\frac{\nabla J(\theta^{(i-1)})}{\|\nabla J(\theta^{(i-1)})\|}.$$

**FIGURE 5.3**

The gradient vector at a point θ is perpendicular to the tangent plane (dotted line) at the isovalue curve crossing θ . The descent direction forms an obtuse angle, ϕ , with the gradient vector.

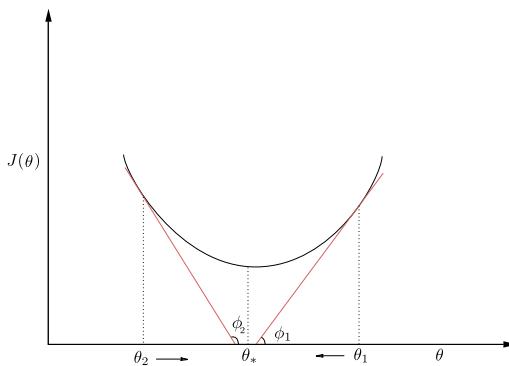
**FIGURE 5.4**

From all the descent directions of unit Euclidean norm (dotted circle centered at $\theta^{(i-1)}$, which for notational simplicity is shown as θ), the negative gradient one leads to the maximum decrease of the cost function.

This is illustrated in Fig. 5.4. Center the unit Euclidean norm ball at $\theta^{(i-1)}$. Then from all the unit norm vectors having their origin at $\theta^{(i-1)}$, choose the one pointing to the negative gradient direction. Thus, for all unit Euclidean norm vectors, the steepest descent direction coincides with the (negative) *gradient descent* direction, and the corresponding update recursion becomes

$$\theta^{(i)} = \theta^{(i-1)} - \mu_i \nabla J(\theta^{(i-1)}) : \text{ gradient descent scheme.} \quad (5.3)$$

Note that we still have to address the second point, concerning the choice of μ_i . The choice must be made in such a way to guarantee convergence of the minimizing sequence. We will come to this issue soon.

**FIGURE 5.5**

Once the algorithm is at θ_1 , the gradient descent will move the point to the left, towards the minimum. The opposite is true for point θ_2 .

Iteration (5.3) is illustrated in Fig. 5.5 for the one-dimensional case. If at the current iteration the algorithm has “landed” at θ_1 , then the derivative of $J(\theta)$ at this point is positive (the tangent of an acute angle, ϕ_1), and this will force the update to move to the left towards the minimum. The scenario is different if the current estimate was θ_2 . The derivative is negative (the tangent of an obtuse angle, ϕ_2) and this will push the update to the right, toward, again, the minimum. Note, however, that it is important how far to the left or to the right one has to move. A large move from, say, θ_1 to the left may land the update on the other side of the optimal value. In such a case, the algorithm may oscillate around the minimum and never converge. A major effort in this chapter will be devoted in providing theoretical frameworks for establishing bounds for the values of the step size that guarantee convergence.

The gradient descent method exhibits approximately *linear convergence*; that is, the error between $\theta^{(i)}$ and the true minimum converges to zero asymptotically in the form of a *geometric series*. However, the convergence rate depends heavily on the eigenvalue spread of the Hessian matrix of $J(\theta)$. The dependence of the convergence rate on the eigenvalues will be unraveled in Section 5.3. For large eigenvalue spreads, the rate of convergence can become extremely slow. On the other hand, the great advantage of the method lies in its low computational requirements.

Finally, it has to be pointed out that we arrived at the scheme in Eq. (5.3) by searching all directions via the unit Euclidean norm. However, there is nothing “sacred” around Euclidean norms. One can employ other norms, such as the ℓ_1 and quadratic $z^T P z$ norms, where P is a positive definite matrix. Under such choices, one will end up with alternative update iterations (see, e.g., [23]). We will return to this point in Chapter 6, when dealing with Newton’s iterative and coordinate descent minimization schemes.

5.3 APPLICATION TO THE MEAN-SQUARE ERROR COST FUNCTION

Let us apply the gradient descent scheme to derive an iterative algorithm to minimize our familiar cost function

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}[(y - \boldsymbol{\theta}^T \mathbf{x})^2] \\ &= \sigma_y^2 - 2\boldsymbol{\theta}^T \mathbf{p} + \boldsymbol{\theta}^T \Sigma_x \boldsymbol{\theta}, \end{aligned} \quad (5.4)$$

where $\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$ is the input covariance matrix and $\mathbf{p} = \mathbb{E}[\mathbf{xy}]$ is the input–output cross-correlation vector (Chapter 4). The respective cost function gradient with respect to $\boldsymbol{\theta}$ is easily seen to be (e.g., Appendix A)

$$\nabla J(\boldsymbol{\theta}) = 2\Sigma_x \boldsymbol{\theta} - 2\mathbf{p}. \quad (5.5)$$

In this chapter, we will also adhere to zero mean jointly distributed input–output random variables, except if otherwise stated. Thus, the covariance and correlation matrices coincide. If this is not the case, the covariance in (5.5) is replaced by the correlation matrix. The treatment is focused on real data and we will point out differences with the complex-valued data case whenever needed.

Employing (5.5), the update recursion in (5.3) becomes

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu (\Sigma_x \boldsymbol{\theta}^{(i-1)} - \mathbf{p}) \\ &= \boldsymbol{\theta}^{(i-1)} + \mu (\mathbf{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)}), \end{aligned} \quad (5.6)$$

where the step size has been considered constant and it has also absorbed the factor 2. The more general case of iteration dependent values of the step size will be discussed soon after. Our goal now becomes that of searching for all values of μ that guarantee convergence. To this end, define

$$\mathbf{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \quad (5.7)$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations (Chapter 4),

$$\Sigma_x \boldsymbol{\theta}_* = \mathbf{p}. \quad (5.8)$$

Subtracting $\boldsymbol{\theta}_*$ from both sides of (5.6) and plugging in (5.7), we obtain

$$\begin{aligned} \mathbf{c}^{(i)} &= \mathbf{c}^{(i-1)} + \mu (\mathbf{p} - \Sigma_x \mathbf{c}^{(i-1)} - \Sigma_x \boldsymbol{\theta}_*) \\ &= \mathbf{c}^{(i-1)} - \mu \Sigma_x \mathbf{c}^{(i-1)} = (I - \mu \Sigma_x) \mathbf{c}^{(i-1)}. \end{aligned} \quad (5.9)$$

Recall that Σ_x is a symmetric positive definite matrix (Chapter 2); hence all its eigenvalues are positive, and moreover (Appendix A.2) it can be written as

$$\Sigma_x = Q \Lambda Q^T, \quad (5.10)$$

where

$$\Lambda := \text{diag}\{\lambda_1, \dots, \lambda_l\} \quad \text{and} \quad Q := [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l],$$

with λ_j , \mathbf{q}_j , $j = 1, 2, \dots, l$, being the eigenvalues and the respective normalized (*orthogonal*) eigenvectors of the covariance matrix,¹ represented by

$$\mathbf{q}_k^T \mathbf{q}_j = \delta_{kj}, \quad k, j = 1, 2, \dots, l \implies Q^T = Q^{-1}.$$

That is, the matrix Q is orthogonal. Plugging the factorization of Σ_x into (5.9), we obtain

$$\mathbf{c}^{(i)} = Q(I - \mu\Lambda)Q^T \mathbf{c}^{(i-1)},$$

or

$$\mathbf{v}^{(i)} = (I - \mu\Lambda)\mathbf{v}^{(i-1)}, \quad (5.11)$$

where

$$\mathbf{v}^{(i)} := Q^T \mathbf{c}^{(i)}, \quad i = 1, 2, \dots \quad (5.12)$$

The previously used “trick” is a standard one and its aim is to “decouple” the various components of $\mathbf{v}^{(i)}$ in (5.6). Indeed, each one of the components $v^{(i)}(j)$, $j = 1, 2, \dots, l$, of $\mathbf{v}^{(i)}$ follows an iteration path, which is independent of the rest of the components; in other words,

$$\begin{aligned} v^{(i)}(j) &= (1 - \mu\lambda_j)v^{(i-1)}(j) = (1 - \mu\lambda_j)^2v^{(i-2)}(j) \\ &= \dots = (1 - \mu\lambda_j)^i v^{(0)}(j), \end{aligned} \quad (5.13)$$

where $v^{(0)}(j)$ is the j th component of $\mathbf{v}^{(0)}$, corresponding to the initial vector. It is readily seen that if

$$|1 - \mu\lambda_j| < 1 \iff -1 < 1 - \mu\lambda_j < 1, \quad j = 1, 2, \dots, l, \quad (5.14)$$

the geometric series tends to zero and

$$\mathbf{v}^{(i)} \rightarrow \mathbf{0} \implies Q^T(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*) \rightarrow \mathbf{0} \implies \boldsymbol{\theta}^{(i)} \rightarrow \boldsymbol{\theta}_*. \quad (5.15)$$

Note that (5.14) is equivalent to

$0 < \mu < 2/\lambda_{\max} : \quad \text{condition for convergence,}$

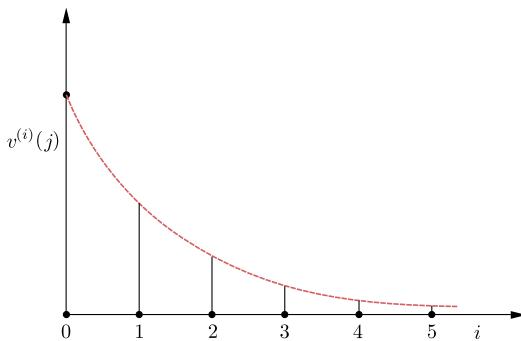
(5.16)

where λ_{\max} denotes the maximum eigenvalue of Σ_x .

Time constant: Fig. 5.6 shows a typical sketch of the evolution of $v^{(i)}(j)$ as a function of the iteration steps for the case $0 < 1 - \mu\lambda_j < 1$. Assume that the envelope, denoted by the red line, is (approximately) of an exponential form, $f(t) = \exp(-t/\tau_j)$. Plug into $f(t)$, as the values corresponding at the time instants $t = iT$ and $t = (i-1)T$, the values of $v^{(i)}(j)$, $v^{(i-1)}(j)$ from (5.13); then the *time constant* results as

$$\tau_j = \frac{-1}{\ln(1 - \mu\lambda_j)},$$

¹ In contrast to other chapters, we denote eigenvectors with \mathbf{q} and not as \mathbf{u} , since at some places the latter is used to denote the input random vector.

**FIGURE 5.6**

Convergence curve for one of the components of the transformed error vector. Note that the curve is of an approximate exponentially decreasing type.

assuming that the sampling time between two successive iterations is $T = 1$. For small values of μ , we can write

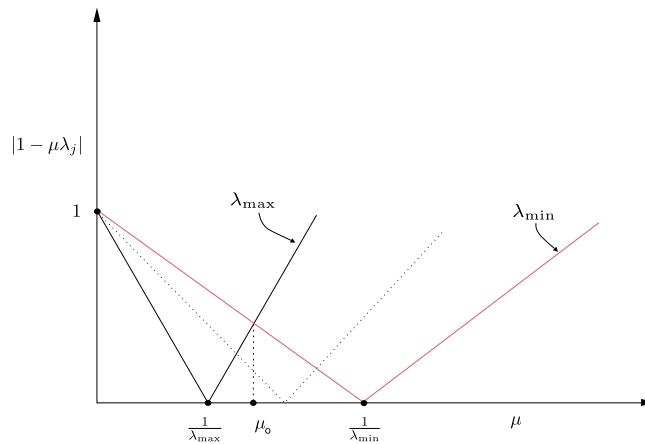
$$\tau_j \approx \frac{1}{\mu \lambda_j}, \quad \text{for } \mu \ll 1.$$

That is, the slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue. However, this is only true for small enough values of μ . For the more general case, this may not be true. Recall that the rate of convergence depends on the value of the term $1 - \mu \lambda_j$. This is also known as the *jth mode*. Its value depends not only on λ_j but also on μ . Let us consider as an example the case of μ taking a value very close to the maximum allowable one, $\mu \simeq 2/\lambda_{\max}$. Then the mode corresponding to the maximum eigenvalue will have an absolute value very close to one. On the other hand, the time constant of the mode corresponding to the minimum eigenvalue will be controlled by the value of $|1 - 2\lambda_{\min}/\lambda_{\max}|$, which can be much smaller than one. In such a case, the mode corresponding to the maximum eigenvalue exhibits slower convergence.

To obtain the optimum value for the step size, one has to select its value in such a way that the resulting maximum absolute mode value is minimized. This is a min/max task,

$$\begin{aligned} \mu_o &= \arg \min_{\mu} \max_j |1 - \mu \lambda_j|, \\ \text{s.t.} \quad & |1 - \mu \lambda_j| < 1, \quad j = 1, 2, \dots, l. \end{aligned}$$

The task can be solved easily graphically. Fig. 5.7 shows the absolute values of the modes (corresponding to the maximum, minimum, and an intermediate eigenvalue). The (absolute) values of the modes initially decrease as μ increases, and then they start increasing. Observe that the optimal value results at the point where the curves for the maximum and minimum eigenvalues intersect. Indeed, this corresponds to the minimum-maximum value. Moving μ away from μ_o , the maximum mode value increases; increasing μ_o , the mode corresponding to the maximum eigenvalue becomes larger, and decreasing it, the mode corresponding to the minimum eigenvalue is increased. At the intersection we have

**FIGURE 5.7**

For each mode, increasing the value of the step size, the time constant starts decreasing and then after a point starts increasing. The full black line corresponds to the maximum eigenvalue, the red one to the minimum, and the dotted curve to an intermediate eigenvalue. The overall optimal, μ_o , corresponds to the value where the red and full black curves intersect.

$$1 - \mu_o \lambda_{\min} = -(1 - \mu_o \lambda_{\max}),$$

which results in

$$\mu_o = \frac{2}{\lambda_{\max} + \lambda_{\min}}. \quad (5.17)$$

At the optimal value, μ_o , there are two slowest modes; one corresponding to λ_{\min} (i.e., $1 - \mu_o \lambda_{\min}$) and another one corresponding to λ_{\max} (i.e., $1 - \mu_o \lambda_{\max}$). They have equal magnitudes but opposite signs, and they are given by

$$\pm \frac{\rho - 1}{\rho + 1},$$

where

$$\rho := \frac{\lambda_{\max}}{\lambda_{\min}}.$$

In other words, the *convergence rate depends on the eigenvalues spread of the covariance matrix*.

Parameter error vector convergence: From the definitions in (5.7) and (5.12), we get

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}_* + Q \mathbf{v}^{(i)} \\ &= \boldsymbol{\theta}_* + [\mathbf{q}_1, \dots, \mathbf{q}_l] [v^{(i)}(1), \dots, v^{(i)}(l)]^T \\ &= \boldsymbol{\theta}_* + \sum_{k=1}^l \mathbf{q}_k v^{(i)}(k), \end{aligned} \quad (5.18)$$

or

$$\theta^{(i)}(j) = \theta_*(j) + \sum_{k=1}^l q_k(j) v^{(0)}(k) (1 - \mu \lambda_k)^i, \quad j = 1, 2, \dots, l. \quad (5.19)$$

In other words, the components of $\theta^{(i)}$ converge to the respective components of the optimum vector θ_* as a weighted average of powers of $1 - \mu \lambda_k$, i.e., $(1 - \mu \lambda_k)^i$. Computing the respective time constant in closed form is not possible; however, we can state lower and upper bounds. The lower bound corresponds to the time constant of the fastest converging mode and the upper bound to the slowest of the modes. For small values of $\mu \ll 1$, we can write

$$\frac{1}{\mu \lambda_{\max}} \leq \tau \leq \frac{1}{\mu \lambda_{\min}}. \quad (5.20)$$

The learning curve: We now turn our focus to the mean-square error (MSE). Recall from (4.8) that

$$J(\theta^{(i)}) = J(\theta_*) + (\theta^{(i)} - \theta_*)^T \Sigma_x (\theta^{(i)} - \theta_*), \quad (5.21)$$

or, mobilizing (5.18) and (5.10) and taking into consideration the orthonormality of the eigenvectors, we obtain

$$\begin{aligned} J(\theta^{(i)}) &= J(\theta_*) + \sum_{j=1}^l \lambda_j |v^{(i)}(j)|^2 \implies \\ J(\theta^{(i)}) &= J(\theta_*) + \sum_{j=1}^l \lambda_j (1 - \mu \lambda_j)^{2i} |v^{(0)}(j)|^2, \end{aligned} \quad (5.22)$$

which converges to the minimum value $J(\theta_*)$ asymptotically. Moreover, observe that this convergence is monotonic, because $\lambda_j (1 - \mu \lambda_j)^2$ is positive. Following similar arguments as before, the respective time constants for each one of the modes are now

$$\tau_j^{\text{mse}} = \frac{-1}{2 \ln(1 - \mu \lambda_j)} \approx \frac{1}{2\mu\lambda_j}. \quad (5.23)$$

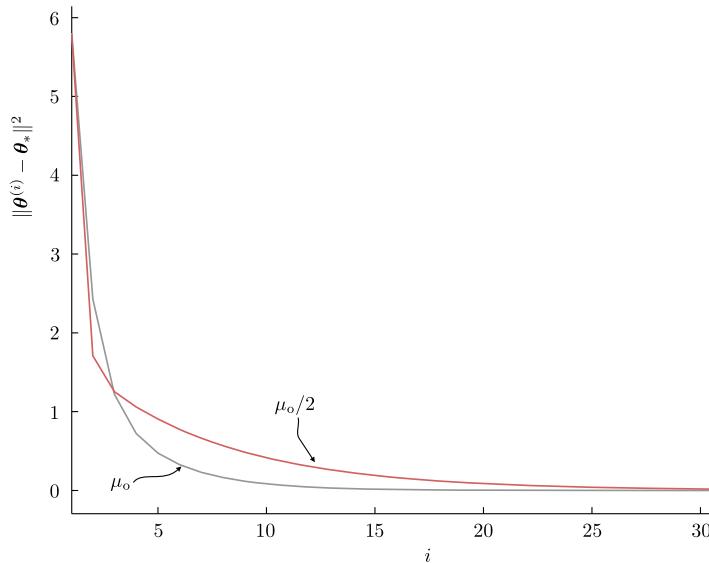
Example 5.1. The aim of the example is to demonstrate what we have said so far, concerning the convergence issues of the gradient descent scheme in (5.6). The cross-correlation vector was chosen to be

$$\mathbf{p} = [0.05, 0.03]^T,$$

and we consider two different covariance matrices,

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that for the case of Σ_2 , both eigenvalues are equal to 1, and for Σ_1 they are $\lambda_1 = 1$ and $\lambda_2 = 0.1$ (for diagonal matrices the eigenvalues are equal to the diagonal elements of the matrix).

**FIGURE 5.8**

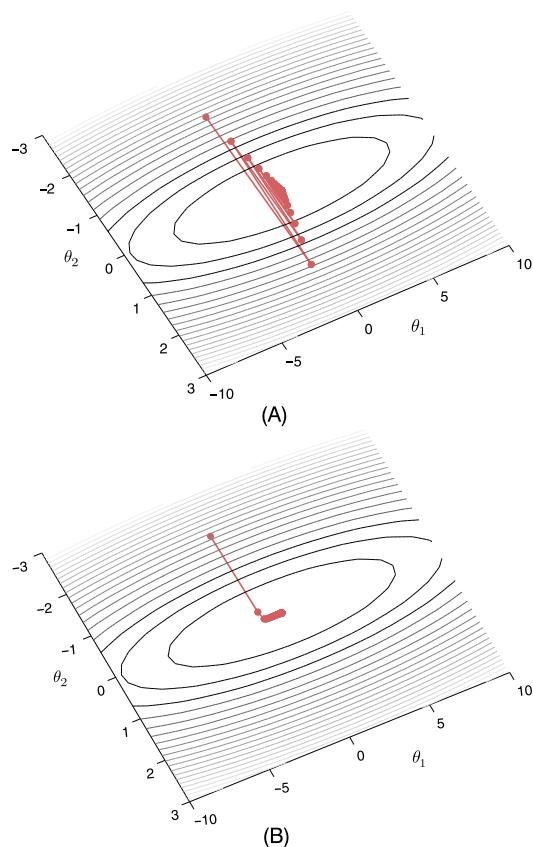
The black curve corresponds to the optimal value $\mu = \mu_o$ and the gray one to $\mu = \mu_o/2$, for the case of an input covariance matrix with unequal eigenvalues.

Fig. 5.8 shows the error curves for two values of μ for the case of Σ_1 ; the gray one corresponds to the optimum value ($\mu_o = 1.81$) and the red one to $\mu = \mu_o/2 = 0.9$. Observe the faster convergence towards zero that is achieved by the optimal value. Note that it may happen, as is the case in Fig. 5.8, that initially the convergence for some $\mu \neq \mu_o$ will be faster compared to μ_o . What the theory guarantees is that, eventually, the curve corresponding to the optimal will tend to zero faster than for any other value of μ . Fig. 5.9 shows the respective trajectories of the successive estimates in the two-dimensional space, together with the isovalue curves; the latter are ellipses, as we can readily deduce if we look carefully at the form of the quadratic cost function written as in (5.21). Compare the zig-zag path, which corresponds to the larger value of $\mu = 1.81$, with the smoother one, obtained for the smaller step size $\mu = 0.9$.

For comparison reasons, to demonstrate the dependence of the convergence speed on the eigenvalues spread, Fig. 5.10 shows the error curves using the same step size, $\mu = 1.81$, for both cases, Σ_1 and Σ_2 . Observe that large eigenvalues spread of the input covariance matrix slows down the convergence rate. Note that if the eigenvalues of the covariance matrix are equal to, say, λ , the isovalue curves are circles; the optimal step size in this case is $\mu = 1/\lambda$ and convergence is achieved in only one step (Fig. 5.11).

TIME-VARYING STEP SIZES

The previous analysis cannot be carried out for the case of an iteration dependent step size. It can be shown (Problem 5.2) that in this case, the gradient descent algorithm converges if

**FIGURE 5.9**

The trajectories of the successive estimates (dots) obtained by the gradient descent algorithm for (A) the larger value of $\mu = 1.81$ and (B) the smaller value of $\mu = 0.9$. In (B), the trajectory toward the minimum is smooth. In contrast, in (A), the trajectory consists of zig-zags.

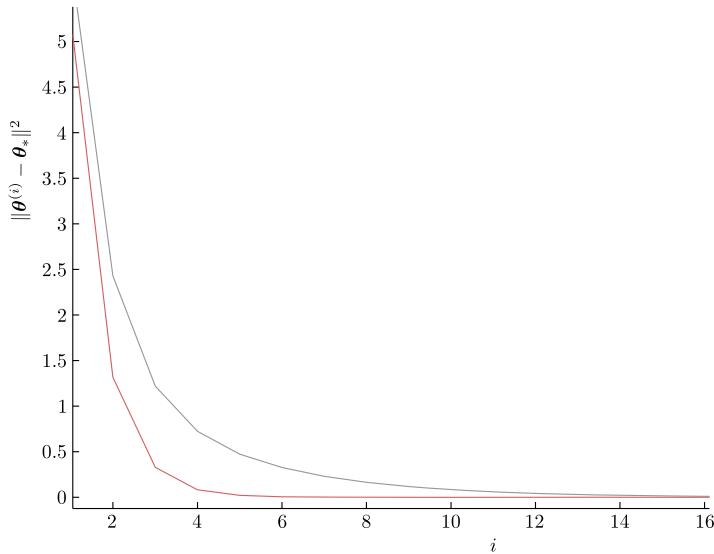
- $\mu_i \rightarrow 0$, as $i \rightarrow \infty$,
- $\sum_{i=1}^{\infty} \mu_i = \infty$.

A typical example of sequences which comply with both conditions are those that satisfy the following:

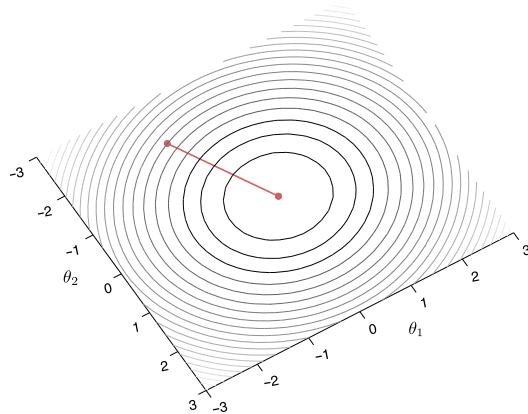
$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty, \quad (5.24)$$

as, for example, the sequence

$$\mu_i = \frac{1}{i}.$$

**FIGURE 5.10**

For the same value of $\mu = 1.81$, the error curves for the case of unequal eigenvalues ($\lambda_1 = 1$ and $\lambda_2 = 0.1$) (gray) and for equal eigenvalues ($\lambda_1 = \lambda_2 = 1$). For the latter case, the isovalue curves are circles; if the optimal value $\mu_o = 1$ is used, the algorithm converges in one step. This is demonstrated in Fig. 5.11.

**FIGURE 5.11**

When the eigenvalues of the covariance matrix are all equal to a value, λ , the use of the optimal $\mu_o = 1/\lambda$ achieves convergence in one step.

Note that the two (sufficient) conditions require that the sequence tends to zero, yet its infinite sum diverges. We will meet this pair of conditions in various parts of this book. The previous conditions

state that the step size has to become smaller and smaller as iterations progress, but this should not take place in a very aggressive manner, so that the algorithm is left to be active for a sufficient number of iterations to learn the solution. If the step size tends to zero very fast, then updates are practically frozen after a few iterations, without the algorithm having acquired enough information to get close to the solution.

5.3.1 THE COMPLEX-VALUED CASE

In Section 4.4.2 we stated that a function $f : \mathbb{C}^l \mapsto \mathbb{R}$ is not differentiable with respect to its complex argument. To deal with such cases, the Wirtinger calculus was introduced. In this section, we use this mathematically convenient tool to derive the corresponding steepest descent direction.

To this end, we again employ a first-order Taylor series approximation [22]. Let

$$\boldsymbol{\theta} = \boldsymbol{\theta}_r + j\boldsymbol{\theta}_i.$$

Then the cost function

$$J(\boldsymbol{\theta}) : \mathbb{C}^l \mapsto [0, +\infty)$$

is approximated as

$$\begin{aligned} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) &= J(\boldsymbol{\theta}_r + \Delta\boldsymbol{\theta}_r, \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i) \\ &= J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i) + \Delta\boldsymbol{\theta}_r^T \nabla_r J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i) + \Delta\boldsymbol{\theta}_i^T \nabla_i J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i), \end{aligned} \quad (5.25)$$

where ∇_r (∇_i) denotes the gradient with respect to $\boldsymbol{\theta}_r$ ($\boldsymbol{\theta}_i$). Taking into account that

$$\Delta\boldsymbol{\theta}_r = \frac{\Delta\boldsymbol{\theta} + \Delta\boldsymbol{\theta}^*}{2}, \quad \Delta\boldsymbol{\theta}_i = \frac{\Delta\boldsymbol{\theta} - \Delta\boldsymbol{\theta}^*}{2j},$$

it is easy to show (Problem 5.3) that

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \operatorname{Re}\{\Delta\boldsymbol{\theta}^H \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})\}, \quad (5.26)$$

where $\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})$ is the CW-derivative, defined in Section 4.4.2 as

$$\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}) = \frac{1}{2} (\nabla_r J(\boldsymbol{\theta}) + j \nabla_i J(\boldsymbol{\theta})).$$

Looking carefully at (5.26), it is straightforward to observe that the direction

$$\Delta\boldsymbol{\theta} = -\mu \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})$$

makes the updated cost equal to

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) - \mu \|\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})\|^2,$$

which guarantees that $J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) < J(\boldsymbol{\theta})$; it is straightforward to see, by taking into account the definition of an inner product, that the above search direction is the one of the largest decrease. Thus, the

counterpart of (5.3) becomes

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}^{(i-1)}) : \text{ complex gradient descent scheme.} \quad (5.27)$$

For the MSE cost function and for the linear estimation model, we get

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E} \left[(\mathbf{y} - \boldsymbol{\theta}^H \mathbf{x})(\mathbf{y} - \boldsymbol{\theta}^H \mathbf{x})^* \right] \\ &= \sigma_y^2 + \boldsymbol{\theta}^H \Sigma_x \boldsymbol{\theta} - \boldsymbol{\theta}^H \mathbf{p} - \mathbf{p}^H \boldsymbol{\theta}, \end{aligned}$$

and taking the gradient with respect to $\boldsymbol{\theta}^*$, by treating $\boldsymbol{\theta}$ as a constant (Section 4.4.2), we obtain

$$\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}) = \Sigma_x \boldsymbol{\theta} - \mathbf{p},$$

and the respective gradient descent iteration is the same as in (5.6).

5.4 STOCHASTIC APPROXIMATION

Solving for the normal equations (Eq. (5.8)) as well as using the gradient descent iterative scheme (for the case of the MSE), one has to have access to the second-order statistics of the involved variables. However, in most of the cases, this is not known and it has to be approximated using a set of observations. In this section, we turn our attention to algorithms that can learn the statistics iteratively via the training set. The origins of such techniques are traced back to 1951, when Robbins and Monro introduced the method of *stochastic approximation* [79], or the *Robbins–Monro algorithm*.

Let us consider the case of a function that is defined in terms of the expected value of another one, namely,

$$f(\boldsymbol{\theta}) = \mathbb{E} [\phi(\boldsymbol{\theta}, \boldsymbol{\eta})], \quad \boldsymbol{\theta} \in \mathbb{R}^l,$$

where $\boldsymbol{\eta}$ is a random vector of unknown distribution. The goal is to compute a root of $f(\boldsymbol{\theta})$. If the distribution was known, the expectation could be computed, at least in principle, and one could use any root-finding algorithm to compute the roots. The problem emerges when the statistics are unknown; hence the exact form of $f(\boldsymbol{\theta})$ is not known. All one has at her/his disposal is a sequence of i.i.d. observations $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \dots$. Robbins and Monro proved that the following algorithm,²

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \phi(\boldsymbol{\theta}_{n-1}, \boldsymbol{\eta}_n) : \text{ Robbins–Monro scheme,} \quad (5.28)$$

starting from an arbitrary initial condition, $\boldsymbol{\theta}_{-1}$, converges³ to a root of $f(\boldsymbol{\theta})$, under some general conditions and provided that (Problem 5.4)

² The original paper dealt with scalar variables only and the method was later extended to more general cases; see [96] for a related discussion.

³ Convergence here is meant to be in probability (see Section 2.6).

$$\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \rightarrow \infty : \text{ convergence conditions.} \quad (5.29)$$

In other words, in the iteration (5.28), we get rid of the expectation operation and use the value of $\phi(\cdot, \cdot)$, which is computed using the current observations and the currently available estimate. That is, the algorithm learns both the statistics as well as the root; two in one! The same comments made for the convergence conditions, met in the iteration dependent step size case in Section 5.3, are valid here as well.

In the context of optimizing a general differentiable cost function of the form

$$J(\boldsymbol{\theta}) = \mathbb{E} [\mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})], \quad (5.30)$$

the Robbins–Monro scheme can be mobilized to find a root of the respected gradient, i.e.,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E} [\nabla \mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})],$$

where the expectation is with respect to the pair (\mathbf{y}, \mathbf{x}) . As we have seen in Chapter 3, such cost functions are also known as the *expected risk* or the *expected loss* in machine learning terminology. Given the sequence of observations $(\mathbf{y}_n, \mathbf{x}_n)$, $n = 0, 1, \dots$, the recursion in (5.28) now becomes

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \nabla \mathcal{L}(\boldsymbol{\theta}_{n-1}, \mathbf{y}_n, \mathbf{x}_n). \quad (5.31)$$

Let us now assume, for simplicity, that the expected risk has a unique minimum, $\boldsymbol{\theta}_*$. Then, according to the Robbins–Monro theorem and using an appropriate sequence μ_n , $\boldsymbol{\theta}_n$ will converge to $\boldsymbol{\theta}_*$. However, although this information is important, it is not by itself enough. In practice, one has to cease iterations after a *finite* number of steps. Hence, one has to know something more concerning the rate of convergence of such a scheme. To this end, two quantities are of interest, namely, the mean and the covariance matrix of the estimator at iteration n , i.e.,

$$\mathbb{E}[\boldsymbol{\theta}_n], \text{ Cov}(\boldsymbol{\theta}_n).$$

It can be shown (see [67]) that if $\mu_n = \mathcal{O}(1/n)^4$ and assuming that iterations have brought the estimate close to the optimal value, then

$$\mathbb{E}[\boldsymbol{\theta}_n] = \boldsymbol{\theta}_* + \frac{1}{n} \mathbf{c} \quad (5.32)$$

and

$$\text{Cov}(\boldsymbol{\theta}_n) = \frac{1}{n} V + \mathcal{O}(1/n^2), \quad (5.33)$$

where \mathbf{c} and V are constants that depend on the form of the expected risk. The above formulas have also been derived under some further assumptions concerning the eigenvalues of the Hessian matrix of the expected risk.⁵ It is worth pointing out here that, in general, the convergence analysis of even

⁴ The symbol \mathcal{O} denotes order of magnitude.

⁵ The proof is a bit technical and the interested reader can look at the provided reference.

simple algorithms is a mathematically tough task, and it is common to carry it out under a number of assumptions. What is important to keep from (5.32) and (5.33) is that both the mean and the variances of the components follow an $\mathcal{O}(1/n)$ pattern (in (5.33), $\mathcal{O}(1/n)$ is the prevailing pattern, since the $\mathcal{O}(1/n^2)$ dependence goes to zero much faster). Furthermore, these formulas indicate that the parameter vector estimate *fluctuates* around the optimal value. Indeed, in contrast to Eq. (5.15), where $\boldsymbol{\theta}^{(i)}$ converges to the optimal value, here it is the corresponding expected value that converges. The spread around the expected value is controlled by the respective covariance matrix.

The previously reported fluctuation depends on the choice of the sequence μ_n , being smaller for smaller values of the step size sequence. However, μ_n cannot be made to decrease very fast due to the two convergence conditions, as discussed before. This is the price one pays for using the *noisy version of the gradient* and it is the reason that such schemes suffer from relatively slow convergence rates. However, this does not mean that such schemes are, necessarily, the poor relatives of other more “elaborate” algorithms. As we will discuss in Chapter 8, their low complexity requirements make this algorithmic family to be the preferred choice in a number of practical applications.

APPLICATION TO THE MSE LINEAR ESTIMATION

Let us apply the Robbins–Monro algorithm to solve for the optimal MSE linear estimator if the covariance matrix and the cross-correlation vector are unknown. We know that the solution corresponds to the root of the gradient of the cost function, which can be written in the form (recall the orthogonality theorem from Chapter 3 and Eq. (5.8))

$$\Sigma_x \boldsymbol{\theta} - \mathbf{p} = \mathbb{E}[\mathbf{x}(\mathbf{x}^T \boldsymbol{\theta} - \mathbf{y})] = \mathbf{0}.$$

Given the training sequence of observations, (y_n, \mathbf{x}_n) , which are assumed to be i.i.d. drawn from the joint distribution of (\mathbf{y}, \mathbf{x}) , the Robbins–Monro algorithm becomes

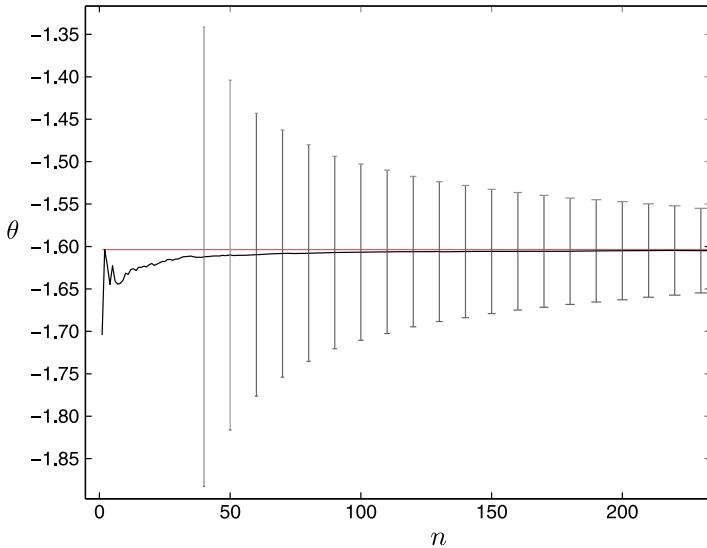
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \mathbf{x}_n (y_n - \mathbf{x}_n^T \boldsymbol{\theta}_{n-1}), \quad (5.34)$$

which converges to the optimal MSE solution provided that the two conditions in (5.29) are satisfied. Compare (5.34) with (5.6). Taking into account the definitions, $\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$, $\mathbf{p} = \mathbb{E}[\mathbf{xy}]$, the former equation results from the latter one by dropping out the expectation operations and using an iteration dependent step size. Observe that the iterations in (5.34) coincide with *time updates*; time has now explicitly entered into the scene. This prompts us to start thinking about modifying such schemes appropriately to track time-varying environments. Algorithms such as the one in (5.34), which result from the generic gradient descent formulation by replacing the expectation by the respective instantaneous observations, are also known as *stochastic gradient descent schemes*.

Remarks 5.1.

- All the algorithms to be derived next can also be applied to nonlinear estimation/filtering tasks of the form

$$\hat{\mathbf{y}} = \sum_{k=1}^l \theta_k \phi_k(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi},$$

**FIGURE 5.12**

The red line corresponds to the true value of the unknown parameter. The black curve corresponds to the average over 200 realizations of the experiment. Observe that the mean value converges to the true value. The bars correspond to the respective standard deviation, which keeps decreasing as n grows.

and the place of \mathbf{x} is taken by $\boldsymbol{\phi}$, where

$$\boldsymbol{\phi} = [\phi_1(\mathbf{x}), \dots, \phi_l(\mathbf{x})]^T.$$

Example 5.2. The aim of this example is to demonstrate the pair of Eqs. (5.32) and (5.33), which characterize the convergence properties of the stochastic gradient scheme.

Data samples were first generated according to the regression model

$$y_n = \boldsymbol{\theta}^T \mathbf{x}_n + \eta_n,$$

where $\boldsymbol{\theta} \in \mathbb{R}^2$ was randomly chosen and then fixed. The elements of \mathbf{x}_n were i.i.d. generated via a normal distribution $\mathcal{N}(0, 1)$ and η_n are samples of a white noise sequence with variance equal to $\sigma^2 = 0.1$. Then the observations (y_n, \mathbf{x}_n) were used in the recursive scheme in (5.34) to obtain an estimate of $\boldsymbol{\theta}$. The experiment was repeated 200 times and the mean and variance of the obtained estimates were computed for each iteration step. Fig. 5.12 shows the resulting curve for one of the parameters (the trend for the other one being similar). Observe that the mean values of the estimates tend to the true value, corresponding to the red line, and the standard deviation keeps decreasing as n grows. The step size was chosen equal to $\mu_n = 1/n$.

5.5 THE LEAST-MEAN-SQUARES ADAPTIVE ALGORITHM

The stochastic gradient descent algorithm in (5.34) converges to the optimal MSE solution provided that μ_n satisfies the two convergence conditions. Once the algorithm has converged, it “locks” at the obtained solution. In a case where the statistics of the involved variables/processes and/or the unknown parameters start changing, the algorithm cannot track the changes. Note that if such changes occur, the error term

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$$

will attain larger values; however, because μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate at time n . This can be overcome if one sets the value of μ_n to a preselected *fixed* value, μ . The resulting algorithm is the celebrated least-mean-squares (LMS) algorithm [102].

Algorithm 5.1 (The LMS algorithm).

- Initialize
 - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$; other values can also be used.
 - Select the value of μ .
- **For** $n = 0, 1, \dots$, **Do**
 - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$
 - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu e_n \mathbf{x}_n$
- **End For**

In case the input is a time series,⁶ u_n , the initialization also involves the samples, $u_{-1}, \dots, u_{-l+1} = 0$, to form the input vectors, $\mathbf{u}_n, n = 0, 1, \dots, l - 2$. The complexity of the algorithm amounts to $2l$ multiplications/additions (MADs) per time update. We have assumed that observations start arriving at time instant $n = 0$, to be in line with most references treating the LMS.

Let us now comment on this simple structure. Assume that the algorithm has converged close to the solution; then the error term is expected to take small values and thus the updates will remain close to the solution. If the statistics and/or the system parameters now start changing, the error values are expected to increase. Given that μ has a constant value, the algorithm has now the “agility” to update the estimates in an attempt to “push” the error to lower values. This small variation of the iterative scheme has important implications. The resulting algorithm is no more a member of the Robbins–Monro stochastic approximation family. Thus, one has to study its convergence conditions as well as its performance properties. Moreover, since the algorithm now has the potential to track changes in the values of the underlying parameters, as well as the statistics of the involved processes/variables, one has to study its performance in nonstationary environments; this is associated to what is known as the *tracking* performance of the algorithm, and it will be treated at the end of the chapter.

⁶ Recall our adopted notation from Chapter 2; in this case we use \mathbf{u}_n in place of \mathbf{x}_n .

5.5.1 CONVERGENCE AND STEADY-STATE PERFORMANCE OF THE LMS IN STATIONARY ENVIRONMENTS

The goal of this subsection is to study the performance of the LMS in stationary environments, that is, to answer the following questions. (a) Does the scheme converge and under which conditions? And if it converges, where does it converge? Although we introduced the scheme having in mind nonstationary environments, we still have to know how it behaves under stationarity; after all, the environment can change very slowly, and it can be considered “locally” stationary.

The convergence properties of the LMS, as well as of any other online/adaptive algorithm, are related to its *transient* characteristics; that is, the period from the initial estimate until the algorithm reaches a “steady-state” mode of operation. In general, analyzing the transient performance of an online algorithm is a formidable task indeed. This is also true even for the very simple structure of the LMS summarized in Algorithm 5.1. The LMS update recursions are equivalent to a time-varying, nonlinear (Problem 5.5), and stochastic in nature estimator. Many papers, some of them of high scientific insight and mathematical skill, have been produced. However, with the exception of a few rare and special cases, the analysis involves approximations. Our goal in this book is not to treat this topic in detail. Our focus will be restricted to the most “primitive” of the techniques, which are easier for the reader to follow compared with more advanced and mathematically elegant theories; after all, even this primitive approach provides results that turn out to be in agreement with what one experiences in practice.

Convergence of the Parameter Error Vector

Define

$$\mathbf{c}_n := \boldsymbol{\theta}_n - \boldsymbol{\theta}_*,$$

where $\boldsymbol{\theta}_*$ is the optimal solution resulting from the normal equations. The LMS update recursion can now be written as

$$\mathbf{c}_n = \mathbf{c}_{n-1} + \mu \mathbf{x}_n (y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n + \boldsymbol{\theta}_*^T \mathbf{x}_n - \boldsymbol{\theta}_*^T \mathbf{x}_n).$$

Because we are going to study the statistical properties of the obtained estimates, we have to switch our notation from that referring to observations to the one involving the respective random variables. Then we can write

$$\begin{aligned} \mathbf{c}_n &= \mathbf{c}_{n-1} + \mu \mathbf{x} (y - \boldsymbol{\theta}_{n-1}^T \mathbf{x} + \boldsymbol{\theta}_*^T \mathbf{x} - \boldsymbol{\theta}_*^T \mathbf{x}) \\ &= \mathbf{c}_{n-1} - \mu \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} + \mu \mathbf{x} \mathbf{e}_* \\ &= (I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1} + \mu \mathbf{x} \mathbf{e}_*, \end{aligned} \tag{5.35}$$

where

$$\mathbf{e}_* = y - \boldsymbol{\theta}_*^T \mathbf{x} \tag{5.36}$$

is the error random variable associated with the optimal $\boldsymbol{\theta}_*$. Compare (5.35) with (5.9). They look similar, yet they are very different. First, the latter of the two involves the expected value, Σ_x , in place of the respective variables. Moreover in (5.35), there is a second term that acts as an external input to

the difference stochastic equation. From (5.35) we obtain

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}\left[(I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1}\right] + \mu \mathbb{E}[\mathbf{x} \mathbf{e}_*]. \quad (5.37)$$

To proceed, it is time to introduce assumptions.

Assumption 1. The involved random variables are jointly linked via the regression model,

$$\mathbf{y} = \boldsymbol{\theta}_o^T \mathbf{x} + \eta, \quad (5.38)$$

where η is the noise variable with variance σ_η^2 and it is assumed to be independent of \mathbf{x} . Moreover, successive samples η_n , which generate the data, are assumed to be i.i.d. We have seen in Remarks 4.2 and Problem 4.4 that in this case, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$ and $\sigma_{e_*}^2 = \sigma_\eta^2$. Also, due to the orthogonality condition, $\mathbb{E}[\mathbf{x} \mathbf{e}_*] = \mathbf{0}$. In addition, a stronger condition will be adopted, and \mathbf{e}_* and \mathbf{x} will be assumed to be *statistically independent*. This is justified by the fact that under the above model, $e_{*,n} = \eta_n$, and the noise sequence has been assumed to be independent of the input.

Assumption 2. (Independence assumption) Assume that \mathbf{c}_{n-1} is statistically independent of both \mathbf{x} and \mathbf{e}_* . No doubt this is a strong assumption, but one we will adopt to simplify computations. Sometimes there is a tendency to “justify” this assumption by resorting to some special cases, which we will not do. If one is not happy with the assumption, he/she has to look for more recent methods, based on more rigorous mathematical analysis; of course, this does not mean that such methods are free of assumptions.

I. *Convergence in the mean:* Having adopted the previous assumptions, (5.37) becomes

$$\begin{aligned} \mathbb{E}[\mathbf{c}_n] &= \mathbb{E}\left[\left(I - \mu \mathbf{x} \mathbf{x}^T\right) \mathbf{c}_{n-1}\right] \\ &= (I - \mu \Sigma_x) \mathbb{E}[\mathbf{c}_{n-1}]. \end{aligned} \quad (5.39)$$

Following similar arguments as in Section 5.3, we obtain

$$\mathbb{E}[\mathbf{v}_n] = (I - \mu \Lambda) \mathbb{E}[\mathbf{v}_{n-1}],$$

where $\Sigma_x = Q \Lambda Q^T$ and $\mathbf{v}_n = Q^T \mathbf{c}_n$. The last equation leads to

$$\mathbb{E}[\boldsymbol{\theta}_n] \longrightarrow \boldsymbol{\theta}_* \quad \text{as } n \longrightarrow \infty,$$

provided that

$$0 < \mu < \frac{2}{\lambda_{\max}}.$$

In other words, in a stationary environment, the LMS converges to the optimal MSE solution *in the mean*. Thus, by fixing the value of the step size to be a constant, we lose something; the obtained estimates, even after convergence, hover around the optimal solution. The obvious task to be considered next is to study the respective covariance matrix.

II. *Error vector covariance matrix:* From (5.39), applying it recursively and assuming for the initial condition that $\mathbb{E}[\mathbf{c}_{-1}] = \mathbf{0}$, we have $\mathbb{E}[\mathbf{c}_n] = \mathbf{0}$. In any case, borrowing the same arguments used

in establishing the convergence in the mean, it turns out that the latter is approximately true for large enough values of n , irrespective of the initialization. Thus, from (5.35) we get

$$\begin{aligned}\Sigma_{c,n} := \mathbb{E}[\mathbf{c}_n \mathbf{c}_n^T] &= \Sigma_{c,n-1} - \mu \mathbb{E}[\mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} \mathbf{c}_{n-1}^T] \\ &\quad - \mu \mathbb{E}[\mathbf{c}_{n-1} \mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T] + \mu^2 \mathbb{E}[\mathbf{e}_*^2 \mathbf{x} \mathbf{x}^T] \\ &\quad + \mu^2 \mathbb{E}[\mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} \mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T],\end{aligned}\tag{5.40}$$

where we have used the independence between \mathbf{e}_* and \mathbf{c}_{n-1} and the fact that \mathbf{e}_* is orthogonal to \mathbf{x} in order to set to zero some of the terms. Taking into consideration the adopted independence assumptions and assuming that the input vector follows a *Gaussian distribution*, (5.40) becomes

$$\begin{aligned}\Sigma_{c,n} &= \Sigma_{c,n-1} - \mu \Sigma_x \Sigma_{c,n-1} - \mu \Sigma_{c,n-1} \Sigma_x \\ &\quad + 2\mu^2 \Sigma_x \Sigma_{c,n-1} \Sigma_x + \mu^2 \Sigma_x \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} \\ &\quad + \mu^2 \sigma_\eta^2 \Sigma_x,\end{aligned}\tag{5.41}$$

where the Gaussian assumption has been exploited to express the term involving fourth-order moments (e.g., [74]) as

$$\mathbb{E}[\mathbf{x} \mathbf{x}^T \Sigma_{c,n-1} \mathbf{x} \mathbf{x}^T] = 2 \Sigma_x \Sigma_{c,n-1} \Sigma_x + \Sigma_x \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Mobilizing the definition of $\mathbf{v}_n = Q^T \mathbf{c}_n$, (5.41) results in (Problem 5.6)

$$\begin{aligned}\Sigma_{v,n} &= Q^T \Sigma_{c,n} Q = \Sigma_{v,n-1} - \mu \Lambda \Sigma_{v,n-1} - \mu \Sigma_{v,n-1} \Lambda \\ &\quad + 2\mu^2 \Lambda \Sigma_{v,n-1} \Lambda + \mu^2 \Lambda \text{trace}\{\Lambda \Sigma_{v,n-1}\} + \mu^2 \sigma_\eta^2 \Lambda.\end{aligned}\tag{5.42}$$

Note that our interest lies at the diagonal elements of $\Sigma_{v,n}$, since these correspond to the variances of the respective elements of $\boldsymbol{\theta}_n - \boldsymbol{\theta}_*$, and correspondingly to $\mathbf{v}_n - \mathbf{v}_*$. Collecting all the *diagonal* elements in a vector, \mathbf{s}_n , a close inspection of the diagonal elements of $\Sigma_{v,n}$ in (5.42) can persuade the reader that the following difference equation is true:

$$\mathbf{s}_n = (I - 2\mu \Lambda + 2\mu^2 \Lambda^2 + \mu^2 \boldsymbol{\lambda} \boldsymbol{\lambda}^T) \mathbf{s}_{n-1} + \mu^2 \sigma_\eta^2 \boldsymbol{\lambda},\tag{5.43}$$

where

$$\boldsymbol{\lambda} := [\lambda_1, \lambda_2, \dots, \lambda_l]^T.$$

It is well known from linear system theory that the difference equation in (5.43) is stable if the eigenvalues of the matrix,

$$\begin{aligned}A &:= I - 2\mu \Lambda + 2\mu^2 \Lambda^2 + \mu^2 \boldsymbol{\lambda} \boldsymbol{\lambda}^T \\ &= (I - \mu \Lambda)^2 + \mu^2 \Lambda^2 + \mu^2 \boldsymbol{\lambda} \boldsymbol{\lambda}^T,\end{aligned}\tag{5.44}$$

have magnitude less than one. This can be guaranteed if the step size μ is chosen such as (Problem 5.7)

$$0 < \mu < \frac{2}{\sum_{i=1}^l \lambda_i},$$

or

$$\boxed{\mu < \frac{2}{\text{trace}\{\Sigma_x\}}.} \quad (5.45)$$

The last condition guarantees that the variances remain *bounded*. Recall the number of assumptions made. Thus, to be on the safe side, μ must be selected so that it is not close to this upper bound.

III. Excess mean-square error: We know that the minimum MSE is achieved at θ_* . Any other weight vector results in higher values of the MSE. We have already said that in the steady state, the estimates obtained via the LMS fluctuate randomly around θ_* ; thus, the MSE will be larger than the minimum J_{\min} . This “extra” error power, denoted as J_{exc} , is known as the *excess* MSE. Also, the ratio

$$\mathcal{M} := \frac{J_{\text{exc}}}{J_{\min}}$$

is known as the *misadjustment*. No doubt, we should seek for the relationship of \mathcal{M} with μ and in practice we would like to adjust μ accordingly, in order to get a value of \mathcal{M} as small as possible. Unfortunately, we will soon see that there is a tradeoff in achieving that. Making \mathcal{M} small, the convergence speed becomes slower and vice versa; there is no free lunch!

By the respective definitions, we have⁷

$$\begin{aligned} \mathbf{e}_n &= \mathbf{y}_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x} \\ &= \mathbf{e}_{*,n} - \mathbf{c}_{n-1}^T \mathbf{x}, \end{aligned}$$

or

$$\mathbf{e}_n^2 = \mathbf{e}_{*,n}^2 + \mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} - 2\mathbf{e}_{*,n} \mathbf{c}_{n-1}^T \mathbf{x}. \quad (5.46)$$

Taking the expectation on both sides and exploiting the assumed independence between \mathbf{c}_{n-1} and \mathbf{x} and $\mathbf{e}_{*,n}$, as well as the orthogonality between $\mathbf{e}_{*,n}$ and \mathbf{x} , we get

$$\begin{aligned} J_n := \mathbb{E}[\mathbf{e}_n^2] &= J_{\min} + \mathbb{E}[\mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1}] \\ &= J_{\min} + \mathbb{E}[\text{trace}\{\mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1}\}] \\ &= J_{\min} + \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}, \end{aligned} \quad (5.47)$$

where the property $\text{trace}\{AB\} = \text{trace}\{BA\}$ has been used. Thus, we can finally write

$$\boxed{J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} : \text{excess MSE at time instant } n.} \quad (5.48)$$

⁷ The time index n is explicitly used for \mathbf{e} , \mathbf{e}_* , \mathbf{y} , since the formula to be derived is also valid for time-varying environments, and it is going to be used later on for the time-varying statistics case.

Let us now elaborate on it a bit more. Taking into account that $QQ^T = I$, we get

$$\begin{aligned} J_{\text{exc},n} &= \text{trace}\{QQ^T \Sigma_x QQ^T \Sigma_{v,n-1} QQ^T\} \\ &= \text{trace}\{Q \Lambda \Sigma_{v,n-1} Q^T\} = \text{trace}\{\Lambda \Sigma_{v,n-1}\} \\ &= \sum_{i=1}^l \lambda_i [\Sigma_{v,n-1}]_{ii} = \lambda^T s_{n-1}, \end{aligned} \quad (5.49)$$

where s_n is the vector of the diagonal elements of $\Sigma_{v,n}$ and obeys the difference equation in (5.43). Assuming that μ has been chosen so that convergence is guaranteed, then for large values of n the *steady state* has been reached. In a more formal way, an online algorithm has reached the steady state if

$$\mathbb{E}[\theta_n] = \mathbb{E}[\theta_{n-1}] = \text{Constant}, \quad (5.50)$$

$$\Sigma_{\theta,n} = \Sigma_{\theta,n-1} = \text{Constant}. \quad (5.51)$$

Thus, in steady state, we assume in Eq. (5.43) that $s_n = s_{n-1}$; if this is exploited in (5.49) this leads to (Problem 5.10)

$$J_{\text{exc},\infty} := \lim_{n \rightarrow \infty} J_{\text{exc},n} \simeq \frac{\mu \sigma_\eta^2 \text{trace}\{\Sigma_x\}}{2 - \mu \text{trace}\{\Sigma_x\}}, \quad (5.52)$$

and for the misadjustment (since under our assumptions $J_{\min} = \sigma_\eta^2$),

$$\mathcal{M} \simeq \frac{\mu \text{trace}\{\Sigma_x\}}{2 - \mu \text{trace}\{\Sigma_x\}},$$

which, for small values of μ , leads to

$$J_{\text{exc},\infty} \simeq \frac{1}{2} \mu \sigma_\eta^2 \text{trace}\{\Sigma_x\} : \text{ excess MSE} \quad (5.53)$$

and

$$\mathcal{M} \simeq \frac{1}{2} \mu \text{trace}\{\Sigma_x\} : \text{ misadjustment.} \quad (5.54)$$

That is, the smaller the value of μ , the smaller the excess MSE.

IV. Time constant. Note that the transient behavior of the LMS is described by the difference equation in (5.43) and its convergence rate (the speed with which it forgets the initial conditions until it settles to its steady-state operation) depends on the eigenvalues of A in (5.44). To simplify the formulas, assume μ to be small enough so that A is approximated as $(I - \mu \Lambda)^2$. Following similar arguments as those used for the gradient descent in Section 5.3, we can write

$$\tau_j^{LMS} \simeq \frac{1}{2\mu\lambda_j}.$$

That is, the time constant for each one of the modes is inversely proportional to μ . Hence, the slower the rate of convergence (small values of μ), the lower the misadjustment and vice versa. Viewing it

differently, the more time the algorithm spends on learning, prior to reaching the steady state, the smaller is its deviation from the optimal.

5.5.2 CUMULATIVE LOSS BOUNDS

In the previously reported analysis method for the LMS performance, there is an underlying assumption that the training samples are generated by a linear model. The focus of the analysis was to investigate how well our algorithm estimates the unknown model once steady state has been reached. This path of analysis is very popular and well suited for a number of tasks, such as system identification.

An alternative route for studying the performance of an algorithm, shedding light from a different angle, is via the so-called *cumulative loss*. Recall that the main goal in machine learning is *prediction*; hence, measuring the prediction accuracy of an algorithm, given a set of observations, becomes the main goal. However, this performance index should be measured against the generalization ability of the algorithm, as pointed out in Chapter 3. In practice, this can be done in different ways, such as via the leave-one-out method (Section 3.13).

For the case of the squared error loss function, the cumulative loss over N observation samples is defined as

$$\mathcal{L}_{\text{cum}} = \sum_{n=0}^{N-1} (y_n - \hat{y}_n)^2 = \sum_{n=0}^{N-1} (y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n)^2 : \quad \text{cumulative loss.} \quad (5.55)$$

Note that $\boldsymbol{\theta}_{n-1}$ has been estimated based on observations up to and including the time instant $n-1$. So, the training pair (y_n, \mathbf{x}_n) can be considered as a test sample, not involved in the training, for measuring the error. It must be pointed out, however, that the cumulative loss is *not* a direct measure of the generalization performance associated with the finally obtained parameter vector. Such a measure should involve $\boldsymbol{\theta}_{N-1}$, tested against a number of test samples.

The goal of the family of methods which build around the cumulative loss is to derive corresponding upper bounds. Our aim here is to outline the essence behind such approaches, without resorting to proofs; these comprise a series of bounds and can become a bit technical. The interested reader can consult the related references. We will return to the cumulative loss and related bounds in the context of the so-called *regret analysis* in Chapter 8.

In the context of the LMS, the following theorem has been proved in [21].

Theorem 5.1. *Let $C = \max_n \|\mathbf{x}_n\|$, $\mu = \beta/C^2$, $0 < \beta < 2$. Then the set of predictions, $\hat{y}_0, \dots, \hat{y}_{N-1}$, generated by Algorithm 5.1 satisfies the following bound:*

$$\sum_{n=0}^{N-1} (y_n - \hat{y}_n)^2 \leq \inf_{\boldsymbol{\theta}} \left\{ \frac{C^2 \|\boldsymbol{\theta}\|^2}{2\beta(1-\beta)c} + \frac{\mathcal{L}(\boldsymbol{\theta}, S)}{(2-\beta)^2 c(1-c)} \right\}, \quad (5.56)$$

where $0 < c < 1$ and

$$\mathcal{L}(\boldsymbol{\theta}, S) = \sum_{n=0}^{N-1} (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2, \quad (5.57)$$

where $S = \{(y_n, \mathbf{x}_n), n = 0, 1, \dots, N-1\}$.

The numerator is the total squared estimation error. The denominator involves two terms. One is the noise/disturbance energy, and the other is the norm of the unknown parameter vector. Assuming that one starts iterations from $\boldsymbol{\theta}_{-1} = \mathbf{0}$, this term measures the energy of the disturbance from the initial guess. It turns out that the LMS scheme is the one that minimizes the following cost:

$$\gamma_{\text{opt}}^2 = \inf_{\{\hat{s}_{n|n-1}\}} \sup_{\{\boldsymbol{\theta}, \eta_n\}} \left(\frac{\sum_{n=0}^{\infty} |\hat{s}_{n|n-1} - s_n|^2}{\mu^{-1} \|\boldsymbol{\theta}\|^2 + \sum_{n=0}^{\infty} |\eta_n|^2} \right).$$

Moreover, it turns out that the optimum corresponds to $\gamma_{\text{opt}}^2 = 1$ [46,83]. Note that, basically, the LMS optimizes a worst-case scenario. It makes the estimation error minimum under the worst (maximum) disturbance circumstances. This type of optimality explains the robust performance of the LMS under “nonideal” environments, which are often met in practice, where a number of modeling assumptions are not valid. Such deviations from the model can be accommodated in η_n , which then “loses” its i.i.d., white, Gaussian, or any other mathematically attractive property.

Finally, it is interesting to point out the similarity of the bound in (5.61) to that in (5.56). Indeed, in the former we make the following substitutions:

$$\hat{s}_{n|n-1} = \hat{y}_n, \quad s_n = y_n, \quad \eta_n = y_n - \boldsymbol{\theta}^T \mathbf{x}_n.$$

Ignoring the values of the constants, the involved quantities are the same. After all, H^∞ is about maximizing the worst-case scenario [55].

5.6 THE AFFINE PROJECTION ALGORITHM

As will soon be verified in the simulations section, a major drawback of the basic LMS scheme is its fairly slow convergence speed. In an attempt to improve upon it, a number of variants have been proposed over the years. The *affine projection algorithm* (APA) belongs to the so-called *data reusing* family, where, at each time instant, past data are reused. Such a rationale helps the algorithm to “learn faster” and improve the convergence speed. However, besides the increased complexity, the faster convergence speed is achieved at the *expense of an increased misadjustment level*.

The APA was proposed originally in [48] and later on in [72]. Let the currently available estimate be $\boldsymbol{\theta}_{n-1}$. According to APA, the updated estimate, $\boldsymbol{\theta}$, must satisfy the following constraints:

$$\mathbf{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \dots, q-1.$$

In other words, we *force* the parameter vector $\boldsymbol{\theta}$ to provide at its output the desired response samples, for the q most recent time instants, where q is a user-defined parameter. At the same time, APA requires $\boldsymbol{\theta}$ to be as close as possible, in the Euclidean norm sense, to the current estimate, $\boldsymbol{\theta}_{n-1}$. Thus, APA, at each time instant, solves the following constrained optimization task:

$$\begin{aligned} \boldsymbol{\theta}_n &= \arg \min_{\boldsymbol{\theta}} \|\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}\|^2 \\ \text{s.t.} \quad \mathbf{x}_{n-i}^T \boldsymbol{\theta} &= y_{n-i}, \quad i = 0, 1, \dots, q-1. \end{aligned} \tag{5.62}$$

If one defines the $q \times l$ matrix

$$X_n = \begin{bmatrix} \mathbf{x}_n^T \\ \vdots \\ \mathbf{x}_{n-q+1}^T \end{bmatrix},$$

then the set of constraints can be compactly written as

$$X_n \boldsymbol{\theta} = \mathbf{y}_n,$$

where

$$\mathbf{y}_n = [y_n \dots y_{n-q+1}]^T.$$

Using Lagrange multipliers in (5.62) (Appendix C) results in (Problem 5.11)

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + X_n^T (X_n X_n^T)^{-1} \mathbf{e}_n, \quad (5.63)$$

$$\mathbf{e}_n = \mathbf{y}_n - X_n \boldsymbol{\theta}_{n-1}, \quad (5.64)$$

provided that $X_n X_n^T$ is invertible. The resulting scheme is summarized in Algorithm 5.2.

Algorithm 5.2 (The affine projection algorithm).

- Initialize
 - $\mathbf{x}_{-1} = \dots = \mathbf{x}_{-q+1} = \mathbf{0}$, $y_{-1} \dots y_{-q+1} = 0$
 - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$ (or any other value).
 - Choose $0 < \mu < 2$ and δ to be small.
- **For** $n = 0, 1, \dots$, **Do**
 - $\mathbf{e}_n = \mathbf{y}_n - X_n \boldsymbol{\theta}_{n-1}$
 - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu X_n^T (\delta I + X_n X_n^T)^{-1} \mathbf{e}_n$
- **End For**

When the input is a time series, the corresponding input vector, denoted as \mathbf{u}_n , is initialized by setting to zero all required samples with negative time index, u_{-1}, u_{-2}, \dots . Note that in the algorithm, a parameter, δ , of a small value has also been used to prevent numerical problems in the associated matrix inversion. Also, a step size μ has been introduced, which controls the size of the update and whose presence will be justified soon. The complexity of APA is increased compared with that of the LMS, due to the involved matrix inversion and matrix operations, requiring $O(q^3)$ MADs. Fast versions of the APA, which exploit the special structure of $X_n X_n^T$, for the case where the involved input-output variables are realizations of stochastic processes, have also been developed (see [42,43]).

The convergence analysis of the APA is more involved than that of the LMS. It turns out that provided that $0 < \mu < 2$, stability of the algorithm is guaranteed. The misadjustment is approximately

given by [2,34,83]

$$\mathcal{M} \simeq \frac{\mu q \sigma_\eta^2}{2 - \mu} \mathbb{E} \left[\frac{1}{\|\mathbf{x}_n\|^2} \right] \text{trace}\{\Sigma_x\}: \text{ misadjustment for the APA.}$$

In words, the misadjustment increases as the parameter q increases; that is, as the number of the reused past data samples increases.

GEOMETRIC INTERPRETATION OF APA

Let us look at the optimization task in (5.62), associated with APA. Each one of the q constraints defines a hyperplane in the l -dimensional space. Hence, since θ_n is constrained to lie on all these hyperplanes, it will lie on their *intersection*. Provided that \mathbf{x}_{n-i} , $i = 0, \dots, q-1$, are linearly independent, these hyperplanes share a nonempty intersection, which is an *affine set* of dimension $l - q$. An affine set is the translation of a linear subspace (i.e., a plane crossing the origin) by a constant vector; that is, it defines a plane in a general position. Thus, θ_n can lie anywhere in this affine set. From the infinite number of points lying in this set, APA selects the one that lies closest, in the Euclidean distance sense, to θ_{n-1} . In other words, θ_n is the *projection* of θ_{n-1} on the affine set defined by the intersection of the q hyperplanes. Recall from geometry that the projection $P_H(\mathbf{a})$ of a point \mathbf{a} on a linear subspace/affine set H is the point in H whose distance from \mathbf{a} is minimum. Fig. 5.13 illustrates the geometry for the case of $q = 2$; this special case of APA is also known as the binormalized data reusing LMS [7].

In the ideal noiseless case, the unknown parameter vector would lie in the intersection of all the hyperplanes defined by (y_n, \mathbf{x}_n) , $n = 0, 1, \dots, q-1$, and this is the information that APA tries to exploit to speed up convergence. However, this is also its drawback. In any practical system, noise is present; thus forcing the updates to lie in the intersection of these hyperplanes is not necessarily good, since their position in space is also determined by the noise. As a matter of fact, the reason that μ is introduced is to account for such cases. An alternative technique, which exploits projections, and at the same time replaces hyperplanes by hyperslabs (whose width depends on the noise variance) to account for the noise, will be treated in Chapter 8. In addition, there, no matrix inversion will be required.

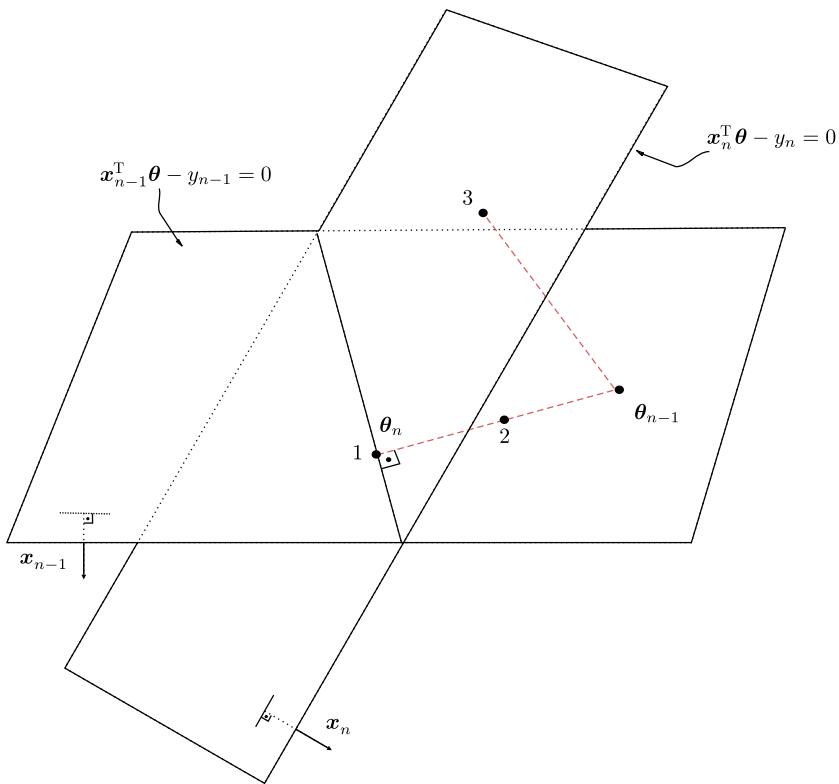
ORTHOGONAL PROJECTIONS

Projections and projection matrices/operators play a crucial part in machine learning, signal processing, and optimization in general; after all, a projection corresponds to a minimization task when the loss is interpreted as a “distance.” Let A be an $l \times k$, $k < l$, matrix with column vectors, \mathbf{a}_i , $i = 1, \dots, k$, and \mathbf{x} an l -dimensional vector. The orthogonal projection of \mathbf{x} on the subspace spanned by the columns of A (assumed to be linearly independent) is given by (Appendix A)

$$P_{\{\mathbf{a}_i\}}(\mathbf{x}) = A(A^T A)^{-1} A^T \mathbf{x}, \quad (5.65)$$

where in complex spaces the transpose operation is replaced by the Hermitian one. One can easily check that $P_{\{\mathbf{a}_i\}}^\perp(\mathbf{x}) := (I - P_{\{\mathbf{a}_i\}})\mathbf{x}$ is orthogonal to $P_{\{\mathbf{a}_i\}}(\mathbf{x})$ and

$$\mathbf{x} = P_{\{\mathbf{a}_i\}}(\mathbf{x}) + P_{\{\mathbf{a}_i\}}^\perp(\mathbf{x}).$$

**FIGURE 5.13**

The geometry associated with the APA, for $q = 2$ and $l = 3$. The intersection of the two hyperplanes is a straight line (affine set of dimension $3 - 2 = 1$); θ_n is the projection of θ_{n-1} on this line (point 1) for $\mu = 1$ and $\delta = 0$. Point 2 corresponds to the case $\mu < 1$. Point 3 is the projection of θ_n on the hyperplane defined by (y_n, \mathbf{x}_n) . This is the case for $q = 1$. The latter case corresponds to the normalized LMS (NLMS) of Section 5.6.1.

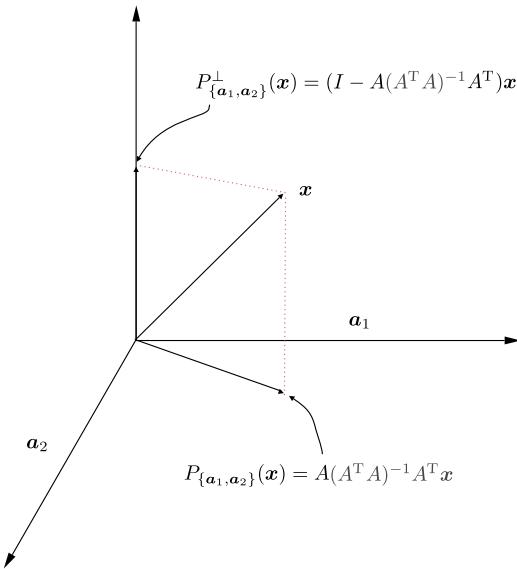
When A has orthonormal columns, we obtain the (familiar from geometry) expansion

$$P_{\{\mathbf{a}_i\}}(\mathbf{x}) = \mathbf{A}\mathbf{A}^T \mathbf{x} = \sum_{i=1}^k (\mathbf{a}_i^T \mathbf{x}) \mathbf{a}_i.$$

Thus, the factor $(\mathbf{A}^T \mathbf{A})^{-1}$, for the general case, accounts for the lack of orthonormality of the columns of A . The matrix $\mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ is known as the respective *projection matrix* and $\mathbf{I} - \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ as the projection matrix on the respective *orthogonal complement space*.

The simplest case occurs when $k = 1$; then the projection of \mathbf{x} onto \mathbf{a}_1 is equal to

$$P_{\{\mathbf{a}_1\}}(\mathbf{x}) = \frac{\mathbf{a}_1 \mathbf{a}_1^T}{\|\mathbf{a}_1\|^2} \mathbf{x},$$

**FIGURE 5.14**

Geometry indicating the orthogonal projection operation on the subspace spanned by $\mathbf{a}_1, \mathbf{a}_2$, using the projection matrix. Note that $A = [\mathbf{a}_1, \mathbf{a}_2]$.

and the corresponding projection matrices are given by

$$P_{\{\mathbf{a}_1\}} = \frac{\mathbf{a}_1 \mathbf{a}_1^T}{\|\mathbf{a}_1\|^2}, \quad P_{\{\mathbf{a}_1\}}^{\perp} = I - \frac{\mathbf{a}_1 \mathbf{a}_1^T}{\|\mathbf{a}_1\|^2}.$$

Fig. 5.14 illustrates the geometry.

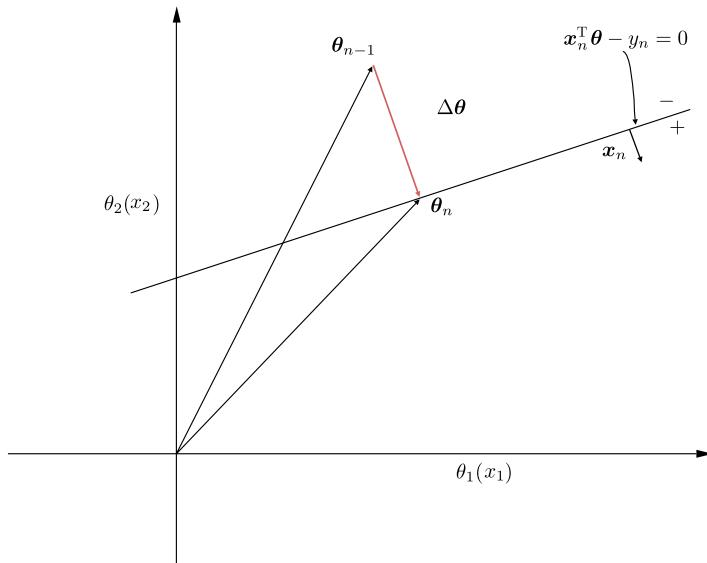
Let us apply the previously reported linear algebra results in the case of the APA of (5.63) and (5.64), and rewrite them as

$$\begin{aligned} \boldsymbol{\theta}_n &= \left(I - X_n^T (X_n X_n^T)^{-1} X_n \right) \boldsymbol{\theta}_{n-1} \\ &\quad + X_n^T (X_n X_n^T)^{-1} \mathbf{y}_n. \end{aligned}$$

The first term on the right-hand side is the projection $P_{\{\mathbf{x}_n, \dots, \mathbf{x}_{n-q+1}\}}^{\perp}(\boldsymbol{\theta}_{n-1})$. This is the most natural. By the definition of the respective affine set, as the intersection of the hyperplanes

$$\mathbf{x}_{n-i}^T \boldsymbol{\theta} - y_{n-i} = 0, \quad i = 0, \dots, q-1,$$

each vector \mathbf{x}_{n-i} is *orthogonal* to the respective hyperplane (Fig. 5.13) (Problem 5.12). Hence, projecting $\boldsymbol{\theta}_{n-1}$ on the intersection of all these hyperplanes is equivalent to projecting on an affine set, which is *orthogonal* to all $\mathbf{x}_n, \dots, \mathbf{x}_{n-q+1}$. Note that the matrix $X_n^T (X_n X_n^T)^{-1} X_n$ is the projection matrix

**FIGURE 5.15**

$\Delta\theta$ is equal to the (signed) distance of θ_{n-1} from the plane times the unit vector $\frac{x_n}{\|x_n\|}$; x_n should be at the origin, but it is drawn so that it is clearly shown that it is perpendicular to the line segment.

that projects on the subspace spanned by x_n, \dots, x_{n-q+1} . The second term accounts for the fact that the affine set on which we project does not include the origin, but it is translated to another point in the space, whose direction is determined by y_n . Fig. 5.15 illustrates the case for $l = 2$ and $q = 1$. Because θ_{n-1} does not lie on the line (plane) $x_n^T \theta - y_n = 0$, whose direction is defined by x_n , we know from geometry (and it is easily checked) that its distance from this line is $s = \frac{|x_n^T \theta_{n-1} - y_n|}{\|x_n\|}$. Also, θ_{n-1} lies on the negative side of the straight line, so that $x_n^T \theta_{n-1} - y_n < 0$. Hence, taking into account the directions of the involved vectors, it turns out that

$$\Delta\theta = \frac{y_n - x_n^T \theta_{n-1}}{\|x_n\|} \frac{x_n}{\|x_n\|}.$$

Thus, for this specific case, the correction

$$\theta_n = \theta_{n-1} + \Delta\theta$$

coincides with the recursion of the APA.

5.6.1 THE NORMALIZED LMS

The NLMS is a special case of the APA corresponding to $q = 1$ (see Fig. 5.13). We treat it separately due to its popularity, and it is summarized in Algorithm 5.3.

Algorithm 5.3 (The normalized LMS).

- Initialization
 - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$, or any other value.
 - Choose $0 < \mu < 2$, and δ a small value.
- **For** $n = 0, 1, 2, \dots$, **Do**
 - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$
 - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu}{\delta + \mathbf{x}_n^T \mathbf{x}_n} \mathbf{x}_n e_n$
- **End For**

The complexity of the NLMS is $3l$ MADs. Stability of the NLMS is guaranteed if

$$0 < \mu < 2.$$

One can look at the NLMS as an LMS whose step size is left to vary with the iterations, as represented by

$$\mu_n = \frac{\mu}{\delta + \mathbf{x}_n^T \mathbf{x}_n},$$

which turns out to have a beneficial effect on the convergence speed, compared with the LMS. More on the performance analysis of the NLMS can be found in, e.g., [15,83,90].

Remarks 5.3.

- To deal with sparse models, the so-called *proportionate* NLMS and related versions of the other algorithms have been derived [11,35]. The idea is to use a separate step size for each one of the parameters. This gives the freedom to the coefficients which correspond to small (or zero) values of the model to adapt at a different rate than the rest, and this has a significant effect on the convergence performance of the algorithm. Such schemes can be considered as the ancestors of the more theoretically elegant sparsity promoting online algorithms, to be treated in Chapter 10.
- Another trend that has received attention more recently is to appropriately (convexly) combine the outputs of two (or more) learning structures. This has the effect of decreasing the sensitivity of the learning algorithms to choices of parameters such as the step size, or to the dimensionality of the problem (size of the filter). The two (or more) algorithms run independently and the mixing parameters of the outputs are learned during the training. In general, this approach turns out to be more robust in the choice of the involved user-defined parameters [8,81].
- In some papers, the user-defined parameter, δ , in the NLMS and the APA is suggested to be given a very small positive value. Often, the explanation for this option is that the use of δ is to avoid division by zero. However, in practice, there are cases, such as the echo cancelation task, where δ needs to be set to quite large values (even larger than 1) to attain good performance. It is fairly recent that the importance of δ was emphasized and a formula for its proper tuning was proposed both for the case of NLMS and APA and their proportionate counterparts [12,73]. There, it is indicated that without the proper setup of this parameter, the performance of these algorithms may be significantly affected, and they may not even converge.

rithm is a deterministic perfect periodic sequence (PPSEQ) with period equal to the impulse response of the system. Such sequences have been used in [24,25] to derive versions of the LMS which not only converge fast but are also of very low computational complexity, requiring only one multiplication, one addition, and one subtraction per update.

5.9 SIMULATION EXAMPLES

Example 5.3. The goal of this example is to demonstrate the sensitivity of the convergence rate of the LMS to the eigenvalues spread of the input covariance matrix. To this end, two experiments were conducted, in the context of a regression/system identification task. Data were generated according to our familiar model

$$y_n = \boldsymbol{\theta}_o^T \mathbf{x}_n + \eta_n.$$

The (unknown) parameters $\boldsymbol{\theta}_o \in \mathbb{R}^{10}$ were randomly chosen from $\mathcal{N}(0, 1)$ and then frozen. In the first experiment, the input vectors were formed by a white noise sequence with samples i.i.d. drawn from $\mathcal{N}(0, 1)$. Thus, the input covariance matrix was diagonal with all the elements being equal to the corresponding noise variance (Section 2.4.3). The noise samples η_n were also i.i.d. drawn from a Gaussian with zero mean and variance $\sigma^2 = 0.01$. In the second experiment, the input vectors were formed by an AR(1) process with coefficient equal to $a_1 = 0.85$ and the corresponding white noise

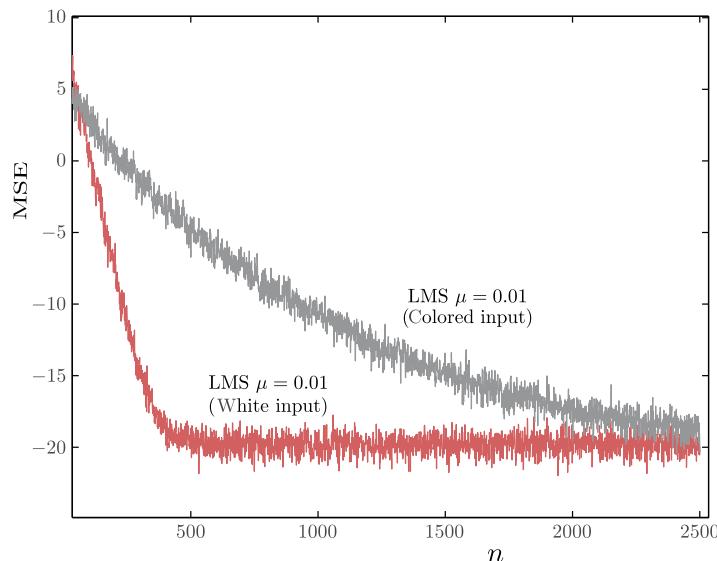


FIGURE 5.16

Observe that for the same step size, the convergence of the LMS is faster when the input is white. The two curves are the result of averaging 100 independent experimental realizations.

excitation was of variance equal to 1 (Section 2.4.4). Thus, the input covariance matrix is no more diagonal and the eigenvalues are not equal. The LMS was run on both cases with the same step size $\mu = 0.01$. Fig. 5.16 summarizes the results. The vertical axis (denoted as MSE) shows the squared error, e_n^2 , in dBs ($10 \log_{10} e_n^2$) and the horizontal axis shows the time instants (iterations) n . Note that both curves level out at the same error floor. However, the convergence rate for the case of the white noise input is significantly higher. The curves shown in the figure are the result of averaging 100 independent experimental realizations.

It must be emphasized that when comparing convergence performance of different algorithms, either all algorithms should converge to the same error floor and compare the respective convergence rates, or all algorithms should have the same convergence rate and compare respective error floors.

Example 5.4. In this example, the dependence of the LMS on the choice of the step size is demonstrated. The unknown parameters $\theta_o \in \mathbb{R}^{10}$ and the data were exactly the same as in the white noise case of Example 5.3.

The LMS was run using the generated samples, with two different step sizes, namely, $\mu = 0.01$ and $\mu = 0.075$. The obtained averaged (over 100 realizations) curves are shown in Fig. 5.17. Observe that the larger the step size, for the same set of observation samples, the faster the convergence becomes albeit at the expense of a higher error floor (misadjustment), in accordance to what was discussed in Section 5.5.1.

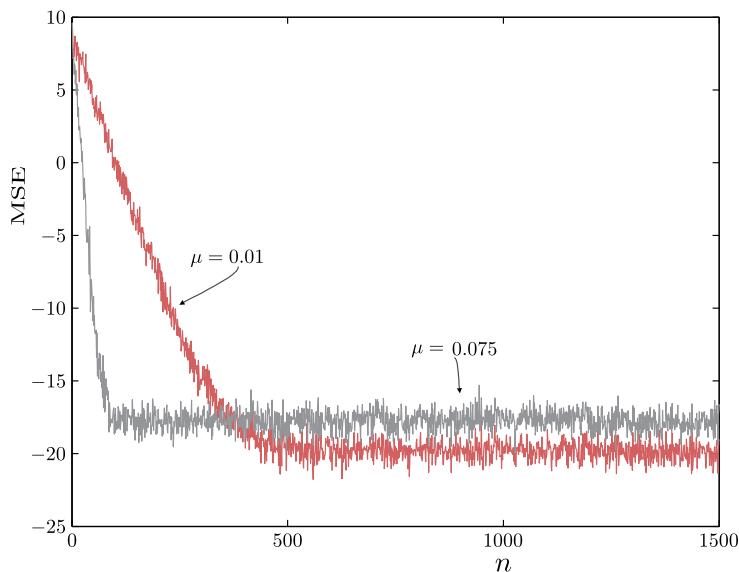
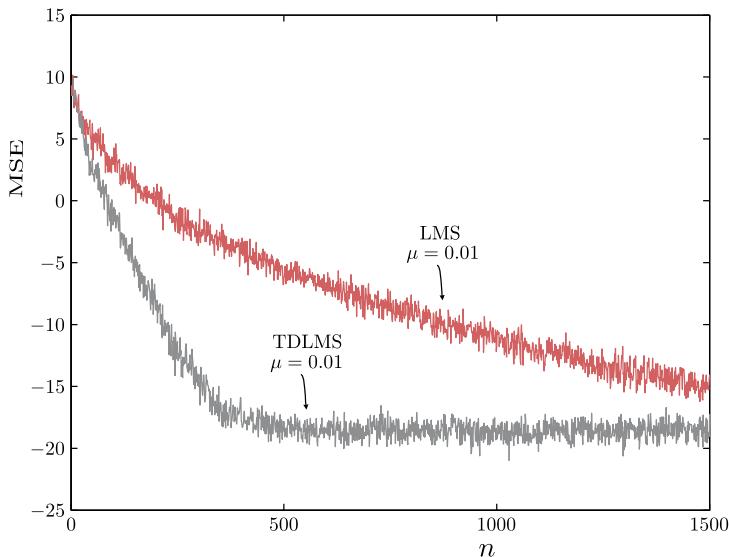


FIGURE 5.17

For the same input, the larger the step size for the LMS, the faster the convergence becomes, albeit at the expense of higher error floor (MSE in dBs).

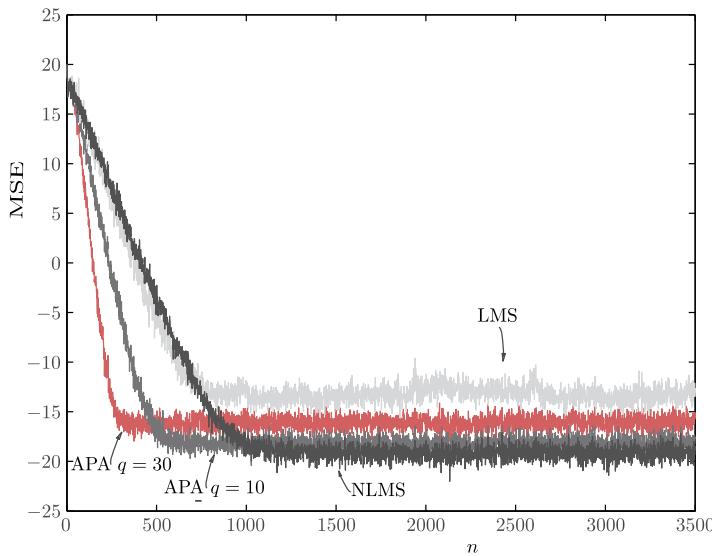
**FIGURE 5.18**

Observe that for the same step size, the convergence of the transform-domain LMS is significantly faster compared to the LMS, for similar error floors. The higher the eigenvalues spread of the input covariance matrix is, the more the obtained performance improvement becomes (MSE in dBs).

Example 5.5 (LMS versus transform-domain LMS). In this example, the stage of the experimental setup is exactly the same as that considered in Example 5.3 for the AR(1) case. The goal is to compare the LMS and the transform-domain LMS. Fig. 5.18 shows the obtained averaged error curves. The step size was the same as the one used in Example 5.3, $\mu = 0.01$; hence the curve for the LMS is the same as the corresponding one appearing in Fig. 5.16. Observe the significantly faster convergence achieved by the transform-domain LMS, due to its (approximate) whitening effect on the input.

Example 5.6. The experimental setup is similar to that of Example 5.4, with the only exception that the unknown parameter vector was of higher dimension, $\theta_o \in \mathbb{R}^{60}$, so that the differences in the performance of the algorithms is more clear. The goal is to compare the LMS, the NLMS, and the APA. The step size of the LMS was chosen equal to $\mu = 0.025$ and for the NLMS $\mu = 0.35$ and $\delta = 0.001$, so that both algorithms had similar convergence rates. The step size for the APA was chosen equal to $\mu = 0.1$, so that $q = 10$ will settle at the same error floor as that of the NLMS. For the APA, we also chose $\delta = 0.001$. The results are shown in Fig. 5.19.

Observe the lower error floor, for the same convergence rate, obtained by the NLMS compared to the LMS and the improved performance obtained by the APA for $q = 10$. Increasing the number of past data samples (re)used in APA to $q = 30$, we can see the improved convergence rate that is obtained, although at the expense of higher error floor, as predicted by the theoretical results reported in Section 5.6.

**FIGURE 5.19**

For the same step size, the NLMS converges at the same rate to a lower error floor compared to the LMS. For the APA, increasing q improves the convergence, at the expense of higher error floors (MSE in dBs).

5.10 ADAPTIVE DECISION FEEDBACK EQUALIZATION

The task of channel equalization was introduced in Fig. 4.12. The input to the equalizer is a stochastic process (random signal), which, according to the notational convention introduced in Section 2.4, will be denoted as u_n . Note that upon receiving the noisy and distorted by the (communications) channel sample, u_n , one has to obtain an estimate of the originally transmitted information sequence, s_n , delayed by L time lags, which accounts for the various delays imposed by the overall transmission system involved. Thus, at time n , the equalizer decides for \hat{s}_{n-L+1} . Ideally, if one knew the true values of the initially transmitted information sequence up to and including time instant $n - L$, represented by $s_{n-L}, s_{n-L-1}, s_{n-L-2} \dots$, it could only be beneficial to use this information, together with the received sequence, u_n , to recover an estimate of \hat{s}_{n-L+1} . This idea is explored in the *decision feedback equalizer* (DFE). The equalizer's output, for the complex-valued data case, is now written as

$$\begin{aligned}\hat{d}_n &= \sum_{i=0}^{L-1} w_i^{f*} u_{n-i} + \sum_{i=0}^{l-1} w_i^{b*} s_{n-L-i} \\ &= \mathbf{w}^H \mathbf{u}_{e,n},\end{aligned}\tag{5.71}$$

where

$$\mathbf{w} := \begin{bmatrix} \mathbf{w}^f \\ \mathbf{w}^b \end{bmatrix} \in \mathbb{C}^{L+l}, \quad \mathbf{u}_{e,n} := \begin{bmatrix} \mathbf{u}_n \\ s_n \end{bmatrix} \in \mathbb{C}^{L+l},$$

Plugging (5.73) into the constraints in (5.72), which can be compactly written as $C^H \mathbf{w} = \mathbf{g}$ (recall $g_i \in \mathbb{R}$), we readily obtain

$$-\mu \lambda_n = (C^H C)^{-1} \mathbf{g} - (C^H C)^{-1} C^H (\mathbf{w}_{n-1} + \mu \mathbf{u}_n e_n^*),$$

and the update recursion becomes

$$\mathbf{w}_n = (I - C(C^H C)^{-1} C^H) (\mathbf{w}_{n-1} + \mu \mathbf{u}_n e_n^*) + C(C^H C)^{-1} \mathbf{g}.$$

Note that $(I - C(C^H C)^{-1} C^H)$ is the orthogonal projection matrix on the intersection (affine set) of the hyperplanes defined by the constraints (recall that $C(C^H C)^{-1} C^H$ is the respective projection matrix on the subspace spanned by \mathbf{c}_i , $i = 1, 2, \dots, m$). In the case the goal is to minimize the output, one has to set in e_n the desired response $d_n = 0$.

The constrained LMS was first treated in [40]. Besides the previously used constraints, other constraints can also be used; for example, for the constrained NLMS, one additionally demands

$$\mathbf{w}_n^H \mathbf{u}_n = d_n$$

(see, e.g., [6]).

5.12 TRACKING PERFORMANCE OF THE LMS IN NONSTATIONARY ENVIRONMENTS

We have already considered the convergence behavior of the LMS and made related comments for the other algorithms that have been discussed. As stated before, convergence is a *transient* phenomenon; that is, it concerns the period from the initial kick-off point to the steady state. The steady state was also discussed for stationary environments, that is, environments in which the unknown parameter vector as well as the underlying statistics of the involved random variables/processes remain unchanged.

We now turn our focus to cases where the true (yet unknown) parameter vector/system undergoes changes. Thus, this affects the output observations and consequently their statistics. Note that the statistics of the input can also change. However, we are not going to consider such cases, as the analysis can become quite involved. Our goal is to study the *tracking* performance of the LMS, that is, the ability of the algorithm to track changes of the unknown parameter vector. Note that tracking is a *steady-state* phenomenon. In other words, we assume that enough time has elapsed so that the influence of the initial conditions has been forgotten and has no effect, any more, on the algorithm. Tracking agility and convergence speed are two *different* properties of an algorithm. An algorithm may converge fast, but it *may not* necessarily have a good tracking performance, and vice versa. We will see such cases later on.

The setting of our discussion will be similar to that of Section 5.5.1. In conformity with the discussion there, we consider the real-data case; similar results hold true for the complex-valued linear estimation scenario. However, in contrast to the adopted model in (5.38), a *time-varying* model is adopted here, using the following assumptions.

Assumption 1. The output observations are generated according to the model

$$\mathbf{y}_n = \boldsymbol{\theta}_{o,n-1}^T \mathbf{x} + \eta, \quad (5.74)$$

which is in line with the prediction model used in LMS, at time n . That is, the unknown set of parameters is a time-varying one. The statistical properties of the input vector \mathbf{x} , as well as the noise variable η , are assumed to be time independent, and this is the reason we have not used the time index; equivalently, in the case where the input is a random process, u_n , it is assumed to be *stationary*. Moreover, the input variables are assumed to be independent of the zero mean noise variable, η . Furthermore, successive samples of η are i.i.d. (white noise sequence) of variance σ_η^2 . So far, we have not gone much beyond Assumption 1, stated in Section 5.5.1.

Assumption 2. The time-varying model follows a random walk variation, represented by

$$\boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}. \quad (5.75)$$

The random vector $\boldsymbol{\omega}$ is assumed to be zero mean with covariance matrix

$$\mathbb{E}[\boldsymbol{\omega}\boldsymbol{\omega}^T] = \Sigma_\omega.$$

Note that the variance of a random walk grows unbounded with time; this is readily shown by applying (5.75) recursively.

A variant of this model would be more sensible to use,

$$\boldsymbol{\theta}_{o,n} = a\boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega},$$

with $|a| < 1$ [106]. However, the analysis gets more involved, so we will stick with the model in (5.75). After all, our goal here is to highlight and have a first touch on the notion of tracking and to get an idea of its effects on the misadjustment in steady state.

Assumption 3. As in Section 5.5.1, we assume that $\mathbf{c}_{n-1} := \boldsymbol{\theta}_{n-1} - \boldsymbol{\theta}_{o,n-1}$ is independent of \mathbf{x} and η . This time, we will also assume independence of \mathbf{c}_n and $\boldsymbol{\omega}$.

Recall from (5.48) that the excess MSE at time n is given by

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Thus, our goal now is to compute $\Sigma_{c,n-1}$ for the time-varying model case. It is straightforward to see that the counterpart of (5.35) now becomes

$$\mathbf{c}_n = (I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1} + \mu \mathbf{x} \eta - \boldsymbol{\omega}. \quad (5.76)$$

Adopting the previously stated three assumptions, as well as the *Gaussian* assumption for \mathbf{x} , and following exactly the same steps used for (5.41), we end up with

$$\begin{aligned} \Sigma_{c,n} = & \Sigma_{c,n-1} - \mu \Sigma_x \Sigma_{c,n-1} - \mu \Sigma_{c,n-1} \Sigma_x + 2\mu^2 \Sigma_x \Sigma_{c,n-1} \Sigma_x \\ & + \mu^2 \Sigma_x \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} + \mu^2 \sigma_\eta^2 \Sigma_x + \Sigma_\omega. \end{aligned} \quad (5.77)$$

Note that if complex data are involved, the only difference is that the fourth term on the right-hand side is not multiplied by 2 (Problem 5.15). This equation governs the propagation of $\Sigma_{c,n}$, which in turn can provide the excess MSE error.

A more convenient form results for small values of μ , where the fourth and fifth terms on the right-hand side can be neglected, being small with respect to $\mu \Sigma_x \Sigma_{c,n-1}$. Moreover, at the steady state, $\Sigma_{c,n} = \Sigma_{c,n-1} := \Sigma_c$, and taking the trace on both sides, we end up with (recall $\text{trace}\{A + B\} =$

$\text{trace}\{A\} + \text{trace}\{B\}$ and $\text{trace}\{AB\} = \text{trace}\{BA\}$)

$$J_{\text{exc}} = \text{trace}\{\Sigma_x \Sigma_c\} = \frac{1}{2} \left(\mu \sigma_\eta^2 \text{trace}\{\Sigma_x\} + \frac{1}{\mu} \text{trace}\{\Sigma_\omega\} \right). \quad (5.78)$$

Note that this is exactly the same approximation as the one resulting from the more sound theory of energy conservation [83].

Compare (5.78) with (5.53); in the current setting, there is another term associated with the noise, which drifts the model around its mean. Thus, (a) the excess MSE is contributed by the inability of the LMS to obtain the optimum value exactly, and (b) there is an extra term measuring its “inertia” to track the changes of the model fast enough. This is the most important outcome of the current discussion. In time-varying environments, the misadjustment increases. Moreover, looking at (5.78) and at the effect of μ , it is observed that small values of μ have a beneficial effect on the first term, but they increase the contribution of the second one. The opposite is true for relatively big values of μ . This is natural. Small step sizes give the algorithm the chance to learn better under stationary environments, but the algorithm cannot track the changes fast enough. Thus, the choice of μ should be the outcome of a tradeoff. Minimizing the excess error in (5.78), it is easily shown to result in

$$\mu_{\text{opt}} = \sqrt{\frac{\text{trace}\{\Sigma_\omega\}}{\sigma_\eta^2 \text{trace}\{\Sigma_x\}}}.$$

Note, however, that such choices for μ are of theoretical importance only. In practice, the time variation of the system can hardly correspond to that of the adopted model; the latter, due to the complexity of the analysis, was chosen to be a simple one in an effort to simplify the mathematical manipulations. Moreover, for the sake of simplifying the analysis, a number of assumptions were adopted. In practice, μ is chosen more according to the user’s practical experience, after experimentation, than based on the theory. The theory, however, has pointed out the tradeoff between the speed of convergence and that of tracking.

More mathematically rigorous analysis of the performance of online/adaptive schemes in nonstationary environments can be obtained from [16,38,47,61,70,83]. Simulation results demonstrating the tracking performance of the LMS compared with other algorithms are given in Example 6.3 in Chapter 6, where the recursive least-squares (RLS) algorithm is presented.

5.13 DISTRIBUTED LEARNING: THE DISTRIBUTED LMS

The focus of our attention is now turned toward a problem that has been of an increasing importance over the last decade or so. There is a growing number of applications where data are received/reside in different sensors/databases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal, i.e., to perform a *common estimation/inference* task. We refer to such tasks as *distributed* or *decentralized* learning. At the heart of this problem lies the concept of *cooperation*, which is another name for the process of exchanging learning experience/information in order to reach a common goal/decision. Human societies have survived because of cooperation (and have disappeared due to lack of cooperation).

THE LEAST-SQUARES FAMILY

6

CONTENTS

6.1	Introduction	253
6.2	Least-Squares Linear Regression: a Geometric Perspective	254
6.3	Statistical Properties of the LS Estimator	257
	The LS Estimator Is Unbiased	257
	Covariance Matrix of the LS Estimator	257
	The LS Estimator Is BLUE in the Presence of White Noise	258
	The LS Estimator Achieves the Cramér–Rao Bound for White Gaussian Noise	259
	Asymptotic Distribution of the LS Estimator	260
6.4	Orthogonalizing the Column Space of the Input Matrix: the SVD Method	260
	Pseudoinverse Matrix and SVD	262
6.5	Ridge Regression: a Geometric Point of View	265
	Principal Components Regression	267
6.6	The Recursive Least-Squares Algorithm	268
	Time-Iterative Computations	269
	Time Updating of the Parameters	270
6.7	Newton's Iterative Minimization Method	271
	6.7.1 RLS and Newton's Method	274
6.8	Steady-State Performance of the RLS	275
6.9	Complex-Valued Data: the Widely Linear RLS	277
6.10	Computational Aspects of the LS Solution	279
	Cholesky Factorization	279
	QR Factorization	279
	Fast RLS Versions	280
6.11	The Coordinate and Cyclic Coordinate Descent Methods	281
6.12	Simulation Examples	283
6.13	Total Least-Squares	286
	Geometric Interpretation of the Total Least-Squares Method	291
Problems		293
	MATLAB® Exercises	296
References		297

6.1 INTRODUCTION

The squared error loss function was at the center of our attention in the previous two chapters. The sum of squared errors cost was introduced in Chapter 3, followed by the mean-square error (MSE) version, treated in Chapter 4. The stochastic gradient descent technique was employed in Chapter 5 to

help us bypass the need to perform expectations for obtaining the second-order statistics of the data, as required by the MSE formulation.

In this chapter, we return to the original formulation of the sum of error squares, and our goal is to look more closely at the resulting family of algorithms and their properties. An emphasis is given on the geometric interpretation of the least-squares (LS) method as well as on some of the most important statistical properties of the resulting solution. The singular value decomposition (SVD) of a matrix is introduced for a first time in this book. Its geometric orthogonalizing properties are discussed and its connection with what is known as dimensionality reduction is established; the latter topic is extensively treated in Chapter 19. Also, a major part of the chapter is dedicated to the recursive LS (RLS) algorithm, which is an online scheme that solves the LS optimization task. The spine of the RLS scheme comprises an efficient update of the inverse (sample) covariance matrix of the input data, whose rationale can also be adopted in the context of different learning methods for developing related online schemes; this is one of the reasons we pay special tribute to the RLS algorithm. The other reason is its popularity in a large number of signal processing/machine learning tasks, due to some attractive properties that this scheme enjoys. Two major optimization schemes are introduced, namely, Newton's method and the coordinate descent method, and their use in solving the LS task is discussed. The bridge between the RLS scheme and Newton's optimization method is established. Finally, at the end of the chapter, a more general formulation of the LS task, known as the total least-squares (TLS) method, is also presented.

6.2 LEAST-SQUARES LINEAR REGRESSION: A GEOMETRIC PERSPECTIVE

The focus of this section is to outline the geometric properties of the LS method. This provides an alternative view on the respective minimization method and helps in its understanding, by revealing a physical structure that is associated with the obtained solution. Geometry is very important when dealing with concepts related to the dimensionality reduction task.

We begin with our familiar linear regression model. Given a set of observations,

$$y_n = \boldsymbol{\theta}^T \mathbf{x}_n + \eta_n, \quad n = 1, 2, \dots, N, \quad y_n \in \mathbb{R}, \quad \mathbf{x}_n \in \mathbb{R}^l, \quad \boldsymbol{\theta} \in \mathbb{R}^l,$$

where η_n denotes the (unobserved) values of a *zero* mean noise source, the task is to obtain an estimate of the unknown parameter vector, $\boldsymbol{\theta}$, so that

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = \arg \min_{\boldsymbol{\theta}} \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2. \quad (6.1)$$

Our stage of discussion is that of real numbers, and we will point out differences with the complex number case whenever needed. Moreover, we assume that our data have been centered around their sample means; alternatively, the intercept, θ_0 , can be absorbed in $\boldsymbol{\theta}$ with a corresponding increase in the dimensionality of \mathbf{x}_n . Define

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N, \quad X := \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times l}. \quad (6.2)$$

Eq. (6.1) can be recast as

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = \arg \min_{\boldsymbol{\theta}} \|\mathbf{e}\|^2,$$

where

$$\mathbf{e} := \mathbf{y} - X\boldsymbol{\theta}$$

and $\|\cdot\|$ denotes the Euclidean norm, which measures the “distance” between the respective vectors in \mathbb{R}^N , i.e., \mathbf{y} and $X\boldsymbol{\theta}$. Indeed, the n th component of the vector \mathbf{e} is equal to $e_n = y_n - \mathbf{x}_n^T \boldsymbol{\theta}$, which, due to the symmetry of the inner product, is equal to $y_n - \boldsymbol{\theta}^T \mathbf{x}_n$; furthermore, the square Euclidean norm of the vector is the sum of the squares of its components, which makes the square norm of \mathbf{e} identical to the sum of squared errors cost in Eq. (6.1).

Let us now denote as $\mathbf{x}_1^c, \dots, \mathbf{x}_l^c \in \mathbb{R}^N$ the columns of X , i.e.,

$$X = [\mathbf{x}_1^c, \dots, \mathbf{x}_l^c].$$

Then the matrix-vector product above can be written as the linear combination of the columns of matrix X , i.e.,

$$\hat{\mathbf{y}} := X\boldsymbol{\theta} = \sum_{i=1}^l \theta_i \mathbf{x}_i^c,$$

and

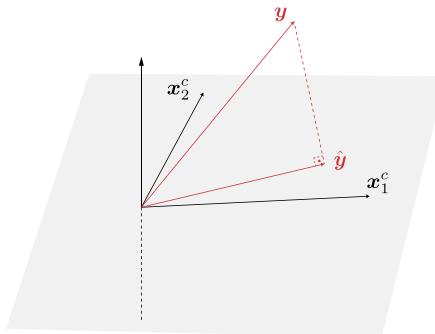
$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}.$$

Note the \mathbf{e} can be viewed as the *error vector* between the vector of the output observations, \mathbf{y} , and $\hat{\mathbf{y}}$; the latter is the prediction of \mathbf{y} based on the input observations, stacked in X , and given a value for $\boldsymbol{\theta}$. Obviously, the N -dimensional vector $\hat{\mathbf{y}}$, being a linear combination of the columns of X , lies in the span $\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$. By definition, the latter is the subspace of \mathbb{R}^l that is generated by all possible linear combinations of the l columns of X (see Appendix A). Thus, naturally, our task now becomes that of selecting $\boldsymbol{\theta}$ so that the error vector between \mathbf{y} and $\hat{\mathbf{y}}$ has minimum norm. In general, the observations vector, \mathbf{y} , does not lie in the subspace spanned by the columns of X , due to the existence of the noise.

According to the Pythagorean theorem of orthogonality for Euclidean spaces, the minimum norm error is obtained if $\hat{\mathbf{y}}$ is chosen as the *orthogonal projection* of \mathbf{y} onto the span $\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$. Recalling the concept of orthogonal projections (Appendix A and Section 5.6, Eq. (5.65)), the orthogonal projection of \mathbf{y} onto the subspace spanned by the columns of X is given by

$$\hat{\mathbf{y}} = X(X^T X)^{-1} X^T \mathbf{y} : \quad \text{LS estimate,}$$

(6.3)

**FIGURE 6.1**

In the figure, y lies outside the two-dimensional (shaded) plane that is defined by the two column vectors, x_1^c and x_2^c , i.e., $\text{span}\{x_1^c, x_2^c\}$. From all the points on this plane, the one that is closest to y , in the minimum error norm sense, is its respective orthogonal projection, i.e., the point \hat{y} . The parameter vector θ associated with this orthogonal projection coincides with the LS estimate.

assuming that $X^T X$ is invertible. Recalling the definition of \hat{y} , the above corresponds to the LS estimate for the unknown set of parameters, as we know it from Chapter 3, i.e.,

$$\hat{\theta} = (X^T X)^{-1} X^T y.$$

The geometry is illustrated in Fig. 6.1.

It is common to describe the LS solution in terms of the *Moore–Penrose pseudoinverse* of X , which for a tall matrix¹ is defined as

$X^\dagger := (X^T X)^{-1} X^T : \quad \text{pseudoinverse of a tall matrix } X,$

(6.4)

and hence we can write

$$\hat{\theta}_{\text{LS}} = X^\dagger y. \quad (6.5)$$

Thus, we have rederived Eq. (3.17) of Chapter 3, this time via geometric arguments. Note that the pseudoinverse is a generalization of the notion of the inverse of a square matrix. Indeed, if X is square, then it is readily seen that the pseudoinverse coincides with X^{-1} . For complex-valued data, the only difference is that transposition is replaced by the Hermitian one.

¹ A matrix, e.g., $X \in \mathbb{R}^{N \times l}$, is called *tall*, if $N > l$. If $N < l$ it is known as *fat*. If $l = N$, it is a *square* matrix.

6.3 STATISTICAL PROPERTIES OF THE LS ESTIMATOR

Some of the statistical properties of the LS estimator were touched on in Chapter 3, for the special case of a random real parameter. Here, we will look at this issue in a more general setting. Assume that there exists a true (yet unknown) parameter/weight vector $\boldsymbol{\theta}_o$ that generates the output (dependent) random variables (stacked in a random vector $\mathbf{y} \in \mathbb{R}^N$), according to the model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta}_o + \boldsymbol{\eta},$$

where $\boldsymbol{\eta}$ is a *zero* mean noise vector. Observe that we have assumed that \mathbf{X} is fixed and not random; that is, the randomness underlying the output variables \mathbf{y} is due solely to the noise. Under the previously stated assumptions, the following properties hold.

THE LS ESTIMATOR IS UNBIASED

The LS estimator for the parameters is given by

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{LS}} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \\ &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta}_o + \boldsymbol{\eta}) = \boldsymbol{\theta}_o + (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \boldsymbol{\eta},\end{aligned}\quad (6.6)$$

or

$$\mathbb{E}[\hat{\boldsymbol{\theta}}_{\text{LS}}] = \boldsymbol{\theta}_o + (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbb{E}[\boldsymbol{\eta}] = \boldsymbol{\theta}_o,$$

which proves the claim.

COVARIANCE MATRIX OF THE LS ESTIMATOR

Let, in addition to the previously adopted assumptions,

$$\mathbb{E}[\boldsymbol{\eta}\boldsymbol{\eta}^T] = \sigma_\eta^2 I,$$

that is, the source generating the noise samples is white. By the definition of the covariance matrix, we get

$$\Sigma_{\hat{\boldsymbol{\theta}}_{\text{LS}}} = \mathbb{E}[(\hat{\boldsymbol{\theta}}_{\text{LS}} - \boldsymbol{\theta}_o)(\hat{\boldsymbol{\theta}}_{\text{LS}} - \boldsymbol{\theta}_o)^T],$$

and substituting $\hat{\boldsymbol{\theta}}_{\text{LS}} - \boldsymbol{\theta}_o$ from (6.6), we obtain

$$\begin{aligned}\Sigma_{\hat{\boldsymbol{\theta}}_{\text{LS}}} &= \mathbb{E}[(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \boldsymbol{\eta}\boldsymbol{\eta}^T \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1}] \\ &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbb{E}[\boldsymbol{\eta}\boldsymbol{\eta}^T] \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \\ &= \sigma_\eta^2 (\mathbf{X}^T \mathbf{X})^{-1}.\end{aligned}\quad (6.7)$$

Note that, for large values of N , we can write

$$\mathbf{X}^T \mathbf{X} = \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \approx N \Sigma_x,$$

where Σ_x is the covariance matrix of our (zero mean) input variables, i.e.,

$$\Sigma_x := \mathbb{E}[\mathbf{x}_n \mathbf{x}_n^T] \approx \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T.$$

Thus, for large values of N , we can write

$$\boxed{\Sigma_{\hat{\theta}_{LS}} \approx \frac{\sigma_\eta^2}{N} \Sigma_x^{-1}.} \quad (6.8)$$

In other words, under the adopted assumptions, the LS estimator is not only unbiased, but its covariance matrix *tends asymptotically to zero*. That is, with high probability, the estimate $\hat{\theta}_{LS}$, which is obtained via a large number of measurements, will be close to the true value θ_o . Viewing it slightly differently, note that the LS solution tends to the MSE solution, which was discussed in Chapter 4. Indeed, for the case of centered data,

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T = \Sigma_x,$$

and

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n y_n = \mathbb{E}[\mathbf{x}\mathbf{y}] = \mathbf{p}.$$

Moreover, we know that for the linear regression modeling case, the normal equations, $\Sigma_x \theta = \mathbf{p}$, result in the solution $\theta = \theta_o$ (Remarks 4.2).

THE LS ESTIMATOR IS BLUE IN THE PRESENCE OF WHITE NOISE

The notion of the best linear unbiased estimator (BLUE) was introduced in Section 4.9.1 in the context of the *Gauss–Markov* theorem. Let $\hat{\theta}$ denote any other *linear* unbiased estimator, under the assumption that

$$\mathbb{E}[\mathbf{\eta}\mathbf{\eta}^T] = \sigma_\eta^2 I.$$

Then, due to the linearity assumption, the estimator will have a linear dependence on the output random variables that are observed, i.e.,

$$\hat{\theta} = H\mathbf{y}, \quad H \in \mathbb{R}^{l \times N}.$$

It will be shown that the variance of such an estimator can never become smaller than that of the LS one, i.e.,

$$\mathbb{E}[(\hat{\theta} - \theta_o)^T (\hat{\theta} - \theta_o)] \geq \mathbb{E}[(\hat{\theta}_{LS} - \theta_o)^T (\hat{\theta}_{LS} - \theta_o)]. \quad (6.9)$$

Indeed, from the respective definitions we have

$$\hat{\theta} = H(X\theta_o + \mathbf{\eta}) = HX\theta_o + H\mathbf{\eta}. \quad (6.10)$$

is expressed as

$$\mathcal{R}(X) = \text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}. \quad (6.28)$$

- Everything that has been said before transfers to complex-valued data, trivially, by replacing transposition with the Hermitian one.

6.5 RIDGE REGRESSION: A GEOMETRIC POINT OF VIEW

In this section, we shed light on the ridge regression task from a different perspective. Instead of a dry optimization task, we are going to look at it by mobilizing statistical geometric arguments that the SVD decomposition offers to us.

Ridge regression was introduced in Chapter 3 as a means to impose bias on the LS solution and also as a major path to cope with overfitting and ill-conditioning problems. In ridge regression, the minimizer results as

$$\hat{\boldsymbol{\theta}}_R = \arg \min_{\boldsymbol{\theta}} \left\{ \|\mathbf{y} - X\boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}\|^2 \right\},$$

where $\lambda > 0$ is a user-defined parameter that controls the importance of the regularizing term. Taking the gradient with respect to $\boldsymbol{\theta}$ and equating to zero results in

$$\hat{\boldsymbol{\theta}}_R = (X^T X + \lambda I)^{-1} X^T \mathbf{y}. \quad (6.29)$$

Looking at (6.29), we readily observe (a) its “stabilizing” effect from the numerical point of view, when $X^T X$ is ill-conditioned and its inversion poses problems, and (b) its biasing effect on the (unbiased) LS solution. Note that ridge regression provides a solution even if $X^T X$ is not invertible, as is the case when $N < l$. Let us now employ the SVD expansion of (6.14) in (6.29). Assuming a full column rank matrix X , we obtain (Problem 6.14)

$$\hat{\mathbf{y}} = X\hat{\boldsymbol{\theta}}_R = U_l D(D^2 + \lambda I)^{-1} D U_l^T \mathbf{y},$$

or

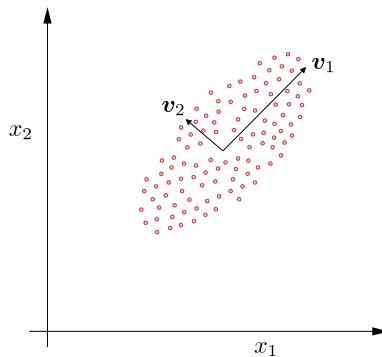
$$\hat{\mathbf{y}} = \sum_{i=1}^l \frac{\sigma_i^2}{\lambda + \sigma_i^2} (\mathbf{u}_i^T \mathbf{y}) \mathbf{u}_i : \quad \text{ridge regression shrinks the weights.}$$

(6.30)

Comparing (6.30) and (6.19), we observe that the components of the projection of \mathbf{y} onto the $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_l\}$ ($\text{span}\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$) are *shrunk* with respect to their LS counterpart. Moreover, the shrinking level depends on the singular values σ_i ; the smaller the value of σ_i , the higher the shrinking of the corresponding component. Let us now turn our attention to the investigation of the geometric interpretation of this algebraic finding. This small diversion will also provide more insight in the interpretation of \mathbf{v}_i and \mathbf{u}_i , $i = 1, 2, \dots, l$, which appear in the SVD method.

Recall that $X^T X$ is a scaled version of the sample covariance matrix for centered regressors. Also, by the definition of the \mathbf{v}_i s, we have

$$(X^T X)\mathbf{v}_i = \sigma_i^2 \mathbf{v}_i, \quad i = 1, 2, \dots, l,$$

**FIGURE 6.4**

The singular vector \mathbf{v}_1 , which is associated with the singular value $\sigma_1 > \sigma_2$, points to the direction where most of the (variance) activity in the data space takes place. The variance in the direction of \mathbf{v}_2 is smaller.

and in a compact form,

$$\begin{aligned} (X^T X) V_l &= V_l \operatorname{diag}\{\sigma_1^2, \dots, \sigma_l^2\} \Rightarrow \\ (X^T X) &= V_l D^2 V_l^T = \sum_{i=1}^l \sigma_i^2 \mathbf{v}_i \mathbf{v}_i^T, \end{aligned} \quad (6.31)$$

where the orthogonality property of V_l has been used for the inversion. Note that in (6.31), the (scaled) sample covariance matrix is written as a sum of rank one matrices, $\mathbf{v}_i \mathbf{v}_i^T$, each one weighted by the square of the respective singular value, σ_i^2 . We are now close to revealing the physical/geometric meaning of the singular values. To this end, define

$$\mathbf{q}_j := X \mathbf{v}_j = \begin{bmatrix} \mathbf{x}_1^T \mathbf{v}_j \\ \vdots \\ \mathbf{x}_N^T \mathbf{v}_j \end{bmatrix} \in \mathbb{R}^N, \quad j = 1, 2, \dots, l. \quad (6.32)$$

Note that \mathbf{q}_j is a vector in the column space of X . Moreover, the respective squared norm of \mathbf{q}_j is given by

$$\sum_{n=1}^N q_j^2(n) = \mathbf{q}_j^T \mathbf{q}_j = \mathbf{v}_j^T X^T X \mathbf{v}_j = \mathbf{v}_j^T \left(\sum_{i=1}^l \sigma_i^2 \mathbf{v}_i \mathbf{v}_i^T \right) \mathbf{v}_j = \sigma_j^2,$$

due to the orthonormality of the \mathbf{v}_j s. That is, σ_j^2 is equal to the (scaled) sample variance of the elements of \mathbf{q}_j . However, by the definition in (6.32), this is the sample variance of the projections of the input vectors (regressors), \mathbf{x}_n , $n = 1, 2, \dots, N$, along the direction \mathbf{v}_j . The larger the value of σ_j , the larger the spread of the (input) data along the respective direction. This is shown in Fig. 6.4, where $\sigma_1 \gg \sigma_2$. From the variance point of view, \mathbf{v}_1 is the more *informative* direction, compared to \mathbf{v}_2 . It is the direction

where most of the activity takes place. This observation is at the heart of *dimensionality reduction*, which will be treated in more detail in Chapter 19. Moreover, from (6.18), we obtain

$$\mathbf{q}_j = X\mathbf{v}_j = \sigma_j \mathbf{u}_j. \quad (6.33)$$

In other words, \mathbf{u}_j points in the direction of \mathbf{q}_j . Thus, (6.30) suggests that while projecting \mathbf{y} onto the column space of X , the directions \mathbf{u}_j associated with larger values of variance are weighted more heavily than the rest. *Ridge regression respects and assigns higher weights to the more informative directions, where most of the data activity takes place.* Alternatively, the less important directions, those associated with small data variance, are shrunk the most.

One final comment concerning ridge regression is that the ridge solutions are not invariant under scaling of the input variables. This becomes obvious by looking at the respective equations. Thus, in practice, often the input variables are standardized to unit variances.

PRINCIPAL COMPONENTS REGRESSION

We have just seen that the effect of the ridge regression is to enforce a shrinking rule on the parameters, which decreases the contribution of the less important of the components \mathbf{u}_i in the respective summation. This can be considered as a soft shrinkage rule. An alternative path is to adopt a hard thresholding rule and keep only the m most significant directions, known as the *principal axes* or *directions*, and forget the rest by setting the respective weights equal to zero. Equivalently, we can write

$$\hat{\mathbf{y}} = \sum_{i=1}^m \hat{\theta}_i \mathbf{u}_i, \quad (6.34)$$

where

$$\hat{\theta}_i = \mathbf{u}_i^T \mathbf{y}, \quad i = 1, 2, \dots, m. \quad (6.35)$$

Furthermore, employing (6.18) we have

$$\hat{\mathbf{y}} = \sum_{i=1}^m \frac{\hat{\theta}_i}{\sigma_i} X \mathbf{v}_i, \quad (6.36)$$

or equivalently, the weights for the expansion of the solution in terms of the input data can be expressed as

$$\boldsymbol{\theta} = \sum_{i=1}^m \frac{\hat{\theta}_i}{\sigma_i} \mathbf{v}_i. \quad (6.37)$$

In other words, the prediction $\hat{\mathbf{y}}$ is performed in a subspace of the column space of X , which is spanned by the m principal axes, that is, the subspace where most of the data activity takes place.

6.6 THE RECURSIVE LEAST-SQUARES ALGORITHM

In previous chapters, we discussed the need for developing recursive algorithms that update the estimates every time a new pair of input–output observations is received. Solving the LS problem using a general purpose solver would amount to $\mathcal{O}(l^3)$ multiplications and additions (MADS), due to the involved matrix inversion. Also, $\mathcal{O}(Nl^2)$ operations are required to compute the (scaled) sample covariance matrix $X^T X$. In this section, the special structure of $X^T X$ will be taken into account in order to obtain a computationally efficient online scheme for the solution of the LS task. Moreover, when dealing with time-recursive techniques, one can also care for time variations of the statistical properties of the involved data. We will allow for such applications, and the sum of squared errors cost will be slightly modified in order to accommodate time-varying environments.

For the needs of the section, our notation will be slightly “enriched” and we will use explicitly the time index, n . Also, to be consistent with the online schemes discussed in Chapter 5, we will assume that the time starts at $n = 0$ and the received observations are (y_n, \mathbf{x}_n) , $n = 0, 1, 2, \dots$. To this end, let us denote the input matrix at time n as

$$X_n^T = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n].$$

Moreover, the cost function in Eq. (6.1) is modified to involve a *forgetting factor*, $0 < \beta \leq 1$. The purpose of its presence is to help the cost function slowly forget past data samples by weighting heavier the more recent observations. This will equip the algorithm with the ability to track changes that occur in the underlying data statistics. Moreover, since we are interested in time-recursive solutions, starting from time $n = 0$, we are forced to introduce regularization. During the initial period, corresponding to time instants $n < l - 1$, the corresponding system of equations will be underdetermined and $X_n^T X_n$ is not invertible. Indeed, we have

$$X_n^T X_n = \sum_{i=0}^n \mathbf{x}_i \mathbf{x}_i^T.$$

In other words, $X_n^T X_n$ is the sum of rank one matrices. Hence, for $n < l - 1$ its rank is necessarily less than l , and it cannot be inverted (see Appendix A). For larger values of n , it can become full rank, provided that at least l of the input vectors are linearly independent, which is usually assumed to be the case. The previous arguments lead to the following modifications of the “conventional” least-squares, known as the *exponentially weighted sum of squared errors* cost function, minimized by

$$\boldsymbol{\theta}_n = \arg \min_{\boldsymbol{\theta}} \left(\sum_{i=0}^n \beta^{n-i} (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2 + \lambda \beta^{n+1} \|\boldsymbol{\theta}\|^2 \right),$$

(6.38)

where β is a user-defined parameter very close to unity, for example, $\beta = 0.999$. In this way, the more recent samples are weighted heavier than the older ones. Note that the regularizing parameter has been made time-varying. This is because for large values of n , no regularization is required. Indeed, for $n > l$, matrix $X_n^T X_n$ becomes, in general, invertible. Moreover, recall from Chapter 3 that the use of regularization also takes precautions for overfitting. However, for very large values of $n \gg l$, this is not a problem, and one wishes to get rid of the imposed bias. The parameter $\lambda > 0$ is also a user-defined variable and its choice will be discussed later on.

Minimizing (6.38) results in

$$\Phi_n \boldsymbol{\theta}_n = \mathbf{p}_n, \quad (6.39)$$

where

$$\Phi_n = \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T + \lambda \beta^{n+1} I \quad (6.40)$$

and

$$\mathbf{p}_n = \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i y_i, \quad (6.41)$$

which for $\beta = 1$ coincides with the ridge regression.

TIME-ITERATIVE COMPUTATIONS

By the respective definitions, we have

$$\Phi_n = \beta \Phi_{n-1} + \mathbf{x}_n \mathbf{x}_n^T, \quad (6.42)$$

and

$$\mathbf{p}_n = \beta \mathbf{p}_{n-1} + \mathbf{x}_n y_n. \quad (6.43)$$

Recall Woodbury's matrix inversion formula (Appendix A.1),

$$(A + BD^{-1}C)^{-1} = A^{-1} - A^{-1}B(D + CA^{-1}B)^{-1}CA^{-1}.$$

Plugging it in (6.42), after the appropriate inversion and substitutions we obtain

$$\boxed{\Phi_n^{-1} = \beta^{-1} \Phi_{n-1}^{-1} - \beta^{-1} \mathbf{k}_n \mathbf{x}_n^T \Phi_{n-1}^{-1}, \quad (6.44)}$$

$$\boxed{\mathbf{k}_n = \frac{\beta^{-1} \Phi_{n-1}^{-1} \mathbf{x}_n}{1 + \beta^{-1} \mathbf{x}_n^T \Phi_{n-1}^{-1} \mathbf{x}_n}. \quad (6.45)}$$

The term \mathbf{k}_n is known as the *Kalman gain*. For notational convenience, define

$$P_n = \Phi_n^{-1}.$$

Also, rearranging the terms in (6.45), we get

$$\mathbf{k}_n = \left(\beta^{-1} P_{n-1} - \beta^{-1} \mathbf{k}_n \mathbf{x}_n^T P_{n-1} \right) \mathbf{x}_n,$$

and taking into account (6.44) results in

$$\mathbf{k}_n = P_n \mathbf{x}_n. \quad (6.46)$$

TIME UPDATING OF THE PARAMETERS

From (6.39) and (6.43)–(6.45) we obtain

$$\begin{aligned}\boldsymbol{\theta}_n &= \left(\beta^{-1} P_{n-1} - \beta^{-1} \mathbf{k}_n \mathbf{x}_n^T P_{n-1} \right) \beta \mathbf{p}_{n-1} + P_n \mathbf{x}_n y_n \\ &= \boldsymbol{\theta}_{n-1} - \mathbf{k}_n \mathbf{x}_n^T \boldsymbol{\theta}_{n-1} + \mathbf{k}_n y_n,\end{aligned}$$

and finally,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mathbf{k}_n e_n, \quad (6.47)$$

where,

$$e_n := y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n. \quad (6.48)$$

The derived algorithm is summarized in Algorithm 6.1.

Note that the basic recursive update of the vector of the parameters follows the same rationale as the LMS and the gradient descent schemes that were discussed in Chapter 5. The updated estimate of the parameters $\boldsymbol{\theta}_n$ at time n equals that of the previous time instant, $n - 1$, plus a correction term that is proportional to the error e_n . As a matter of fact, this is the *generic scheme* that we are going to meet for all the recursive algorithms in this book, including those used for the very “trendy” case of neural networks. The main difference from algorithm to algorithm lies in how one computes the multiplicative factor for the error. In the case of the LMS, this factor was equal to $\mu \mathbf{x}_n$. For the case of the RLS, this factor is \mathbf{k}_n . As we shall soon see, the RLS algorithm is closely related to an alternative to the gradient descent family of optimization algorithms, known as Newton’s iterative optimization of cost functions.

Algorithm 6.1 (The RLS algorithm).

- Initialize
 - $\boldsymbol{\theta}_{-1} = \mathbf{0}$; any other value is also possible.
 - $P_{-1} = \lambda^{-1} I$; $\lambda > 0$ a user-defined variable.
 - Select β ; close to 1.
- For $n = 0, 1, \dots$, Do
 - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$
 - $\mathbf{z}_n = P_{n-1} \mathbf{x}_n$
 - $\mathbf{k}_n = \frac{\mathbf{z}_n}{\beta + \mathbf{x}_n^T \mathbf{z}_n}$
 - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mathbf{k}_n e_n$
 - $P_n = \beta^{-1} P_{n-1} - \beta^{-1} \mathbf{k}_n \mathbf{z}_n^T$
- End For

Remarks 6.2.

- The complexity of the RLS algorithm is of the order $\mathcal{O}(l^2)$ per iteration, due to the matrix-product operations. That is, there is an order of magnitude difference compared to the LMS and the other schemes that were discussed in Chapter 5. In other words, RLS does not scale well with dimensionality.

- The RLS algorithm shares similar numerical behavior with the Kalman filter, which was discussed in Section 4.10; P_n may lose its positive definite and symmetric nature, which then leads the algorithm to divergence. To remedy such a tendency, symmetry preserving versions of the RLS algorithm have been derived; see [65,68]. Note that the use of $\beta < 1$ has a beneficial effect on the error propagation [30,34]. In [58], it is shown that for $\beta = 1$ the error propagation mechanism is of a random walk type, and hence the algorithm is unstable. In [5], it is pointed out that due to numerical errors the term $\frac{1}{\beta + \mathbf{x}_n^T P_{n-1} \mathbf{x}_n}$ may become negative, leading to divergence. The numerical performance of the RLS becomes a more serious concern in implementations using limited precision, such as fixed point arithmetic. Compared to the LMS, RLS would require the use of higher precision implementations; otherwise, divergence may occur after a few iteration steps. This adds further to its computational disadvantage compared to the LMS.
- The choice of λ in the initialization step has been considered in [46]. The related theoretical analysis suggests that λ has a direct influence on the convergence speed, and it should be chosen so as to be a small positive for high signal-to-noise (SNR) ratios and a large positive constant for low SNRs.
- In [56], it has been shown that the RLS algorithm can be obtained as a special case of the Kalman filter.
- The main advantage of the RLS is that it converges to the steady state much faster than the LMS and the rest of the members of the LMS family. This can be justified by the fact that the RLS can be seen as an offspring of Newton's iterative optimization method.
- Distributed versions of the RLS have been proposed in [8,39,40].

6.7 NEWTON'S ITERATIVE MINIMIZATION METHOD

The gradient descent formulation was presented in Chapter 5. It was noted that it exhibits a linear convergence rate and a heavy dependence on the condition number of the Hessian matrix associated with the cost function. Newton's method is a way to overcome this dependence on the condition number and at the same time improve upon the rate of convergence toward the solution.

In Section 5.2, a first-order Taylor expansion was used around the current value $J(\boldsymbol{\theta}^{(i-1)})$. Let us now consider a second-order Taylor expansion (assume $\mu_i = 1$) that involves second-order derivatives,

$$\begin{aligned} J(\boldsymbol{\theta}^{(i-1)} + \Delta\boldsymbol{\theta}^{(i)}) &= J(\boldsymbol{\theta}^{(i-1)}) + (\nabla J(\boldsymbol{\theta}^{(i-1)}))^T \Delta\boldsymbol{\theta}^{(i)} \\ &\quad + \frac{1}{2} (\Delta\boldsymbol{\theta}^{(i)})^T \nabla^2 J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}. \end{aligned}$$

Recall that the second derivative, $\nabla^2 J$, being the derivative of the gradient vector, is an $l \times l$ matrix (see Appendix A). Assuming that $\nabla^2 J(\boldsymbol{\theta}^{(i-1)})$ to be positive definite (this is always the case if $J(\boldsymbol{\theta})$ is a strictly convex function⁵), the above turns out to be a convex quadratic function with respect to the

⁵ See Chapter 8 for related definitions.

step $\Delta\theta^{(i)}$; the latter is computed so as to *minimize* the above second-order approximation. Due to the convexity, there is a single minimum, which results by equating the corresponding gradient to $\mathbf{0}$, which results in

$$\Delta\theta^{(i)} = -\left(\nabla^2 J(\theta^{(i-1)})\right)^{-1} \nabla J(\theta^{(i-1)}). \quad (6.49)$$

Note that this is indeed a descent direction, because

$$\nabla^T J(\theta^{(i-1)}) \Delta\theta^{(i)} = -\nabla^T J(\theta^{(i-1)}) \left(\nabla^2 J(\theta^{(i-1)})\right)^{-1} \nabla J(\theta^{(i-1)}) < 0,$$

due to the positive definite nature of the Hessian⁶; equality to zero is achieved only at a minimum, where the gradient becomes zero. Thus, the iterative scheme takes the following form:

$$\theta^{(i)} = \theta^{(i-1)} - \mu_i \left(\nabla^2 J(\theta^{(i-1)})\right)^{-1} \nabla J(\theta^{(i-1)}): \quad \text{Newton's iterative scheme.} \quad (6.50)$$

Fig. 6.5 illustrates the method. Note that if the cost function is quadratic, then the minimum is achieved in one iteration!

As is apparent from the derived recursion, the difference of Newton's schemes with the gradient descent family of algorithms lies in the step size. This is no more a scalar, i.e., μ_i . The step size involves the inverse of a matrix; that is, the second derivative of the cost function with respect to the parameters' vector. Here lies the power and at the same time the drawback of this type of algorithms. The use of the second derivative provides extra information concerning the local shape of the cost (see below) and as a consequence leads to faster convergence. At the same time, it increases substantially

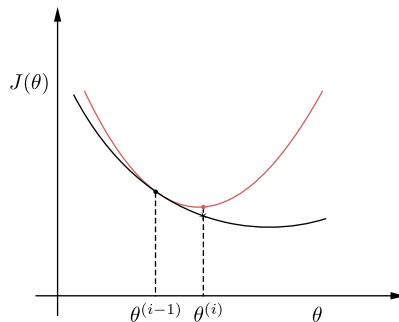
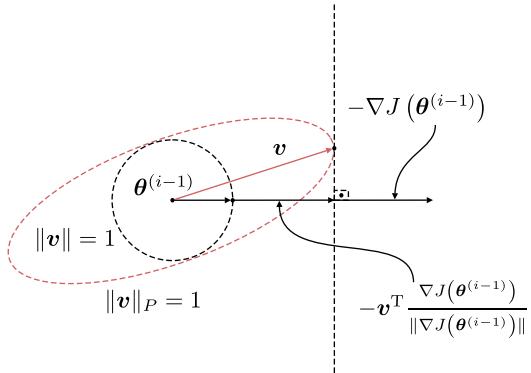


FIGURE 6.5

According to Newton's method, a local quadratic approximation of the cost function is considered (red curve), and the correction pushes the new estimate toward the minimum of this approximation. If the cost function is quadratic, then convergence can be achieved in one step.

⁶ Recall from Appendix A that if A is positive definite, then $\mathbf{x}^T A \mathbf{x} > 0, \forall \mathbf{x}$.

**FIGURE 6.6**

The graphs of the unit Euclidean (black circle) and quadratic (red ellipse) norms centered at $\theta^{(i-1)}$ are shown. In both cases, the goal is to move as far as possible in the direction of $-\nabla J(\theta^{(i-1)})$, while remaining at the ellipse (circle). The result is different for the two cases. The Euclidean norm corresponds to the gradient descent and the quadratic norm to Newton's method.

the computational complexity. Also, matrix inversions need always a careful treatment. The associated matrix may become ill-conditioned, its determinant gets small values, and in such cases numerical stability issues have to be carefully considered.

Observe that in the case of Newton's algorithm, the correction direction is not that of 180° with respect to $\nabla J(\theta^{(i-1)})$, as is the case for the gradient descent method. An alternative point of view is to look at (6.50) as the steepest descent direction under the following norm (see Section 5.2):

$$\|v\|_P = (v^T P v)^{1/2},$$

where P is a symmetric positive definite matrix. For our case, we set

$$P = \nabla^2 J(\theta^{(i-1)}).$$

Then searching for the respective normalized steepest descent direction, i.e.,

$$\begin{aligned} v &= \arg \min_z z^T \nabla J(\theta^{(i-1)}) \\ \text{s.t. } \|z\|_P^2 &= 1, \end{aligned}$$

results in the normalized vector pointing in the same direction as the one in (6.49) (Problem 6.15). For $P = I$, the gradient descent algorithm results. The geometry is illustrated in Fig. 6.6. Note that Newton's direction accounts for the *local shape* of the cost function.

The convergence rate for Newton's method is, in general, high and it becomes quadratic close to the solution. Assuming θ_* to be the minimum, quadratic convergence means that at each iteration i ,

the deviation from the optimum value follows the following pattern:

$$\boxed{\ln \ln \frac{1}{||\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*||^2} \propto i : \text{ quadratic convergence rate.}} \quad (6.51)$$

In contrast, for the linear convergence, the iterations approach the optimal according to

$$\boxed{\ln \frac{1}{||\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*||^2} \propto i : \text{ linear convergence rate.}} \quad (6.52)$$

Furthermore, the presence of the Hessian in the correction term remedies, to a large extent, the influence of the condition number of the Hessian matrix on the convergence [6] (Problem 6.16).

6.7.1 RLS AND NEWTON'S METHOD

The RLS algorithm can be rederived following Newton's iterative scheme applied to the MSE and adopting *stochastic approximation* arguments. Let

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}[(y - \boldsymbol{\theta}^T \mathbf{x})^2] = \frac{1}{2} \sigma_y^2 + \frac{1}{2} \boldsymbol{\theta}^T \Sigma_x \boldsymbol{\theta} - \boldsymbol{\theta}^T \mathbf{p},$$

or

$$-\nabla J(\boldsymbol{\theta}) = \mathbf{p} - \Sigma_x \boldsymbol{\theta} = \mathbb{E}[\mathbf{x}\mathbf{y}] - \mathbb{E}[\mathbf{x}\mathbf{x}^T]\boldsymbol{\theta} = \mathbb{E}[\mathbf{x}(y - \mathbf{x}^T \boldsymbol{\theta})] = \mathbb{E}[\mathbf{x}\mathbf{e}],$$

and

$$\nabla^2 J(\boldsymbol{\theta}) = \Sigma_x.$$

Newton's iteration becomes

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Sigma_x^{-1} \mathbb{E}[\mathbf{x}\mathbf{e}].$$

Following stochastic approximation arguments and replacing iteration steps with time updates and expectations with observations, we obtain

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \Sigma_x^{-1} \mathbf{x}_n e_n.$$

Let us now adopt the approximation,

$$\Sigma_x \simeq \frac{1}{n+1} \Phi_n = \left(\frac{1}{n+1} \lambda \beta^{n+1} I + \frac{1}{n+1} \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T \right),$$

and set

$$\mu_n = \frac{1}{n+1}.$$

Then

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mathbf{k}_n e_n,$$

with

$$\mathbf{k}_n = P_n \mathbf{x}_n,$$

where

$$P_n = \left(\sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T + \lambda \beta^{n+1} I \right)^{-1},$$

which then, by using similar steps as for (6.42)–(6.44), leads to the RLS scheme.

Note that this point of view explains the fast converging properties of the RLS and its relative insensitivity to the condition number of the covariance matrix.

Remarks 6.3.

- In Section 5.5.2, it was pointed out that the LMS is optimal with respect to a min/max robustness criterion. However, this is not true for the RLS. It turns out that while LMS exhibits the best worst-case performance, the RLS is expected to have better performance on average [23].

6.8 STEADY-STATE PERFORMANCE OF THE RLS

Compared to the stochastic gradient techniques, which were considered in Chapter 5, we do not have to worry whether RLS converges and where it converges. The RLS computes the *exact* solution of the minimization task in (6.38) in an iterative way. Asymptotically and for $\beta = 1$, $\lambda = 0$ RLS solves the MSE optimization task. However, we have to consider its steady-state performance for $\beta \neq 1$. Even for the stationary case, $\beta \neq 1$ results in an excess mean-square error. Moreover, it is important to get a feeling of its tracking performance in time-varying environments. To this end, we adopt the same setting as the one followed in Section 5.12. We will not provide all the details of the proof, because this follows similar steps as in the LMS case. We will point out where differences arise and state the results. For the detailed derivation, the interested reader may consult [15,48,57]; in the latter one, the energy conservation theory is employed.

As in Chapter 5, we adopt the following models:

$$\mathbf{y}_n = \boldsymbol{\theta}_{o,n-1}^T \mathbf{x}_n + \boldsymbol{\eta}_n \quad (6.53)$$

and

$$\boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}_n, \quad (6.54)$$

with

$$\mathbb{E}[\boldsymbol{\omega}_n \boldsymbol{\omega}_n^T] = \boldsymbol{\Sigma}_{\omega}.$$

Hence, taking into account (6.53), (6.54), and the RLS iteration involving the respective random variables, we get

$$\boldsymbol{\theta}_n - \boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{n-1} + \mathbf{k}_n \mathbf{e}_n - \boldsymbol{\theta}_{o,n-1} - \boldsymbol{\omega}_n,$$

Table 6.1 The steady-state excess MSE, for small values of μ and β .

Algorithm	Excess MSE, J_{exc} , at steady state
LMS	$\frac{1}{2}\mu\sigma_\eta^2\text{trace}\{\Sigma_x\} + \frac{1}{2}\mu^{-1}\text{trace}\{\Sigma_\omega\}$
APA	$\frac{1}{2}\mu\sigma_\eta^2\text{trace}\{\Sigma_x\}\mathbb{E}\left[\frac{q}{\ x\ ^2}\right] + \frac{1}{2}\mu^{-1}\text{trace}\{\Sigma_x\}\text{trace}\{\Sigma_\omega\}$
RLS	$\frac{1}{2}(1-\beta)\sigma_\eta^2 l + \frac{1}{2}(1-\beta)^{-1}\text{trace}\{\Sigma_\omega\Sigma_x\}$

For $q = 1$, the normalized LMS results. Under a Gaussian input assumption and for long system orders, l , in the APA, $\mathbb{E}\left[\frac{q}{\|x\|^2}\right] \approx \frac{q}{\sigma_x^2(l-2)}$ [11].

or

$$\begin{aligned}\mathbf{c}_n &:= \boldsymbol{\theta}_n - \boldsymbol{\theta}_{o,n} = \mathbf{c}_{n-1} + \mathbf{P}_n \mathbf{x}_n \mathbf{e}_n - \boldsymbol{\omega}_n \\ &= (\mathbf{I} - \mathbf{P}_n \mathbf{x}_n \mathbf{x}_n^T) \mathbf{c}_{n-1} + \mathbf{P}_n \mathbf{x}_n \mathbf{\eta}_n - \boldsymbol{\omega}_n,\end{aligned}$$

which is the counterpart of (5.76). Note that the time indices for the input and the noise variables can be omitted, because their statistics is assumed to be time-invariant.

We adopt the same assumptions as in Section 5.12. In addition, we assume that \mathbf{P}_n is changing slowly compared to \mathbf{c}_n . Hence, every time \mathbf{P}_n appears inside an expectation, it is substituted by its mean $\mathbb{E}[\mathbf{P}_n]$, i.e.,

$$\mathbb{E}[\mathbf{P}_n] = \mathbb{E}[\Phi_n^{-1}],$$

where

$$\Phi_n = \lambda\beta^{n+1}\mathbf{I} + \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T$$

and

$$\mathbb{E}[\Phi_n] = \lambda\beta^{n+1}\mathbf{I} + \frac{1 - \beta^{n+1}}{1 - \beta} \Sigma_x.$$

Assuming $\beta \simeq 1$, the variance at the steady state of Φ_n can be considered small and we can adopt the following approximation:

$$\mathbb{E}[\mathbf{P}_n] \simeq [\mathbb{E}[\Phi_n]]^{-1} = \left[\beta^{n+1} \lambda \mathbf{I} + \frac{1 - \beta^{n+1}}{1 - \beta} \Sigma_x \right]^{-1}.$$

Based on all the previously stated assumptions, repeating carefully the same steps as in Section 5.12, we end up with the result shown in Table 6.1, which holds for small values of β . For comparison reasons, the excess MSE is shown together with the values obtained for the LMS as well as the APA algorithms. In stationary environments, one simply sets $\Sigma_\omega = 0$.

According to Table 6.1, the following remarks are in order.

Remarks 6.4.

- For stationary environments, the performance of the RLS is independent of Σ_x . Of course, if one knows that the environment is stationary, then ideally $\beta = 1$ should be the choice. Yet recall that for $\beta = 1$, the algorithm has stability problems.
- Note that for small μ and $\beta \simeq 1$, there is an “equivalence” of $\mu \simeq 1 - \beta$, for the two parameters in the LMS and RLS. That is, larger values of μ are beneficial to the tracking performance of LMS, while smaller values of β are required for faster tracking; this is expected because the algorithm forgets the past.
- It is clear from Table 6.1 that an algorithm may converge to the steady state quickly, but it may not necessarily track fast. It all depends on the specific scenario. For example, under the modeling assumptions associated with Table 6.1, the optimal value μ_{opt} for the LMS (Section 5.12) is given by

$$\mu_{\text{opt}} = \sqrt{\frac{\text{trace}\{\Sigma_\omega\}}{\sigma_\eta^2 \text{trace}\{\Sigma_x\}}},$$

which corresponds to

$$J_{\min}^{\text{LMS}} = \sqrt{\sigma_\eta^2 \text{trace}\{\Sigma_x\} \text{trace}\{\Sigma_\omega\}}.$$

Optimizing with respect to β for the RLS, it is easily shown that

$$\begin{aligned}\beta_{\text{opt}} &= 1 - \sqrt{\frac{\text{trace}\{\Sigma_\omega \Sigma_x\}}{\sigma_\eta^2 l}}, \\ J_{\min}^{\text{RLS}} &= \sqrt{\sigma_\eta^2 l \text{trace}\{\Sigma_\omega \Sigma_x\}}.\end{aligned}$$

Hence, the ratio

$$\boxed{\frac{J_{\min}^{\text{LMS}}}{J_{\min}^{\text{RLS}}} = \sqrt{\frac{\text{trace}\{\Sigma_x\} \text{trace}\{\Sigma_\omega\}}{l \text{trace}\{\Sigma_\omega \Sigma_x\}}}}$$

depends on Σ_ω and Σ_x . Sometimes LMS tracks better, yet in other problems RLS is the winner. Having said that, it must be pointed out that the RLS always converges faster, and the difference in the rate, compared to the LMS, increases with the condition number of the input covariance matrix.

6.9 COMPLEX-VALUED DATA: THE WIDELY LINEAR RLS

Following similar arguments as in Section 5.7, let

$$\boldsymbol{\varphi} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{v} \end{bmatrix}, \quad \tilde{\mathbf{x}}_n = \begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_n^* \end{bmatrix},$$

with

Because only one component is updated at each iteration, this greatly simplifies the update mechanism. The method is known as *coordinate descent*.

Based on this rationale, a number of variants of the basic coordinate descent have been proposed. The *cyclic coordinate descent* (CCD) method in its simplest form entails a *cyclic* update with respect to one coordinate per iteration cycle; that is, at the i th iteration the following minimization is solved:

$$\theta_k^{(i)} := \arg \min_{\theta} J(\theta_1^{(i)}, \dots, \theta_{k-1}^{(i)}, \theta, \theta_{k+1}^{(i-1)}, \dots, \theta_l^{(i-1)}).$$

In words, all components but θ_k are assumed constant; those components θ_j , $j < k$, are fixed to their updated values, $\theta_j^{(i)}$, $j = 1, 2, \dots, k-1$, and the rest, θ_j , $j = k+1, \dots, l$, to the available estimates, $\theta_j^{(i-1)}$, from the previous iteration. The nice feature of such a technique is that a simple closed form solution for the minimizer may be obtained. A revival of such techniques has happened in the context of sparse learning models (Chapter 10) [18,67]. Convergence issues of CCD have been considered in [36,62]. CCD algorithms for the LS task have also been considered in [66] and the references therein. Besides the basic CCD scheme, variants are also available, using different scenarios for the choice of the direction to be updated each time, in order to improve convergence, ranging from a random choice to a change of the coordinate systems, which is known as an *adaptive coordinate descent* scheme [35].

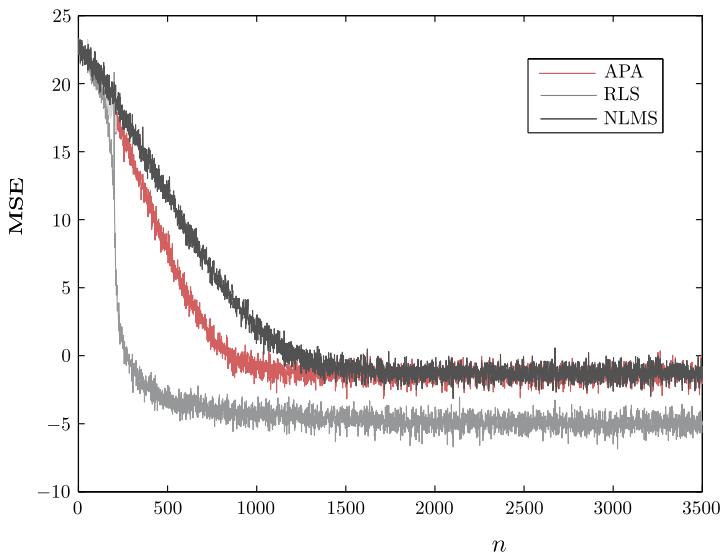
6.12 SIMULATION EXAMPLES

In this section, simulation examples are presented concerning the convergence and tracking performance of the RLS compared to algorithms of the gradient descent family, which have been derived in Chapter 5.

Example 6.1. The focus of this example is to demonstrate the comparative performance, with respect to the convergence rate of the RLS, NLMS, and APA algorithms, which have been discussed in Chapter 5. To this end, we generate data according to the regression model

$$y_n = \boldsymbol{\theta}_o^T \mathbf{x}_n + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^{200}$. Its elements are generated randomly according to the normalized Gaussian. The noise samples are i.i.d. generated via the zero mean Gaussian with variance equal to $\sigma_\eta^2 = 0.01$. The elements of the input vector are also i.i.d. generated via the normalized Gaussian. Using the generated samples (y_n, \mathbf{x}_n) , $n = 0, 1, \dots$, as the training sequence for all three previously stated algorithms, the convergence curves of Fig. 6.8 are obtained. The curves show the squared error in dBs ($10 \log_{10}(e_n^2)$), averaged over 100 different realizations of the experiments, as a function of the time index n . The following parameters were used for the involved algorithms. (a) For the NLMS, we used $\mu = 1.2$ and $\delta = 0.001$; (b) for the APA, we used $\mu = 0.2$, $\delta = 0.001$, and $q = 30$; and (c) for the RLS, we used $\beta = 1$ and $\lambda = 0.1$. The parameters for the NLMS algorithm and the APA were chosen so that both algorithms converge to the same error floor. The improved performance of the APA concerning the convergence rate compared to the NLMS is readily seen. However, both algorithms fall short when compared to the RLS. Note that the RLS converges to lower error floor, because no forgetting factor

**FIGURE 6.8**

MSE curves as a function of the number of iterations for NLMS, APA, and RLS. The RLS converges faster and at lower error floor.

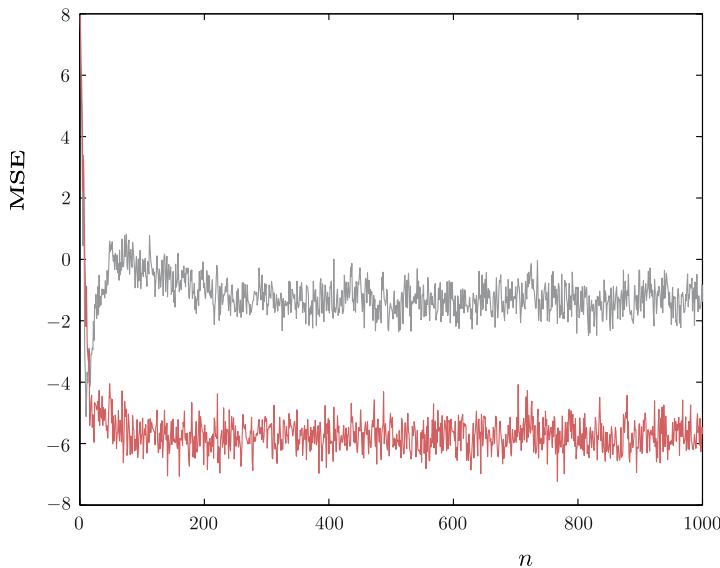
was used. To be consistent, a forgetting factor $\beta < 1$ should have been used in order for this algorithm to settle at the same error floor as the other two algorithms; this would have a beneficial effect on the convergence rate. However, having chosen $\beta = 1$, it is demonstrated that the RLS can converge really fast, even to lower error floors. On the other hand, this improved performance is obtained at substantial higher complexity. In case the input vector is part of a random process, and the special time shift structure can be exploited, as discussed in Section 6.10, the lower-complexity versions are at the disposal of the designer. A further comparative performance example, including another family of online algorithms, will be given in Chapter 8.

However, it has to be stressed that this notable advantage (between RLS- and LMS-type schemes) in convergence speed from the initial conditions to steady state may not be the case concerning the tracking performance, when the algorithms have to track time-varying environments. This is demonstrated next.

Example 6.2. This example focuses on the comparative tracking performance of the RLS and NLMS. Our goal is to demonstrate some cases where the RLS fails to do as well as the NLMS. Of course, it must be kept in mind that according to the theory, the comparative performance is very much dependent on the specific application.

For the needs of our example, let us mobilize the time-varying model of the parameters given in (6.54) in its more practical version and generate the data according to the following linear system:

$$y_n = \mathbf{x}_n^T \boldsymbol{\theta}_{o,n-1} + \eta_n, \quad (6.63)$$

**FIGURE 6.9**

For a fast time-varying parameter model, the RLS (gray) fails to track it, in spite of its very fast initial convergence, compared to the NLMS (red).

where

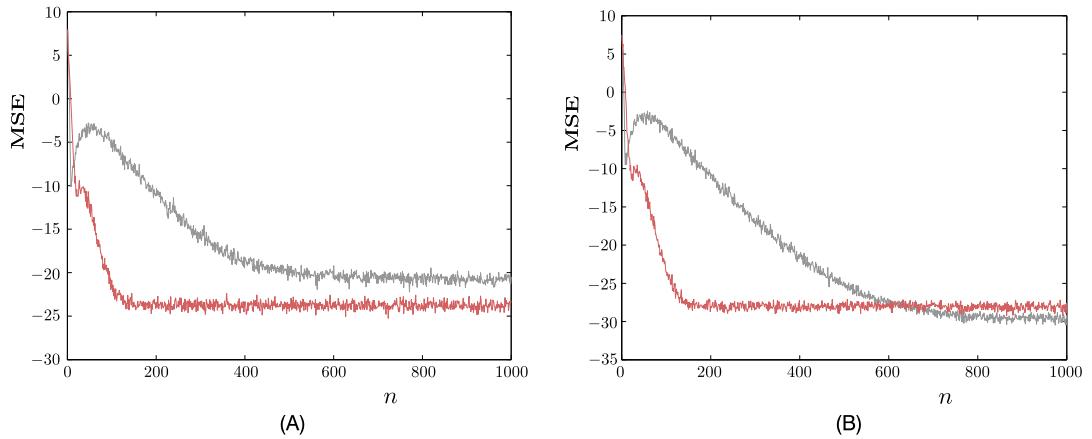
$$\boldsymbol{\theta}_{o,n} = \alpha \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}_n,$$

with $\boldsymbol{\theta}_{o,n} \in \mathbb{R}^5$. It turns out that such a time-varying model is closely related (for the right choice of the involved parameters) to what is known in communications as a Rayleigh fading channel, if the parameters comprising $\boldsymbol{\theta}_{o,n}$ are thought to represent the impulse response of such a channel [57]. Rayleigh fading channels are very common and can adequately model a number of transmission channels in wireless communications. Playing with the parameter α and the variance of the corresponding noise source, $\boldsymbol{\omega}$, one can achieve fast or slow time-varying scenarios. In our case, we chose $\alpha = 0.97$ and the noise followed a Gaussian distribution of zero mean and covariance matrix $\Sigma_{\omega} = 0.1I$.

Concerning the data generation, the input samples were generated i.i.d. from a Gaussian $\mathcal{N}(0, 1)$, and the noise was also Gaussian of zero mean value and variance equal to $\sigma_{\eta}^2 = 0.01$. Initialization of the time-varying model ($\boldsymbol{\theta}_{o,0}$) was randomly done by drawing samples from $\mathcal{N}(0, 1)$.

Fig. 6.9 shows the obtained MSE curve as a function of the iterations for the NLMS and the RLS. For the RLS, the forgetting factor was set equal to $\beta = 0.995$ and for the NLMS, $\mu = 0.5$ and $\delta = 0.001$. Such a choice resulted in the best performance, for both algorithms, after extensive experimentation. The curves are the result of averaging out 200 independent runs.

Fig. 6.10 shows the resulting curves for medium and slow time-varying channels, corresponding to $\Sigma_{\omega} = 0.01I$ and $\Sigma_{\omega} = 0.001I$, respectively.

**FIGURE 6.10**

MSE curves as a function of iteration for (A) a medium and (B) a slow time-varying parameter model. The red curve corresponds to the NLMS and the gray one to the RLS.

6.13 TOTAL LEAST-SQUARES

In this section, the LS task will be formulated from a different perspective. Assume zero mean (centered) data and our familiar linear regression model, employing the observed samples,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta},$$

as in Section 6.2. We have seen that the LS task is equivalent to (orthogonally) projecting \mathbf{y} onto the $\text{span}\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$ of the columns of \mathbf{X} , hence making the error

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

orthogonal to the column space of \mathbf{X} . Equivalently, this can be written as

$$\begin{aligned} & \text{minimize} && \|\mathbf{e}\|^2, \\ & \text{s.t.} && \mathbf{y} - \mathbf{e} \in \mathcal{R}(\mathbf{X}), \end{aligned} \tag{6.64}$$

where $\mathcal{R}(\mathbf{X})$ is the range space of \mathbf{X} (see Remarks 6.1 for the respective definition). Moreover, once $\hat{\boldsymbol{\theta}}_{\text{LS}}$ has been obtained, we can write

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\theta}}_{\text{LS}} = \mathbf{y} - \mathbf{e}$$

or

$$[\mathbf{X} : \mathbf{y} - \mathbf{e}] \begin{bmatrix} \hat{\boldsymbol{\theta}}_{\text{LS}} \\ -1 \end{bmatrix} = \mathbf{0}, \tag{6.65}$$

CLASSIFICATION: A TOUR OF THE CLASSICS

7

CONTENTS

7.1	Introduction	301
7.2	Bayesian Classification	302
	The Bayesian Classifier Minimizes the Misclassification Error	303
	7.2.1 Average Risk	304
7.3	Decision (Hyper)Surfaces	307
	7.3.1 The Gaussian Distribution Case	309
	<i>Minimum Distance Classifiers</i>	311
7.4	The Naive Bayes Classifier	315
7.5	The Nearest Neighbor Rule	315
7.6	Logistic Regression	317
7.7	Fisher's Linear Discriminant	322
	7.7.1 Scatter Matrices	323
	7.7.2 Fisher's Discriminant: the Two-Class Case	325
	7.7.3 Fisher's Discriminant: the Multiclass Case	328
7.8	Classification Trees	329
7.9	Combining Classifiers	333
	No Free Lunch Theorem	334
	Some Experimental Comparisons	334
	Schemes for Combining Classifiers	335
7.10	The Boosting Approach	337
	The AdaBoost Algorithm	337
	The Log-Loss Function	341
7.11	Boosting Trees	343
Problems		345
	MATLAB® Exercises	347
References		349

7.1 INTRODUCTION

The classification task was introduced in Chapter 3. There, it was pointed out that, in principle, one could employ the same loss functions as those used for regression in order to optimize the design of a classifier; however, for most cases in practice, this is not the most reasonable way to attack such problems. This is because in classification the output random variable, y , is of a *discrete nature*; hence, different measures than those used for the regression task are more appropriate for quantifying performance quality.

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

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

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

7.2 BAYESIAN CLASSIFICATION

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

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

Bayesian classification rule: Given a set of M classes, ω_i , $i = 1, 2, \dots, M$, and the respective *posterior probabilities* $P(\omega_i|\mathbf{x})$, classify an unknown feature vector, \mathbf{x} , according to the following rule¹:

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

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

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

¹ Recall that probability values for discrete random variables are denoted by capital P , and PDFs, for continuous random variables, by lower case p .

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

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

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

and the respective threshold value x_r is computed by the equation

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

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

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

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

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

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

7.3 DECISION (HYPER)SURFACES

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

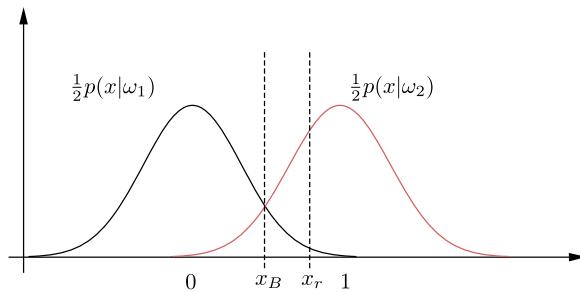
$$g : \mathbb{R}^l \mapsto \mathbb{R},$$

and it comprises all the points such that

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

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

$$g(\mathbf{x}) := P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0. \quad (7.15)$$

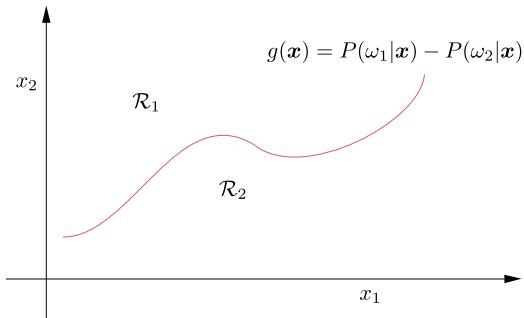
**FIGURE 7.2**

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

Indeed, we decide in favor of class ω_1 (region \mathcal{R}_1) if \mathbf{x} falls on the positive side of the hypersurface defined in (7.15), and in favor of ω_2 for the points falling on the negative side (region \mathcal{R}_2). This is illustrated in Fig. 7.3. At this point, recall the reject option from Remarks 7.1. Points where no decision is taken are those that lie close to the decision hypersurface.

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

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

**FIGURE 7.3**

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

7.3.1 THE GAUSSIAN DISTRIBUTION CASE

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

$$p(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^{l/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, 2, \dots, M. \quad (7.16)$$

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

$$g_i(\mathbf{x}) := \ln(p(\mathbf{x}|\omega_i)P(\omega_i)) = \ln p(\mathbf{x}|\omega_i) + \ln P(\omega_i), \quad i = 1, 2, \dots, M, \quad (7.17)$$

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

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

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) = 0, \quad (7.18)$$

which, after plugging into (7.17) the specific forms of the Gaussian conditionals, and after a bit of trivial algebra, becomes

$$\begin{aligned} g(\mathbf{x}) &= \underbrace{\frac{1}{2} \left(\mathbf{x}^T \Sigma_2^{-1} \mathbf{x} - \mathbf{x}^T \Sigma_1^{-1} \mathbf{x} \right)}_{\text{quadratic terms}} \\ &\quad + \underbrace{\boldsymbol{\mu}_1^T \Sigma_1^{-1} \mathbf{x} - \boldsymbol{\mu}_2^T \Sigma_2^{-1} \mathbf{x}}_{\text{linear terms}} \\ &\quad - \underbrace{\frac{1}{2} \boldsymbol{\mu}_1^T \Sigma_1^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_2^T \Sigma_2^{-1} \boldsymbol{\mu}_2 + \ln \frac{P(\omega_1)}{P(\omega_2)} + \frac{1}{2} \ln \frac{|\Sigma_2|}{|\Sigma_1|}}_{\text{constant terms}} = 0. \end{aligned} \quad (7.19)$$

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

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

and

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

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

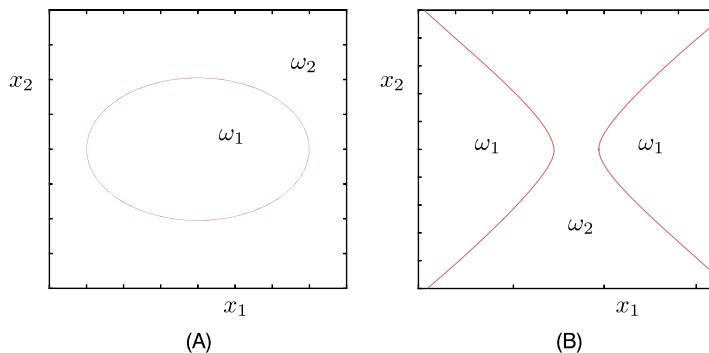


FIGURE 7.4

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

Looking carefully at (7.19), it is readily noticed that once the covariance matrices for the two classes become equal, the quadratic terms cancel out and the discriminant function becomes linear; thus, the corresponding hypersurface is a *hyperplane*. That is, under the previous assumptions, the optimal Bayesian classifier becomes a *linear* classifier, which after some straightforward algebraic manipulations (try it) can be written as

$$g(\mathbf{x}) = \boldsymbol{\theta}^T (\mathbf{x} - \mathbf{x}_0) = 0, \quad (7.20)$$

$$\theta := \Sigma^{-1}(\mu_1 - \mu_2), \quad (7.21)$$

$$x_0 := \frac{1}{2}(\mu_1 + \mu_2) - \ln \frac{P(\omega_1)}{P(\omega_2)} \frac{\mu_1 - \mu_2}{||\mu_1 - \mu_2||^2_{\Sigma^{-1}}}, \quad (7.22)$$

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

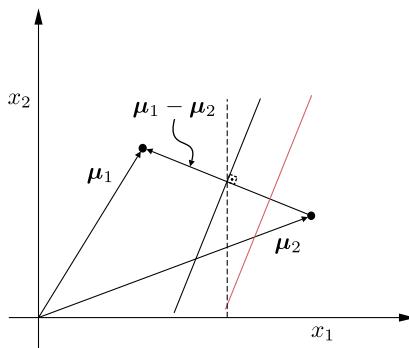
$$||\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2||_{\Sigma^{-1}} := \sqrt{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)}$$

is the Σ^{-1} norm of the vector $(\mu_1 - \mu_2)$; alternatively, this is also known as the *Mahalanobis distance* between μ_1 and μ_2 . The Mahalanobis distance is a generalization of the Euclidean distance; note that for $\Sigma = I$ it becomes the Euclidean distance.

Fig. 7.5 shows three cases for the two-dimensional space. The full black line corresponds to the case of equiprobable classes with a covariance matrix of the special form, $\Sigma = \sigma^2 I$. The corresponding decision hyperplane, according to Eq. (7.20), is now written as

$$g(\mathbf{x}) \equiv (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T (\mathbf{x} - \mathbf{x}_0) \equiv 0. \quad (7.23)$$

The separating line (hyperplane) crosses the middle point of the line segment joining the mean value points, μ_1 and μ_2 ($x_0 = \frac{1}{2}(\mu_1 + \mu_2)$). Also, it is perpendicular to this segment, defined by the vector $\mu_1 - \mu_2$, as is readily verified by the above hyperplane definition. Indeed, for any point x on the

**FIGURE 7.5**

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

decision curve (full black line in Fig. 7.5), since the middle point x_0 also lies on this curve, the corresponding vector $x - x_0$ is parallel to this line. Hence, the inner product in Eq. (7.23) being equal to zero means that the decision line is perpendicular to $\mu_1 - \mu_2$. The red line corresponds to the case where $P(\omega_1) > P(\omega_2)$. It gets closer to the mean value point of class ω_2 , thus enlarging the region where one decides in favor of the more probable class. Note that in this case, the logarithm of the ratio in Eq. (7.22) is positive. Finally, the dotted line corresponds to the equiprobable case with the common covariance matrix being of a more general form, $\Sigma \neq \sigma^2 I$. The separating hyperplane crosses x_0 but it is rotated in order to be perpendicular to the vector $\Sigma^{-1}(\mu_1 - \mu_2)$, according to (7.20)–(7.21). For each one of the three cases, an unknown point is classified according to the side of the respective hyperplane on which it lies.

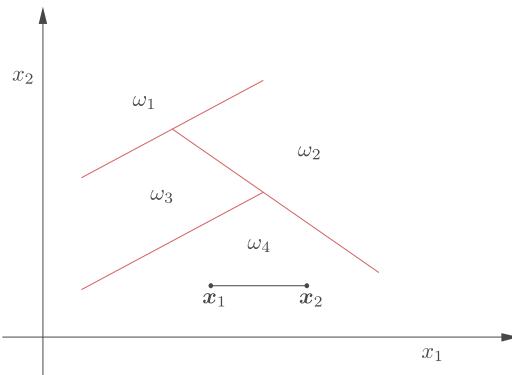
What was said before for the two-class task is generalized to the more general M -class problem; the separating hypersurfaces of two contiguous regions $\mathcal{R}_i, \mathcal{R}_j$ associated with classes ω_i, ω_j obey the same arguments as the ones adopted before. For example, assuming that all covariance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

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

Minimum Distance Classifiers

There are two special cases, where the optimal Bayesian classifier becomes very simple to compute and it also has a strong geometric flavor.

- *Minimum Euclidean distance classifier:* Under the assumptions of (a) Gaussian distributed data in each one of the classes, i.e., Eq. (7.16), (b) equiprobable classes, and (c) common covariance matrix

**FIGURE 7.6**

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

in all classes of the special form $\Sigma = \sigma^2 I$ (individual features are independent and share a common variance), the Bayesian classification rule becomes equivalent to

$$\text{Assign } \mathbf{x} \text{ to class } \omega_i : i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T (\mathbf{x} - \boldsymbol{\mu}_j), \quad j = 1, 2, \dots, M. \quad (7.24)$$

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

For the case of the two classes, this classification rule corresponds to the full black line of Fig. 7.5. Indeed, recalling our geometry basics, any point that lies on the left side of this hyperplane that bisects the segment $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$ is closer to $\boldsymbol{\mu}_1$ than to $\boldsymbol{\mu}_2$, in the Euclidean distance sense. The opposite is true for any point lying on the right of the hyperplane.

- *Minimum Mahalanobis distance classifier:* Under the previously adopted assumptions, but with the covariance matrix being of the more general form $\Sigma \neq \sigma^2 I$, the rule becomes

$$\text{Assign } \mathbf{x} \text{ to class } \omega_i : i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_j), \quad j = 1, 2, \dots, M. \quad (7.25)$$

Thus, instead of looking for the minimum Euclidean distance, one searches for the minimum Mahalanobis distance; the latter is a weighted form of the Euclidean distance, in order to account for the shape of the underlying Gaussian distributions [38]. For the two-class case, this rule corresponds to the dotted line of Fig. 7.5.

Remarks 7.2.

- In statistics, adopting the Gaussian assumption for the data distribution is sometimes called *linear discriminant analysis* (LDA) or *quadratic discriminant analysis* (QDA), depending on the

adopted assumptions with respect to the underlying covariance matrices, which will lead to linear or quadratic discriminant functions, respectively. In practice, the ML method is usually employed in order to obtain estimates of the unknown parameters, namely, the mean values and the covariance matrices. Recall from Example 3.7 of Chapter 3 that the ML estimate of the mean value of a Gaussian PDF, obtained via N observations, \mathbf{x}_n , $n = 1, 2, \dots, N$, is equal to

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

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

$$\hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T. \quad (7.26)$$

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

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

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

Example 7.2. Consider a two-class classification task in the two-dimensional space, with $P(\omega_1) = P(\omega_2) = 1/2$. Generate 100 points, 50 from each class. The data from each class, ω_i , $i = 1, 2$, stem from a corresponding Gaussian, $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, where

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

and (a)

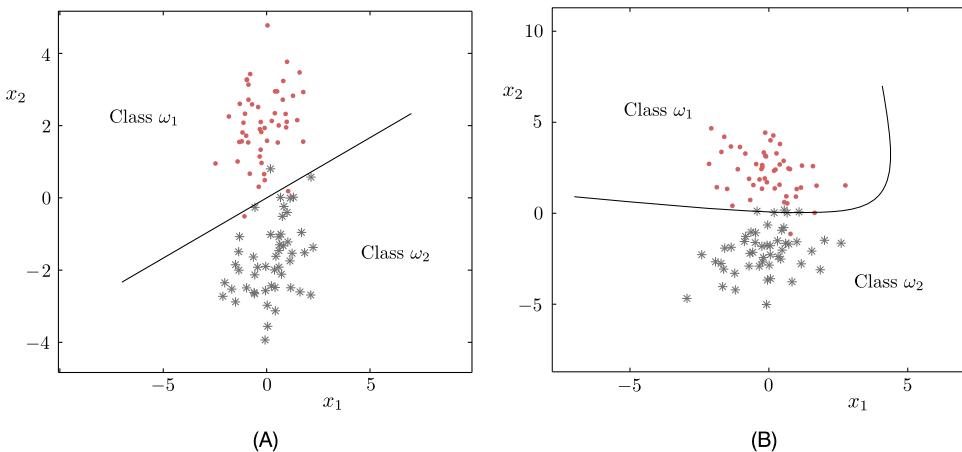
$$\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1.2 & 0.4 \\ 0.4 & 1.2 \end{bmatrix},$$

or (b)

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

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

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

**FIGURE 7.7**

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

respectively, sharing a common covariance matrix

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

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

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

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

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

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

Hence, the pattern is assigned to class ω_1 , because its distance from μ_1 is smaller compared to that from μ_2 . Verify that if the Euclidean distance were used instead, the pattern would be assigned to class ω_2 .

7.4 THE NAIVE BAYES CLASSIFIER

We have already seen that in the case the covariance matrix is to be estimated, the number of unknown parameters is of the order of $\mathcal{O}(l^2/2)$. For high-dimensional spaces, besides the fact that this estimation task is a formidable one, it also requires a large number of data points, in order to obtain statistically good estimates and avoid overfitting, as discussed in Chapter 3. In such cases, one has to be content with suboptimal solutions. Indeed, adopting an optimal method while using bad estimates of the involved parameters can lead to a bad overall performance.

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

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

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

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

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

7.5 THE NEAREST NEIGHBOR RULE

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

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

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

It turns out that this conceptually simple rule tends to the Bayesian classifier if (a) $N \rightarrow \infty$, (b) $k \rightarrow \infty$, and (c) $k/N \rightarrow 0$. In practical terms, these conditions mean that N and k must be large, yet k must be relatively small with respect to N . More specifically, it can be shown that the classification errors P_{NN} and $P_{k\text{NN}}$ satisfy, asymptotically, the following bounds [9]:

$$P_B \leq P_{\text{NN}} \leq 2P_B \quad (7.27)$$

for the $k = 1$ NN rule, and

$$P_B \leq P_{k\text{NN}} \leq P_B + \sqrt{\frac{2P_{\text{NN}}}{k}} \quad (7.28)$$

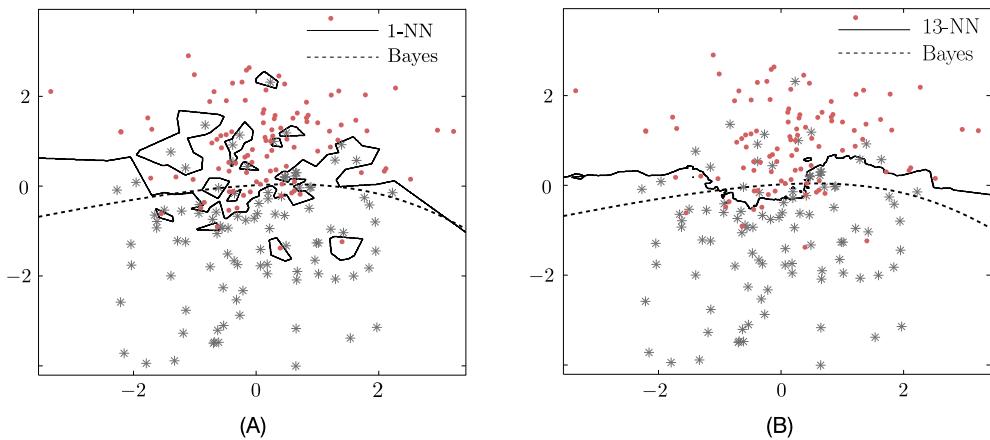
for the more general k -NN version; P_B is the error corresponding to the optimal Bayesian classifier. These two formulas are quite interesting. Take for example (7.27). It says that the simple NN rule will never give an error larger than twice the optimal one. If, for example, $P_B = 0.01$, then $P_{\text{NN}} \leq 0.02$. This is not bad for such a simple classifier. All this says is that if one has an easy task (as indicated by the very low value of P_B), the NN rule can also do a good job. This, of course, is not the case if the problem is not an easy one and larger error values are involved. The bound in (7.28) says that for large values of k (provided, of course, N is large enough), the performance of the k -NN tends to that of the optimal classifier. In practice, one has to make sure that k does not get values close to N , but remains a relatively small fraction of it.

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

A major drawback of the k -NN rule is that every time a new pattern is considered, its distance from all the training points has to be computed, then selecting the k closest to it points. To this end, various searching techniques have been suggested over the years. The interested reader may consult [38] for a related discussion.

Remarks 7.3.

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

**FIGURE 7.8**

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

set, represented by

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

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

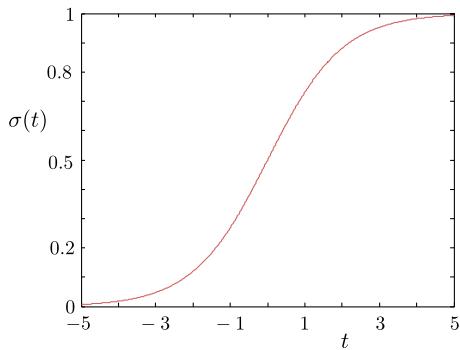
7.6 LOGISTIC REGRESSION

In Bayesian classification, the assignment of a pattern in a class is performed based on the posterior probabilities, $P(\omega_i|\mathbf{x})$. The posteriors are estimated via the respective conditional PDFs, which is not, in general, an easy task. The goal in this section is to model the posterior probabilities directly, via the *logistic regression* method. This name has been established in the statistics community, although the model refers to classification and not to regression. This is a typical example of the discriminative modeling approach, where the distribution of data is not taken into account.

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

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = \boldsymbol{\theta}^T \mathbf{x} : \quad \text{two-class logistic regression,}$$

(7.29)

**FIGURE 7.9**

The sigmoid link function.

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

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

and defining

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

it is readily seen that the model in (7.29) is equivalent to

$$P(\omega_1|\mathbf{x}) = \sigma(t), \quad (7.30)$$

$$\sigma(t) := \frac{1}{1 + \exp(-t)}, \quad (7.31)$$

and

$$P(\omega_2|\mathbf{x}) = 1 - P(\omega_1|\mathbf{x}) = \frac{\exp(-t)}{1 + \exp(-t)}. \quad (7.32)$$

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

Although it may sound a bit “mystical” as to how one thought of such a model, it suffices to look more carefully at (7.17)–(7.18) to demystify it. Let us assume that the data in a two-class task follow Gaussian distributions with $\Sigma_1 = \Sigma_2 \equiv \Sigma$. Under such assumptions, and taking into account the Bayes theorem, one can write

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = \ln \frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x}|\omega_2)P(\omega_2)} \quad (7.33)$$

$$= \ln p(\mathbf{x}|\omega_1) + \ln P(\omega_1) - (\ln p(\mathbf{x}|\omega_2) + \ln P(\omega_2)) \quad (7.34)$$

$$= g(\mathbf{x}). \quad (7.35)$$

Furthermore, we know that under the previous assumptions, $g(\mathbf{x})$ is given by Eqs. (7.20)–(7.22); hence, we can write

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \text{constants}, \quad (7.36)$$

where “constants” refers to all terms that do not depend on \mathbf{x} . In other words, when the distributions that describe the data are Gaussians with a common covariance matrix, then the log ratio of the posteriors is a *linear* function. Thus, in logistic regression, all we do is to go one step ahead and adopt such a linear model, irrespective of the underlying data distributions.

Moreover, even if the data are distributed according to Gaussians, it may still be preferable to adopt the logistic regression formulation instead of that in (7.36). In the latter formulation, the covariance matrix has to be estimated, amounting to $\mathcal{O}(l^2/2)$ parameters. The logistic regression formulation only involves $l+1$ parameters. That is, once we know about the linear dependence of the log ratio on \mathbf{x} , we can use this a priori information to simplify the model. Of course, assuming that the Gaussian assumption is valid, if one can obtain good estimates of the covariance matrix, employing this extra information can lead to more efficient estimates, in the sense of lower variance. The issue is treated in [12]. This is natural, because more information concerning the distribution of the data is exploited. In practice, it turns out that using the logistic regression is, in general, a safer bet compared to the linear discriminant analysis (LDA).

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

$$P(y_1, \dots, y_N; \boldsymbol{\theta}) = \prod_{n=1}^N \left(\sigma(\boldsymbol{\theta}^T \mathbf{x}_n) \right)^{y_n} \left(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_n) \right)^{1-y_n}. \quad (7.37)$$

Indeed, if \mathbf{x}_n originates from class ω_1 , then $y_n = 1$ and the corresponding probability is given by $\sigma(\boldsymbol{\theta}^T \mathbf{x}_n)$. On the other hand, if \mathbf{x}_n comes from ω_2 , then $y_n = 0$ and the respective probability is given by $1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_n)$. Assuming independence among the successive observations, the likelihood is given as the product of the respective probabilities.

Usually, we consider the negative log-likelihood given by

$$L(\boldsymbol{\theta}) = - \sum_{n=1}^N (y_n \ln s_n + (1 - y_n) \ln(1 - s_n)), \quad (7.38)$$

where

$$s_n := \sigma(\boldsymbol{\theta}^T \mathbf{x}_n). \quad (7.39)$$

The log-likelihood cost function in (7.38) is also known as the *cross-entropy* error. Minimization of $L(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is carried out iteratively by any iterative minimization scheme, such as the gradient descent or Newton’s method. Both schemes need the computation of the respective gradient, which in turn is based on the derivative of the sigmoid link function (Problem 7.6)

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t)). \quad (7.40)$$

The gradient is given by (Problem 7.7)

$$\begin{aligned}\nabla L(\boldsymbol{\theta}) &= \sum_{n=1}^N (s_n - y_n) \mathbf{x}_n \\ &= X^T (\mathbf{s} - \mathbf{y}),\end{aligned}\tag{7.41}$$

where

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

The Hessian matrix is given by (Problem 7.8)

$$\begin{aligned}\nabla^2 L(\boldsymbol{\theta}) &= \sum_{n=1}^N s_n(1-s_n) \mathbf{x}_n \mathbf{x}_n^T \\ &= X^T R X,\end{aligned}\tag{7.42}$$

where

$$R := \text{diag}\{s_1(1-s_1), \dots, s_N(1-s_N)\}.\tag{7.43}$$

Note that because $0 < s_n < 1$, by definition of the sigmoid link function, matrix R is positive definite (see Appendix A); hence, the Hessian matrix is also positive definite (Problem 7.9). This is a necessary and sufficient condition for convexity.² Thus, the negative log-likelihood function is convex, which guarantees the existence of a unique minimum (e.g., [1] and Chapter 8).

Two of the possible iterative minimization schemes to be used are

- Gradient descent (Section 5.2)

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i X^T (\mathbf{s}^{(i-1)} - \mathbf{y}).\tag{7.44}$$

- Newton's scheme (Section 6.7)

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu_i \left(X^T R^{(i-1)} X \right)^{-1} X^T (\mathbf{s}^{(i-1)} - \mathbf{y}) \\ &= \left(X^T R^{(i-1)} X \right)^{-1} X^T R^{(i-1)} \mathbf{z}^{(i-1)},\end{aligned}\tag{7.45}$$

where

$$\mathbf{z}^{(i-1)} := X \boldsymbol{\theta}^{(i-1)} - \left(R^{(i-1)} \right)^{-1} (\mathbf{s}^{(i-1)} - \mathbf{y}).\tag{7.46}$$

Eq. (7.45) is a weighted version of the LS solution (Chapters 3 and 6); however, the involved quantities are iteration dependent and the resulting scheme is known as *iterative reweighted least-squares* scheme (IRLS) [36].

² Convexity is discussed in more detail in Chapter 8.

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

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

$$P(\omega_m | \mathbf{x}) = \frac{\exp(\boldsymbol{\theta}_m^T \mathbf{x})}{\sum_{j=1}^M \exp(\boldsymbol{\theta}_j^T \mathbf{x})} : \text{ multiclass logistic regression.} \quad (7.47)$$

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

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

Let us define, for notational convenience,

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

and

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

The likelihood function is now written as

$$P(\mathbf{y}; \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = \prod_{n=1}^N \prod_{m=1}^M (\phi_{nm})^{y_{nm}}, \quad (7.48)$$

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

$$L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = - \sum_{n=1}^N \sum_{m=1}^M y_{nm} \ln \phi_{nm}, \quad (7.49)$$

which is the generalization of the cross-entropy cost function for the case of M classes. Minimization with respect to $\boldsymbol{\theta}_m$, $m = 1, \dots, M$, takes place iteratively. To this end, the following gradients are used (Problems 7.10–7.12):

$$\frac{\partial \phi_{nm}}{\partial t_j} = \phi_{nm} (\delta_{mj} - \phi_{nj}), \quad (7.50)$$

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

$$\nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = \sum_{n=1}^N (\phi_{nj} - y_{nj}) \mathbf{x}_n. \quad (7.51)$$

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

$$\nabla_{\boldsymbol{\theta}_k} \nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = \sum_{n=1}^N \phi_{nj} (\delta_{kj} - \phi_{nk}) \mathbf{x}_n \mathbf{x}_n^T. \quad (7.52)$$

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

Remarks 7.4.

- *Probit regression:* Instead of using the logistic sigmoid function in (7.30) (for the two-class case), other functions can also be adopted. A popular function in the statistical community is the *probit* function, which is defined as

$$\begin{aligned} \Phi(t) &:= \int_{-\infty}^t \mathcal{N}(z|0, 1) dz \\ &= \frac{1}{2} \left(1 + \frac{1}{\sqrt{2}} \text{erf}(t) \right), \end{aligned} \quad (7.53)$$

where erf is the *error* function defined as

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

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

7.7 FISHER'S LINEAR DISCRIMINANT

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

$$g(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 = 0. \quad (7.54)$$

We have dealt with the task of designing linear classifiers in the framework of the LS method in Chapter 3. In this section, the unknown parameter vector will be estimated via a path that exploits a number of important notions relevant to classification. The method is known as *Fisher's discriminant* and it can be dressed up with different interpretations.

PARAMETER LEARNING: A CONVEX ANALYTIC PATH

8

CONTENTS

8.1	Introduction	352
8.2	Convex Sets and Functions	352
8.2.1	Convex Sets	353
8.2.2	Convex Functions	354
8.3	Projections Onto Convex Sets	357
8.3.1	Properties of Projections	361
8.4	Fundamental Theorem of Projections Onto Convex Sets	365
8.5	A Parallel Version of POCS	369
8.6	From Convex Sets to Parameter Estimation and Machine Learning	369
8.6.1	Regression	369
8.6.2	Classification	373
8.7	Infinitely Many Closed Convex Sets: the Online Learning Case	374
8.7.1	Convergence of APSM	376
<i>Some Practical Hints</i>		378
8.8	Constrained Learning	380
8.9	The Distributed APSM	382
8.10	Optimizing Nonsmooth Convex Cost Functions	384
8.10.1	Subgradients and Subdifferentials	385
8.10.2	Minimizing Nonsmooth Continuous Convex Loss Functions: the Batch Learning Case	388
<i>The Subgradient Method</i>		388
<i>The Generic Projected Subgradient Scheme</i>		391
<i>The Projected Gradient Method (PGM)</i>		391
<i>Projected Subgradient Method</i>		392
8.10.3	Online Learning for Convex Optimization	393
<i>The PEGASOS Algorithm</i>		395
8.11	Regret Analysis	396
Regret Analysis of the Subgradient Algorithm		398
8.12	Online Learning and Big Data Applications: a Discussion	399
Approximation, Estimation, and Optimization Errors		400
Batch Versus Online Learning		402
8.13	Proximal Operators	405
8.13.1	Properties of the Proximal Operator	407
8.13.2	Proximal Minimization	409
<i>Resolvent of the Subdifferential Mapping</i>		411
8.14	Proximal Splitting Methods for Optimization	412
The Proximal Forward-Backward Splitting Operator		413
Alternating Direction Method of Multipliers (ADMM)		414

Mirror Descent Algorithms	415
8.15 Distributed Optimization: Some Highlights	417
Problems	417
MATLAB® Exercises	420
References	422

8.1 INTRODUCTION

The theory of convex sets and functions has a rich history and has been the focus of intense study for over a century in mathematics. In the terrain of applied sciences and engineering, the revival of interest in convex functions and optimization is traced back to the early 1980s. In addition to the increased processing power that became available via the use of computers, certain theoretical developments were catalytic in demonstrating the power of such techniques. The advent of the so-called interior point methods opened a new path in solving the classical linear programming task. Moreover, it was increasingly realized that, despite its advantages, the least-squares (LS) method also has a number of drawbacks, particularly in the presence of non-Gaussian noise and the presence of outliers. It has been demonstrated that the use of alternative cost functions, which may not even be differentiable, can alleviate a number of problems that are associated with the LS methods. Furthermore, the increased interest in robust learning methods brought into the scene the need for nontrivial constraints, which the optimized solution has to respect. In the machine learning community, the discovery of support vector machines, to be treated in Chapter 11, played an important role in popularizing convex optimization techniques.

The goal of this chapter is to present some basic notions and definitions related to convex analysis and optimization in the context of machine learning and signal processing. Convex optimization is a discipline in itself, and it cannot be summarized in one chapter. The emphasis here is on computationally light techniques with a focus on online versions, which are gaining in importance in the context of big data applications. A related discussion is also part of this chapter.

The material revolves around two families of algorithms. One goes back to the classical work of Von Neumann on projections on convex sets, which is reviewed together with its more recent online versions. The notions of projection and related properties are treated in some detail. The method of projections, in the context of constrained optimization, has been gaining in popularity recently.

The other family of algorithms that is considered builds around the notion of subgradient for optimizing nondifferentiable convex functions and generalizations of the gradient descent family, discussed in Chapter 5. Furthermore, a powerful tool for analyzing the performance of online algorithms for convex optimization, known as regret analysis, is discussed and a related case study is presented. Finally, some current trends in convex optimization, involving proximal and mirror descent methods are introduced.

8.2 CONVEX SETS AND FUNCTIONS

Although most of the algorithms we will discuss in this chapter refer to vector variables in Euclidean spaces, which is in line with what we have done so far in this book, the definitions and some of the

fundamental theorems will be stated in the context of the more general case of Hilbert spaces.¹ This is because the current chapter will also serve the needs of subsequent chapters, whose setting is that of infinite-dimensional Hilbert spaces. For those readers who are not familiar with such spaces, all they need to know is that a Hilbert space is a generalization of the Euclidean one, allowing for infinite dimensions. To serve the needs of these readers, the differences between Euclidean and the more general Hilbert spaces in the theorems will be carefully pointed out whenever this is required.

8.2.1 CONVEX SETS

Definition 8.1. A nonempty subset C of a Hilbert space \mathbb{H} , $C \subseteq \mathbb{H}$, is called *convex* if $\forall \mathbf{x}_1, \mathbf{x}_2 \in C$ and $\forall \lambda \in [0, 1]$ the following holds true²:

$$\mathbf{x} := \lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2 \in C. \quad (8.1)$$

Note that if $\lambda = 1$, $\mathbf{x} = \mathbf{x}_1$, and if $\lambda = 0$, $\mathbf{x} = \mathbf{x}_2$. For any other value of λ in $[0, 1]$, \mathbf{x} lies in the line segment joining \mathbf{x}_1 and \mathbf{x}_2 . Indeed, from (8.1) we can write

$$\mathbf{x} - \mathbf{x}_2 = \lambda(\mathbf{x}_1 - \mathbf{x}_2), \quad 0 \leq \lambda \leq 1.$$

Fig. 8.1 shows two examples of convex sets, in the two-dimensional Euclidean space, \mathbb{R}^2 . In Fig. 8.1A, the set comprises all points whose Euclidean (ℓ_2) norm is less than or equal to one,

$$C_2 = \left\{ \mathbf{x} : \sqrt{x_1^2 + x_2^2} \leq 1 \right\}.$$

Sometimes we refer to C_2 as the ℓ_2 -ball of radius equal to one. Note that the set includes all the points *on and inside the circle*. The set in Fig. 8.1B comprises all the points *on and inside the rhombus* defined by

$$C_1 = \left\{ \mathbf{x} : |x_1| + |x_2| \leq 1 \right\}.$$

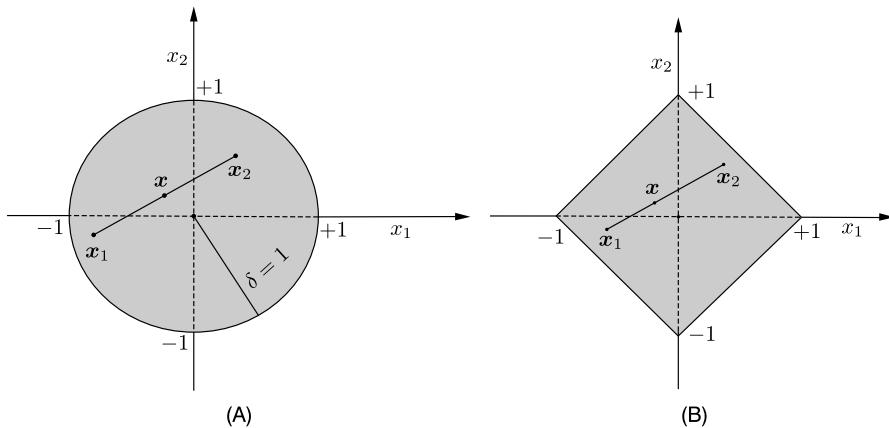
Because the sum of the absolute values of the components of a vector defines the ℓ_1 norm, that is, $\|\mathbf{x}\|_1 := |x_1| + |x_2|$, in analogy to C_2 we call the set C_1 as the ℓ_1 ball of radius equal to one. In contrast, the sets whose ℓ_2 and ℓ_1 norms are equal to one, or in other words,

$$\bar{C}_2 = \left\{ \mathbf{x} : x_1^2 + x_2^2 = 1 \right\}, \quad \bar{C}_1 = \left\{ \mathbf{x} : |x_1| + |x_2| = 1 \right\},$$

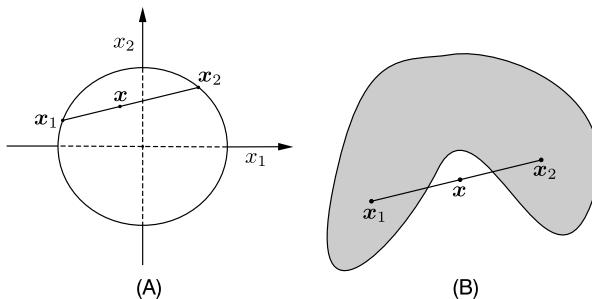
are not convex (Problem 8.2). Fig. 8.2 shows two examples of nonconvex sets.

¹ The mathematical definition of a Hilbert space is provided in the appendix associated with this chapter and which can be downloaded from the book's website.

² In conformity with Euclidean vector spaces and for the sake of notational simplicity, we will keep the same notation and denote the elements of a Hilbert space with lower case bold letters.

**FIGURE 8.1**

(A) The ℓ_2 ball of radius $\delta = 1$ comprises all points with Euclidean norm less than or equal to $\delta = 1$. (B) The ℓ_1 ball consists of all the points with ℓ_1 norm less than or equal to $\delta = 1$. Both are convex sets.

**FIGURE 8.2**

Examples of two nonconvex sets. In both cases, the point \mathbf{x} does not lie on the same set in which \mathbf{x}_1 and \mathbf{x}_2 belong. In (A) the set comprises all the points whose Euclidean norm is equal to one.

8.2.2 CONVEX FUNCTIONS

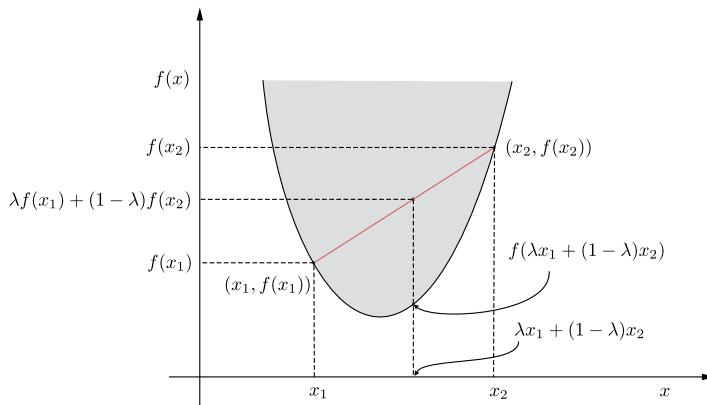
Definition 8.2. A function

$$f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$$

is called *convex* if \mathcal{X} is convex and if $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ the following holds true:

$$f(\lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda) f(\mathbf{x}_2), \quad \lambda \in [0, 1]. \quad (8.2)$$

The function is called *strictly convex* if (8.2) holds true with strict inequality when $\lambda \in (0, 1)$, $\mathbf{x}_1 \neq \mathbf{x}_2$. The geometric interpretation of (8.2) is that the line segment joining the points $(\mathbf{x}_1, f(\mathbf{x}_1))$ and

**FIGURE 8.3**

The line segment joining the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ lies above the graph of $f(x)$. The shaded region corresponds to the epigraph of the function.

$(x_2, f(x_2))$ lies above the graph of $f(x)$, as shown in Fig. 8.3. We say that a function is *concave* (*strictly concave*) if the negative, $-f$, is convex (*strictly convex*). Next, we state three important theorems.

Theorem 8.1 (First-order convexity condition). *Let $\mathcal{X} \subseteq \mathbb{R}^l$ be a convex set and let*

$$f : \mathcal{X} \mapsto \mathbb{R}$$

be a differentiable function. Then f is convex if and only if $\forall \mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$\boxed{f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla^T f(\mathbf{x})(\mathbf{y} - \mathbf{x}).} \quad (8.3)$$

The proof of the theorem is given in Problem 8.3. The theorem generalizes to nondifferentiable convex functions; it will be discussed in this context in Section 8.10.

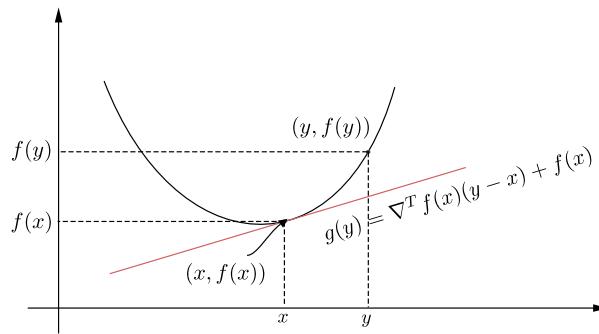
Fig. 8.4 illustrates the geometric interpretation of this theorem. It means that the graph of the convex function is located above the graph of the affine function

$$g : \mathbf{y} \mapsto \nabla^T f(\mathbf{x})(\mathbf{y} - \mathbf{x}) + f(\mathbf{x}),$$

which defines the tangent hyperplane of the graph at the point $(\mathbf{x}, f(\mathbf{x}))$.

Theorem 8.2 (Second-order convexity condition). *Let $\mathcal{X} \subseteq \mathbb{R}^l$ be a convex set. Then a twice differentiable function $f : \mathcal{X} \mapsto \mathbb{R}$ is convex (*strictly convex*) if and only if the Hessian matrix is positive semidefinite (positive definite).*

The proof of the theorem is given in Problem 8.5. Recall that in previous chapters, when we dealt with the squared error loss function, we commented that it is a convex one. Now we are ready to justify

**FIGURE 8.4**

The graph of a convex function is above the tangent plane at any point of the respective graph.

this argument. Consider the quadratic function,

$$f(\mathbf{x}) := \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + \mathbf{b}^T \mathbf{x} + c,$$

where Q is a positive definite matrix. Taking the gradient, we have

$$\nabla f(\mathbf{x}) = Q\mathbf{x} + \mathbf{b},$$

and the Hessian matrix is equal to Q , which by assumption is positive definite, hence f is a (strictly) convex function.

In the sequel, two very important notions in convex analysis and optimization are defined.

Definition 8.3. The *epigraph* of a function f is defined as the set of points

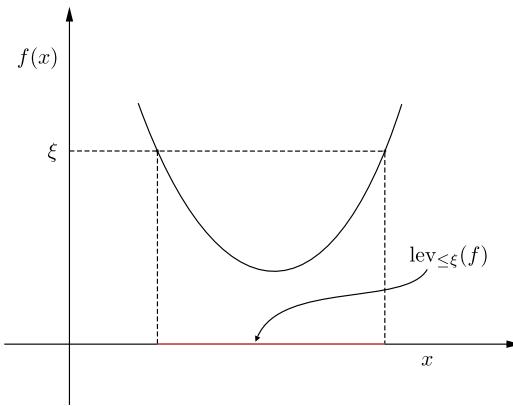
$$\boxed{\text{epi}(f) := \{(\mathbf{x}, r) \in \mathcal{X} \times \mathbb{R} : f(\mathbf{x}) \leq r\}} : \text{ epigraph.} \quad (8.4)$$

From a geometric point of view, the epigraph is the set of all points in $\mathbb{R}^l \times \mathbb{R}$ that lie on and above the graph of $f(\mathbf{x})$, as indicated by the gray shaded region in Fig. 8.3. It is important to note that a function is convex if and only if its epigraph is a convex set (Problem 8.6).

Definition 8.4. Given a real number ξ , the *lower level set* of function $f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$, at height ξ , is defined as

$$\boxed{\text{lev}_{\leq \xi}(f) := \{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) \leq \xi\}} : \text{ level set at } \xi. \quad (8.5)$$

In words, it is the set of all points at which the function takes a value less than or equal to ξ . The geometric interpretation of the level set is shown in Fig. 8.5. It can easily be shown (Problem 8.7) that if a function f is convex, then its lower level set is convex for any $\xi \in \mathbb{R}$. The converse is not true. We can easily check that the function $f(x) = -\exp(x)$ is not convex (as a matter of fact it is concave) and all its lower level sets are convex.

**FIGURE 8.5**

The level set at height ξ comprises all the points in the interval denoted as the “red” segment on the x -axis.

Theorem 8.3 (Local and global minimizers). *Let a convex function $f : \mathcal{X} \rightarrow \mathbb{R}$. Then if a point \mathbf{x}_* is a local minimizer, it is also a global one, and the set of all minimizers is convex. Further, if the function is strictly convex, the minimizer is unique.*

Proof. Inasmuch as the function is convex we know that, $\forall \mathbf{x} \in \mathcal{X}$,

$$f(\mathbf{x}) \geq f(\mathbf{x}_*) + \nabla^T f(\mathbf{x}_*)(\mathbf{x} - \mathbf{x}_*),$$

and because at the minimizer the gradient is zero, we get

$$f(\mathbf{x}) \geq f(\mathbf{x}_*), \quad (8.6)$$

which proves the claim. Let us now denote

$$f_* = \min_{\mathbf{x}} f(\mathbf{x}). \quad (8.7)$$

Note that the set of all minimizers coincides with the level set at height f_* . Then, because the function is convex, we know that the level set $\text{lev}_{f_*}(f)$ is convex, which verifies the convexity of the set of minimizers. Finally, for strictly convex functions, the inequality in (8.6) is a strict one, which proves the uniqueness of the (global) minimizer. The theorem is also true, even if the function is not differentiable (Problem 8.10). \square

8.3 PROJECTIONS ONTO CONVEX SETS

The projection onto a hyperplane in finite-dimensional Euclidean spaces was discussed and used in the context of the affine projection algorithm (APA) in Chapter 5. The notion of projection will now be gen-

25 dBs. The parameters for the algorithms were chosen for optimized performance (after experimentation) and for similar convergence rate. For the LMS, $\mu = 0.035$, and for the APSM, $\epsilon = \sqrt{2\sigma}$, $q = 20$, $\mu_k(n) = 0.2M_k(n)$. The combination weights were chosen according to the Metropolis rule and the data combination matrix was the identity one (no observations are exchanged). Fig. 8.27 shows the benefits of the data reuse offered by the APSM. The curves show the mean-square deviation (MSD) $= \frac{1}{K} \sum_{k=1}^K ||\theta_k(n) - \theta_o||^2$ as a function of the number of iterations.

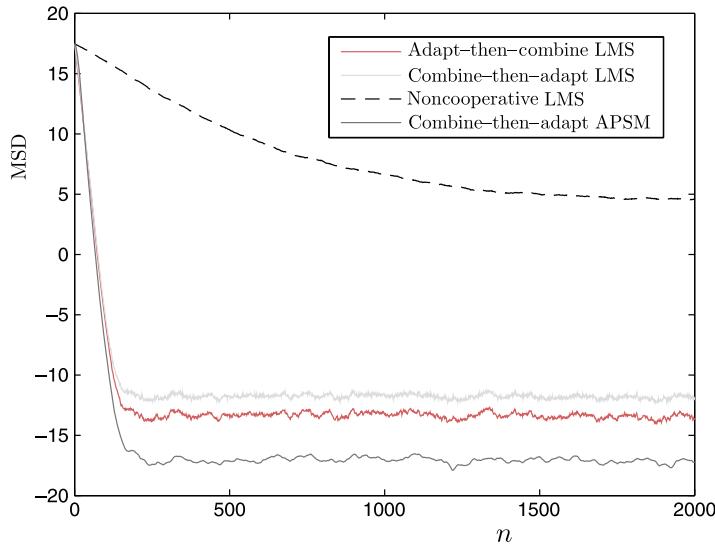


FIGURE 8.27

The MSD as a function of the number of iterations. The improved performance due to the data reuse offered by the diffusion APSM is readily observed. Moreover, observe the significant performance improvement offered by all cooperation schemes, compared to the noncooperative LMS; for the latter, only one node is used.

8.10 OPTIMIZING NONSMOOTH CONVEX COST FUNCTIONS

Estimating parameters via the use of convex loss functions in the presence of a set of constraints is an established and well-researched field in optimization, with numerous applications in a wide range of disciplines. The mainstream methods follow either the Lagrange multiplier philosophy [11,15] or the rationale behind the so-called *interior point* methods [15,90]. In this section, we will focus on an alternative path and consider iterative schemes, which can be considered as the generalization of the gradient descent method, discussed in Chapter 5. The reason is that such techniques give rise to variants that scale well with the dimensionality and have inspired a number of algorithms, which have been suggested for online learning within the machine learning and signal processing communities. Later on, we will move to more advanced techniques that build on the so-called *operator/mapping* and *fixed point* theoretic framework.

Although the stage of our discussion will be that of Euclidean spaces, \mathbb{R}^l , everything that will be said can be generalized to infinite-dimensional Hilbert spaces; we will consider such cases in Chapter 11.

8.10.1 SUBGRADIENTS AND SUBDIFFERENTIALS

We have already met the first-order convexity condition in (8.3) and it was shown that this is a sufficient and necessary condition for convexity, provided, of course, that the gradient exists. The condition basically states that the graph of the convex function lies above the hyperplanes, which are tangent at any point $(\mathbf{x}, f(\mathbf{x}))$ that lies on this graph.

Let us now move a step forward and assume a function

$$f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$$

to be convex and continuous, but *nonsmooth*. This means that there are points where the gradient is not defined. Our goal now becomes that of generalizing the notion of gradient, for the case of convex functions.

Definition 8.7. A vector $\mathbf{g} \in \mathbb{R}^l$ is said to be the *subgradient* of a convex function f at a point $\mathbf{x} \in \mathcal{X}$ if the following is true:

$$\boxed{f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}), \quad \forall \mathbf{y} \in \mathcal{X} : \quad \text{subgradient.}} \quad (8.46)$$

It turns out that this vector is *not* unique. All the subgradients of a (convex) function at a point comprise a set.

Definition 8.8. The *subdifferential* of a convex function f at $\mathbf{x} \in \mathcal{X}$, denoted as $\partial f(\mathbf{x})$, is defined as the *set*

$$\boxed{\partial f(\mathbf{x}) := \{\mathbf{g} \in \mathbb{R}^l : f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}), \forall \mathbf{y} \in \mathcal{X}\} : \quad \text{subdifferential.}} \quad (8.47)$$

If f is differentiable at a point \mathbf{x} , then $\partial f(\mathbf{x})$ becomes a singleton, that is,

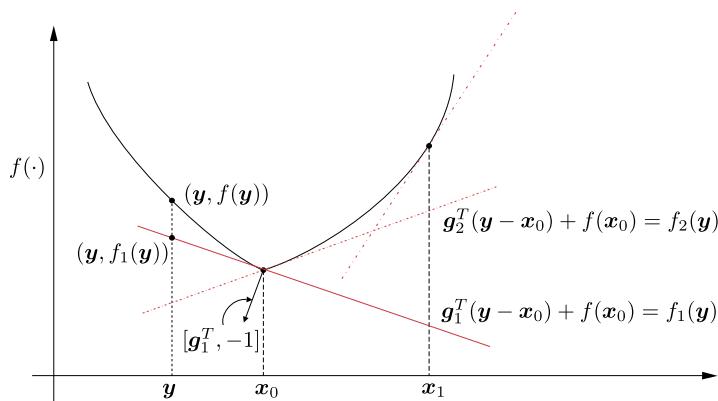
$$\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}.$$

Note that if $f(\mathbf{x})$ is convex, then the set $\partial f(\mathbf{x})$ is *nonempty and convex*. Moreover, $f(\mathbf{x})$ is differentiable at a point \mathbf{x} if and only if it has a unique subgradient [11]. From now on, we will denote a subgradient of f at the point \mathbf{x} as $f'(\mathbf{x})$.

Fig. 8.28 gives a geometric interpretation of the notion of the subgradient. Each one of the subgradients at the point \mathbf{x}_0 defines a hyperplane that *supports* the graph of f . At \mathbf{x}_0 , there is an infinity of subgradients, which comprise the subdifferential (set) at \mathbf{x}_0 . At \mathbf{x}_1 , the function is differentiable and there is a *unique* subgradient that coincides with the gradient at \mathbf{x}_1 .

Example 8.4. Let $x \in \mathbb{R}$ and

$$f(x) = |x|.$$

**FIGURE 8.28**

At x_0 , there is an infinity of subgradients, each one defining a hyperplane in the extended $(x, f(x))$ space. All these hyperplanes pass through the point $(x_0, f(x_0))$ and support the graph of $f(\cdot)$. At the point x_1 , there is a unique subgradient that coincides with the gradient and defines the unique tangent hyperplane at the respective point of the graph.

Show that

$$\partial f(x) = \begin{cases} \text{sgn}(x), & \text{if } x \neq 0, \\ g \in [-1, 1], & \text{if } x = 0, \end{cases}$$

where $\text{sgn}(\cdot)$ is the sign function, being equal to 1 if its argument is positive and -1 if the argument is negative.

Indeed, if $x > 0$, then

$$g = \frac{dx}{dx} = 1,$$

and similarly $g = -1$, if $x < 0$. For $x = 0$, any $g \in [-1, 1]$ satisfies

$$g(y - 0) + 0 = gy \leq |y|,$$

and it is a subgradient. This is illustrated in Fig. 8.29.

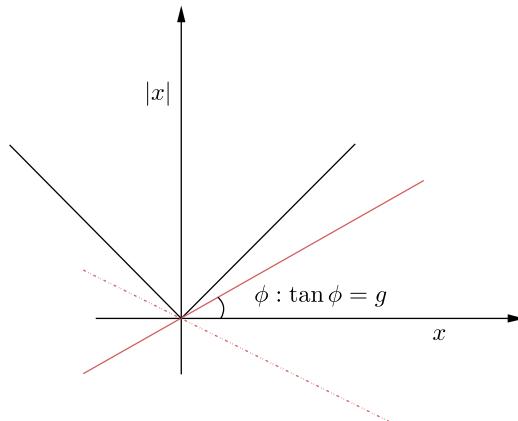
Lemma 8.2. *Given a convex function $f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$, a point $\mathbf{x}_* \in \mathcal{X}$ is a minimizer of f if and only if the zero vector belongs to its subdifferential set, that is,*

$\mathbf{0} \in \partial f(\mathbf{x}_*)$

(8.48)

Proof. The proof is straightforward from the definition of a subgradient. Indeed, assume that $\mathbf{0} \in \partial f(\mathbf{x}_*)$. Then the following is valid:

$$f(\mathbf{y}) \geq f(\mathbf{x}_*) + \mathbf{0}^T(\mathbf{y} - \mathbf{x}_*), \quad \forall \mathbf{y} \in \mathcal{X},$$

**FIGURE 8.29**

All lines with slope in $[-1, 1]$ comprise the subdifferential at $x = 0$.

and \mathbf{x}_* is a minimizer. If now \mathbf{x}_* is a minimizer, then we have

$$f(\mathbf{y}) \geq f(\mathbf{x}_*) = f(\mathbf{x}_*) + \mathbf{0}^T(\mathbf{y} - \mathbf{x}_*),$$

and hence $\mathbf{0} \in \partial f(\mathbf{x}_*)$. □

Example 8.5. Let the metric *distance* function

$$d_C(\mathbf{x}) := \min_{\mathbf{y} \in C} \|\mathbf{x} - \mathbf{y}\|.$$

This is the Euclidean distance of a point from its projection on a closed convex set C , as defined in Section 8.3. Then show (Problem 8.24) that the subdifferential is given by

$$\partial d_C(\mathbf{x}) = \begin{cases} \frac{\mathbf{x} - P_C(\mathbf{x})}{\|\mathbf{x} - P_C(\mathbf{x})\|}, & \mathbf{x} \notin C, \\ N_C(\mathbf{x}) \cap B[\mathbf{0}, 1], & \mathbf{x} \in C, \end{cases} \quad (8.49)$$

where

$$N_C(\mathbf{x}) := \{\mathbf{g} \in \mathbb{R}^l : \mathbf{g}^T(\mathbf{y} - \mathbf{x}) \leq 0, \forall \mathbf{y} \in C\},$$

and

$$B[\mathbf{0}, 1] := \{\mathbf{x} \in \mathbb{R}^l : \|\mathbf{x}\| \leq 1\}.$$

Moreover, if \mathbf{x} is an *interior* point of C , then

$$\partial d_C(\mathbf{x}) = \{\mathbf{0}\}.$$

Observe that for all points $\mathbf{x} \notin C$ as well as for all interior points of C , the subgradient is a singleton, which means that $d_C(\mathbf{x})$ is differentiable. Recall that the function $d_C(\cdot)$ is nonnegative, convex, and continuous [44]. Note that (8.49) is also generalized to infinite-dimensional Hilbert spaces.

8.10.2 MINIMIZING NONSMOOTH CONTINUOUS CONVEX LOSS FUNCTIONS: THE BATCH LEARNING CASE

Let J be a cost function⁷

$$J : \mathbb{R}^l \mapsto [0, +\infty),$$

and let C be a closed convex set, $C \subseteq \mathbb{R}^l$. Our task is to compute a minimizer with respect to an unknown parameter vector, that is,

$$\begin{aligned} \boldsymbol{\theta}_* &= \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \\ \text{s.t. } \boldsymbol{\theta} &\in C, \end{aligned} \tag{8.50}$$

and we will assume that the set of solutions is *nonempty*; J is assumed to be convex, continuous, but not necessarily differentiable at all points. We have already seen examples of such loss function, such as the ϵ -insensitive linear function in (8.33) and the hinge one (8.37). The ℓ_1 norm function is another example, and it will be treated in Chapters 9 and 10.

The Subgradient Method

Our starting point is the simplest of the cases, where $C = \mathbb{R}^l$; that is, the minimizing task is unconstrained. The first thought that comes into mind is to consider the generalization of the gradient descent method, which was introduced in Chapter 5, and replace the gradient by the subgradient operation. The resulting scheme is known as the *subgradient algorithm* [74,75].

Starting from an arbitrary estimate, $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^l$, the update recursions become

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i J'(\boldsymbol{\theta}^{(i-1)}) : \quad \text{subgradient algorithm,}$$

(8.51)

where J' denotes *any* subgradient of the cost function, and μ_i is a step size sequence judiciously chosen so that convergence is guaranteed. In spite of the similarity in the appearance with our familiar gradient descent scheme, there are some major differences. The reader may have noticed that the new algorithm was not called subgradient “descent.” This is because the update in (8.51) is not necessarily performed in the descent direction. Thus, during the operation of the algorithm, the value of the cost function may increase. Recall that in the gradient descent methods, the value of the cost function is guaranteed to decrease with each iteration step, which also led to a linear convergence rate, as we have pointed out in Chapter 5.

In contrast here, concerning the subgradient method, such comments cannot be stated. To establish convergence, a different route has to be adopted. To this end, let us define

$$J_*^{(i)} := \min \{J(\boldsymbol{\theta}^{(i)}), J(\boldsymbol{\theta}^{(i-1)}), \dots, J(\boldsymbol{\theta}^{(0)})\}, \tag{8.52}$$

⁷ Recall that all the methods to be reported can be extended to general Hilbert spaces, \mathbb{H} .

which can also be recursively obtained by

$$J_*^{(i)} = \min \{ J_*^{(i-1)}, J(\boldsymbol{\theta}^{(i)}) \}.$$

Then the following holds true.

Proposition 8.3. *Let J be a convex cost function. Assume that the subgradients at all points are bounded, that is,*

$$\|J'(\mathbf{x})\| \leq G, \quad \forall \mathbf{x} \in \mathbb{R}^l.$$

Let us also assume that the step size sequence be a diminishing one, such as

$$\sum_{i=1}^{\infty} \mu_i = \infty, \quad \sum_{i=1}^{\infty} \mu_i^2 < \infty.$$

Then

$$\lim_{i \rightarrow \infty} J_*^{(i)} = J(\boldsymbol{\theta}_*),$$

where $\boldsymbol{\theta}_$ is a minimizer; assuming that the set of minimizers is not empty.*

Proof. We have

$$\begin{aligned} \|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*\|^2 &= \|\boldsymbol{\theta}^{(i-1)} - \mu_i J'(\boldsymbol{\theta}^{(i-1)}) - \boldsymbol{\theta}_*\|^2 \\ &= \|\boldsymbol{\theta}^{(i-1)} - \boldsymbol{\theta}_*\|^2 - 2\mu_i J'^T(\boldsymbol{\theta}^{(i-1)})(\boldsymbol{\theta}^{(i-1)} - \boldsymbol{\theta}_*) \\ &\quad + \mu_i^2 \|J'(\boldsymbol{\theta}^{(i-1)})\|^2. \end{aligned} \tag{8.53}$$

By the definition of the subgradient, we have

$$J(\boldsymbol{\theta}_*) - J(\boldsymbol{\theta}^{(i-1)}) \geq J'^T(\boldsymbol{\theta}^{(i-1)})(\boldsymbol{\theta}_* - \boldsymbol{\theta}^{(i-1)}). \tag{8.54}$$

Plugging (8.54) in (8.53) and after some algebraic manipulations, by applying the resulting inequality recursively (Problem 8.25), we finally obtain

$$J_*^{(i)} - J(\boldsymbol{\theta}_*) \leq \frac{\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}_*\|^2}{2 \sum_{k=1}^i \mu_k} + \frac{\sum_{k=1}^i \mu_k^2}{2 \sum_{k=1}^i \mu_k} G^2. \tag{8.55}$$

Leaving i to grow to infinity and taking into account the assumptions, the claim is proved. \square

There are a number of variants of this proof. Other choices for the diminishing sequence can also guarantee convergence, such as $\mu_i = 1/\sqrt{i}$. Moreover, in certain cases, some of the assumptions may be relaxed. Note that the assumption of the subgradient being bounded is guaranteed if J is γ -Lipschitz continuous (Problem 8.26), that is, there is $\gamma > 0$ such that

$$|J(\mathbf{y}) - J(\mathbf{x})| \leq \gamma \|\mathbf{y} - \mathbf{x}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^l.$$

SPARSITY-AWARE LEARNING: CONCEPTS AND THEORETICAL FOUNDATIONS

9

CONTENTS

9.1	Introduction	427
9.2	Searching for a Norm	428
9.3	The Least Absolute Shrinkage and Selection Operator (LASSO)	431
9.4	Sparse Signal Representation	436
9.5	In Search of the Sparsest Solution	440
	The ℓ_2 Norm Minimizer	441
	The ℓ_0 Norm Minimizer	442
	The ℓ_1 Norm Minimizer	442
	Characterization of the ℓ_1 Norm Minimizer	443
	Geometric Interpretation	444
9.6	Uniqueness of the ℓ_0 Minimizer	447
9.6.1	Mutual Coherence	449
9.7	Equivalence of ℓ_0 and ℓ_1 Minimizers: Sufficiency Conditions	451
9.7.1	Condition Implied by the Mutual Coherence Number	451
9.7.2	The Restricted Isometry Property (RIP)	452
	Constructing Matrices That Obey the RIP of Order k	453
9.8	Robust Sparse Signal Recovery From Noisy Measurements	455
9.9	Compressed Sensing: the Glory of Randomness	456
	Compressed Sensing	456
9.9.1	Dimensionality Reduction and Stable Embeddings	458
9.9.2	Sub-Nyquist Sampling: Analog-to-Information Conversion	460
9.10	A Case Study: Image Denoising	463
Problems	465	
	MATLAB® Exercises	468
References	469	

9.1 INTRODUCTION

In Chapter 3, the notion of regularization was introduced as a tool to address a number of problems that are usually encountered in machine learning. Improving the performance of an estimator by shrinking the norm of the minimum variance unbiased (MVU) estimator, guarding against overfitting, coping with ill-conditioning, and providing a solution to an underdetermined set of equations are some notable examples where regularization has provided successful answers. Some of the advantages were demon-

strated via the ridge regression concept, where the sum of squared errors cost function was combined, in a tradeoff rationale, with the squared Euclidean norm of the desired solution.

In this and the next chapter, our interest will be on alternatives in the Euclidean norm, and in particular the focus will revolve around the ℓ_1 norm; this is the sum of the absolute values of the components comprising a vector. Although seeking a solution to a problem via the ℓ_1 norm regularization of a cost function has been known and used since the 1970s, it is only recently that it has become the focus of attention of a massive volume of research in the context of compressed sensing. At the heart of this problem lies an underdetermined set of linear equations, which, in general, accepts an infinite number of solutions. However, in a number of cases, an extra piece of information is available: the true model, whose estimate we want to obtain, is sparse; that is, only a few of its coordinates are nonzero. It turns out that a large number of commonly used applications can be cast under such a scenario and can benefit by sparse modeling.

Besides its practical significance, sparsity-aware learning has offered to the scientific community novel theoretical tools and solutions to problems that only a few years ago seemed intractable. This is also a reason that this is an interdisciplinary field of research encompassing scientists from, for example, mathematics, statistics, machine learning, and signal processing. Moreover, it has already been applied in many areas, ranging from biomedicine to communications and astronomy. In this and the following chapters, I made an effort to present in a unifying way the basic notions and ideas that run across this field. The goal is to provide the reader with an overview of the major contributions that have taken place in the theoretical and algorithmic fronts and have been consolidated as a distinct scientific area.

In the current chapter, the focus is on presenting the main concepts and theoretical foundations related to sparsity-aware learning techniques. We start by reviewing various norms. Then we move on to establish conditions on the recovery of sparse vectors, or vectors that are sparse in a transform domain, using less observations than the dimension of the corresponding space. Geometry plays an important part in our approach. Finally, some theoretical advances that tie sparsity and sampling theory are presented. At the end of the chapter, a case study concerning image denoising is discussed.

9.2 SEARCHING FOR A NORM

Mathematicians have been very imaginative in proposing various norms in order to equip linear spaces. Among the most popular norms used in functional analysis are the ℓ_p norms. To tailor things to our needs, given a vector $\boldsymbol{\theta} \in \mathbb{R}^l$, its ℓ_p norm is defined as

$$\|\boldsymbol{\theta}\|_p := \left(\sum_{i=1}^l |\theta_i|^p \right)^{1/p}. \quad (9.1)$$

For $p = 2$, the Euclidean or ℓ_2 norm is obtained, and for $p = 1$, (9.1) results in the ℓ_1 norm, that is,

$$\|\boldsymbol{\theta}\|_1 = \sum_{i=1}^l |\theta_i|. \quad (9.2)$$

If we let $p \rightarrow \infty$, then we get the ℓ_∞ norm; let $|\theta_{i_{\max}}| := \max \{|\theta_1|, |\theta_2|, \dots, |\theta_l|\}$, and note that

$$\|\boldsymbol{\theta}\|_\infty := \lim_{p \rightarrow \infty} \left(|\theta_{i_{\max}}|^p \sum_{i=1}^l \left(\frac{|\theta_i|}{|\theta_{i_{\max}}|} \right)^p \right)^{1/p} = |\theta_{i_{\max}}|, \quad (9.3)$$

that is, $\|\boldsymbol{\theta}\|_\infty$ is equal to the maximum of the absolute values of the coordinates of $\boldsymbol{\theta}$. One can show that all the ℓ_p norms are true norms for $p \geq 1$; that is, satisfy all four requirements that a function $\mathbb{R}^l \mapsto [0, \infty)$ must respect in order to be called a norm, that is,

1. $\|\boldsymbol{\theta}\|_p \geq 0$,
2. $\|\boldsymbol{\theta}\|_p = 0 \Leftrightarrow \boldsymbol{\theta} = \mathbf{0}$,
3. $\|\alpha\boldsymbol{\theta}\|_p = |\alpha| \|\boldsymbol{\theta}\|_p$, $\forall \alpha \in \mathbb{R}$,
4. $\|\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2\|_p \leq \|\boldsymbol{\theta}_1\|_p + \|\boldsymbol{\theta}_2\|_p$.

The third condition enforces the norm function to be (*positively*) *homogeneous* and the fourth one is the *triangle inequality*. These properties also guarantee that any function that is a norm is also a convex one (Problem 9.3). Though strictly speaking, if we allow $p > 0$ to take values less than one in (9.1), the resulting function is not a true norm (Problem 9.8), we may still call them norms, although we know that this is an abuse of the definition of a norm. An interesting case, which will be used extensively in this chapter, is the ℓ_0 norm, which can be obtained as the limit, for $p \rightarrow 0$, of

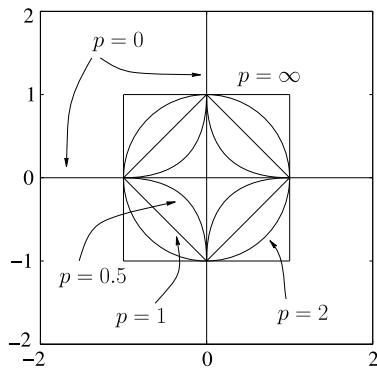
$$\|\boldsymbol{\theta}\|_0 := \lim_{p \rightarrow 0} \|\boldsymbol{\theta}\|_p^p = \lim_{p \rightarrow 0} \sum_{i=1}^l |\theta_i|^p = \sum_{i=1}^l \chi_{(0, \infty)}(|\theta_i|), \quad (9.4)$$

where $\chi_{\mathcal{A}}(\cdot)$ is the characteristic function with respect to a set \mathcal{A} , defined as

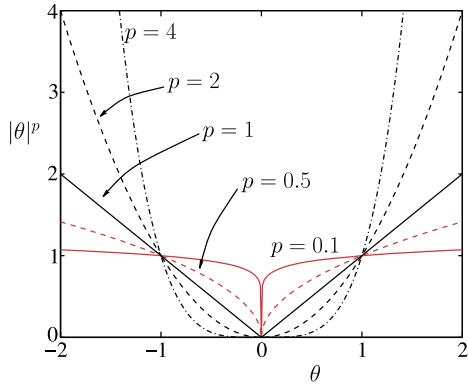
$$\chi_{\mathcal{A}}(\tau) := \begin{cases} 1, & \text{if } \tau \in \mathcal{A}, \\ 0, & \text{if } \tau \notin \mathcal{A}. \end{cases}$$

That is, the ℓ_0 norm is equal to the number of nonzero components of the respective vector. It is very easy to check that this function is not a true norm. Indeed, this is not homogeneous, that is, $\|\alpha\boldsymbol{\theta}\|_0 \neq |\alpha| \|\boldsymbol{\theta}\|_0$, $\forall \alpha \neq 1$. Fig. 9.1 shows the isovalue curves, in the two-dimensional space, that correspond to $\|\boldsymbol{\theta}\|_p = 1$, for $p = 0, 0.5, 1, 2$, and ∞ . Observe that for the Euclidean norm the isovalue curve has the shape of a circle and for the ℓ_1 norm the shape of a rhombus. We refer to them as the ℓ_2 and the ℓ_1 balls, respectively, by slightly “abusing” the meaning of a ball.¹ Observe that in the case of the ℓ_0 norm, the isovalue curve comprises both the horizontal and the vertical axes, excluding the $(0, 0)$ element. If we restrict the size of the ℓ_0 norm to be less than one, then the corresponding set of points becomes a singleton, that is, $(0, 0)$. Also, the set of all the two-dimensional points that have ℓ_0 norm less than or equal to two is the \mathbb{R}^2 space. This slightly “strange” behavior is a consequence of the discrete nature of this “norm.”

¹ Strictly speaking, a ball must also contain all the points in the interior, that is, all concentric spheres of smaller radius (Chapter 8).

**FIGURE 9.1**

The isovalue curves for $\|\theta\|_p = 1$ and for various values of p , in the two-dimensional space. Observe that for the ℓ_0 norm, the respective values cover the two axes with the exception of the point $(0, 0)$. For the ℓ_1 norm, the isovalue curve is a rhombus, and for the ℓ_2 (Euclidean) norm, it is a circle.

**FIGURE 9.2**

Observe that the epigraph, that is, the region above the graph, is nonconvex for values $p < 1$, indicating the non-convexity of the respective $|\theta|^p$ function. The value $p = 1$ is the smallest one for which convexity is retained. Also note that, for large values of $p > 1$, the contribution of small values of $|\theta| < 1$ to the respective norm becomes insignificant.

Fig. 9.2 shows the graph of $|\theta|^p$, which is the individual contribution of each component of a vector to the ℓ_p norm, for different values of p . Observe that (a) for $p < 1$, the region that is formed above the graph (epigraph, see Chapter 8) is not a convex one, which verifies what we have already said, that is, the respective function is not a true norm; and (b) for values of the argument $|\theta| > 1$, the larger the value of $p \geq 1$ and the larger the value of $|\theta|$, the higher the contribution of the respective component to the norm. Hence, if ℓ_p norms, $p \geq 1$, are used in the regularization method, components with large values

become the dominant ones and the optimization algorithm will concentrate on these by penalizing them to get smaller so that the overall cost can be reduced. The opposite is true for values $|\theta| < 1$; ℓ_p , $p > 1$, norms tend to push the contribution of such components to zero. The ℓ_1 norm is the only one (among $p \geq 1$) that retains relatively large values even for small values of $|\theta| < 1$ and, hence, components with small values can still have a say in the optimization process and can be penalized by being pushed to smaller values. Hence, if the ℓ_1 norm is used to replace the ℓ_2 one in Eq. (3.41), *only those components of the vector that are really significant in reducing the model misfit measuring term in the regularized cost function will be kept, and the rest will be forced to zero.* The same tendency, yet more aggressive, is true for $0 \leq p < 1$. The extreme case is when one considers the ℓ_0 norm. Even a small increase of a component from zero makes its contribution to the norm large, so the optimizing algorithm has to be very “cautious” in making an element nonzero.

In a nutshell, from all the true norms ($p \geq 1$), the ℓ_1 is the only one that shows respect to small values. The rest of the ℓ_p norms, $p > 1$, just squeeze them to make their values even smaller, and care mainly for the large values. We will return to this point very soon.

9.3 THE LEAST ABSOLUTE SHRINKAGE AND SELECTION OPERATOR (LASSO)

In Chapter 3, we discussed some of the benefits in adopting the regularization method for enhancing the performance of an estimator. In this chapter, we will see and study more reasons that justify the use of regularization. The first one refers to what is known as the *interpretation* power of an estimator. For example, in the regression task, we want to select those components θ_i of $\boldsymbol{\theta}$ that have the most important say in the formation of the output variable. This is very important if the number of parameters, l , is large and we want to concentrate on the most important of them. In a classification task, not all features are informative, hence one would like to keep the most informative of them and make the less informative ones equal to zero. Another related problem refers to those cases where we know, a priori, that a number of the components of a parameter vector are zero, but we do not know which ones. Now, the discussion at the end of the previous section becomes more meaningful. Can we use, while regularizing, an appropriate norm that can assist the optimization process (a) in unveiling such zeros or (b) to put more emphasis on the most significant of its components, those that play a decisive role in reducing the misfit measuring term in the regularized cost function, and set the rest of them equal to zero? Although the ℓ_p norms, with $p < 1$, seem to be the natural choice for such a regularization, the fact that they are not convex makes the optimization process hard. The ℓ_1 norm is the one that is “closest” to them, yet it retains the computationally attractive property of convexity.

The ℓ_1 norm has been used for such problems for a long time. In the 1970s, it was used in seismology [27,86], where the reflected signal that indicates changes in the various earth substrates is a sparse one, that is, very few values are relatively large and the rest are small and insignificant. Since then, it has been used to tackle similar problems in different applications (e.g., [40,80]). However, one can trace two papers that were catalytic in providing the spark for the current strong interest around the ℓ_1 norm. One came from statistics [89], which addressed the LASSO task (first formulated, to our knowledge, in [80]), to be discussed next, and the other came from the signal analysis community [26], which formulated the *basis pursuit*, to be discussed in a later section.

We first address our familiar regression task

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}, \quad \mathbf{y}, \boldsymbol{\eta} \in \mathbb{R}^N, \quad \boldsymbol{\theta} \in \mathbb{R}^l, \quad N \geq l,$$

and obtain the estimate of the unknown parameter $\boldsymbol{\theta}$ via the sum of squared error cost, regularized by the ℓ_1 norm, that is, for $\lambda \geq 0$,

$$\hat{\boldsymbol{\theta}} := \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} L(\boldsymbol{\theta}, \lambda) \quad (9.5)$$

$$\begin{aligned} &:= \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left(\sum_{n=1}^N (y_n - \mathbf{x}_n^T \boldsymbol{\theta})^2 + \lambda \|\boldsymbol{\theta}\|_1 \right) \\ &= \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left((\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1 \right). \end{aligned} \quad (9.6)$$

Following the discussion with respect to the bias term given in Section 3.8 and in order to simplify the analysis, we will assume hereafter, without harming generality, that the data are of zero mean values. If this is not the case, the data can be centered by subtracting their respective sample means.

It turns out that the task in (9.6) can be equivalently written in the following two formulations:

$$\begin{aligned} \hat{\boldsymbol{\theta}} : \quad &\min_{\boldsymbol{\theta} \in \mathbb{R}^l} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}), \\ \text{s.t.} \quad &\|\boldsymbol{\theta}\|_1 \leq \rho, \end{aligned} \quad (9.7)$$

or

$$\begin{aligned} \hat{\boldsymbol{\theta}} : \quad &\min_{\boldsymbol{\theta} \in \mathbb{R}^l} \|\boldsymbol{\theta}\|_1, \\ \text{s.t.} \quad &(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \leq \epsilon, \end{aligned} \quad (9.8)$$

given the user-defined parameters $\rho, \epsilon \geq 0$. The formulation in (9.7) is known as the LASSO and the one in (9.8) as the *basis pursuit denoising* (BPDN) (e.g., [15]). All three formulations are equivalent for specific choices of λ, ϵ , and ρ (see, e.g., [14]). Observe that the minimized cost function in (9.6) corresponds to the Lagrangian of the formulation in (9.7). However, this functional dependence among λ, ϵ , and ρ is hard to compute, unless the columns of \mathbf{X} are mutually orthogonal. Moreover, this equivalence does not necessarily imply that all three formulations are equally easy or difficult to solve. As we will see later in this chapter, algorithms have been developed along each one of the previous formulations. From now on, we will refer to all three formulations as the LASSO task, in a slight abuse of the standard terminology, and the specific formulation will be apparent from the context, if not stated explicitly.

We know that ridge regression admits a closed form solution, that is,

$$\hat{\boldsymbol{\theta}}_R = \left(\mathbf{X}^T \mathbf{X} + \lambda I \right)^{-1} \mathbf{X}^T \mathbf{y}.$$

In contrast, this is not the case for LASSO, and its solution requires iterative techniques. It is straightforward to see that LASSO can be formulated as a standard convex quadratic problem with linear

inequalities. Indeed, we can rewrite (9.6) as

$$\begin{aligned} \min_{\{\theta_i, u_i\}_{i=1}^l} & (y - X\theta)^T(y - X\theta) + \lambda \sum_{i=1}^l u_i, \\ \text{s.t. } & \begin{cases} -u_i \leq \theta_i \leq u_i, & i = 1, 2, \dots, l, \\ u_i \geq 0, \end{cases} \end{aligned}$$

which can be solved by any standard convex optimization method (e.g., [14, 101]). The reason that developing algorithms for the LASSO task has been at the center of an intense research activity is due to the emphasis on obtaining *efficient* algorithms by exploiting the specific nature of this task, especially for cases where l is very large, as is often the case in practice.

In order to get better insight into the nature of the solution that is obtained by LASSO, let us assume that the regressors are mutually orthogonal and of unit norm, hence $X^T X = I$. Orthogonality of the input matrix helps to decouple the coordinates and results to l one-dimensional problems that can be solved analytically. For this case, the LS estimate becomes

$$\hat{\theta}_{LS} = (X^T X)^{-1} X^T y = X^T y,$$

and the ridge regression gives

$$\hat{\theta}_R = \frac{1}{1+\lambda} \hat{\theta}_{LS}, \quad (9.9)$$

that is, every component of the LS estimate is simply shrunk by the *same* factor, $\frac{1}{1+\lambda}$; see, also, Section 6.5.

In the case of ℓ_1 regularization, the minimized Lagrangian function is no more differentiable, due to the presence of the absolute values in the ℓ_1 norm. So, in this case, we have to consider the notion of the subdifferential. It is known (Chapter 8) that if the zero vector belongs to the subdifferential set of a convex function at a point, this means that this point corresponds to a minimum of the function. Taking the subdifferential of the Lagrangian defined in (9.6) and recalling that the subdifferential set of a differentiable function includes as its *single* element the respective gradient, the estimate $\hat{\theta}_1$, resulting from the ℓ_1 regularized task, must satisfy

$$\mathbf{0} \in -2X^T y + 2X^T X\theta + \lambda \partial \|\theta\|_1,$$

where ∂ stands for the subdifferential set (Chapter 8). If X has orthonormal columns, the previous equation can be written component-wise as follows:

$$0 \in -\hat{\theta}_{LS,i} + \hat{\theta}_{1,i} + \frac{\lambda}{2} \partial |\hat{\theta}_{1,i}|, \quad \forall i, \quad (9.10)$$

where the subdifferential of the function $|\cdot|$, derived in Example 8.4 (Chapter 8), is given as

$$\partial |\theta| = \begin{cases} \{1\}, & \text{if } \theta > 0, \\ \{-1\}, & \text{if } \theta < 0, \\ [-1, 1], & \text{if } \theta = 0. \end{cases}$$

Thus, we can now write for each component of the LASSO optimal estimate

$$\hat{\theta}_{1,i} = \begin{cases} \hat{\theta}_{\text{LS},i} - \frac{\lambda}{2}, & \text{if } \hat{\theta}_{1,i} > 0, \\ \hat{\theta}_{\text{LS},i} + \frac{\lambda}{2}, & \text{if } \hat{\theta}_{1,i} < 0. \end{cases} \quad (9.11)$$

Note that (9.11) can only be true if $\hat{\theta}_{\text{LS},i} > \frac{\lambda}{2}$, and (9.12) only if $\hat{\theta}_{\text{LS},i} < -\frac{\lambda}{2}$. Moreover, in the case where $\hat{\theta}_{1,i} = 0$, (9.10) and the subdifferential of $|\cdot|$ suggest that necessarily $|\hat{\theta}_{\text{LS},i}| \leq \frac{\lambda}{2}$. Concluding, we can write in a more compact way that

$$\hat{\theta}_{1,i} = \text{sgn}(\hat{\theta}_{\text{LS},i}) \left(\left| \hat{\theta}_{\text{LS},i} \right| - \frac{\lambda}{2} \right)_+ : \quad \text{soft thresholding operation,} \quad (9.13)$$

where $(\cdot)_+$ denotes the “positive part” of the respective argument; it is equal to the argument if this is nonnegative, and zero otherwise. This is very interesting indeed. In contrast to the ridge regression which shrinks all coordinates of the unregularized LS solution by the same factor, LASSO forces all coordinates, whose absolute value is less than or equal to $\lambda/2$, to zero, and the rest of the coordinates are reduced, in absolute value, by the same amount $\lambda/2$. This is known as *soft thresholding*, to distinguish it from the *hard thresholding* operation; the latter is defined as $\theta \cdot \chi_{(0,\infty)}(|\theta| - \frac{\lambda}{2})$, $\theta \in \mathbb{R}$, where $\chi_{(0,\infty)}(\cdot)$ stands for the characteristic function with respect to the set $(0, \infty)$. Fig. 9.3 shows the graphs illustrating the effect that the ridge regression, LASSO, and hard thresholding have on the unregularized LS solution, as a function of its value (horizontal axis). Note that our discussion here, simplified via the orthonormal input matrix case, has quantified what we said before about the tendency of the ℓ_1 norm to push small values to become *exactly zero*. This will be further strengthened, via a more rigorous mathematical formulation, in Section 9.5.

Example 9.1. Assume that the unregularized LS solution, for a given regression task, $y = X\theta + \eta$, is given by

$$\hat{\theta}_{\text{LS}} = [0.2, -0.7, 0.8, -0.1, 1.0]^T.$$

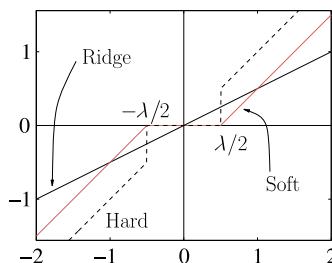


FIGURE 9.3

Curves relating the output (vertical) to the input (horizontal) for the hard thresholding and the soft thresholding operators shown together with the linear operator that is associated with the ridge regression, for the same value of $\lambda = 1$.

Derive the solutions for the corresponding ridge regression and ℓ_1 norm regularization tasks. Assume that the input matrix X has orthonormal columns and that the regularization parameter is $\lambda = 1$. Also, what is the result of hard thresholding the vector $\hat{\theta}_{\text{LS}}$ with threshold equal to 0.5?

We know that the corresponding solution for the ridge regression is

$$\hat{\theta}_R = \frac{1}{1 + \lambda} \hat{\theta}_{\text{LS}} = [0.1, -0.35, 0.4, -0.05, 0.5]^T.$$

The solution for the ℓ_1 norm regularization is given by soft thresholding, with threshold equal to $\lambda/2 = 0.5$, and hence the corresponding vector is

$$\hat{\theta}_1 = [0, -0.2, 0.3, 0, 0.5]^T.$$

The result of the hard thresholding operation is the vector $[0, -0.7, 0.8, 0, 1.0]^T$.

Remarks 9.1.

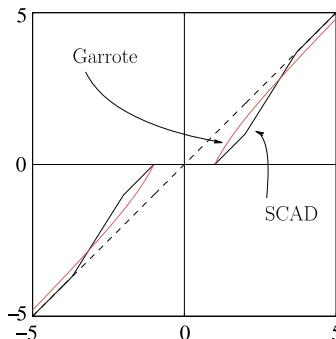
- The hard and soft thresholding rules are only two possibilities out of a larger number of alternatives. Note that the hard thresholding operation is defined via a discontinuous function, and this makes this rule unstable in the sense of being very sensitive to small changes of the input. Moreover, this shrinking rule tends to exhibit large variance in the resulting estimates. The soft thresholding rule is a continuous function, but, as readily seen from the graph in Fig. 9.3, it introduces bias even for the large values of the input argument. In order to ameliorate such shortcomings, a number of alternative thresholding operators have been introduced and studied both theoretically and experimentally. Although these are not within the mainstream of our interest, we provide two popular examples for the sake of completeness—the *smoothly clipped absolute deviation* (SCAD) thresholding rule,

$$\hat{\theta}_{\text{SCAD}} = \begin{cases} \text{sgn}(\theta) (|\theta| - \lambda_{\text{SCAD}})_+, & |\theta| \leq 2\lambda_{\text{SCAD}}, \\ \frac{(\alpha - 1)\theta - \alpha\lambda_{\text{SCAD}} \text{sgn}(\theta)}{\alpha - 2}, & 2\lambda_{\text{SCAD}} < |\theta| \leq \alpha\lambda_{\text{SCAD}}, \\ \theta, & |\theta| > \alpha\lambda_{\text{SCAD}}, \end{cases}$$

and the *nonnegative garrote* thresholding rule,

$$\hat{\theta}_{\text{garr}} = \begin{cases} 0, & |\theta| \leq \lambda_{\text{garr}}, \\ \theta - \frac{\lambda_{\text{garr}}^2}{\theta}, & |\theta| > \lambda_{\text{garr}}. \end{cases}$$

Fig. 9.4 shows the respective graphs. Observe that in both cases, an effort has been made to remove the discontinuity (associated with the hard thresholding) and to remove/reduce the bias for large values of the input argument. The parameter $\alpha > 2$ is a user-defined one. For a more detailed related discussion, the interested reader can refer, for example, to [2]. In [83], a generalized thresholding rule is suggested that encompasses all previously mentioned ones as special cases. Moreover, the proposed framework is general enough to provide means for designing novel thresholding rules and/or incorporating a priori information associated with the sparsity level, i.e., the number of nonzero components, of the sparse vector to be recovered.

**FIGURE 9.4**

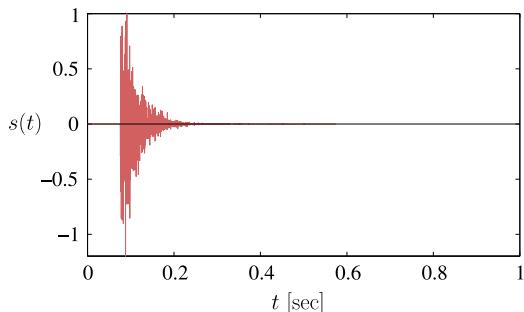
Output (vertical)–input (horizontal) graph for the SCAD and nonnegative garrote rules with parameters $\alpha = 3.7$ and $\lambda_{\text{SCAD}} = \lambda_{\text{garr}} = 1$. Observe that both rules smoothen out the discontinuity associated with the hard thresholding rule. Note, also, that the SCAD rule removes the bias associated with the soft thresholding rule for large values of the input variable. On the contrary, the garrote thresholding rule allows some bias for large input values, which diminishes as λ_{garr} gets smaller and smaller.

9.4 SPARSE SIGNAL REPRESENTATION

In the previous section, we brought into our discussion the need to take special care for zeros. Sparsity is an attribute that is met in a plethora of natural signals, because nature tends to be parsimonious. The notion of and need for parsimonious models was also discussed in Chapter 3, in the context of inverse problems in machine learning tasks. In this section, we will briefly present a number of application cases where the existence of zeros in a mathematical expansion is of paramount importance; hence, it justifies our search for and development of related analysis tools.

In Chapter 4, we discussed the task of echo cancellation. In a number of cases, the echo path, represented by a vector comprising the values of the impulse response samples, is a sparse one. This is the case, for example, in internet telephony and in acoustic and network environments (e.g., [3,10,73]). Fig. 9.5 shows the impulse response of such an echo path. The impulse response of the echo path is of short duration; however, the delay with which it appears is not known. So, in order to model it, one has to use a long impulse response, yet only a relatively small number of the coefficients will be significant and the rest will be close to zero. Of course, one could ask, why not use an LMS or an RLS, and eventually the significant coefficients will be identified? The answer is that this turns out not to be the most efficient way to tackle such problems, because the convergence of the algorithm can be very slow. In contrast, if one embeds, somehow, into the problem the a priori information concerning the existence of (almost) zero coefficients, then the convergence speed can be significantly increased and also better error floors can be attained.

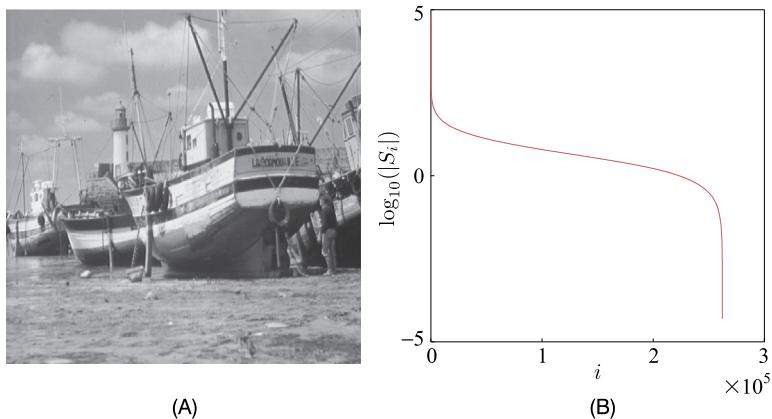
A similar situation occurs in wireless communication systems, which involve multipath channels. A typical application is in high-definition television (HDTV) systems, where the involved communication channels consist of a few nonnegligible coefficients, some of which may have quite large time delays with respect to the main signal (see, e.g., [4,32,52,77]). If the information signal is transmitted at high symbol rates through such a dispersive channel, then the introduced intersymbol interference

**FIGURE 9.5**

The impulse response function of an echo path in a telephone network. Observe that although it is of relatively short duration, it is not a priori known where exactly in time it will occur.

(ISI) has a span of several tens up to hundreds of symbol intervals. This in turn implies that quite long channel estimators are required at the receiver's end in order to reduce effectively the ISI component of the received signal, although only a small part of it has values substantially different from zero. The situation is even more demanding when the channel-frequency response exhibits deep nulls. More recently, sparsity has been exploited in channel estimation for multicarrier systems, both for single antenna as well as for multiple-input-multiple-output (MIMO) systems [46,47]. A thorough, in-depth treatment related to sparsity in multipath communication systems is provided in [5].

Another example, which might be more widely known, is that of signal compression. It turns out that if the signal modalities with which we communicate (e.g., speech) and also sense the world (e.g.,

**FIGURE 9.6**

(A) A 512×512 pixel image and (B) the magnitude of its DCT components in descending order and logarithmic scale. Note that more than 95% of the total energy is contributed by only 5% of the largest components.

images, audio) are transformed into a suitably chosen domain, then they are sparsely represented; only a relatively small number of the signal components in this domain are large, and the rest are close to zero. As an example, Fig. 9.6A shows an image and Fig. 9.6B shows the plot of the magnitude of the obtained discrete cosine transform (DCT) components, which are computed by writing the corresponding image array as a vector in lexicographic order. Note that more than 95% of the total energy is contributed by only 5% of the largest components. This is at the heart of any compression technique. Only the large coefficients are chosen to be coded and the rest are considered to be zero. Hence, significant gains are obtained in memory/bandwidth requirements while storing/transmitting such signals, without much perceptual loss. Depending on the modality, different transforms are used. For example, in JPEG-2000, an image array, represented in terms of a vector that contains the intensity of the gray levels of the image pixels, is transformed via the discrete wavelet transform (DWT), resulting in a transformed vector that comprises only a few large components.

Let

$$\tilde{s} = \Phi^H s, \quad s, \tilde{s} \in \mathbb{C}^l, \quad (9.14)$$

where s is the vector of the “raw” signal samples, \tilde{s} is the (complex-valued) vector of the transformed ones, and Φ is the $l \times l$ transformation matrix. Often, this is an orthonormal/unitary matrix, $\Phi^H \Phi = I$. Basically, a transform is nothing more than a projection of a vector on a new set of coordinate axes, which comprise the columns of the transformation matrix Φ . Celebrated examples of such transforms are the wavelet, the discrete Fourier (DFT), and the discrete cosine (DCT) transforms (e.g., [87]). In such cases, where the transformation matrix is orthonormal, one can write

$$s = \Psi \tilde{s}, \quad (9.15)$$

where $\Psi = \Phi$. Eq. (9.14) is known as the *analysis* and (9.15) as the *synthesis* equation.

Compression via such transforms exploits the fact that many signals in nature, which are rich in context, can be *compactly* represented in an appropriately chosen basis, depending on the modality of the signal. Very often, the construction of such bases tries to “imitate” the sensory systems that the human brain has developed in order to sense these signals; and we know that nature (in contrast to modern humans) does not like to waste resources. A standard compression task comprises the following stages: (a) obtain the l components of \tilde{s} via the analysis step (9.14); (b) keep the k most significant of them; (c) code these values, as well as their respective locations in the transformed vector \tilde{s} ; and (d) obtain the (approximate) original signal s when needed (after storage or transmission), via the synthesis equation (Eq. (9.15)), where in place of \tilde{s} only its k most significant components are used, which are the ones that were coded, while the rest are set equal to zero. However, there is something unorthodox in this process of compression as it has been practiced until very recently. One processes (transforms) large signal vectors of l coordinates, where l in practice can be quite large, and then uses only a small percentage of the transformed coefficients, while the rest are simply ignored. Moreover, one has to store/transmit the location of the respective large coefficients that are finally coded.

A natural question that is raised is the following: Because \tilde{s} in the synthesis equation is (approximately) sparse, can one compute it via an alternative path to the analysis equation in (9.14)? The issue here is to investigate whether one could use a more informative way of sampling the available raw data so that fewer than l samples/observations are sufficient to recover all the necessary information. The ideal case would be to recover it via a set of k such samples, because this is the number of the signif-

icant free parameters. On the other hand, if this sounds a bit extreme, can one obtain N ($k < N \ll l$) such signal-related measurements, from which s can eventually be retrieved? It turns out that such an approach is possible and it leads to the solution of an *underdetermined* system of linear equations, under the constraint that the unknown target vector is a sparse one.

The importance of such techniques becomes even more apparent when, instead of an orthonormal basis, as discussed before, a more general type of expansion is adopted, in terms of what is known as *overcomplete dictionaries*. A dictionary [65] is a collection of parameterized waveforms, which are discrete-time signal samples, represented as vectors $\psi_i \in \mathbb{C}^l$, $i \in \mathcal{I}$, where \mathcal{I} is an integer index set. For example, the columns of a DFT or a discrete wavelet (DWT) matrix comprise a dictionary. These are two examples of what are known as *complete* dictionaries, which consist of l (orthonormal) vectors, that is, a number equal to the length of the signal vector. However, in many cases in practice, using such dictionaries is very restrictive. Let us take, for example, a segment of audio signal, from a news media or a video, that needs to be processed. This consists, in general, of different types of signals, namely, speech, music, and environmental sounds. For each type of these signals, different signal vectors may be more appropriate in the expansion for the analysis. For example, music signals are characterized by a strong harmonic content and the use of sinusoids seems to be best for compression, while for speech signals a Gabor-type signal expansion (sinusoids of various frequencies weighted by sufficiently narrow pulses at different locations in time [31,87]) may be a better choice. The same applies when one deals with an image. Different parts of an image, such as parts that are smooth or contain sharp edges, may demand a different expansion vector set for obtaining the best overall performance. The more recent tendency, in order to satisfy such needs, is to use *overcomplete* dictionaries. Such dictionaries can be obtained, for example, by concatenating different dictionaries together, for example, a DFT and a DWT matrix to result in a combined $l \times 2l$ transformation matrix. Alternatively, a dictionary can be “trained” in order to effectively represent a set of available signal exemplars, a task that is often referred to as dictionary learning [75,78,90,100]. While using such overcomplete dictionaries, the synthesis equation takes the form

$$s = \sum_{i \in \mathcal{I}} \theta_i \psi_i. \quad (9.16)$$

Note that, now, the analysis is an ill-posed problem, because the elements $\{\psi_i\}_{i \in \mathcal{I}}$ (usually called *atoms*) of the dictionary are not linearly independent, and there is not a unique set of parameters $\{\theta_i\}_{i \in \mathcal{I}}$ that generates s . Moreover, we expect most of these parameters to be (nearly) zero. Note that in such cases, the cardinality of \mathcal{I} is larger than l . This necessarily leads to underdetermined systems of equations with infinitely many solutions. The question that is now raised is whether we can exploit the fact that most of these parameters are known to be zero, in order to come up with a unique solution. If yes, under which conditions is such a solution possible? We will return to the task of learning dictionaries in Chapter 19.

Besides the previous examples, there are a number of cases where an underdetermined system of equations is the result of our inability to obtain a sufficiently large number of measurements, due to physical and technical constraints. This is the case in MRI imaging, which will be presented in more detail in Section 10.3.

9.5 IN SEARCH OF THE SPARSEST SOLUTION

Inspired by the discussion in the previous section, we now turn our attention to the task of solving underdetermined systems of equations by imposing the sparsity constraint on the solution. We will develop the theoretical setup in the context of regression and we will adhere to the notation that has been adopted for this task. Moreover, we will focus on the real-valued data case in order to simplify the presentation. The theory can be readily extended to the more general complex-valued data case (see, e.g., [64, 99]). We assume that we are given a set of observations/measurements, $\mathbf{y} := [y_1, y_2, \dots, y_N]^T \in \mathbb{R}^N$, according to the linear model

$$\mathbf{y} = X\boldsymbol{\theta}, \quad \mathbf{y} \in \mathbb{R}^N, \quad \boldsymbol{\theta} \in \mathbb{R}^l, \quad l > N, \quad (9.17)$$

where X is the $N \times l$ input matrix, which is assumed to be of full row rank, that is, $\text{rank}(X) = N$. Our starting point is the noiseless case. The linear system of equations in (9.17) is an underdetermined one and accepts an infinite number of solutions. The set of possible solutions lies in the intersection of the N hyperplanes² in the l -dimensional space,

$$\left\{ \boldsymbol{\theta} \in \mathbb{R}^l : y_n = \mathbf{x}_n^T \boldsymbol{\theta} \right\}, \quad n = 1, 2, \dots, N.$$

We know from geometry that the intersection of N nonparallel hyperplanes (which in our case is guaranteed by the fact that X has been assumed to be full row rank, hence \mathbf{x}_n , $n = 1, 2, \dots, N$, are linearly independent) is a plane of dimensionality $l - N$ (e.g., the intersection of two [nonparallel] [hyper]planes in the three-dimensional space is a straight line, that is, a plane of dimensionality equal to one). In a more formal way, the set of all possible solutions, to be denoted as Θ , is an *affine* set. An affine set is the translation of a linear subspace by a constant vector. Let us pursue this a bit further, because we will need it later on.

Let the null space of X be denoted as $\text{null}(X)$ (sometimes denoted as $\mathcal{N}(X)$), and it is defined as the linear subspace

$$\text{null}(X) = \left\{ \mathbf{z} \in \mathbb{R}^l : X\mathbf{z} = \mathbf{0} \right\}.$$

Obviously, if $\boldsymbol{\theta}_0$ is a solution to (9.17), that is, $\boldsymbol{\theta}_0 \in \Theta$, then it is easy to verify that $\forall \boldsymbol{\theta} \in \Theta, X(\boldsymbol{\theta} - \boldsymbol{\theta}_0) = \mathbf{0}$, or $\boldsymbol{\theta} - \boldsymbol{\theta}_0 \in \text{null}(X)$. As a result,

$$\Theta = \boldsymbol{\theta}_0 + \text{null}(X),$$

and Θ is an affine set. We also know from linear algebra basics (and it is easy to show it; see Problem 9.9) that the null space of a full row rank matrix, $N \times l$, $l > N$, is a subspace of dimensionality $l - N$. Fig. 9.7 illustrates the case for one measurement sample in the two-dimensional space, $l = 2$ and $N = 1$. The set of solutions Θ is a straight line, which is the translation of the linear subspace crossing the origin (the $\text{null}(X)$). Therefore, if one wants to select a *single* point among all the points that lie in the affine set of solutions, Θ , then an extra constraint/a priori knowledge has to be imposed.

In the sequel, three such possibilities are examined.

² In \mathbb{R}^l , a hyperplane is of dimension $l - 1$. A plane has dimension lower than $l - 1$.

THE ℓ_2 NORM MINIMIZER

Our goal now becomes to pick a point in (the affine set) Θ that corresponds to the minimum ℓ_2 norm. This is equivalent to solving the following constrained task:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^l} \quad & \|\theta\|_2^2, \\ \text{s.t.} \quad & \mathbf{x}_n^T \theta = y_n, \quad n = 1, 2, \dots, N. \end{aligned} \quad (9.18)$$

We already know from Section 6.4 (and one can rederive it by employing Lagrange multipliers; see Problem 9.10) that the previous optimization task accepts a *unique* solution given in closed form as

$$\hat{\theta} = X^T (X X^T)^{-1} y. \quad (9.19)$$

The geometric interpretation of this solution is provided in Fig. 9.7A, for the case of $l = 2$ and $N = 1$. The radius of the Euclidean norm ball keeps increasing, until it touches the plane that contains the solutions. This point is the one with the minimum ℓ_2 norm or, equivalently, the point that lies closest to the origin. Equivalently, the point $\hat{\theta}$ can be seen as the (metric) projection of $\mathbf{0}$ onto Θ .

Minimizing the ℓ_2 norm in order to solve a linear set of underdetermined equations has been used in various applications. The closest to us is in the context of determining the unknown parameters in an expansion using an overcomplete dictionary of functions (vectors) [35]. A main drawback of this method is that it is not sparsity preserving. There is no guarantee that the solution in (9.19) will give zeros even if the true model vector θ has zeros. Moreover, the method is *resolution-limited* [26]. This means that even if there may be a sharp contribution of specific atoms in the dictionary, this is not portrayed in the obtained solution. This is a consequence of the fact that the information provided by $X X^T$

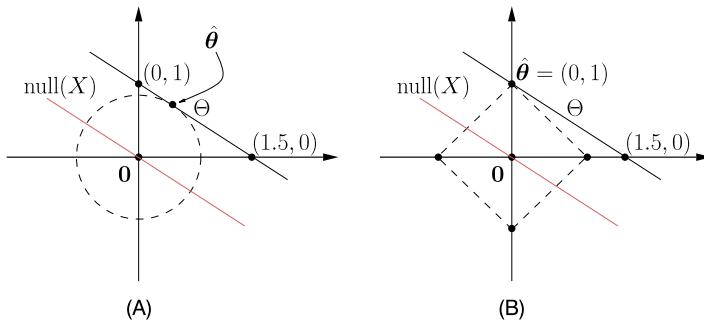


FIGURE 9.7

The set of solutions Θ is an affine set (gray line), which is a translation of the $\text{null}(X)$ subspace (red line). (A) The ℓ_2 norm minimizer: The dotted circle corresponds to the smallest ℓ_2 ball that intersects the set Θ . As such, the intersection point, $\hat{\theta}$, is the ℓ_2 norm minimizer of the task in (9.18). Note that the vector $\hat{\theta}$ contains no zero component. (B) The ℓ_1 norm minimizer: The dotted rhombus corresponds to the smallest ℓ_1 ball that intersects Θ . Hence, the intersection point, $\hat{\theta}$, is the solution of the constrained ℓ_1 minimization task of (9.21). Note that the obtained estimate $\hat{\theta} = (0, 1)$ contains a zero.

is a global one, containing all atoms of the dictionary in an “averaging” fashion, and the final result tends to smoothen out the individual contributions, especially when the dictionary is overcomplete.

THE ℓ_0 NORM MINIMIZER

Now we turn our attention to the ℓ_0 norm (once more, it is pointed out that this is an abuse of the definition of the norm), and we make sparsity our new flag under which a solution will be obtained. The task now becomes

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \quad & \|\boldsymbol{\theta}\|_0, \\ \text{s.t.} \quad & \mathbf{x}_n^T \boldsymbol{\theta} = y_n, \quad n = 1, 2, \dots, N, \end{aligned} \quad (9.20)$$

that is, from all the points that lie on the plane of all possible solutions, find the *sparsest* one, that is, the one with the lowest number of nonzero elements. As a matter of fact, such an approach is within the spirit of *Occam’s razor* rule—it corresponds to the smallest number of parameters that can explain the obtained observations. The points that are now raised are:

- Is a solution to this problem unique, and under which conditions?
- Can a solution be obtained with low enough complexity in realistic time?

We postpone the answer to the first question until later. As for the second one, the news is not good. Minimizing the ℓ_0 norm under a set of linear constraints is a task of combinatorial nature, and as a matter of fact, the problem is, in general, NP-hard [72]. The way to approach the problem is to consider all possible combinations of zeros in $\boldsymbol{\theta}$, removing the respective columns of X in (9.17), and check whether the system of equations is satisfied; keep as solutions the ones with the smallest number of nonzero elements. Such a searching technique exhibits complexity of an exponential dependence on l . Fig. 9.7A illustrates the two points ((1.5, 0) and (0, 1)) that comprise the solution set of minimizing the ℓ_0 norm for the single measurement (constraint) case.

THE ℓ_1 NORM MINIMIZER

The current task is now given by

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \quad & \|\boldsymbol{\theta}\|_1, \\ \text{s.t.} \quad & \mathbf{x}_n^T \boldsymbol{\theta} = y_n, \quad n = 1, 2, \dots, N. \end{aligned} \quad (9.21)$$

Fig. 9.7B illustrates the geometry. The ℓ_1 ball is increased until it touches the affine set of the possible solutions. For this specific geometry, the solution is the point (0, 1), which is a sparse solution. In our discussion in Section 9.2, we saw that the ℓ_1 norm is the one, out of all ℓ_p , $p \geq 1$ norms, that bears some similarity with the sparsity promoting (nonconvex) ℓ_p , $p < 1$ “norms.” Also, we have commented that the ℓ_1 norm encourages zeros when the respective values are small. In the sequel, we will state one lemma that establishes this zero favoring property in a more formal way. The ℓ_1 norm minimization task is also known as *basis pursuit* and it was suggested for decomposing a vector signal in terms of the atoms of an overcomplete dictionary [26].

The ℓ_1 minimizer can be brought into the standard linear programming (LP) form and then can be solved by recalling any related method; the simplex method and the more recent interior point method are two possibilities (see, e.g., [14,33]). Indeed, consider the LP task

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T \mathbf{x}, \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

To verify that our ℓ_1 minimizer can be cast in the previous form, note first that any l -dimensional vector $\boldsymbol{\theta}$ can be decomposed as

$$\boldsymbol{\theta} = \mathbf{u} - \mathbf{v}, \quad \mathbf{u} \geq \mathbf{0}, \mathbf{v} \geq \mathbf{0}.$$

Indeed, this holds true if, for example,

$$\mathbf{u} := \boldsymbol{\theta}_+, \quad \mathbf{v} := (-\boldsymbol{\theta})_+,$$

where \mathbf{x}_+ stands for the vector obtained after keeping the positive components of \mathbf{x} and setting the rest equal to zero. Moreover, note that

$$\|\boldsymbol{\theta}\|_1 = [1, 1, \dots, 1] \begin{bmatrix} \boldsymbol{\theta}_+ \\ (-\boldsymbol{\theta})_+ \end{bmatrix} = [1, 1, \dots, 1] \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}.$$

Hence, our ℓ_1 minimization task can be recast in the LP form if

$$\begin{aligned} \mathbf{c} &:= [1, 1, \dots, 1]^T, \quad \mathbf{x} := [\mathbf{u}^T, \mathbf{v}^T]^T, \\ \mathbf{A} &:= [\mathbf{X}, -\mathbf{X}], \quad \mathbf{b} := \mathbf{y}. \end{aligned}$$

CHARACTERIZATION OF THE ℓ_1 NORM MINIMIZER

Lemma 9.1. *An element $\boldsymbol{\theta}$ in the affine set Θ of the solutions of the underdetermined linear system (9.17) has minimal ℓ_1 norm if and only if the following condition is satisfied:*

$$\left| \sum_{i: \theta_i \neq 0} \operatorname{sgn}(\theta_i) z_i \right| \leq \sum_{i: \theta_i = 0} |z_i|, \quad \forall \mathbf{z} \in \operatorname{null}(X). \quad (9.22)$$

Moreover, the ℓ_1 minimizer is unique if and only if the inequality in (9.22) is a strict one for all $\mathbf{z} \neq \mathbf{0}$ (see, e.g., [74] and Problem 9.11).

Remarks 9.2.

- The previous lemma has a very interesting and important consequence. If $\hat{\boldsymbol{\theta}}$ is the *unique* minimizer of (9.21), then

$$\operatorname{card}\{i : \hat{\theta}_i = 0\} \geq \dim(\operatorname{null}(X)), \quad (9.23)$$

where $\text{card}\{\cdot\}$ denotes the cardinality of a set. In words, the number of zero coordinates of the unique minimizer cannot be smaller than the dimension of the null space of X . Indeed, if this is not the case, then the unique minimizer could have fewer zeros than the dimensionality of $\text{null}(X)$. This means that we can always find a $z \in \text{null}(X)$, which has zeros in the same locations where the coordinates of the unique minimizer are zero, and at the same time it is not identically zero, that is, $z \neq 0$ (Problem 9.12). However, this would violate (9.22), which in the case of uniqueness holds as a strict inequality.

Definition 9.1. A vector θ is called k -sparse if it has *at most* k nonzero components.

Remarks 9.3.

- If the minimizer of (9.21) is *unique*, then it is a k -sparse vector with

$$k \leq N.$$

This is a direct consequence of Remarks 9.2, and the fact that for the matrix X ,

$$\dim(\text{null}(X)) = l - \text{rank}(X) = l - N.$$

Hence, the number of the nonzero elements of the unique minimizer must be at most equal to N . If one resorts to geometry, all the previously stated results become crystal clear.

GEOMETRIC INTERPRETATION

Assume that our target solution resides in the three-dimensional space and that we are given one measurement,

$$y_1 = \mathbf{x}_1^T \boldsymbol{\theta} = x_{11}\theta_1 + x_{12}\theta_2 + x_{13}\theta_3.$$

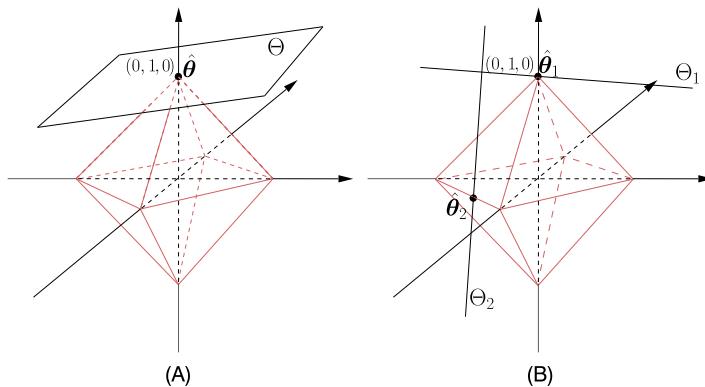
Then the solution lies in the two-dimensional (hyper)plane, which is described by the previous equation. To get the minimal ℓ_1 solution we keep increasing the size of the ℓ_1 ball³ (all the points that lie on the sides/faces of this ball have equal ℓ_1 norm) until it touches this plane. The only way that these two geometric objects have a single point in common (unique solution) is when they meet at a vertex of the diamond. This is shown in Fig. 9.8A. In other words, the resulting solution is 1-sparse, having two of its components equal to zero. This complies with the finding stated in Remarks 9.3, because now $N = 1$. For any other orientation of the plane, this will either cut across the ℓ_1 ball or will share with the diamond an edge or a side. In both cases, there will be infinite solutions.

Let us now assume that we are given an extra measurement,

$$y_2 = x_{21}\theta_1 + x_{22}\theta_2 + x_{23}\theta_3.$$

The solution now lies in the intersection of the two previous planes, which is a straight line. However, now, we have more alternatives for a unique solution. A line, for example, Θ_1 , can either touch the ℓ_1 ball at a vertex (1-sparse solution) or, as shown in Fig. 9.8B, touch the ℓ_1 ball at one of its edges, for

³ Observe that in the three-dimensional space the ℓ_1 ball looks like a diamond.

**FIGURE 9.8**

(A) The ℓ_1 ball intersecting with a plane. The only possible scenario, for the existence of a unique common intersecting point of the ℓ_1 ball with a plane in the Euclidean \mathbb{R}^3 space, is for the point to be located at one of the vertices of the ℓ_1 ball, that is, to be a 1-sparse vector. (B) The ℓ_1 ball intersecting with lines. In this case, the sparsity level of the unique intersecting point is relaxed; it could be a 1- or a 2-sparse vector.

example, Θ_2 . The latter case corresponds to a solution that lies on a two-dimensional subspace; hence it will be a 2-sparse vector. This also complies with the findings stated in Remarks 9.3, because in this case we have $N = 2$, $l = 3$, and the sparsity level for a unique solution can be either 1 or 2.

Note that uniqueness is associated with the particular geometry and orientation of the affine set, which is the set of all possible solutions of the underdetermined system of equations. For the case of the squared ℓ_2 norm, the solution is always unique. This is a consequence of the (hyper)spherical shape formed by the Euclidean norm. From a mathematical point of view, the squared ℓ_2 norm is a strictly convex function. This is not the case for the ℓ_1 norm, which is convex, albeit not a strictly convex function (Problem 9.13).

Example 9.2. Consider a sparse vector parameter $[0, 1]^T$, which we assume to be unknown. We will use one measurement to *sense* it. Based on this single measurement, we will use the ℓ_1 minimizer of (9.21) to recover its true value. Let us see what happens.

We will consider three different values of the “sensing” (input) vector \mathbf{x} in order to obtain the measurement $y = \mathbf{x}^T \boldsymbol{\theta}$: (a) $\mathbf{x} = [\frac{1}{2}, 1]^T$, (b) $\mathbf{x} = [1, 1]^T$, and (c) $\mathbf{x} = [2, 1]^T$. The resulting measurement, after sensing $\boldsymbol{\theta}$ by \mathbf{x} , is $y = 1$ for all three previous cases.

Case a: The solution will lie on the straight line

$$\Theta = \left\{ [\theta_1, \theta_2]^T \in \mathbb{R}^2 : \frac{1}{2}\theta_1 + \theta_2 = 1 \right\},$$

which is shown in Fig. 9.9A. For this setting, expanding the ℓ_1 ball, this will touch the straight line (our solution’s affine set) at the vertex $[0, 1]^T$. This is a unique solution, hence it is sparse, and it coincides with the true value.

Case b: The solution lies on the straight line

$$\Theta = \left\{ [\theta_1, \theta_2]^T \in \mathbb{R}^2 : \theta_1 + \theta_2 = 1 \right\},$$

which is shown in Fig. 9.9B. For this setup, there is an infinite number of solutions, including two sparse ones.

Case c: The affine set of solutions is described by

$$\Theta = \left\{ [\theta_1, \theta_2]^T \in \mathbb{R}^2 : 2\theta_1 + \theta_2 = 1 \right\},$$

which is sketched in Fig. 9.9C. The solution in this case is sparse, but it is not the correct one.

This example is quite informative. *If we sense (measure) our unknown parameter vector with appropriate sensing (input) data, the use of the ℓ_1 norm can unveil the true value of the parameter vector, even if the system of equations is underdetermined, provided that the true parameter is sparse.* This becomes our new goal; to investigate whether what we have just said can be generalized, and under which conditions it holds true. In such a case, the choice of the regressors (which we called sensing vectors) and hence the input matrix (which we will refer to more and more frequently as the sensing matrix) acquire extra significance. It is not enough for the designer to care only for the rank of the matrix, that

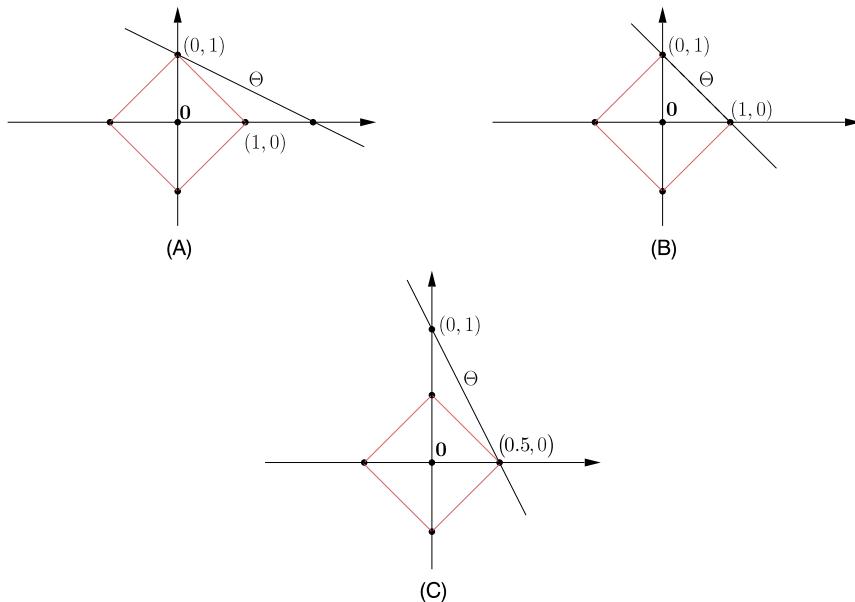


FIGURE 9.9

(A) Sensing with $x = [\frac{1}{2}, 1]^T$. (B) Sensing with $x = [1, 1]^T$. (C) Sensing with $x = [2, 1]^T$. The choice of the sensing vector x is crucial to unveiling the true sparse solution $(0,1)$. Only the sensing vector $x = [\frac{1}{2}, 1]^T$ identifies uniquely the desired $(0,1)$.

is, the linear independence of the sensing vectors. One has to make sure that the corresponding affine set of the solutions has such an orientation so that the touch with the ℓ_1 ball (as this increases from zero to meet this plane) is a “gentle” one; that is, they meet at a single point, and more importantly, at the correct one, which is the point that represents the true value of the sparse parameter vector.

Remarks 9.4.

- Often in practice, the columns of the input matrix, X , are normalized to unit ℓ_2 norm. Although ℓ_0 norm is insensitive to the values of the nonzero components of θ , this is not the case with the ℓ_1 and ℓ_2 norms. Hence, while trying to minimize the respective norms and at the same time fulfill the constraints, components that correspond to columns of X with high energy (norm) are favored over the rest. Hence, the latter become more popular candidates to be pushed to zero. In order to avoid such situations, the columns of X are normalized to unity by dividing each element of the column vector by the respective (Euclidean) norm.

9.6 UNIQUENESS OF THE ℓ_0 MINIMIZER

Our first goal is to derive *sufficient* conditions that guarantee uniqueness of the ℓ_0 minimizer, which has been defined in Section 9.5.

Definition 9.2. The *spark* of a full row rank $N \times l$ ($l \geq N$) matrix, X , denoted as $\text{spark}(X)$, is the *smallest* number of its linearly *dependent* columns.

According to the previous definition, *any* $m < \text{spark}(X)$ column of X is necessarily *linearly independent*. The spark of a square, $N \times N$, full rank matrix is equal to $N + 1$.

Remarks 9.5.

- In contrast to the rank of a matrix, which can be easily determined, its spark can only be obtained by resorting to a combinatorial search over all possible combinations of the columns of the respective matrix (see, e.g., [15,37]). The notion of the spark was used in the context of sparse representation, under the name *uniqueness representation property*, in [53]. The name “spark” was coined in [37]. An interesting discussion relating this matrix index with indices used in other disciplines is given in [15].
- Note that the notion of “spark” is related to the notion of the minimum Hamming weight of a linear code in coding theory (e.g., [60]).

Example 9.3. Consider the following matrix:

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

The matrix has rank equal to 4 and spark equal to 3. Indeed, any pair of columns is linearly independent. On the other hand, the first, second, and fifth columns are linearly dependent. The same is also true

$$\|\boldsymbol{\theta} - \boldsymbol{\theta}_*\|_2 \leq C_0 k^{-\frac{1}{2}} \|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|_1 + C_1 \sqrt{\epsilon}, \quad (9.36)$$

for some constants C_1 , C_0 , and $\boldsymbol{\theta}_k$ as defined in Theorem 9.3.

This is also an elegant result. If the model is exact and $\epsilon = 0$, we obtain (9.32). If not, the higher the uncertainty (noise) term in the model, the higher our ambiguity about the solution. Note, also, that the ambiguity about the solution depends on how far the true model is from $\boldsymbol{\theta}_k$. If the true model is k -sparse, the first term on the right-hand side of the inequality is zero. The values of C_1 , C_0 depend on δ_{2k} but they are small, for example, close to five or six [23].

The important conclusion here is that *adopting the ℓ_1 norm and the associated LASSO optimization for solving inverse problems (which in general, as we noted in Chapter 3, tend to be ill-conditioned) is a stable one and the noise is not amplified excessively during the recovery process.*

9.9 COMPRESSED SENSING: THE GLORY OF RANDOMNESS

The way in which this chapter was deployed followed, more or less, the sequence of developments that took place during the evolution of the sparsity-aware parameter estimation field. We intentionally made an effort to follow such a path, because this is also indicative of how science evolves in most cases. The starting point had a rather strong mathematical flavor: to develop conditions for the solution of an underdetermined linear system of equations, under the sparsity constraint and in a mathematically tractable way, that is, using convex optimization. In the end, the accumulation of a sequence of individual contributions revealed that the solution can be (uniquely) recovered if the unknown quantity is sensed via randomly chosen data samples. This development has, in turn, given birth to a new field with strong theoretical interest as well as with an enormous impact on practical applications. This new emerged area is known as *compressed sensing* or *compressive sampling* (CS), and it has changed our view on how to sense and process signals efficiently.

COMPRESSED SENSING

In CS, the goal is to directly acquire as few samples as possible that encode the minimum information needed to obtain a compressed signal representation. In order to demonstrate this, let us return to the data compression example discussed in Section 9.4. There, it was commented that the “classical” approach to compression was rather unorthodox, in the sense that first all (i.e., a number of l) samples of the signal are used, and then they are processed to obtain l transformed values, from which only a small subset is used for coding. In the CS setting, the procedure changes to the following one.

Let X be an $N \times l$ sensing matrix, which is applied to the (unknown) signal vector, s , in order to obtain the observations, y , and let Ψ be the dictionary matrix that describes the domain where the signal s accepts a sparse representation, that is,

$$\begin{aligned} s &= \Psi \boldsymbol{\theta}, \\ y &= Xs. \end{aligned} \quad (9.37)$$

Assuming that at most k of the components of $\boldsymbol{\theta}$ are nonzero, this can be obtained by the following optimization task:

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \quad & \|\boldsymbol{\theta}\|_1, \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{X}\Psi\boldsymbol{\theta}, \end{aligned} \tag{9.38}$$

provided that the combined matrix $\mathbf{X}\Psi$ complies with the RIP, and the number of observations, N , is large enough, as dictated by the bound in (9.33). Note that s needs not be stored and can be obtained any time, once $\boldsymbol{\theta}$ is known. Moreover, as we will soon discuss, there are techniques that allow observations, y_n , $n = 1, 2, \dots, N$, to be acquired directly from an analog signal $s(t)$, prior to obtaining its sample (vector) version, \mathbf{s} ! Thus, from such a perspective, CS fuses the data acquisition and the compression steps together.

There are different ways to obtain a sensing matrix, \mathbf{X} , that lead to a product $\mathbf{X}\Psi$, which satisfies the RIP. It can be shown (Problem 9.19) that if Ψ is orthonormal and \mathbf{X} is a random matrix, which is constructed as discussed at the end of Section 9.7.2, then the product $\mathbf{X}\Psi$ obeys the RIP, provided that (9.33) is satisfied. An alternative way to obtain a combined matrix that respects the RIP is to consider another orthonormal matrix Φ , whose columns have low coherence with the columns of Ψ (coherence between two matrices is defined in (9.25), where now the place of \mathbf{x}_i^c is taken by a column of Φ and that of \mathbf{x}_j^c by a column of Ψ). For example, Φ could be the DFT matrix and $\Psi = I$, or vice versa. Then choose N rows of Φ uniformly at random to form \mathbf{X} in (9.37). In other words, for such a case, the sensing matrix can be written as $R\Phi$, where R is an $N \times l$ matrix that extracts N rows uniformly at random. The notion of incoherence (low coherence) between the sensing and the basis matrices is closely related to RIP. The more incoherent the two matrices, the lower the number of the required observations for the RIP to hold (e.g., [21, 79]). Another way to view incoherence is that the rows of Φ cannot be sparsely represented in terms of the columns of Ψ . It turns out that if the sensing matrix \mathbf{X} is a random one, formed as described in Section 9.7.2, then the RIP and the incoherence with any Ψ are satisfied with high probability.

It gets even better when we say that all the previously stated philosophy can be extended to the more general type of signals, which are not necessarily sparse or sparsely represented in terms of the atoms of a dictionary, and they are known as *compressible*. A signal vector is said to be compressible if its expansion in terms of a basis consists of just a few large parameters θ_i and the rest are small. In other words, the signal vector is *approximately* sparse in some basis. Obviously, this is the most interesting case in practice, where exact sparsity is scarcely (if ever) met. Reformulating the arguments used in Section 9.8, the CS task for this case can be cast as

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \quad & \|\boldsymbol{\theta}\|_1, \\ \text{s.t.} \quad & \|\mathbf{y} - \mathbf{X}\Psi\boldsymbol{\theta}\|_2^2 \leq \epsilon, \end{aligned} \tag{9.39}$$

and everything that has been said in Section 9.8 is also valid for this case, if in place of \mathbf{X} we consider the product $\mathbf{X}\Psi$.

Remarks 9.9.

- An important property in CS is that the sensing matrix, which provides the observations, may be chosen independently on the matrix Ψ , that is, the basis/dictionary in which the signal is sparsely

represented. In other words, the sensing matrix can be “universal” and can be used to provide the observations for reconstructing any sparse or sparsely represented signal in any dictionary, provided the RIP is not violated.

- Each observation, y_n , is the result of an inner product of the signal vector with a row x_n^T of the sensing matrix X . Assuming that the signal vector s is the result of a sampling process on an analog signal, $s(t)$, y_n can be directly obtained, to a good approximation, by taking the inner product (integral) of $s(t)$ with a sensing waveform, $x_n(t)$, that corresponds to x_n . For example, if X is formed by ± 1 , as described in Section 9.7.2, then the configuration shown in Fig. 9.10 results in y_n . An important aspect of this approach, besides avoiding computing and storing the l components of s , is that multiplying by ± 1 is a relatively easy operation. It is equivalent to changing the polarity of the signal and it can be implemented by employing inverters and mixers. It is a process that can be performed, in practice, at much higher rates than sampling. The sampling system shown in Fig. 9.10 is referred to as *random demodulator* [58,91]. It is one among the popular analog-to-digital (A/D) conversion architectures, which exploit the CS rationale in order to sample at rates much lower than those required for classical sampling. We will come back to this soon.

One of the very first CS-based acquisition systems was an imaging system called the *one-pixel camera* [84], which followed an approach resembling the conventional digital CS. According to this, light of an image of interest is projected onto a random base generated by a micromirror device. A sequence of projected images is collected by a *single photodiode* and used for the reconstruction of the full image using conventional CS techniques. This was among the most catalytic examples that spread the rumor about the practical power of CS. CS is an example of common wisdom: “There is nothing more practical than a good theory!”

9.9.1 DIMENSIONALITY REDUCTION AND STABLE EMBEDDINGS

We will now shed light on what we have said so far in this chapter from a different point of view. In both cases, either when the unknown quantity was a k -sparse vector in a high-dimensional space, \mathbb{R}^l , or when the signal s was (approximately) sparsely represented in some dictionary ($s = \Psi\theta$), we chose to work in a lower-dimensional space (\mathbb{R}^N), that is, the space of the observations, y . This is a typical task of dimensionality reduction (see Chapter 19). The main task in any (linear) dimensionality reduction technique is to choose the proper matrix X , which dictates the projection to the lower-dimensional space. In general, there is always a loss of information by projecting from \mathbb{R}^l to \mathbb{R}^N , with $N < l$, in

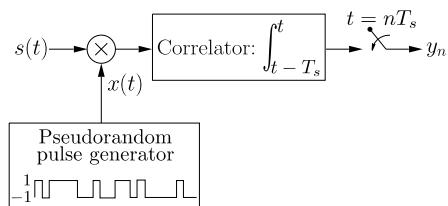


FIGURE 9.10

Sampling an analog signal $s(t)$ in order to generate the sample/observation y_n at time instant n . The sampling period T_s is much shorter than that required by the Nyquist sampling.

the sense that we cannot recover any vector $\theta_l \in \mathbb{R}^l$ from its projection $\theta_N \in \mathbb{R}^N$. Indeed, take any vector $\theta_{l-N} \in \text{null}(X)$ that lies in the $(l - N)$ -dimensional null space of the (full row rank) X (see Section 9.5). Then all vectors $\theta_l + \theta_{l-N} \in \mathbb{R}^l$ share the same projection in \mathbb{R}^N . However, what we have discovered in this chapter is that if the original vector is sparse, then we can recover it exactly. This is because all the k -sparse vectors do not lie anywhere in \mathbb{R}^l , but rather in a subset of it, that is, in a *union of subspaces*, each one having dimensionality k . If the signal s is sparse in some dictionary Ψ , then one has to search for it in the union of all possible k -dimensional subspaces of \mathbb{R}^l , which are spanned by k -column vectors from Ψ [8,62]. Of course, even in this case, where sparse vectors are involved, no projection can guarantee unique recovery. The guarantee is provided if the projection in the lower-dimensional space is a *stable embedding*. A stable embedding in a lower-dimensional space must guarantee that if $\theta_1 \neq \theta_2$, then their projections also remain different. Yet this is not enough. A stable embedding must guarantee that distances are (approximately) preserved; that is, vectors that lie far apart in the high-dimensional space have projections that also lie far apart. Such a property guarantees robustness to noise. The sufficient conditions, which have been derived and discussed throughout this chapter, and guarantee the recovery of a sparse vector lying in \mathbb{R}^l from its projections in \mathbb{R}^N , are conditions that guarantee stable embeddings. The RIP and the associated bound on N provide a condition on X that leads to stable embeddings. We commented on this norm preserving property of RIP in the related section. The interesting fact that came from the theory is that we can achieve such stable embeddings via random projection matrices.

Random projections for dimensionality reduction are not new and have extensively been used in pattern recognition, clustering, and data mining (see, e.g., [1,13,34,82,87]). The advent of the big data era resparked the interest in random projection-aided data analysis algorithms (e.g., [55,81]) for two major reasons. The first is that data processing is computationally lighter in the lower-dimensional space, because it involves operations with matrices or vectors represented with fewer parameters. Moreover, the projection of the data to lower-dimensional spaces can be realized via well-structured matrices at a computational cost significantly lower compared to that implied by general matrix-vector multiplications [29,42]. The reduced computational power required by these methods renders them appealing when dealing with excessively large data volumes. The second reason is that there exist randomized algorithms which access the data matrix a (usually fixed) number of times that is much smaller than the number of accesses performed by ordinary methods [28,55]. This is very important whenever the full amount of data does not fit in fast memory and has to be accessed in parts from slow memory devices, such as hard discs. In such cases, the computational time is often dominated by the cost of memory access.

The spirit underlying CS has been exploited in the context of pattern recognition too. In this application, one need not return to the original high-dimensional space after the information digging activity in the low-dimensional subspace. Since the focus in pattern recognition is to identify the class of an object/pattern, this can be performed in the observations subspace, provided that there is no class-related information loss. In [17], it is shown, using compressed sensing arguments, that if the data are approximately linearly separable in the original high-dimensional space and the data have a sparse representation, even in an unknown basis, then projecting randomly in the observations subspace retains the structure of linear separability.

Manifold learning is another area where random projections have also applied. A manifold is, in general, a nonlinear k -dimensional surface, embedded in a higher-dimensional (ambient) space. For example, the surface of a sphere is a two-dimensional manifold in a three-dimensional space. In [7,96],

the compressed sensing rationale is extended to signal vectors that live along a k -dimensional submanifold of the space \mathbb{R}^l . It is shown that if choosing a matrix X to project and a sufficient number N of observations, then the corresponding submanifold has a stable embedding in the observations subspace, under the projection matrix X ; that is, pair-wise Euclidean and geodesic distances are approximately preserved after the projection mapping. More on these issues can be found in the given references and in, for example, [8]. We will come to the manifold learning task in Chapter 19.

9.9.2 SUB-NYQUIST SAMPLING: ANALOG-TO-INFORMATION CONVERSION

In our discussion in the remarks presented before, we touched on a very important issue—that of going from the analog domain to the discrete one. The topic of A/D conversion has been at the forefront of research and technology since the seminal works of Shannon, Nyquist, Whittaker, and Kotelnikof were published; see, for example, [92] for a thorough related review. We all know that if the highest frequency of an analog signal $s(t)$ is less than $F/2$, then Shannon's theorem suggests that no loss of information is achieved if the signal is sampled, at least, at the Nyquist rate of $F = 1/T$, where T is the corresponding sampling period, and the signal can be perfectly recovered by its samples

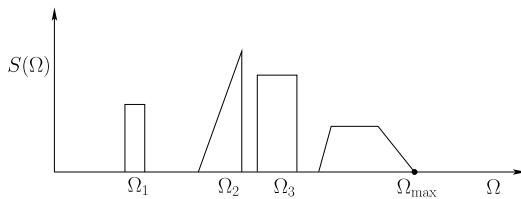
$$s(t) = \sum_n s(nT) \operatorname{sinc}(Ft - n),$$

where sinc is the sampling function

$$\operatorname{sinc}(t) = \frac{\sin(\pi t)}{\pi t}.$$

While this has been the driving force behind the development of signal acquisition devices, the increasing complexity of emerging applications demands increasingly higher sampling rates that cannot be accommodated by today's hardware technology. This is the case, for example, in wideband communications, where conversion speeds, as dictated by Shannon's bound, have become more and more difficult to obtain. Consequently, alternatives to high-rate sampling are attracting strong interest, with the goal of reducing the sampling rate by exploiting the *underlying structure* of the signals at hand. For example, in many applications, the signal comprises a few frequencies or bands; see Fig. 9.11 for an illustration. In such cases, sampling at the Nyquist rate is inefficient. This is an old problem investigated by a number of authors, leading to techniques that allow low-rate sampling whenever the locations of the nonzero bands in the frequency spectrum are known (see, e.g., [61, 93, 94]). CS theory has inspired research to study cases where the locations (carrier frequencies) of the bands are not known *a priori*. A typical application of this kind, of high practical interest, lies within the field of cognitive radio (e.g., [68, 88, 103]).

The process of sampling an analog signal with a rate lower than the Nyquist one is referred to as *analog-to-information* sampling or *sub-Nyquist* sampling. Let us focus on two among the most popular CS-based A/D converters. The first is the *random demodulator* (RD), which was first presented in [58] and later improved and theoretically developed in [91]. RD in its basic configuration is shown in Fig. 9.10, and it is designed for acquiring at sub-Nyquist rates sparse multitone signals, that is, signals having a sparse DFT. This implies that the signal comprises a few frequency components, but these components are constrained to correspond to integral frequencies. This limitation was pointed out in [91], and potential solutions have been sought according to the general framework proposed in [24]

**FIGURE 9.11**

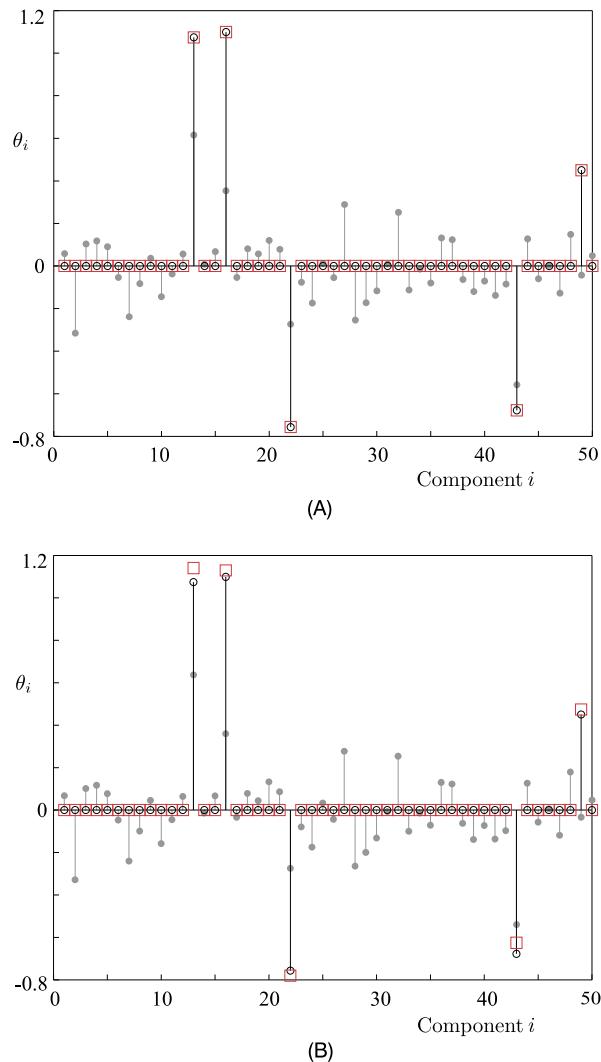
The Fourier transform of an analog signal, $s(t)$, which is sparse in the frequency domain; only a limited number of frequency bands contribute to its spectrum content $S(\Omega)$, where Ω stands for the angular frequency. Nyquist's theory guarantees that sampling at a frequency larger than or equal to twice the maximum Ω_{\max} is sufficient to recover the original analog signal. However, this theory does not exploit information related to the sparse structure of the signal in the frequency domain.

and/or the heuristic approach described in [45]. Moreover, more elaborate RD designs, such as the *random-modulation preintegrator* (RMPI) [102], have the potential to deal with signals that are sparse in any domain.

Another CS-based sub-Nyquist sampling strategy that has received much attention is the *modulated wideband converter* (MWC) [68,69,71]. The MWC is very efficient in acquiring multiband signals such as the one depicted in Fig. 9.11. This concept has also been extended to accommodate signals with different characteristics, such as signals consisting of short pulses [66]. An in-depth investigation which sheds light on the similarities and differences between the RD and MWC sampling architectures can be found in [59].

Note that both RD and MWC sample the signal uniformly in time. In [97], a different approach is adopted, leading to much easier implementations. In particular, the preprocessing stage is avoided and nonuniformly spread in time samples are acquired directly from the raw signal. In total, fewer samples are obtained compared to Nyquist sampling. Then CS-based reconstruction is mobilized in order to recover the signal under consideration based on the values of the samples and the time information. Like in the basic RD case, the nonuniform sampling approach is suitable for signals sparse in the DFT basis. From a practical point of view, there are still a number of hardware implementation-related issues that more or less concern all the approaches above and need to be solved (see, e.g., [9,25,63]).

An alternative path to sub-Nyquist sampling embraces a different class of analog signals, known as *multipulse* signals, that is, signals that consist of a stream of short pulses. Sparsity now refers to the time domain, and such signals may not even be bandlimited. Signals of this type can be met in a number of applications, such as radar, ultrasound, bioimaging, and neuronal signal processing (see, e.g., [41]). An approach known as *finite rate of innovation sampling* passes an analog signal having k degrees of freedom per second through a linear time-invariant filter, and then samples at a rate of $2k$ samples per second. Reconstruction is performed via rooting a high-order polynomial (see, e.g., [12, 95] and the references therein). In [66], the task of sub-Nyquist sampling is treated using CS theory arguments and an expansion in terms of Gabor functions; the signal is assumed to consist of a sum of a few pulses of finite duration, yet of unknown shape and time positions. More on this topic can be obtained in [43,51,70] and the references therein.

**FIGURE 9.12**

(A) Noiseless case. The values of the true vector, which generated the data for Example 9.5, are shown with stems topped with open circles. The recovered points are shown with squares. An exact recovery of the signal has been obtained. The stems topped with gray-filled circles correspond to the minimum Euclidean norm LS solution. (B) This figure corresponds to the noisy counterpart of that in (A). In the presence of noise, exact recovery is not possible and the higher the variance of the noise, the less accurate the results.

Example 9.5. We are given a set of $N = 20$ observations stacked in the $\mathbf{y} \in \mathbb{R}^N$ vector. These were taken by applying a sensing matrix X on an “unknown” vector in \mathbb{R}^{50} , which is known to be sparse

with $k = 5$ nonzero components; the location of these nonzero components in the unknown vector is not known. The sensing matrix was a random matrix with elements drawn from a normal distribution $\mathcal{N}(0, 1)$, and then the columns were normalized to unit norm. There are two scenarios for the observations. In the first one, we are given their exact values while in the second one, white Gaussian noise of variance $\sigma^2 = 0.025$ was added.

In order to recover the unknown sparse vector, the CS matching pursuit (CoSaMP, Chapter 10) algorithm was used for both scenarios.

The results are shown in Fig. 9.12A and B for the noiseless and noisy scenarios, respectively. The values of the true unknown vector θ are represented with black stems topped with open circles. Note that all but five of them are zero. In Fig. 9.12A, exact recovery of the unknown values is achieved; the estimated values of θ_i , $i = 1, 2, \dots, 50$, are indicated with squares in red color. In the noisy case of Fig. 9.12B, the resulting estimates, which are denoted with squares, deviate from the correct values. Note that estimated values very close to zero ($|\theta| \leq 0.01$) have been omitted from the figure in order to facilitate visualization. In both figures, the stemmed gray-filled circles correspond to the minimum ℓ_2 norm LS solution. The advantages of adopting a sparsity promoting approach to recover the solution are obvious. The CoSaMP algorithm was provided with the exact number of sparsity. The reader is advised to reproduce the example and play with different values of the parameters and see how results are affected.

9.10 A CASE STUDY: IMAGE DENOISING

We have already discussed CS as a notable application of sparsity-aware learning. Although CS has acquired a lot of fame, a number of classical signal processing and machine learning tasks lend themselves to efficient modeling via sparsity-related arguments. Two typical examples are the following.

- *Denoising:* The problem in signal denoising is that instead of the actual signal samples, $\tilde{\mathbf{y}}$, a noisy version of the corresponding observations, \mathbf{y} , is available; that is, $\mathbf{y} = \tilde{\mathbf{y}} + \boldsymbol{\eta}$, where $\boldsymbol{\eta}$ is the vector of noise samples. Under the sparse modeling framework, the unknown signal $\tilde{\mathbf{y}}$ is modeled as a sparse representation in terms of a specific known dictionary Ψ , that is, $\tilde{\mathbf{y}} = \Psi\boldsymbol{\theta}$. Moreover, the dictionary is allowed to be redundant (overcomplete). Then the denoising procedure is realized in two steps.

First, an estimate of the sparse representation vector, $\hat{\boldsymbol{\theta}}$, is obtained via the ℓ_0 norm minimizer or via any LASSO formulation, for example,

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^I} \|\boldsymbol{\theta}\|_1, \quad (9.40)$$

$$\text{s.t. } \|\mathbf{y} - \Psi\hat{\boldsymbol{\theta}}\|_2^2 \leq \epsilon. \quad (9.41)$$

Second, the estimate of the true signal is computed as $\hat{\mathbf{y}} = \hat{\boldsymbol{\theta}}\hat{\Psi}$. In Chapter 19, we will study the case where the dictionary is not fixed and known, but is estimated from the data.

- *Linear inverse problems:* Such problems, which come under the more general umbrella of what is known as *signal restoration*, go one step beyond denoising. Now, the available observations are *distorted* as well as noisy versions of the true signal samples; that is, $\mathbf{y} = H\tilde{\mathbf{y}} + \boldsymbol{\eta}$, where H is a known linear operator. For example, H may correspond to the blurring point spread function of an

SPARSITY-AWARE LEARNING: ALGORITHMS AND APPLICATIONS

10

CONTENTS

10.1 Introduction	473
10.2 Sparsity Promoting Algorithms	474
10.2.1 Greedy Algorithms	474
<i>OMP Can Recover Optimal Sparse Solutions: Sufficiency Condition</i>	477
<i>The LARS Algorithm</i>	478
<i>Compressed Sensing Matching Pursuit (CSMP) Algorithms</i>	479
10.2.2 Iterative Shrinkage/Thresholding (IST) Algorithms	480
10.2.3 Which Algorithm? Some Practical Hints	487
10.3 Variations on the Sparsity-Aware Theme	492
10.4 Online Sparsity Promoting Algorithms	499
10.4.1 LASSO: Asymptotic Performance	500
10.4.2 The Adaptive Norm-Weighted LASSO	502
10.4.3 Adaptive CoSaMP Algorithm	504
10.4.4 Sparse-Adaptive Projection Subgradient Method <i>Projection Onto the Weighted ℓ_1 Ball</i>	505
10.4.5 Online LASSO	507
10.5 Learning Sparse Analysis Models	510
10.5.1 Compressed Sensing for Sparse Signal Representation in Coherent Dictionaries	512
10.5.2 Cosparsity	513
10.6 A Case Study: Time-Frequency Analysis	516
Gabor Transform and Frames	516
Time-Frequency Resolution	517
Gabor Frames	518
Time-Frequency Analysis of Echolocation Signals Emitted by Bats	519
Problems	523
MATLAB® Exercises	524
References	525

10.1 INTRODUCTION

This chapter is the follow-up to the previous one concerning sparsity-aware learning. The emphasis now is on the algorithmic front. Following the theoretical advances concerning sparse modeling, a true scientific happening occurred in trying to derive algorithms tailored for the efficient solution of the related constrained optimization tasks. Our goal is to present the main directions that have been followed and to provide in a more explicit form some of the most popular algorithms. We will discuss batch as

well as online algorithms. This chapter can also be considered as a complement to Chapter 8, where some aspects of convex optimization were introduced; a number of algorithms discussed there are also appropriate for tasks involving sparsity-related constraints/regularization.

Besides describing various algorithmic families, some variants of the basic sparsity promoting ℓ_1 and ℓ_0 norms are discussed. Furthermore, some typical examples are considered and a case study concerning time-frequency analysis is presented. Finally, a discussion concerning the issue “synthesis versus analysis” models is provided.

10.2 SPARSITY PROMOTING ALGORITHMS

In the previous chapter, our emphasis was on highlighting some of the most important aspects underlying the theory of sparse signal/parameter vector recovery from an underdetermined set of linear equations. We now turn our attention to the algorithmic aspects of the problem (e.g., [52,54]). The issue now becomes that of discussing *efficient* algorithmic schemes, which can achieve the recovery of the unknown set of parameters. In Sections 9.3 and 9.5, we saw that the constrained ℓ_1 norm minimization (basis pursuit) can be solved via linear programming techniques and the LASSO task via convex optimization schemes. However, such general purpose techniques tend to be inefficient, because they often require many iterations to converge, and the respective computational resources can be excessive for practical applications, especially in high-dimensional spaces \mathbb{R}^l . As a consequence, a huge research effort has been invested for the goal of developing efficient algorithms that are tailored to these specific tasks. Our aim here is to provide the reader with some general trends and philosophies that characterize the related activity. We will focus on the most commonly used and cited algorithms, which at the same time are structurally simple, so the reader can follow them, without deeper knowledge of optimization. Moreover, these algorithms involve, in one way or another, arguments that are directly related to notions we have already used while presenting the theory; thus, they can also be exploited from a pedagogical point of view in order to strengthen the reader’s understanding of the topic. We start our review with the class of batch algorithms, where all data are assumed to be available prior to the application of the algorithm, and then we will move on to online/time-adaptive schemes. Furthermore, our emphasis is on algorithms that are appropriate for any sensing matrix. This is stated in order to point out that in the literature, efficient algorithms have also been developed for specific forms of highly structured sensing matrices, and exploiting their particular structure can lead to reduced computational demands [61,93].

There are three rough types of families along which this algorithmic activity has grown: (a) greedy algorithms, (b) iterative shrinkage schemes, and (c) convex optimization techniques. We have used the word rough because in some cases, it may be difficult to assign an algorithm to a specific family.

10.2.1 GREEDY ALGORITHMS

Greedy algorithms have a long history; see, for example, [114] for a comprehensive list of references. In the context of dictionary learning, a greedy algorithm known as *matching pursuit* was introduced in [88]. A greedy algorithm is built upon a series of *locally* optimal *single-term* updates. In our context, the goals are (a) to unveil the “active” columns of the sensing matrix X , that is, those columns that correspond to the nonzero locations of the unknown parameters, and (b) to estimate the respective

sparse parameter vector. The set of indices that correspond to the nonzero vector components is also known as the *support*. To this end, the set of active columns of X (and the support) is increased by one at each iteration step. In the sequel, an updated estimate of the unknown sparse vector is obtained. Let us assume that at the $(i - 1)$ th iteration step, the algorithm has selected the columns denoted as $\mathbf{x}_{j_1}^c, \mathbf{x}_{j_2}^c, \dots, \mathbf{x}_{j_{i-1}}^c$, with $j_1, j_2, \dots, j_{i-1} \in \{1, 2, \dots, l\}$. These indices are the elements of the currently available support, $S^{(i-1)}$. Let $X^{(i-1)}$ be the $N \times (i - 1)$ matrix, having $\mathbf{x}_{j_1}^c, \mathbf{x}_{j_2}^c, \dots, \mathbf{x}_{j_{i-1}}^c$ as its columns. Let also the current estimate of the solution be $\boldsymbol{\theta}^{(i-1)}$, which is a $(i - 1)$ -sparse vector, with zeros at all locations with index outside the support. The *orthogonal matching pursuit* (OMP) scheme given in Algorithm 10.1 builds up recursively a sparse solution.

Algorithm 10.1 (The OMP algorithm).

The algorithm is initialized with $\boldsymbol{\theta}^{(0)} := \mathbf{0}$, $\mathbf{e}^{(0)} := \mathbf{y}$, and $S^{(0)} = \emptyset$. At iteration step i , the following computational steps are performed:

1. Select the column $\mathbf{x}_{j_i}^c$ of X , which is *maximally* correlated to (forms the smallest angle with) the respective error vector, $\mathbf{e}^{(i-1)} := \mathbf{y} - X\boldsymbol{\theta}^{(i-1)}$, that is,

$$\mathbf{x}_{j_i}^c : j_i := \arg \max_{j=1,2,\dots,l} \frac{|\mathbf{x}_j^{cT} \mathbf{e}^{(i-1)}|}{\|\mathbf{x}_j^c\|_2}.$$

2. Update the support and the corresponding set of active columns, $S^{(i)} = S^{(i-1)} \cup \{j_i\}$ and $X^{(i)} = [X^{(i-1)}, \mathbf{x}_{j_i}^c]$.
3. Update the estimate of the parameter vector: Solve the least-squares (LS) problem that minimizes the norm of the error, using the active columns of X only, that is,

$$\tilde{\boldsymbol{\theta}} := \arg \min_{z \in \mathbb{R}^i} \|\mathbf{y} - X^{(i)} z\|_2^2.$$

Obtain $\boldsymbol{\theta}^{(i)}$ by inserting the elements of $\tilde{\boldsymbol{\theta}}$ in the respective locations (j_1, j_2, \dots, j_i) , which comprise the support (the rest of the elements of $\boldsymbol{\theta}^{(i)}$ retain their zero values).

4. Update the error vector

$$\mathbf{e}^{(i)} := \mathbf{y} - X^{(i)} \boldsymbol{\theta}^{(i)}.$$

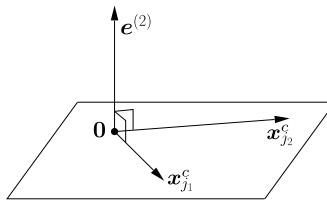
The algorithm terminates if the norm of the error becomes less than a preselected user-defined constant, ϵ_0 . The following observations are in order.

Remarks 10.1.

- Because $\boldsymbol{\theta}^{(i)}$, in Step 3, is the result of an LS task, we know from Chapter 6 that the error vector is orthogonal to the subspace spanned by the active columns involved, that is,

$$\mathbf{e}^{(i)} \perp \text{span}\{\mathbf{x}_{j_1}^c, \dots, \mathbf{x}_{j_i}^c\}.$$

This guarantees that in the next step, taking the correlation of the columns of X with $\mathbf{e}^{(i)}$, none of the previously selected columns will be reselected; they result to zero correlation, being orthogonal to $\mathbf{e}^{(i)}$ (see Fig. 10.1).

**FIGURE 10.1**

The error vector at the i th iteration is orthogonal to the subspace spanned by the currently available set of active columns. Here is an illustration for the case of the three-dimensional Euclidean space \mathbb{R}^3 , and for $i = 2$.

- The column which has maximal correlation (maximum absolute value of the inner product) with the currently available error vector is the one that maximally reduces (compared to any other column) the ℓ_2 norm of the error, when \mathbf{y} is approximated by linearly combining the currently available active columns. This is the point where the heart of the greedy strategy beats. This minimization is with respect to a *single term*, keeping the rest fixed, as they have been obtained from the previous iteration steps (Problem 10.1).
- Starting with all the components being zero, if the algorithm stops after k_0 iteration steps, the result will be a k_0 -sparse solution.
- Note that there is no optimality in this searching strategy. The only guarantee is that the ℓ_2 norm of the error vector is decreased at every iteration step. In general, there is no guarantee that the algorithm can obtain a solution close to the true one (see, for example, [38]). However, under certain constraints on the structure of X , performance bounds can be obtained (see, for example, [37, 115, 123]).
- The complexity of the algorithm amounts to $\mathcal{O}(k_0 l N)$ operations, which are contributed by the computations of the correlations, plus the demands raised by the solution of the LS task in step 3, whose complexity depends on the specific algorithm used. The k_0 is the sparsity level of the delivered solution and, hence, the total number of iteration steps that are performed.

Another more qualitative argument that justifies the selection of the columns based on their correlation with the error vector is the following. Assume that the matrix X is orthonormal. Let $\mathbf{y} = X\boldsymbol{\theta}$. Then \mathbf{y} lies in the subspace spanned by the active columns of X , that is, those that correspond to the nonzero components of $\boldsymbol{\theta}$. Hence, the rest of the columns are orthogonal to \mathbf{y} , because X is assumed to be orthonormal. Taking the correlation of \mathbf{y} , at the first iteration step, with all the columns, it is certain that one among the active columns will be chosen. The inactive columns result in zero correlation. A similar argument holds true for all subsequent steps, because all activity takes place in a subspace that is orthogonal to all the inactive columns of X . In the more general case, where X is not orthonormal, we can still use the correlation as a measure that quantifies geometric similarity. The smaller the correlation/magnitude of the inner product is, the more orthogonal the two vectors are. This brings us back to the notion of mutual coherence, which is a measure of the maximum correlation (smallest angle) among the columns of X .

The algorithm requires as input the sparsity level k . Iterations carry on until a halting criterion is met. The value of t , which determines the largest in magnitude values in steps 1 and 3a, depends on the specific algorithm. In the compressive sampling matching pursuit (CoSaMP) [93], $t = 2k$ (Problem 10.3), and in the subspace pursuit (SP) [33], $t = k$.

Having stated the general scheme, a major difference with OMP becomes readily apparent. In OMP, only one column is selected per iteration step. Moreover, this remains in the active set for all subsequent steps. If, for some reason, this was not a good choice, the scheme cannot recover from such a bad decision. In contrast, the support and, hence, the active columns of X are continuously updated in CSMP, and the algorithm has the ability to correct a previously bad decision, as more information is accumulated and iterations progress. In [33], it is shown that if the measurements are exact ($\mathbf{y} = X\boldsymbol{\theta}$), then SP can recover the k -sparse true vector in a finite number of iteration steps, provided that X satisfies the RIP with $\delta_{3k} < 0.205$. If the measurements are noisy, performance bounds have been derived, which hold true for $\delta_{3k} < 0.083$. For the CoSaMP, performance bounds have been derived for $\delta_{4k} < 0.1$.

10.2.2 ITERATIVE SHRINKAGE/THRESHOLDING (IST) ALGORITHMS

This family of algorithms also have a long history (see, for example, [44, 69, 70, 73]). However, in the “early” days, most of the developed algorithms had some sense of heuristic flavor, without establishing a clear bridge with optimizing a cost function. Later attempts were substantiated by sound theoretical arguments concerning issues such as convergence and convergence rate [31, 34, 50, 56].

The general form of this algorithmic family has a striking resemblance to the classical linear algebra iterative schemes for approximating the solution of large linear systems of equations, known as *stationary iterative* or *iterative relaxation* methods. The classical Gauss–Seidel and Jacobi algorithms (e.g., [65]), in numerical analysis can be considered members of this family. Given a linear system of l equations with l unknowns, $\mathbf{z} = \mathbf{Ax}$, the basic iteration at step i has the following form:

$$\begin{aligned}\mathbf{x}^{(i)} &= (I - Q\mathbf{A})\mathbf{x}^{(i-1)} + Q\mathbf{z} \\ &= \mathbf{x}^{(i-1)} + Q\mathbf{e}^{(i-1)}, \quad \mathbf{e}^{(i-1)} := \mathbf{z} - \mathbf{A}\mathbf{x}^{(i-1)},\end{aligned}$$

which does not come as a surprise. It is of the same form as most of the iterative schemes for numerical solutions! The matrix Q is chosen in order to guarantee convergence, and different choices lead to different algorithms with their pros and cons. It turns out that this algorithmic form can also be applied to underdetermined systems of equations, $\mathbf{y} = X\boldsymbol{\theta}$, with a “minor” modification, which is imposed by the sparsity constraint of the target vector. This leads to the following general form of iterative computation:

$$\boldsymbol{\theta}^{(i)} = T_i(\boldsymbol{\theta}^{(i-1)} + Q\mathbf{e}^{(i-1)}), \quad \mathbf{e}^{(i-1)} = \mathbf{y} - X\boldsymbol{\theta}^{(i-1)},$$

starting from an initial guess of $\boldsymbol{\theta}^{(0)}$ (usually $\boldsymbol{\theta}^{(0)} = \mathbf{0}$, $\mathbf{e}^{(0)} = \mathbf{y}$). In certain cases, Q can be made to be iteration dependent. The function T_i is a nonlinear thresholding that is applied *entry-wise*, that is, *component-wise*. Depending on the specific scheme, this can be either the hard thresholding function, denoted as H_k , or the soft thresholding function, denoted as S_α . Hard thresholding, as we already know, keeps the k largest components of a vector unaltered and sets the rest equal to zero. Soft thresholding was introduced in Section 9.3. All components with magnitude less than a threshold value α are forced

to zero and the rest are reduced in magnitude by α ; that is, the j th component of a vector $\boldsymbol{\theta}$, after soft thresholding, becomes

$$(S_\alpha(\boldsymbol{\theta}))_j = \text{sgn}(\theta_j)(|\theta_j| - \alpha)_+.$$

Depending on (a) the choice of T_i , (b) the specific value of the parameter k or α , and (c) the matrix Q , different instances occur. The most common choice for Q is μX^T , and the generic form of the main iteration becomes

$$\boxed{\boldsymbol{\theta}^{(i)} = T_i \left(\boldsymbol{\theta}^{(i-1)} + \mu X^T \mathbf{e}^{(i-1)} \right)}, \quad (10.2)$$

where μ is a relaxation (user-defined) parameter, which can also be left to vary with each iteration step. The choice of X^T is intuitively justified, once more, by the near-orthogonal nature of X . For the first iteration step and for a linear system of the form $\mathbf{y} = X\boldsymbol{\theta}$, starting from a zero initial guess, we have $X^T \mathbf{y} = X^T X \boldsymbol{\theta} \approx \boldsymbol{\theta}$ and we are close to the solution.

Although intuition is most important in scientific research, it is not enough, by itself, to justify decisions and actions. The generic scheme in (10.2) has been reached from different paths, following different perspectives that lead to different choices of the involved parameters. Let us spend some more time on that, with the aim of making the reader more familiar with techniques that address optimization tasks of nondifferentiable loss functions. The term in parentheses in (10.2) coincides with the gradient descent iteration step if the cost function is the unregularized sum of squared errors cost (LS), that is,

$$J(\boldsymbol{\theta}) = \frac{1}{2} \|\mathbf{y} - X\boldsymbol{\theta}\|_2^2.$$

In this case, the gradient descent rationale leads to

$$\begin{aligned} \boldsymbol{\theta}^{(i-1)} - \mu \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}} &= \boldsymbol{\theta}^{(i-1)} - \mu X^T (X\boldsymbol{\theta}^{(i-1)} - \mathbf{y}) \\ &= \boldsymbol{\theta}^{(i-1)} + \mu X^T \mathbf{e}^{(i-1)}. \end{aligned}$$

The gradient descent can alternatively be viewed as the result of minimizing a regularized version of the linearized cost function (verify it),

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left\{ J(\boldsymbol{\theta}^{(i-1)}) + (\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)})^T \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}} \right. \\ &\quad \left. + \frac{1}{2\mu} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|_2^2 \right\}. \end{aligned} \quad (10.3)$$

One can adopt this view of the gradient descent philosophy as a kick-off point to minimize iteratively the following LASSO task:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left\{ L(\boldsymbol{\theta}, \lambda) = \frac{1}{2} \|\mathbf{y} - X\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1 = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1 \right\}.$$

The difference now is that the loss function comprises two terms: one that is smooth (differentiable) and a nonsmooth one. Let the current estimate be $\boldsymbol{\theta}^{(i-1)}$. The updated estimate is obtained by

$$\begin{aligned}\boldsymbol{\theta}^{(i)} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} & \left\{ J(\boldsymbol{\theta}^{(i-1)}) + (\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)})^T \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}} \right. \\ & \left. + \frac{1}{2\mu} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1 \right\},\end{aligned}$$

which, after ignoring constants, is equivalently written as

$$\boldsymbol{\theta}^{(i)} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left\{ \frac{1}{2} \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 + \lambda \mu \|\boldsymbol{\theta}\|_1 \right\}, \quad (10.4)$$

where

$$\tilde{\boldsymbol{\theta}} := \boldsymbol{\theta}^{(i-1)} - \mu \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}}. \quad (10.5)$$

Following exactly the same steps as those that led to the derivation of (9.13) from (9.6) (after replacing $\hat{\boldsymbol{\theta}}_{LS}$ with $\tilde{\boldsymbol{\theta}}$), we obtain

$$\boldsymbol{\theta}^{(i)} = S_{\lambda\mu}(\tilde{\boldsymbol{\theta}}) = S_{\lambda\mu} \left(\boldsymbol{\theta}^{(i-1)} - \mu \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}} \right) \quad (10.6)$$

$$= S_{\lambda\mu} \left(\boldsymbol{\theta}^{(i-1)} + \mu X^T \boldsymbol{e}^{(i-1)} \right). \quad (10.7)$$

This is very interesting and practically useful. The only effect of the presence of the nonsmooth ℓ_1 norm in the loss function is an extra simple thresholding operation, which as we know is an operation performed *individually* on each component. It can be shown (e.g., [11, 95]) that this algorithm converges to a minimizer $\boldsymbol{\theta}_*$ of the LASSO (9.6), provided that $\mu \in (0, 1/\lambda_{\max}(X^T X))$, where $\lambda_{\max}(\cdot)$ denotes the maximum eigenvalue of $X^T X$. The convergence rate is dictated by the rule

$$L(\boldsymbol{\theta}^{(i)}, \lambda) - L(\boldsymbol{\theta}_*, \lambda) \approx O(1/i),$$

which is known as *sublinear* global rate of convergence. Moreover, it can be shown that

$$L(\boldsymbol{\theta}^{(i)}, \lambda) - L(\boldsymbol{\theta}_*, \lambda) \leq \frac{C \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}_*\|_2^2}{2i}.$$

The latter result indicates that if one wants to achieve an accuracy of ϵ , then this can be obtained by at most $\lfloor \frac{C \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}_*\|_2^2}{2\epsilon} \rfloor$ iterations, where $\lfloor \cdot \rfloor$ denotes the floor function.

In [34], (10.2) was obtained from a nearby corner, building upon arguments from the classical *proximal point* methods in optimization theory (e.g., [105]). The original LASSO regularized cost function is modified to the *surrogate objective*,

$$J(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) = \frac{1}{2} \|\mathbf{y} - X\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1 + \frac{1}{2} d(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}),$$

where

$$d(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) := c \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 - \|X\boldsymbol{\theta} - X\tilde{\boldsymbol{\theta}}\|_2^2.$$

If c is appropriately chosen (larger than the largest eigenvalue of $X^T X$), the surrogate objective is guaranteed to be strictly convex. Then it can be shown (Problem 10.4) that the minimizer of the surrogate objective is given by

$$\hat{\boldsymbol{\theta}} = S_{\lambda/c} \left(\tilde{\boldsymbol{\theta}} + \frac{1}{c} X^T (\mathbf{y} - X\tilde{\boldsymbol{\theta}}) \right). \quad (10.8)$$

In the iterative formulation, $\tilde{\boldsymbol{\theta}}$ is selected to be the previously obtained estimate; in this way, one tries to keep the new estimate close to the previous one. The procedure readily results in our generic scheme in (10.2), using soft thresholding with parameter λ/c . It can be shown that such a strategy converges to a minimizer of the original LASSO problem. The same algorithm was reached in [56], using *majorization-minimization* techniques from optimization theory. So, from this perspective, the IST family has strong ties with algorithms that belong to the convex optimization category.

In [118], the *sparse reconstruction by separable approximation* (SpaRSA) algorithm is proposed, which is a modification of the standard IST scheme. The starting point is (10.3); however, the multiplying factor, $\frac{1}{2\mu}$, instead of being constant, is now allowed to change from iteration to iteration according to a rule. This results in a speedup in the convergence of the algorithm. Moreover, inspired by the homotopy family of algorithms, where λ is allowed to vary, SpaRSA can be extended to solve a sequence of problems that are associated with a corresponding sequence of values of λ . Once a solution has been obtained for a particular value of λ , it can be used as a “warm start” for a nearby value. Solutions can therefore be computed for a range of values, at a small extra computational cost, compared to solving for a single value from a “cold start.” This technique abides by the *continuation strategy*, which has been used in the context of other algorithms as well (e.g., [66]). Continuation has been shown to be a very successful tool to increase the speed of convergence.

An interesting variation of the basic IST scheme has been proposed in [11], which improves the convergence rate to $O(1/i^2)$, by only a simple modification with almost no extra computational burden. The scheme is known as *fast iterative shrinkage-thresholding algorithm* (FISTA). This scheme is an evolution of [96], which introduced the basic idea for the case of differentiable costs, and consists of the following steps:

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= S_{\lambda, \mu} \left(\mathbf{z}^{(i)} + \mu X^T (\mathbf{y} - X\mathbf{z}^{(i)}) \right), \\ \mathbf{z}^{(i+1)} &:= \boldsymbol{\theta}^{(i)} + \frac{t_i - 1}{t_{i+1}} (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(i-1)}), \end{aligned}$$

where

$$t_{i+1} := \frac{1 + \sqrt{1 + 4t_i^2}}{2},$$

with initial points $t_1 = 1$ and $\mathbf{z}^{(1)} = \boldsymbol{\theta}^{(0)}$. In words, in the thresholding operation, $\boldsymbol{\theta}^{(i-1)}$ is replaced by $\mathbf{z}^{(i)}$, which is a specific linear combination of two successive updates of $\boldsymbol{\theta}$. Hence, at a marginal increase of the computational cost, a substantial increase in convergence speed is achieved.

vectors $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, 3000$, having elements drawn from $\mathcal{N}(0, 1)$. In this way, the online algorithms do not estimate the signal itself, but its sparse wavelet representation, $\boldsymbol{\theta}$. The observations are corrupted by additive white Gaussian noise of variance $\sigma_n^2 = 0.1$. Regarding SpAPSM, the extrapolation parameter μ_n is set equal to $1.8 \times \mathcal{M}_n$, $\omega_i^{(n)}$ are all getting the same value $1/q$, the hyperslabs parameter ϵ was set equal to $1.3\sigma_n$, and $q = 390$. The parameters for all algorithms were selected in order to optimize their performance. Because the sparsity level of the signal may change (from $k = 100$ up to $k = 110$) and because in practice it is not possible to know in advance the exact value of k , we feed the algorithms with an overestimate, k , of the true sparsity value, and in particular we use $\hat{k} = 150$ (i.e., 50% overestimation up to the 1500th iteration).

The results are shown in Fig. 10.11. Note the enhanced performance obtained via the SpAPSM algorithm. However, it has to be pointed out that the complexity of the AdCoSAMP is much lower compared to the other two algorithms, for the choice of $q = 390$ for the SpAPSM. The interesting observation is that SpAPSM achieves a better performance compared to OCCD-TWL, albeit at significantly lower complexity. If on the other hand complexity is of major concern, use of SpAPSM offers the flexibility to use generalized thresholding operators, which lead to improved performance for small values of q , at complexity comparable to that of LMS-based sparsity promoting algorithms [79,80].

10.5 LEARNING SPARSE ANALYSIS MODELS

Our whole discussion so far has been spent in the terrain of signals that are either sparse themselves or that can be sparsely represented in terms of the atoms of a dictionary in a synthesis model, as introduced in (9.16), that is,

$$\mathbf{s} = \sum_{i \in \mathcal{I}} \theta_i \boldsymbol{\psi}_i.$$

As a matter of fact, most of the research activity has been focused on the synthesis model. This may be partly due to the fact that the synthesis modeling path may provide a more intuitively appealing structure to describe the generation of the signal in terms of the elements (atoms) of a dictionary. Recall from Section 9.9 that the sparsity assumption was imposed on $\boldsymbol{\theta}$ in the synthesis model and the corresponding optimization task was formulated in (9.38) and (9.39) for the exact and noisy cases, respectively.

However, this is not the only way to approach the task of sparse modeling. Very early in this chapter, in Section 9.4, we referred to the analysis model

$$\tilde{\mathbf{s}} = \Phi^H \mathbf{s}$$

and pointed out that in a number of real-life applications, the resulting transform $\tilde{\mathbf{s}}$ is sparse. To be fair, the most orthodox way to deal with the underlying model sparsity would be to consider $\|\Phi^H \mathbf{s}\|_0$. Thus, if one wants to estimate \mathbf{s} , a very natural way would be to cast the related optimization task as

$$\begin{aligned} \min_{\mathbf{s}} \quad & \|\Phi^H \mathbf{s}\|_0, \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{Xs}, \text{ or } \|\mathbf{y} - \mathbf{Xs}\|_2^2 \leq \epsilon, \end{aligned} \tag{10.28}$$

depending on whether the measurements via a sensing matrix X are exact or noisy. Strictly speaking, the total variation minimization approach, which was used in Example 10.1, falls under this analysis model formulation umbrella, because what is minimized is the ℓ_1 norm of the gradient transform of the image.

The optimization tasks in either of the two formulations given in (10.28) build around the assumption that the signal of interest has *sparse analysis representation*. The obvious question that is now raised is whether the optimization tasks in (10.28) and their counterparts in (9.38) or (9.39) are any different. One of the first efforts to shed light on this problem was in [51]. There, it was pointed out that the two tasks, though related, are in general different. Moreover, their comparative performance depends on the specific problem at hand. Let us consider, for example, the case where the involved dictionary corresponds to an orthonormal transformation matrix (e.g., DFT). In this case, we already know that the analysis and synthesis matrices are related as

$$\Phi = \Psi = \Psi^{-H},$$

which leads to an equivalence between the two previously stated formulations. Indeed, for such a transform we have

$$\underbrace{\tilde{s}}_{\text{Analysis}} = \underbrace{\Phi^H s}_{\text{Synthesis}} \Leftrightarrow \underbrace{s}_{\text{Synthesis}} = \underbrace{\Phi \tilde{s}}_{\text{Analysis}}.$$

Using the last formula in (10.28), the tasks in (9.38) or (9.39) are readily obtained by replacing θ by s . However, this reasoning cannot be extended to the case of overcomplete dictionaries; in these cases, the two optimization tasks may lead to different solutions.

The previous discussion concerning the comparative performance between the synthesis or analysis-based sparse representations is not only of “philosophical” value. It turns out that often in practice, the nature of certain overcomplete dictionaries does not permit the use of the synthesis-based formulation. These are the cases where the columns of the overcomplete dictionary exhibit a high degree of dependence; that is, the coherence of the matrix, as defined in Section 9.6.1, has large values. Typical examples of such overcomplete dictionaries are the Gabor frames, the curvelet frames, and the oversampled DFT. The use of such dictionaries leads to enhanced performance in a number of applications (e.g., [111,112]). Take as an example the case of our familiar DFT transform. This transform provides a representation of our signal samples in terms of sampled exponential sinusoids, whose integral frequencies are multiples of $\frac{2\pi}{l}$, that is,

$$s := \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{l-1} \end{bmatrix} = \sum_{i=0}^{l-1} \tilde{s}_i \psi_i, \quad (10.29)$$

where, now, \tilde{s}_i are the DFT coefficients and ψ_i is the sampled sinusoid with frequency equal to $\frac{2\pi}{l}i$, that is,

$$\boldsymbol{\psi}_i = \begin{bmatrix} 1 \\ \exp\left(-j\frac{2\pi}{l}i\right) \\ \vdots \\ \exp\left(-j\frac{2\pi}{l}i(l-1)\right) \end{bmatrix}. \quad (10.30)$$

However, this is not necessarily the most efficient representation. For example, it is highly unlikely that a signal comprises only integral frequencies and only such signals can result in a sparse representation using the DFT basis. Most probably, in general, there will be frequencies lying in between the frequency samples of the DFT basis that result in nonsparse representations. Using these extra frequencies, a much better representation of the frequency content of the signal can be obtained. However, in such a dictionary, the atoms are no longer linearly independent, and the coherence of the respective (dictionary) matrix increases.

Once a dictionary exhibits high coherence, then there is no way of finding a sensing matrix X so that $X\Psi$ obeys the RIP. Recall that at the heart of sparsity-aware learning lies the concept of stable embedding, which allows the recovery of a vector/signal after projecting it on a lower-dimensional space, which is what all the available conditions (e.g., RIP) guarantee. However, no stable embedding is possible with highly coherent dictionaries. Take as an extreme example the case where the first and second atoms are identical. Then no sensing matrix X can achieve a signal recovery that distinguishes the vector $[1, 0, \dots, 0]^T$ from $[0, 1, 0, \dots, 0]^T$. Can one then conclude that for highly coherent overcomplete dictionaries, compressed sensing techniques are not possible? Fortunately, the answer to this is negative. After all, our goal in compressed sensing has always been the recovery of the signal $\mathbf{s} = \Psi\boldsymbol{\theta}$ and not the identification of the sparse vector $\boldsymbol{\theta}$ in the synthesis model representation. The latter was just a means to an end. While the unique recovery of $\boldsymbol{\theta}$ cannot be guaranteed for highly coherent dictionaries, this does not necessarily cause any problems for the recovery of \mathbf{s} , using a small set of measurement samples. The escape route will come by considering the analysis model formulation.

10.5.1 COMPRESSED SENSING FOR SPARSE SIGNAL REPRESENTATION IN COHERENT DICTIONARIES

Our goal in this subsection is to establish conditions that guarantee recovery of a signal vector, which accepts a sparse representation in a redundant and coherent dictionary, using a small number of signal-related measurements. Let the dictionary at hand be a tight frame Ψ (see appendix of the chapter, that can be downloaded from the book's website). Then our signal vector is written as

$$\mathbf{s} = \Psi\boldsymbol{\theta}, \quad (10.31)$$

where $\boldsymbol{\theta}$ is assumed to be k -sparse. Recalling the properties of a tight frame, as they are summarized in the appendix, the coefficients in the expansion (10.31) can be written as $\langle \boldsymbol{\psi}_i, \mathbf{s} \rangle$, and the respective vector as

$$\boldsymbol{\theta} = \Psi^T \mathbf{s},$$

because a tight frame is self-dual. Then the analysis counterpart of the synthesis formulation in (9.39) can be cast as

$$\begin{aligned} \min_s \quad & \|\Psi^T s\|_1, \\ \text{s.t.} \quad & \|y - Xs\|_2^2 \leq \epsilon. \end{aligned} \quad (10.32)$$

The goal now is to investigate the accuracy of the recovered solution to this convex optimization task. It turns out that similar strong theorems are also valid for this problem, as with the case of the synthesis formulation, which was studied in Chapter 9.

Definition 10.1. Let Σ_k be the union of all subspaces spanned by all subsets of k columns of Ψ . A sensing matrix X obeys the restricted isometry property adapted to Ψ (Ψ -RIP) with δ_k , if

$$(1 - \delta_k) \|s\|_2^2 \leq \|Xs\|_2^2 \leq (1 + \delta_k) \|s\|_2^2 : \quad \Psi - \text{RIP condition}, \quad (10.33)$$

for all $s \in \Sigma_k$.

The union of subspaces, Σ_k , is the image under Ψ of all k -sparse vectors. This is the difference with the RIP definition given in Section 9.7.2. All the random matrices discussed earlier in this chapter can be shown to satisfy this form of RIP, with overwhelming probability, provided the number of observations, N , is at least of the order of $k \ln(l/k)$. We are now ready to establish the main theorem concerning our ℓ_1 minimization task.

Theorem 10.2. *Let Ψ be an arbitrary tight frame and X a sensing matrix that satisfies the Ψ -RIP with $\delta_{2k} \leq 0.08$, for some positive k . Then the solution, s_* , of the minimization task in (10.32) satisfies the property*

$$\|s - s_*\|_2 \leq C_0 k^{-\frac{1}{2}} \|\Psi^T s - (\Psi^T s)_k\|_1 + C_1 \sqrt{\epsilon}, \quad (10.34)$$

where C_0, C_1 are constants depending on δ_{2k} and $(\Psi^T s)_k$ denotes the best k -sparse approximation of $\Psi^T s$, which results by setting all but the k largest in magnitude components of $\Psi^T s$ equal to zero.

The bound in (10.34) is the counterpart of that given in (9.36). In other words, the previous theorem states that if $\Psi^T s$ decays rapidly, then s can be reconstructed from just a few (compared to the signal length l) observations. The theorem was first given in [25] and it is the first time that such a theorem provides results for the sparse analysis model formulation in a general context.

10.5.2 COSPARSITY

In the sparse synthesis formulation, one searches for a solution in a union of subspaces that are formed by all possible combinations of k columns of the dictionary, Ψ . Our signal vector lies in one of these subspaces—the one that is spanned by the columns of Ψ whose indices lie in the support set (Section 10.2.1). In the sparse analysis approach, things get different. The kick-off point is the sparsity of the transform $\tilde{s} := \Phi^T s$, where Φ defines the transformation matrix or analysis operator. Because \tilde{s} is assumed to be sparse, there exists an index set \mathcal{I} such that $\forall i \in \mathcal{I}, \tilde{s}_i = 0$. In other words, $\forall i \in \mathcal{I}, \phi_i^T s := \langle \phi_i, s \rangle = 0$, where ϕ_i stands for the i th column of Φ . Hence, the subspace in which s resides is

In [92], conditions are derived that guarantee the equivalence of the ℓ_0 and ℓ_1 tasks, in (10.36) and (10.37), respectively; this is done in a way similar to that for the sparse synthesis modeling. Also, in [92], a greedy algorithm inspired by the orthogonal matching pursuit, discussed in Section 10.2.1, has been derived. A thorough study on greedy-like algorithms applicable to the cosparse model can be found in [62]. In [103] an iterative analysis thresholding is proposed and theoretically investigated. Other algorithms that solve the ℓ_1 optimization in the analysis modeling framework can be found in, for example, [21,49,108]. NESTA can also be used for the analysis formulation. Moreover, a critical aspect affecting the performance of algorithms obeying the cosparse analysis model is the choice of the analysis matrix Φ . It turns out that it is not always the best practice to use fixed and predefined matrices. As a promising alternative, problem-tailored analysis matrices can be learned using the available data (e.g., [106,119]).

10.6 A CASE STUDY: TIME-FREQUENCY ANALYSIS

The goal of this section is to demonstrate how all the previously stated theoretical findings can be exploited in the context of a real application. Sparse modeling has been applied to almost everything. So picking up a typical application would not be easy. We preferred to focus on a less “publicized” application, i.e., that of analyzing echolocation signals emitted by bats. However, the analysis will take place within the framework of time-frequency representation, which is one of the research areas that significantly inspired the evolution of compressed sensing theory. Time-frequency analysis of signals has been a field of intense research for a number of decades, and it is one of the most powerful signal processing tools. Typical applications include speech processing, sonar sounding, communications, biological signals, and EEG processing, to name but a few (see, e.g., [13,20,57]).

GABOR TRANSFORM AND FRAMES

It is not our intention to present the theory behind the Gabor transform. Our goal is to outline some basic related notions and use them as a vehicle for the less familiar reader to better understand how redundant dictionaries are used and to get better acquainted with their potential performance benefits.

The Gabor transform was introduced in the mid-1940s by Dennis Gabor (1900–1979), a Hungarian-British engineer. His most notable scientific achievement was the invention of holography, for which he won the Nobel Prize for Physics in 1971.

The discrete version of the Gabor transform can be seen as a special case of the short-time Fourier transform (STFT) (e.g., [57,87]). In the standard DFT transform, the full length of a time sequence, comprising l samples, is used all in “one go” in order to compute the corresponding frequency content. However, the latter can be time-varying, so the DFT will provide an average information, which cannot be of much use. The Gabor transform (and the STFT in general) introduces time localization via the use of a window function, which slides along the signal segment in time, and at each time instant focuses on a different part of the signal. This is a way that allows one to follow the slow time variations which take place in the frequency domain. The time localization in the context of the Gabor transform is achieved via a Gaussian window function, that is,

$$g(n) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{n^2}{2\sigma^2}\right). \quad (10.38)$$

Fig. 10.13A shows the Gaussian window, $g(n - m)$, centered at time instant m . The choice of the window spreading factor, σ , will be discussed later on.

Let us now construct the atoms of the Gabor dictionary. Recall that in the case of the signal representation in terms of the DFT in (10.29), each frequency is represented only once, by the corresponding sampled sinusoid, (10.30). In the Gabor transform, each frequency appears l times; the corresponding sampled sinusoid is multiplied by the Gaussian window sequence, each time shifted by one sample. Thus, at the i th frequency bin, we have l atoms, $\mathbf{g}^{(m,i)}$, $m = 0, 1, \dots, l - 1$, with elements given by

$$g^{(m,i)}(n) = g(n - m)\psi_i(n), \quad n, m, i = 0, 1, \dots, l - 1, \quad (10.39)$$

where $\psi_i(n)$ is the n th element of the vector $\boldsymbol{\psi}_i$ in (10.30). This results in an overcomplete dictionary comprising l^2 atoms in the l -dimensional space. Fig. 10.13B illustrates the effect of multiplying different sinusoids with Gaussian pulses of different spread and at different time delays. Fig. 10.14 is a graphical interpretation of the atoms involved in the Gabor dictionary. Each node (m, i) in this time-frequency plot corresponds to an atom of frequency equal to $\frac{2\pi}{l}i$ and delay equal to m .

Note that the windowing of a signal of finite duration inevitably introduces boundary effects, especially when the delay m gets close to the time segment edges, 0 and $l - 1$. A solution that facilitates the theoretical analysis is to use a modulo l arithmetic to wrap around at the edge points (this is equivalent to extending the signal periodically); see, for example, [113].

Once the atoms have been defined, they can be stacked one next to the other to form the columns of the $l \times l^2$ Gabor dictionary, G . It can be shown that the Gabor dictionary is a tight frame [125].

TIME-FREQUENCY RESOLUTION

By definition of the Gabor dictionary, it is readily understood that the choice of the window spread, as measured by σ , must be a critical factor, since it controls the localization in time. As known from our

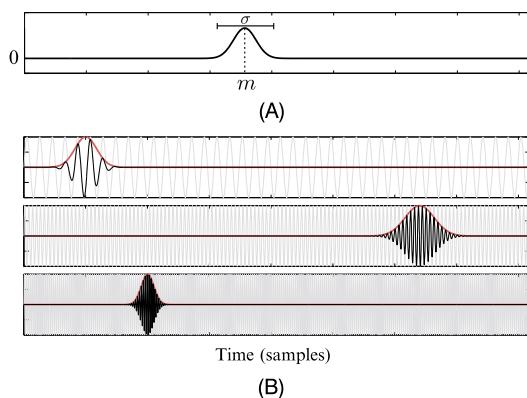
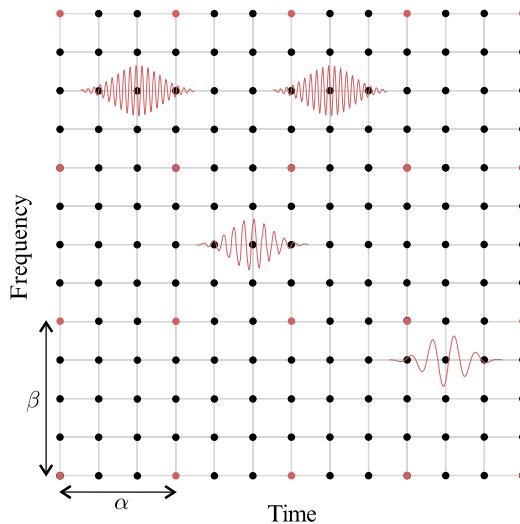


FIGURE 10.13

(A) The Gaussian window with spreading factor σ centered at time instant m . (B) Pulses obtained by windowing three different sinusoids with Gaussian windows of different spread and applied at different time instants.

**FIGURE 10.14**

Each atom of the Gabor dictionary corresponds to a node in the time-frequency grid. That is, it is a sampled windowed sinusoid whose frequency and location in time are given by the coordinates of the respective node. In practice, this grid may be subsampled by factors α and β for the two axes, respectively, in order to reduce the number of the involved atoms.

Fourier transform basics, when the pulse becomes short, in order to increase the time resolution, its corresponding frequency content spreads out, and vice versa. From Heisenberg's principle, we know that we can never achieve high time and frequency resolution simultaneously; one is gained at the expense of the other. It is here where the Gaussian shape in the Gabor transform is justified. It can be shown that the Gaussian window gives the optimal tradeoff between time and frequency resolution [57, 87]. The time-frequency resolution tradeoff is demonstrated in Fig. 10.15, where three sinusoids are shown windowed with different pulse durations. The diagram shows the corresponding spread in the time-frequency plot. The value of σ_t indicates the time spread and σ_f the spread of the respective frequency content around the basic frequency of each sinusoid.

GABOR FRAMES

In practice, l^2 can take large values, and it is desirable to see whether one can reduce the number of the involved atoms without sacrificing the frame-related properties. This can be achieved by an appropriate subsampling, as illustrated in Fig. 10.14. We only keep the atoms that correspond to the red nodes. That is, we subsample by keeping every α nodes in time and every β nodes in frequency in order to form the dictionary, that is,

$$G_{(\alpha,\beta)} = \{\mathbf{g}^{(m\alpha,i\beta)}\}, \quad m = 0, 1, \dots, \frac{l}{\alpha} - 1, \quad i = 0, 1, \dots, \frac{l}{\beta} - 1,$$

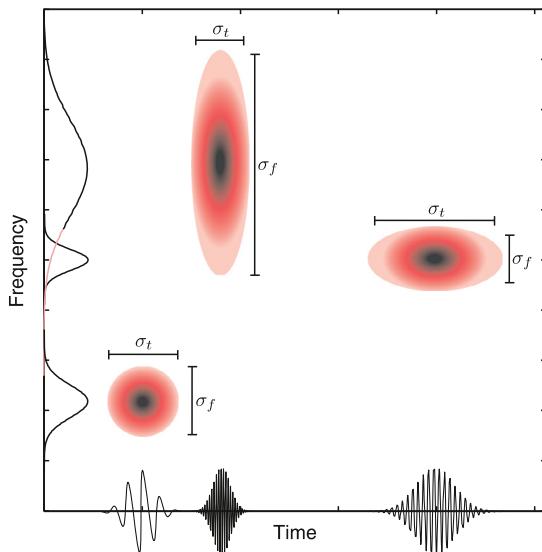


FIGURE 10.15

The shorter the width of the pulsed (windowed) sinusoid is in time, the wider the spread of its frequency content around the frequency of the sinusoid. The Gaussian-like curves along the frequency axis indicate the energy spread in frequency of the respective pulses. The values of σ_t and σ_f indicate the spread in time and frequency, respectively.

where α and β are divisors of l . Then it can be shown (e.g., [57]) that if $\alpha\beta < l$ the resulting dictionary retains its frame properties. Once $G_{(\alpha,\beta)}$ is obtained, the canonical dual frame is readily available via (10.47) (adjusted for complex data), from which the corresponding set of expansion coefficients, θ , results.

TIME-FREQUENCY ANALYSIS OF ECHolocation SIGNALS EMITTED BY BATS

Bats use echolocation for navigation (flying around at night), for prey detection (small insects), and for prey approaching and catching; each bat adaptively changes the shape and frequency content of its calls in order to better serve the previous tasks. Echolocation is used in a similar way for sonars. Bats emit calls as they fly, and “listen” to the returning echoes in order to build up a sonic map of their surroundings. In this way, bats can infer the distance and the size of obstacles as well as of other flying creatures/insects. Moreover, all bats emit special types of calls, called social calls, which are used for socializing, flirting, and so on. The fundamental characteristics of the echolocation calls, for example, the frequency range and average time duration, differ from species to species because, thanks to evolution, bats have adapted their calls in order to become better suited to the environment in which a species operates.

Time-frequency analysis of echolocation calls provides information about the species (species identification) as well as the specific task and behavior of the bats in certain environments. Moreover, the

bat biosonar system is studied in order for humans to learn more about nature and get inspired for subsequent advances in applications such as sonar navigation systems, radars, and medical ultrasonic devices.

Fig. 10.16 shows a case of a recorded echolocation signal from bats. Zooming at two different parts of the signal, we can observe that the frequency is changing with time. In Fig. 10.17, the DFT of the signal is shown, but there is not much information that can be drawn from it except that the signal is compressible in the frequency domain; most of the activity takes place within a short range of frequencies.

Our echolocation signal was a recording of total length $T = 21.845$ ms [75]. Samples were taken at the sampling frequency $f_s = 750$ kHz, which results in a total of $l = 16,384$ samples. Although the signal itself is not sparse in the time domain, we will take advantage of the fact that it is sparse in a transformed domain. We will assume that the signal is sparse in its expansion in terms of the Gabor dictionary.

Our goal in this example is to demonstrate that one does not really need all 16,384 samples to perform time-frequency analysis; all the processing can be carried out using a reduced number of observations, by exploiting the theory of compressed sensing. To form the observations vector, \mathbf{y} , the number of observations was chosen to be $N = 2048$. This amounts to a reduction of eight times with

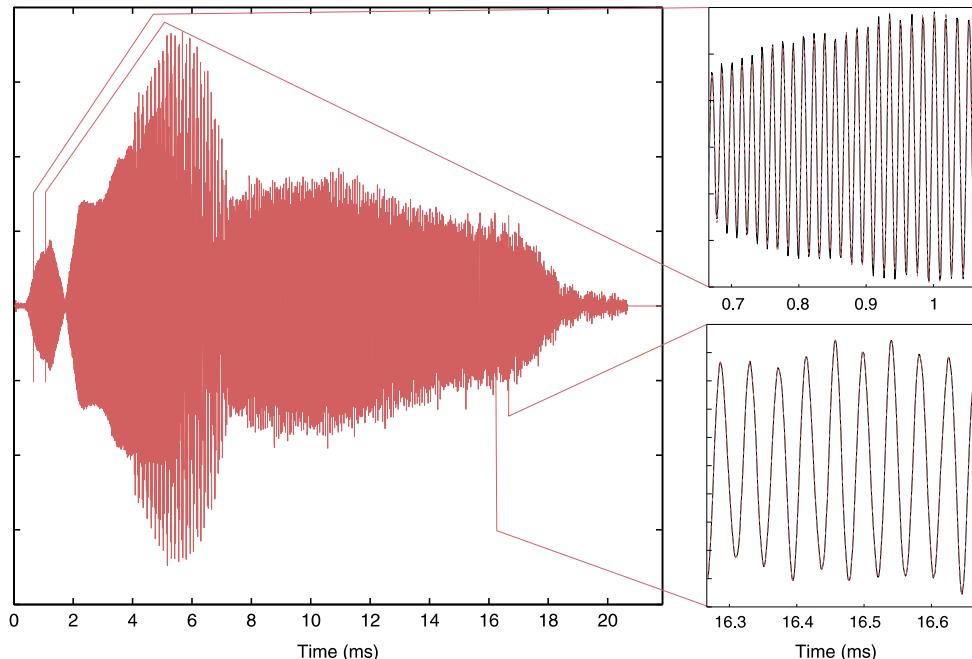


FIGURE 10.16

The recorded echolocation signal. The frequency of the signal is time-varying, which is indicated by focusing on two different parts of the signal.

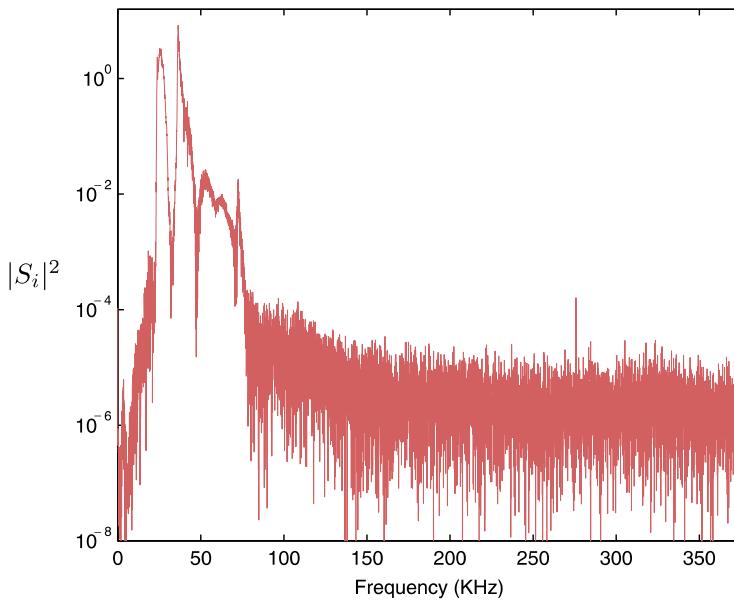


FIGURE 10.17

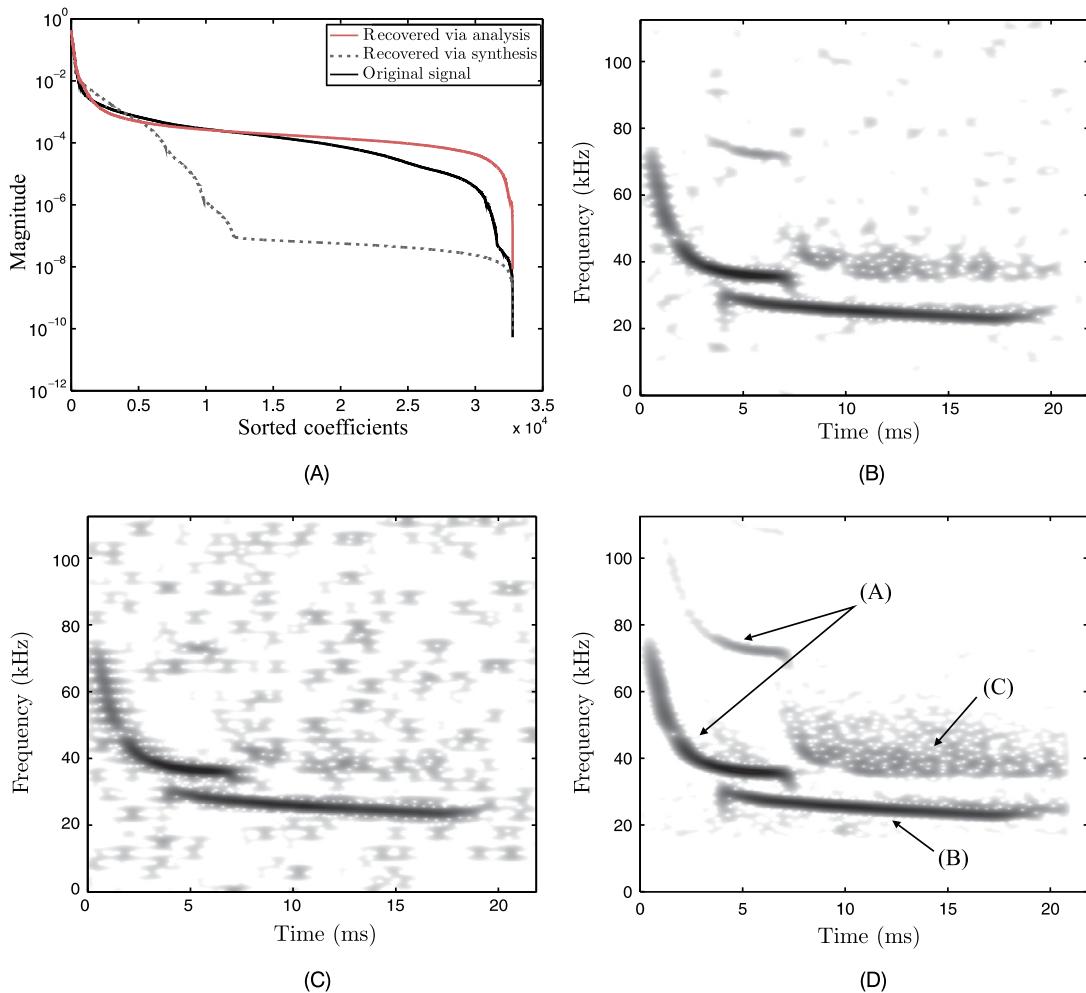
Plot of the energy of the DFT transform coefficients, S_i . Observe that most of the frequency activity takes place within a short frequency range.

respect to the number of available samples. The observations vector was formed as

$$\mathbf{y} = \mathbf{X}\mathbf{s},$$

where \mathbf{X} is an $N \times l$ sensing matrix comprising ± 1 generated in a random way. This means that once we obtain \mathbf{y} , we do not need to store the original samples anymore, leading to a savings in memory requirements. Ideally, one could have obtained the reduced number of observations by sampling directly the analog signal at sub-Nyquist rates, as has already been discussed at the end of Section 9.9. Another goal is to use both the analysis and synthesis models and demonstrate their difference.

Three different spectrograms were computed. Two of them, shown in Fig. 10.18B and C, correspond to the reconstructed signals obtained by the analysis (10.37) and the synthesis (9.37) formulations, respectively. In both cases, the NESTA algorithm was used and the $G_{(128,64)}$ frame was employed. Note that the latter dictionary is redundant by a factor of 2. The spectrograms are the result of plotting the time-frequency grid and coloring each node (t, i) according to the energy $|\theta|^2$ of the coefficient associated with the respective atom in the Gabor dictionary. The full Gabor transform was applied to the reconstructed signals to obtain the spectrograms, in order to get better coverage of the time-frequency grid. The scale is logarithmic and the darker areas correspond to larger values. The spectrogram of the original signal obtained via the full Gabor transform is shown in Fig. 10.18D. It is evident that the analysis model resulted in a more clear spectrogram, which resembles the original one better. When

**FIGURE 10.18**

(A) Plot of the magnitude of the coefficients, sorted in decreasing order, in the expansion in terms of the $G_{(128,64)}$ Gabor frame. The results correspond to the analysis and synthesis model formulations. The third curve corresponds to the case of analyzing the original vector signal directly, by projecting it on the dual frame. (B) The spectrogram from the analysis and (C) the spectrogram from the synthesis formulations. (D) The spectrogram corresponding to the $G_{(64,32)}$ frame using the analysis formulation. For all cases, the number of observations used was one-eighth of the total number of signal samples. A, B, and C indicate different parts of the signal, as explained in the text.

the frame $G_{(64,32)}$ is employed, which is a highly redundant Gabor dictionary comprising $8l$ atoms, then the analysis model results in a recovered signal whose spectrogram is visually indistinguishable from the original one in Fig. 10.18D.

Fig. 10.18A is the plot of the magnitude of the corresponding Gabor transform coefficients, sorted in decreasing values. The synthesis model provides a sparser representation in the sense that the coefficients decrease much faster. The third curve is the one that results if we multiply the dual frame matrix $\tilde{G}_{(128,64)}$ directly with the vector of the original signal samples, and it is shown for comparison reasons.

To conclude, the curious reader may wonder what these curves in Fig. 10.18D mean after all. The call denoted by (A) belongs to a *Pipistrellus pipistrellus* (!) and the call denoted by (B) is either a social call or belongs to a different species. The signal (C) is the return echo from the signal (A). The large spread in time of (C) indicates a highly reflective environment [75].

PROBLEMS

- 10.1** Show that the step in a greedy algorithm that selects the column of the sensing matrix in order to maximize the correlation between the column and the currently available error vector $e^{(i-1)}$ is equivalent to selecting the column that reduces the ℓ_2 norm of the error vector.

Hint: All the parameters obtained in previous steps are fixed, and the optimization is with respect to the new column as well as the corresponding weighting coefficient in the estimate of the parameter vector.

- 10.2** Prove the proposition that if there is a sparse solution to the linear system $\mathbf{y} = X\boldsymbol{\theta}$ such that

$$k_0 = \|\boldsymbol{\theta}\|_0 < \frac{1}{2} \left(1 + \frac{1}{\mu(X)} \right),$$

where $\mu(X)$ is the mutual coherence of X , then the column selection procedure in a greedy algorithm will always select a column among the active columns of X , which correspond to the support of $\boldsymbol{\theta}$; that is, the columns that take part in the representation of \mathbf{y} in terms of the columns of X .

Hint: Assume that

$$\mathbf{y} = \sum_{i=1}^{k_0} \theta_i \mathbf{x}_i^c.$$

- 10.3** Give an explanation to justify why in step 4 of the CoSaMP algorithm the value of t is taken to be equal to $2k$.

- 10.4** Show that if

$$J(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) = \frac{1}{2} \|\mathbf{y} - X\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1 + \frac{1}{2} d(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}),$$

where

$$d(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) := c \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 - \|X\boldsymbol{\theta} - X\tilde{\boldsymbol{\theta}}\|_2^2,$$

then minimization results in

$$\hat{\boldsymbol{\theta}} = S_{\lambda/c} \left(\frac{1}{c} X^T (\mathbf{y} - X\tilde{\boldsymbol{\theta}}) + \tilde{\boldsymbol{\theta}} \right).$$

LEARNING IN REPRODUCING KERNEL HILBERT SPACES

11

CONTENTS

11.1	Introduction	532
11.2	Generalized Linear Models	532
11.3	Volterra, Wiener, and Hammerstein Models	533
11.4	Cover's Theorem: Capacity of a Space in Linear Dichotomies	536
11.5	Reproducing Kernel Hilbert Spaces	539
11.5.1	Some Properties and Theoretical Highlights	541
11.5.2	Examples of Kernel Functions <i>Constructing Kernels</i>	543
	<i>String Kernels</i>	546
		547
11.6	Representer Theorem	548
11.6.1	Semiparametric Representer Theorem	550
11.6.2	Nonparametric Modeling: a Discussion	551
11.7	Kernel Ridge Regression	551
11.8	Support Vector Regression	554
11.8.1	The Linear ϵ -Insensitive Optimal Regression	555
	<i>The Solution</i>	556
	<i>Solving the Optimization Task</i>	557
11.9	Kernel Ridge Regression Revisited	561
11.10	Optimal Margin Classification: Support Vector Machines	562
11.10.1	Linearly Separable Classes: Maximum Margin Classifiers	564
	<i>The Solution</i>	567
	<i>The Optimization Task</i>	568
11.10.2	Nonseparable Classes	569
	<i>The Solution</i>	570
	<i>The Optimization Task</i>	570
11.10.3	Performance of SVMs and Applications	574
11.10.4	Choice of Hyperparameters	574
11.10.5	Multiclass Generalizations	575
11.11	Computational Considerations	576
11.12	Random Fourier Features	577
11.12.1	Online and Distributed Learning in RKHS	579
11.13	Multiple Kernel Learning	580
11.14	Nonparametric Sparsity-Aware Learning: Additive Models	582
11.15	A Case Study: Authorship Identification	584
Problems		587
	MATLAB® Exercises	589
References		590

11.1 INTRODUCTION

Our emphasis in this chapter will be on learning nonlinear models. The necessity of adopting nonlinear models has already been discussed in Chapter 3, in the context of the classification as well as the regression tasks. For example, recall that given two jointly distributed random vectors $(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^k \times \mathbb{R}^l$, we know that the optimal estimate of \mathbf{y} given $\mathbf{x} = \mathbf{x}$ in the mean-square error (MSE) sense is the corresponding conditional mean, that is, $\mathbb{E}[\mathbf{y}|\mathbf{x}]$, which in general is a nonlinear function of \mathbf{x} .

There are different ways of dealing with nonlinear modeling tasks. Our emphasis in this chapter will be on a path through the so-called reproducing kernel Hilbert spaces (RKHS). The technique consists of mapping the input variables to a new space, such that the originally nonlinear task is transformed into a linear one. From a practical point of view, the beauty behind these spaces is that their rich structure allows us to perform inner product operations in a very efficient way, with complexity independent of the dimensionality of the respective RKHS. Moreover, note that the dimension of such spaces can even be infinite.

We start the chapter by reviewing some more “traditional” techniques concerning Volterra series expansions, and then we move slowly to exploring the RKHS concept. Cover’s theorem, the basic properties of RKHSs, and their defining kernels are discussed. Kernel ridge regression and the support vector machine (SVM) framework is presented. Approximation techniques concerning the kernel function and the kernel matrix, such as random Fourier features (RFFs) and the Nyström method, and their implication to online and distributed learning in RKHS are discussed. Finally, some more advanced concepts related to sparsity and multikernel representations are presented. A case study in the context of text mining is presented at the end of the chapter.

11.2 GENERALIZED LINEAR MODELS

Given $(\mathbf{y}, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$, a generalized linear estimator $\hat{\mathbf{y}}$ of \mathbf{y} has the form

$$\hat{\mathbf{y}} = f(\mathbf{x}) := \theta_0 + \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}), \quad (11.1)$$

where ϕ_1, \dots, ϕ_K are *preselected* (nonlinear) functions. A popular family of functions is the polynomial one, for example,

$$\hat{\mathbf{y}} = \theta_0 + \sum_{i=1}^l \theta_i \mathbf{x}_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^l \theta_{im} \mathbf{x}_i \mathbf{x}_m + \sum_{i=1}^l \theta_{ii} \mathbf{x}_i^2. \quad (11.2)$$

Assuming $l = 2$ ($\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]^T$), (11.2) can be brought into the form of (11.1) by setting $K = 5$ and $\phi_1(\mathbf{x}) = \mathbf{x}_1$, $\phi_2(\mathbf{x}) = \mathbf{x}_2$, $\phi_3(\mathbf{x}) = \mathbf{x}_1 \mathbf{x}_2$, $\phi_4(\mathbf{x}) = \mathbf{x}_1^2$, $\phi_5(\mathbf{x}) = \mathbf{x}_2^2$. The generalization of (11.2) to r th-order polynomials is readily obtained and it will contain products of the form $\mathbf{x}_1^{p_1} \mathbf{x}_2^{p_2} \cdots \mathbf{x}_l^{p_l}$, with $p_1 + p_2 + \cdots + p_l \leq r$. It turns out that the number of free parameters, K , for an r th-order polynomial is equal to

$$K = \frac{(l+r)!}{r!l!}.$$

Just to get a feeling for $l = 10$ and $r = 3$, $K = 286$. The use of polynomial expansions is justified by the Weierstrass theorem, stating that every continuous function defined on a compact (closed and bounded) subset $S \subset \mathbb{R}^l$ can be uniformly approximated as closely as desired, with an arbitrarily small error, ϵ , by a polynomial function (for example, [95]). Of course, in order to achieve a good enough approximation, one may have to use a large value of r . Besides polynomial functions, other types of functions can also be used, such as splines and trigonometric functions.

A common characteristic of this type of models is that the basis functions in the expansion are *preselected* and they are fixed and independent of the data. The advantage of such a path is that the associated models are linear with respect to the unknown set of free parameters, and they can be estimated by following any one of the methods described for linear models, presented in Chapters 4–8. However, one has to pay a price for that. As shown in [7], for an expansion involving K fixed functions, the squared approximation error *cannot* be made smaller than order $(\frac{1}{K})^{\frac{2}{l}}$. In other words, for high-dimensional spaces and in order to get a small enough error, one has to use large values of K . This is another face of the curse of dimensionality problem. In contrast, one can get rid of the dependence of the approximation error on the input space dimensionality, l , if the expansion involves data dependent functions, which are optimized with respect to the specific data set. This is, for example, the case for a class of neural networks, to be discussed in Chapter 18. In this case, the price one pays is that the dependence on the free parameters is now nonlinear, making the optimization with regard to the unknown parameters a harder task.

11.3 VOLTERRA, WIENER, AND HAMMERSTEIN MODELS

Let us start with the case of modeling nonlinear systems, where the involved input–output entities are time series/discrete-time signals denoted as (u_n, d_n) , respectively. The counterpart of polynomial modeling in (11.2) is now known as the Volterra series expansion.

These types of models will not be pursued any more in this book, and they are briefly discussed here in order to put the nonlinear modeling task in a more general context as well as for historical reasons. *Thus, this section can be bypassed in a first reading.*



FIGURE 11.1

The nonlinear filter is excited by u_n and provides in its output d_n .

Volterra was an Italian mathematician (1860–1940) who made major contributions to mathematics as well as physics and biology. One of his landmark theories is the development of Volterra series, which is used to solve integral and integro-differential equations. He was one of the Italian professors who refused to take an oath of loyalty to the fascist regime of Mussolini and he was obliged to resign from his university post.

Fig. 11.1 shows an unknown nonlinear system/filter with the respective input–output signals. The output of a discrete-time Volterra model can be written as

(dimensionality, number of free parameters) the same number of free parameters will be required to describe the same objects after the mapping in \mathbb{R}^K . In other words, after the mapping, we embed an l -dimensional manifold in a K -dimensional space in such a way that the data in the two classes become linearly separable.

We have by now fully justified the need for mapping the task from the original low-dimensional space to a higher-dimensional one, via a set of nonlinear functions. However, life is not easy to work in high-dimensional spaces. A large number of parameters are needed; this in turn poses computational complexity problems, and raises issues related to the generalization and overfitting performance of the designed predictors. In the sequel, we will address the former of the two problems by making a “careful” mapping to a higher-dimensional space of a specific structure. The latter problem will be addressed via regularization, as has already been discussed in various parts in previous chapters.

11.5 REPRODUCING KERNEL HILBERT SPACES

Consider a linear space \mathbb{H} of real-valued functions defined on a set² $\mathcal{X} \subseteq \mathbb{R}^l$. Furthermore, suppose that \mathbb{H} is a Hilbert space; that is, it is equipped with an inner product operation, $\langle \cdot, \cdot \rangle_{\mathbb{H}}$, that defines a corresponding norm $\|\cdot\|_{\mathbb{H}}$ and \mathbb{H} is complete with respect to this norm.³ From now on, and for notational simplicity, we omit the subscript \mathbb{H} from the inner product and norm notations, and we are going to use them only if it is necessary to avoid confusion.

Definition 11.1. A Hilbert space \mathbb{H} is called *reproducing kernel Hilbert space* (RKHS) if there exists a function

$$\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$$

with the following properties:

- for every $x \in \mathcal{X}$, $\kappa(\cdot, x)$ belongs to \mathbb{H} ;
- $\kappa(\cdot, \cdot)$ has the so-called *reproducing property*, that is,

$f(x) = \langle f, \kappa(\cdot, x) \rangle, \forall f \in \mathbb{H}, \forall x \in \mathcal{X} : \text{ reproducing property.}$

(11.8)

In words, the kernel function is a function of two arguments. Fixing the value of one of them to, say, $x \in \mathcal{X}$, then the kernel becomes a function of single argument (associated with the \cdot) and this function belongs to \mathbb{H} . The reproducing property means that the value of any function $f \in \mathbb{H}$, at any $x \in \mathcal{X}$, is equal to the respective inner product, performed in \mathbb{H} , between f and $\kappa(\cdot, x)$.

A direct consequence of the reproducing property, if we set $f(\cdot) = \kappa(\cdot, y)$, $y \in \mathcal{X}$, is that

$$\langle \kappa(\cdot, y), \kappa(\cdot, x) \rangle = \kappa(x, y) = \kappa(y, x). \quad (11.9)$$

² Generalization to more general sets is also possible.

³ For the unfamiliar reader, a Hilbert space is the generalization of Euclidean space allowing for infinite dimensions. More rigorous definitions and related properties are given in the appendix of Chapter 8.

Definition 11.2. Let \mathbb{H} be an RKHS, associated with a kernel function $\kappa(\cdot, \cdot)$, and \mathcal{X} a set of elements. Then the mapping

$$\mathcal{X} \ni \mathbf{x} \longmapsto \phi(\mathbf{x}) := \kappa(\cdot, \mathbf{x}) \in \mathbb{H}: \text{ feature map}$$

is known as *feature map* and the space \mathbb{H} as the *feature space*.

In other words, if \mathcal{X} is the set of our observation vectors, the feature mapping maps each vector to the RKHS \mathbb{H} . Note that, in general, \mathbb{H} can be of infinite dimension and its elements are functions. That is, each training point is mapped to a function. In special cases, where \mathbb{H} is of a finite dimension, K , the image can be represented as an equivalent vector $\phi(\mathbf{x}) \in \mathbb{R}^K$. From now on, the general infinite-dimensional case will be treated and the images will be denoted as functions, $\phi(\cdot)$.

Let us now see what we have gained by choosing to perform the feature mapping from the original space to a high-dimensional RKHS one. Let $\mathbf{x}, \mathbf{y} \in \mathcal{X} \subseteq \mathbb{R}^l$. Then the inner product of the respective mapping images is written as

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \langle \kappa(\cdot, \mathbf{x}), \kappa(\cdot, \mathbf{y}) \rangle,$$

or

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \kappa(\mathbf{x}, \mathbf{y}): \text{ kernel trick.}$$

In other words, employing this type of mapping to our problem, we can perform inner product operations in \mathbb{H} in a very efficient way; that is, via a function evaluation performed in the original low-dimensional space! This property is also known as the *kernel trick*, and it facilitates significantly the computations. As will become apparent soon, the way this property is exploited in practice involves the following steps:

1. Map (implicitly) the input training data to an RKHS

$$\mathbf{x}_n \longmapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

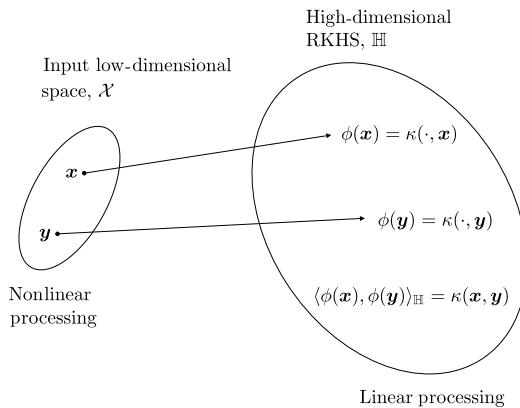
2. Solve a *linear* estimation task in \mathbb{H} , involving the images $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.
3. Cast the algorithm that solves for the unknown parameters in terms of inner product operations in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

4. Replace each inner product by a kernel evaluation, that is,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

It is apparent that one does not need to perform any explicit mapping of the data. All is needed is to perform the kernel operations at the final step. Note that the specific form of $\kappa(\cdot, \cdot)$ does not concern the analysis. Once the algorithm for the prediction, \hat{y} , has been derived, one can use different choices for $\kappa(\cdot, \cdot)$. As we will see, different choices for $\kappa(\cdot, \cdot)$ correspond to different types of nonlinearity. Fig. 11.6 illustrates the rationale behind the procedure. In practice, the four steps listed above are equivalent to (a) work in the original (low-dimensional Euclidean space) and express all operations in terms of inner products and (b) at the final step substitute the inner products with kernel evaluations.

**FIGURE 11.6**

The nonlinear task in the original low-dimensional space is mapped to a linear one in the high-dimensional RKHS \mathbb{H} . Using feature mapping, inner product operations are efficiently performed via kernel evaluations in the original low-dimensional spaces.

Example 11.1. The goal of this example is to demonstrate that one can map the input space into another of higher dimension (finite-dimensional for this case⁴), where the corresponding inner product can be computed as a function performed in the lower-dimensional original one. Consider the case of the two-dimensional space and the mapping into a three-dimensional one, i.e.,

$$\mathbb{R}^2 \ni \mathbf{x} \longmapsto \phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \in \mathbb{R}^3.$$

Then, given two vectors $\mathbf{x} = [x_1, x_2]^T$ and $\mathbf{y} = [y_1, y_2]^T$, it is straightforward to see that

$$\phi^T(\mathbf{x})\phi(\mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2.$$

That is, the inner product in the three-dimensional space, after the mapping, is given in terms of a function of the variables in the original space.

11.5.1 SOME PROPERTIES AND THEORETICAL HIGHLIGHTS

The reader who has no “mathematical anxieties” can bypass this subsection during a first reading.

Let \mathcal{X} be a set of points. Typically \mathcal{X} is a compact (closed and bounded) subset of \mathbb{R}^l . Consider a function

$$\kappa : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R}.$$

⁴ If a space of functions has finite dimension, then it is equivalent (isometrically isomorphic) to a finite Euclidean linear/vector space.

Definition 11.3. The function κ is called a *positive definite kernel* if

$$\boxed{\sum_{n=1}^N \sum_{m=1}^N a_n a_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \geq 0 : \text{ positive definite kernel},} \quad (11.10)$$

for any real numbers, a_n, a_m , any points $\mathbf{x}_n, \mathbf{x}_m \in \mathcal{X}$, and any $N \in \mathbb{N}$.

Note that (11.10) can be written in an equivalent form. Define the so-called *kernel matrix* \mathcal{K} of order N ,

$$\mathcal{K} := \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \quad (11.11)$$

Then (11.10) is written as

$$\mathbf{a}^T \mathcal{K} \mathbf{a} \geq 0, \quad (11.12)$$

where

$$\mathbf{a} = [a_1, \dots, a_N]^T.$$

Because (11.10) is true for any $\mathbf{a} \in \mathbb{R}^N$, (11.12) suggests that for a kernel to be positive definite, it suffices for the corresponding kernel matrix to be positive semidefinite.⁵

Lemma 11.1. *The reproducing kernel associated with an RKHS \mathbb{H} is a positive definite kernel.*

The proof of the lemma is given in Problem 11.3. Note that the opposite is also true. It can be shown [82,106] that if $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a positive definite kernel, then there exists an RKHS \mathbb{H} of functions on \mathcal{X} such that $\kappa(\cdot, \cdot)$ is a reproducing kernel of \mathbb{H} . This establishes the equivalence between reproducing and positive definite kernels. Historically, the theory of positive definite kernels was developed first in the context of integral equations by Mercer [76], and the connection to RKHS was developed later on (see, for example, [3]).

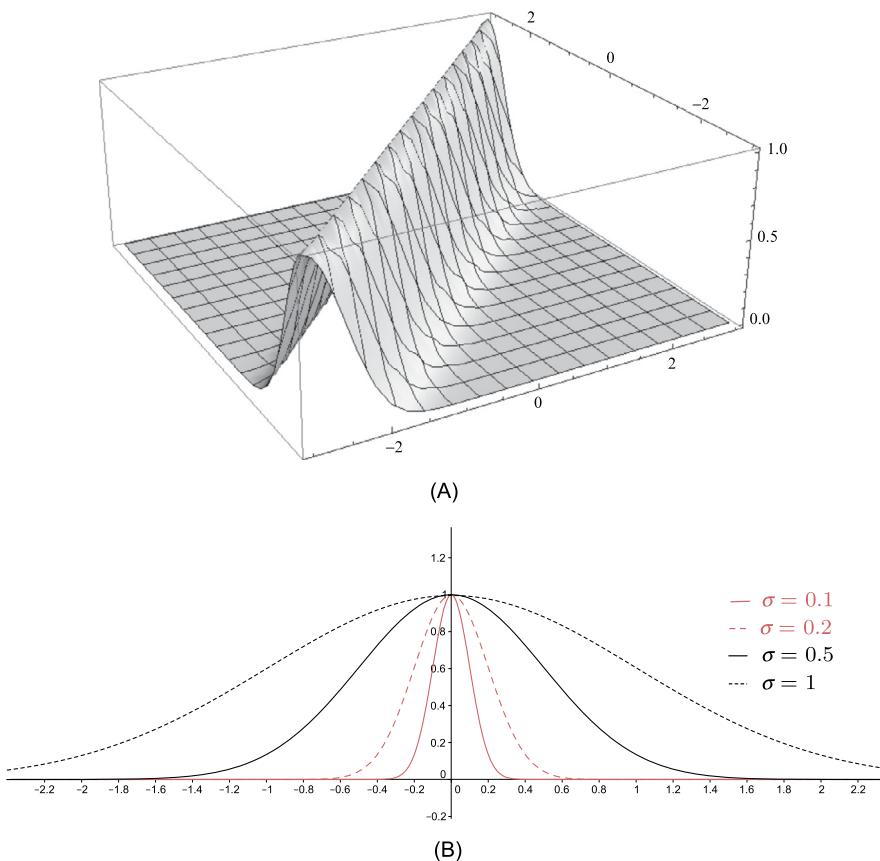
Lemma 11.2. *Let \mathbb{H} be an RKHS on the set \mathcal{X} with reproducing kernel $\kappa(\cdot, \cdot)$. Then the linear span of the function $\kappa(\cdot, \mathbf{x})$, $\mathbf{x} \in \mathcal{X}$, is dense in \mathbb{H} , that is,*

$$\mathbb{H} = \overline{\text{span}\{\kappa(\cdot, \mathbf{x}), \mathbf{x} \in \mathcal{X}\}}. \quad (11.13)$$

The proof of the lemma is given in Problem 11.4. The overbar denotes the closure of a set. In other words, \mathbb{H} can be constructed by *all* possible linear combinations of the kernel function computed in \mathcal{X} , as well as the *limit points* of sequences of such combinations. Simply stated, \mathbb{H} can be *fully generated from the knowledge* of $\kappa(\cdot, \cdot)$.

The interested reader can obtain more theoretical results concerning RKHSs from, for example, [64,83,87,103,106,108].

⁵ It may be slightly confusing that the definition of a positive definite kernel requires a positive semidefinite kernel matrix. However, this is what has been the accepted definition.

**FIGURE 11.7**

(A) The Gaussian kernel for $\mathcal{X} = \mathbb{R}$, $\sigma = 0.5$. (B) The function $\kappa(\cdot, 0)$ for different values of σ .

11.5.2 EXAMPLES OF KERNEL FUNCTIONS

In this subsection, we present some typical examples of kernel functions, which are commonly used in various applications.

- The *Gaussian* kernel is among the most popular ones, and it is given by our familiar form

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

with $\sigma > 0$ being a parameter. Fig. 11.7A shows the Gaussian kernel as a function of $x, y \in \mathcal{X} = \mathbb{R}^2$ and $\sigma = 0.5$. Fig. 11.7B shows the graph of resulting function if we set one of the kernel's arguments to 0, i.e., $\kappa(\cdot, 0)$, for various values of σ .

The dimension of the RKHS generated by the Gaussian kernel is *infinite*. A proof that the Gaussian kernel satisfies the required properties can be obtained, for example, from [108].

- The *homogeneous polynomial* kernel has the form

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^r,$$

where r is a parameter.

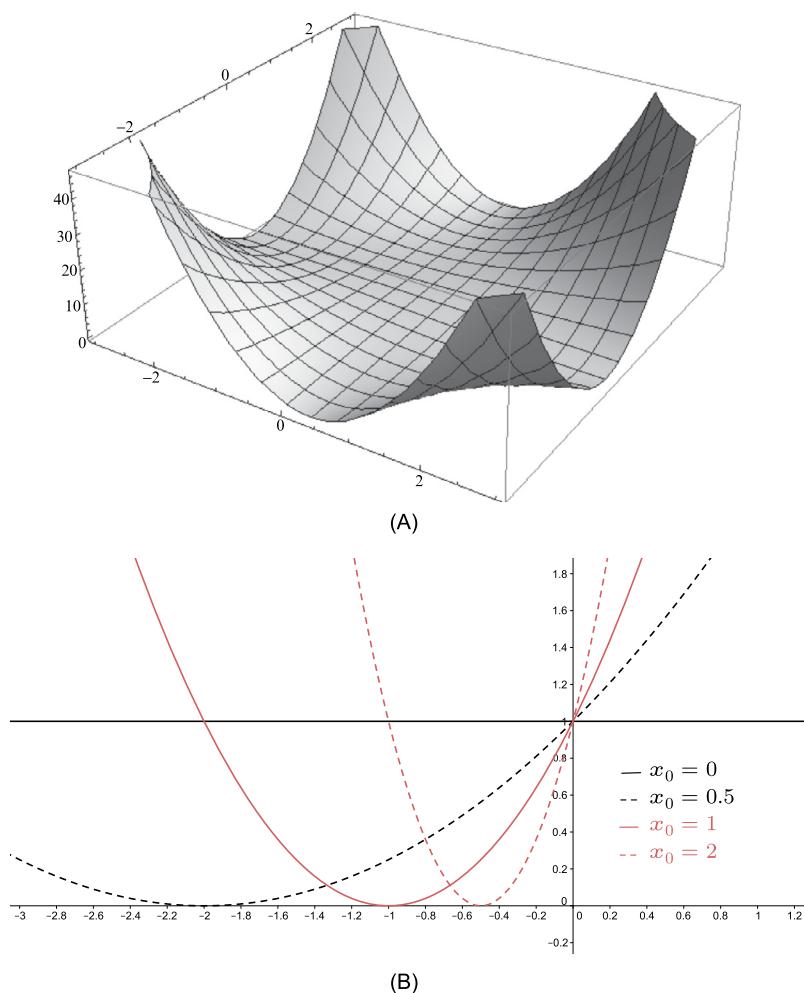


FIGURE 11.8

(A) The inhomogeneous polynomial kernel for $\mathcal{X} = \mathbb{R}$, $r = 2$. (B) The graph of $\kappa(\cdot, x_0)$ for different values of x_0 .

- The *inhomogeneous* polynomial kernel is given by

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^r,$$

where $c \geq 0$ and $r > 0$, $r \in \mathbb{N}$, are parameters. The graph of the kernel is given in Fig. 11.8A. In Fig. 11.8B the graphs of the function $\phi(\cdot, x_0)$ are shown for different values of x_0 . The dimensionality of the RKHS associated with polynomial kernels is finite.

- The *Laplacian* kernel is given by

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp(-t \|\mathbf{x} - \mathbf{y}\|),$$

where $t > 0$ is a parameter. The dimensionality of the RKHS associated with the Laplacian kernel is infinite.

- The *spline* kernels are defined as

$$\kappa(\mathbf{x}, \mathbf{y}) = B_{2p+1}(\|\mathbf{x} - \mathbf{y}\|^2),$$

where the B_n spline is defined via the $n + 1$ convolutions of the unit interval $[-\frac{1}{2}, \frac{1}{2}]$, that is,

$$B_n(\cdot) := \bigotimes_{i=1}^{n+1} \chi_{[-\frac{1}{2}, \frac{1}{2}]}(\cdot),$$

and $\chi_{[-\frac{1}{2}, \frac{1}{2}]}(\cdot)$ is the characteristic function on the respective interval.⁶

- The *sampling function* or *sinc* kernel is of particular interest from a signal processing point of view. This kernel function is defined as

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}.$$

Recall that we have met this function in Chapter 9 while discussing sub-Nyquist sampling. Let us now consider the set of all squared integrable functions, which are *bandlimited*, that is,

$$\mathcal{F}_B = \left\{ f : \int_{-\infty}^{+\infty} |f(x)|^2 dx < +\infty, \text{ and } |F(\omega)| = 0, |\omega| > \pi \right\},$$

where $F(\omega)$ is the respective Fourier transform

$$F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(x) e^{-j\omega x} dx.$$

It turns out that \mathcal{F}_B is an RKHS whose reproducing kernel is the sinc function (e.g., [51]), that is,

$$\kappa(x, y) = \text{sinc}(x - y).$$

⁶ It is equal to one if the variable belongs to the interval and zero otherwise.

This takes us back to the classical sampling theorem through the RKHS route. Without going into details, a by-product of this view is the Shannon sampling theorem; any bandlimited function can be written as⁷

$$f(x) = \sum_n f(n) \operatorname{sinc}(x - n). \quad (11.14)$$

Constructing Kernels

Besides the previous examples, one can construct more kernels by applying the following properties (Problem 11.6, [106]):

- If

$$\begin{aligned} \kappa_1(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{X} &\mapsto \mathbb{R}, \\ \kappa_2(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{X} &\mapsto \mathbb{R} \end{aligned}$$

are kernels, then

$$\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y}) + \kappa_2(\mathbf{x}, \mathbf{y})$$

and

$$\kappa(\mathbf{x}, \mathbf{y}) = \alpha \kappa_1(\mathbf{x}, \mathbf{y}), \quad \alpha > 0,$$

and

$$\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y}) \kappa_2(\mathbf{x}, \mathbf{y})$$

are also kernels.

- Let

$$f : \mathcal{X} \mapsto \mathbb{R}.$$

Then

$$\kappa(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) f(\mathbf{y})$$

is a kernel.

- Let a function

$$g : \mathcal{X} \mapsto \mathbb{R}^l$$

and a kernel function

$$\kappa_1(\cdot, \cdot) : \mathbb{R}^l \times \mathbb{R}^l \mapsto \mathbb{R}.$$

Then

$$\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(g(\mathbf{x}), g(\mathbf{y}))$$

⁷ The key point behind the proof is that in \mathcal{F}_B , the kernel $\kappa(x, y)$ can be decomposed in terms of a set of orthogonal functions, that is, $\operatorname{sinc}(x - n)$, $n = 0, \pm 1, \pm 2, \dots$.

is also a kernel.

- Let A be a positive definite $l \times l$ matrix. Then

$$\kappa(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A \mathbf{y}$$

is a kernel.

- If

$$\kappa_1(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R},$$

then

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp(\kappa_1(\mathbf{x}, \mathbf{y}))$$

is also a kernel, and if $p(\cdot)$ is a polynomial with nonnegative coefficients,

$$\kappa(\mathbf{x}, \mathbf{y}) = p(\kappa_1(\mathbf{x}, \mathbf{y}))$$

is also a kernel.

The interested reader will find more information concerning kernels and their construction in, for example, [52, 106, 108].

String Kernels

So far, our discussion has been focused on input data that were vectors in a Euclidean space. However, as we have already pointed out, the input data need not necessarily be vectors, and they can be elements of more general sets.

Let us denote by \mathcal{S} an alphabet set; that is, a set with a finite number of elements, which we call *symbols*. For example, this can be the set of all capital letters in the Latin alphabet. Bypassing the path of formal definitions, a *string* is a finite sequence of any length of symbols from \mathcal{S} . For example, two cases of strings are

$$T_1 = \text{"MYNAMEISSERGIOS"}, \quad T_2 = \text{"HERNAMEISDESPONIA"}.$$

In a number of applications, such as in text mining, spam filtering, text summarization, and bioinformatics, it is important to quantify how “similar” two strings are. However, kernels, by their definition, are similarity measures; they are constructed so as to express inner products in the high-dimensional feature space. An inner product is a similarity measure. Two vectors are most similar if they point to the same direction. Starting from this observation, there has been a lot of activity on defining kernels that measure similarity between strings. Without going into details, let us give such an example.

Let us denote by \mathcal{S}^* the set of all possible strings that can be constructed using symbols from \mathcal{S} . Also, a string s is said to be a *substring* of x if $x = bsa$, where a and b are other strings (possibly empty) from the symbols of \mathcal{S} . Given two strings $x, y \in \mathcal{S}^*$, define

$$\kappa(x, y) := \sum_{s \in \mathcal{S}^*} w_s \phi_s(x) \phi_s(y), \tag{11.15}$$

where $w_s \geq 0$, and $\phi_s(x)$ is the number of times substring s appears in x . It turns out that this is indeed a kernel, in the sense that it complies with (11.10); such kernels constructed from strings are known as *string kernels*.

Obviously, a number of different variants of this kernel are available. The so-called k -*spectrum* kernel considers common substrings only of length k . For example, for the two strings given before, the value of the 6-spectrum string kernel in (11.15) is equal to one (one common substring of length 6 is identified and appears once in each one of the two strings: “NAMEIS”). The interested reader can obtain more on this topic from, for example, [106]. We will use the notion of the string kernel in the case study in Section 11.15.

11.6 REPRESENTER THEOREM

The theorem to be stated in this section is of major importance from a practical point of view. It allows us to perform *empirical* loss function optimization, based on a finite set of training samples, in a very efficient way even if the function to be estimated belongs to a very high (even infinite)-dimensional RKHS \mathbb{H} .

Theorem 11.2. *Let*

$$\Omega : [0, +\infty) \mapsto \mathbb{R}$$

be an arbitrary strictly monotonic increasing function. Let also

$$\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$$

be an arbitrary loss function. Then each minimizer $f \in \mathbb{H}$ of the regularized minimization task

$$\min_{f \in \mathbb{H}} J(f) := \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \Omega(\|f\|^2) \quad (11.16)$$

admits a representation of the form⁸

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n),$$

(11.17)

where $\theta_n \in \mathbb{R}$, $n = 1, 2, \dots, N$.

⁸ The property holds also for regularization of the form $\Omega(\|f\|)$, since the quadratic function is strictly monotonic on $[0, \infty)$, and the proof follows a similar line.

Proof. The linear span, $A := \text{span}\{\kappa(\cdot, \mathbf{x}_1), \dots, \kappa(\cdot, \mathbf{x}_N)\}$, forms a closed subspace. Then each $f \in \mathbb{H}$ can be decomposed into two parts (see (8.20)), i.e.,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + f_{\perp},$$

where f_{\perp} is the part of f that is orthogonal to A . From the reproducing property, we obtain

$$\begin{aligned} f(\mathbf{x}_m) &= \langle f, \kappa(\cdot, \mathbf{x}_m) \rangle = \left\langle \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \kappa(\cdot, \mathbf{x}_m) \right\rangle \\ &= \sum_{n=1}^N \theta_n \kappa(\mathbf{x}_m, \mathbf{x}_n), \end{aligned}$$

where we used the fact that $\langle f_{\perp}, \kappa(\cdot, \mathbf{x}_n) \rangle = 0$, $n = 1, 2, \dots, N$. In other words, the expansion in (11.17) guarantees that at the training points, the value of f does not depend on f_{\perp} . Hence, the first term in (11.16), corresponding to the empirical loss, does *not* depend on f_{\perp} . Moreover, for all f_{\perp} we have

$$\begin{aligned} \Omega(\|f\|^2) &= \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2 + \|f_{\perp}\|^2\right) \\ &\geq \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2\right). \end{aligned}$$

Thus, for *any* choice of θ_n , $n = 1, 2, \dots, N$, the cost function in (11.16) is minimized for $f_{\perp} = 0$. Thus, the claim is proved. \square

The theorem was first shown in [61]. In [2], the conditions under which the theorem exists were investigated and related sufficient and necessary conditions were derived. The importance of this theorem is that in order to optimize (11.16) with respect to f , one uses the expansion in (11.17) and minimization is carried out with respect to the *finite* set of parameters θ_n , $n = 1, 2, \dots, N$.

Note that when working in high/infinite-dimensional spaces, the presence of a regularizer can hardly be avoided; otherwise, the obtained solution will suffer from overfitting, as only a finite number of data samples are used for training. The effect of regularization on the generalization performance and stability of the associated solution has been studied in a number of classical papers (for example, [18, 39, 79]).

Usually, a bias term is often added and it is assumed that the minimizing function admits the following representation:

$$\tilde{f} = f + b, \tag{11.18}$$

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n). \tag{11.19}$$

In practice, the use of a bias term (which does not enter in the regularization) turns out to improve performance. First, it enlarges the class of functions in which the solution is searched and potentially leads to better performance. Moreover, due to the penalization imposed by the regularizing term, $\Omega(\|f\|^2)$, the minimizer pushes the values which the function takes at the training points to smaller values. The existence of b tries to “absorb” some of this action (see, for example, [108]).

Remarks 11.2.

- We will use the expansion in (11.17) in a number of cases. However, it is interesting to apply this expansion to the RKHS of the bandlimited functions and see what comes out. Assume that the available samples from a function f are $f(n)$, $n = 1, 2, \dots, N$ (assuming the case of normalized sampling period $x_s = 1$). Then according to the representer theorem, we can write the following approximation:

$$f(x) \approx \sum_{n=1}^N \theta_n \operatorname{sinc}(x - n). \quad (11.20)$$

Taking into account the orthonormality of the $\operatorname{sinc}(\cdot - n)$ functions, we get $\theta_n = f(n)$, $n = 1, 2, \dots, N$. However, note that in contrast to (11.14), which is exact, (11.20) is only an approximation. On the other hand, (11.20) can be used even if the obtained samples are contaminated by noise.

11.6.1 SEMIPARAMETRIC REPRESENTER THEOREM

The use of the bias term is also theoretically justified by the generalization of the representer theorem [103]. The essence of this theorem is to expand the solution into two parts, i.e., one that lies in an RKHS \mathbb{H} , and another that is given as a linear combination of a set of preselected functions.

Theorem 11.3. *Let us assume that in addition to the assumptions adopted in Theorem 11.2, we are given the set of real-valued functions*

$$\psi_m : \mathcal{X} \mapsto \mathbb{R}, \quad m = 1, 2, \dots, M,$$

with the property that the $N \times M$ matrix with elements $\psi_m(\mathbf{x}_n)$, $n = 1, 2, \dots, N$, $m = 1, 2, \dots, M$, has rank M . Then any

$$\tilde{f} = f + h, \quad f \in \mathbb{H}, \quad h \in \operatorname{span}\{\psi_m, m = 1, 2, \dots, M\},$$

solving the minimization task

$$\min_{\tilde{f}} J(\tilde{f}) := \sum_{n=1}^N \mathcal{L}(y_n, \tilde{f}(\mathbf{x}_n)) + \Omega(\|\tilde{f}\|^2) \quad (11.21)$$

admits the following representation:

$$\tilde{f}(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + \sum_{m=1}^M b_m \psi_m(\cdot).$$

(11.22)

Obviously, the use of a bias term is a special case of the expansion above. An example of successful application of this theorem was demonstrated in [13] in the context of image denoising. A set of nonlinear functions in place of ψ_m were used to account for the edges (nonsmooth jumps) in an image. The part of f lying in the RKHS accounted for the smooth parts in the image.

11.6.2 NONPARAMETRIC MODELING: A DISCUSSION

Note that searching a model function in an RKHS space is a typical task of *nonparametric* modeling. In contrast to the parametric modeling in Eq. (11.1), where the unknown function is *parameterized* in terms of a set of basis functions, the minimization in (11.16) or (11.21) is performed with regard to functions that are *constrained to belong in a specific space*. In the more general case, minimization could be performed with regard to any (continuous) function, for example,

$$\min_f \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \phi(f),$$

where $\mathcal{L}(\cdot, \cdot)$ can be any loss function and ϕ an appropriately chosen regularizing functional. Note, however, that in this case, the presence of the regularization is crucial. If there is no regularization, then any function that *interpolates* the data is a solution; such techniques have also been used in interpolation theory (for example, [78,90]). The regularization term, $\phi(f)$, helps to smoothen out the function to be recovered. To this end, functions of derivatives have been employed. For example, if the minimization cost is chosen as

$$\sum_{n=1}^N (y_n - f(x_n))^2 + \lambda \int (f''(x))^2 dx,$$

then the solution is a cubic spline; that is, a piecewise cubic function with knots at the points x_n , $n = 1, 2, \dots, N$, and it is continuously differentiable to the second order. The choice of λ controls the degree of smoothness of the approximating function; the larger its value, the smoother the minimizer becomes.

If, on the other hand, f is constrained to lie in an RKHS and the minimizing task is as in (11.16), then the resulting function is of the form given in (11.17), where a kernel function is placed at each input training point. It must be pointed out that the parametric form that now results was not in our original intentions. It came out as a by-product of the theory. However, it should be stressed that, in contrast to the parametric methods, now the number of parameters to be estimated is not fixed but it depends on the number of the training points. Recall that this is an important difference and it was carefully pointed out when parametric methods were introduced and defined in Chapter 3.

11.7 KERNEL RIDGE REGRESSION

Ridge regression was introduced in Chapter 3 and it has also been treated in more detail in Chapter 6. Here, we will state the task in a general RKHS. The path to be followed is the typical one used to extend techniques which have been developed for linear models to the more general RKHS.

We assume that the generation mechanism of the data, represented by the training set $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, is modeled via a nonlinear regression task

$$y_n = g(\mathbf{x}_n) + \eta_n, \quad n = 1, 2, \dots, N. \quad (11.23)$$

Let us denote by f the estimate of the unknown g . Sometimes, f is called the *hypothesis* and the space \mathbb{H} in which f is searched is known as the *hypothesis space*. We will further assume that f lies in an RKHS, associated with a kernel

$$\kappa : \mathbb{R}^l \times \mathbb{R}^l \mapsto \mathbb{R}.$$

Motivated by the representer theorem, we adopt the following expansion:

$$f(\mathbf{x}) = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}, \mathbf{x}_n).$$

According to the kernel ridge regression approach, the unknown coefficients are estimated by the following task:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \\ J(\boldsymbol{\theta}) &:= \sum_{n=1}^N \left(y_n - \sum_{m=1}^N \theta_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \right)^2 + C \langle f, f \rangle, \end{aligned} \quad (11.24)$$

where C is the regularization parameter.⁹ Eq. (11.24) can be rewritten as (Problem 11.7)

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathcal{K}\boldsymbol{\theta})^T (\mathbf{y} - \mathcal{K}\boldsymbol{\theta}) + C\boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta}, \quad (11.25)$$

where

$$\mathbf{y} = [y_1, \dots, y_N]^T, \quad \boldsymbol{\theta} = [\theta_1, \dots, \theta_N]^T,$$

and \mathcal{K} is the kernel matrix defined in (11.11); the latter is fully determined by the kernel function and the training points. Following our familiar-by-now arguments, minimization of $J(\boldsymbol{\theta})$ with regard to $\boldsymbol{\theta}$ leads to

$$(\mathcal{K}^T \mathcal{K} + C\mathcal{K}^T) \hat{\boldsymbol{\theta}} = \mathcal{K}^T \mathbf{y}$$

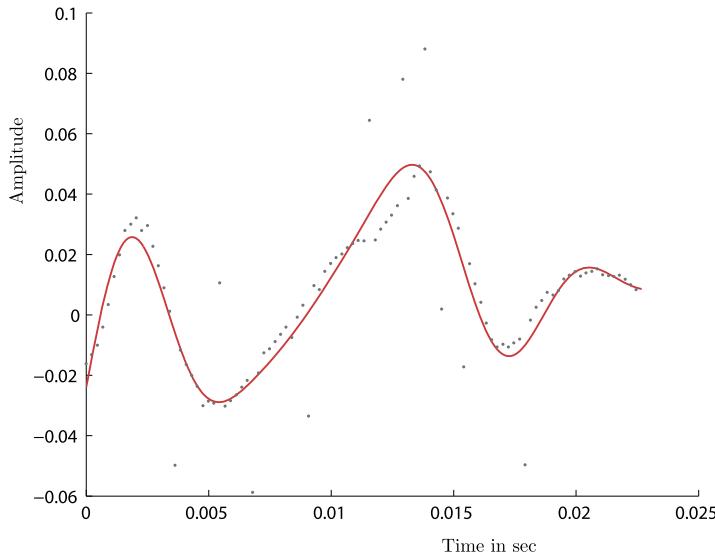
or

$$(\mathcal{K} + CI) \hat{\boldsymbol{\theta}} = \mathbf{y} : \quad \text{kernel ridge regression,} \quad (11.26)$$

where $\mathcal{K}^T = \mathcal{K}$ has been assumed to be invertible.¹⁰ Once $\hat{\boldsymbol{\theta}}$ has been obtained, given an unknown

⁹ For the needs of this chapter, we denote the regularization constant as C , not to be confused with the Lagrange multipliers, to be introduced soon.

¹⁰ This is true, for example, for the Gaussian kernel [103].

**FIGURE 11.9**

Plot of the data used for training together with the fitted (prediction) curve obtained via the kernel ridge regression, for Example 11.2. The Gaussian kernel was used.

vector $\mathbf{x} \in \mathbb{R}^l$, the corresponding prediction value of the dependent variable is given by

$$\hat{y} = \sum_{n=1}^N \hat{\theta}_n \kappa(\mathbf{x}, \mathbf{x}_n) = \hat{\boldsymbol{\theta}}^T \boldsymbol{\kappa}(\mathbf{x}),$$

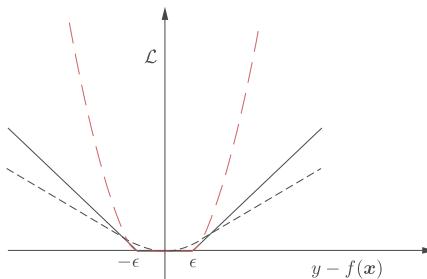
where

$$\boldsymbol{\kappa}(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \dots, \kappa(\mathbf{x}, \mathbf{x}_N)]^T.$$

Employing (11.26), we obtain

$$\boxed{\hat{y}(\mathbf{x}) = \mathbf{y}^T (\mathcal{K} + CI)^{-1} \boldsymbol{\kappa}(\mathbf{x}).} \quad (11.27)$$

Example 11.2. In this example, the prediction power of the kernel ridge regression in the presence of noise and outliers will be tested. The original data were samples from a music recording of *Blade Runner* by Vangelis Papathanasiou. A white Gaussian noise was then added at a 15 dB level and a number of outliers were intentionally randomly introduced and “hit” some of the values (10% of them). The kernel ridge regression method was used, employing the Gaussian kernel with $\sigma = 0.004$. We allowed for a bias term to be present (see Problem 11.8). The prediction (fitted) curve, $\hat{y}(x)$, for various values of x is shown in Fig. 11.9 together with the (noisy) data used for training.

**FIGURE 11.10**

The Huber loss function (dotted-gray), the linear ϵ -insensitive (full-gray), and the quadratic ϵ -insensitive (red) loss functions, for $\epsilon = 0.7$.

11.8 SUPPORT VECTOR REGRESSION

The squared error loss function, associated with the LS method, is not always the best criterion for optimization, in spite of its merits. In the case of the presence of a non-Gaussian noise with long tails (i.e., high probability for relatively large values) and, hence, with an increased number of noise *outliers*, the square dependence on the error of the squared error criterion gets biased toward values associated with the presence of outliers. Recall from Chapter 3 that the LS method is equivalent to the maximum likelihood estimation under the assumption of white Gaussian noise. Moreover, under this assumption, the LS estimator achieves the Cramér–Rao bound and it becomes a minimum variance estimator. However, under other noise scenarios, one has to look for alternative criteria.

The task of optimization in the presence of outliers was studied by Huber [54], whose goal was to obtain a strategy for choosing the loss function that “matches” the noise model best. He proved that under the assumption that the noise has a symmetric probability density function (PDF), the optimal minimax strategy¹¹ for regression is obtained via the following loss function:

$$\mathcal{L}(y, f(\mathbf{x})) = |y - f(\mathbf{x})|,$$

which gives rise to the *least-modulus* method. Note from Section 5.8 that the stochastic gradient online version for this loss function leads to the sign-error LMS. Huber also showed that if the noise comprises two components, one corresponding to a Gaussian and another to an arbitrary PDF (which remains symmetric), then the best in the minimax sense loss function is given by

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} \epsilon|y - f(\mathbf{x})| - \frac{\epsilon^2}{2}, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ \frac{1}{2}|y - f(\mathbf{x})|^2, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases}$$

for some parameter ϵ . This is known as the Huber loss function, and it is shown in Fig. 11.10. A loss function that can approximate the Huber one and, as we will see, turns out to have some nice compu-

¹¹ The best L_2 approximation of the worst noise model. Note that this is a pessimistic scenario, because the only information that is used is the symmetry of the noise PDF.

tational properties, is the so-called *linear* ϵ -insensitive loss function, defined as (see also Chapter 8)

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})| - \epsilon, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases} \quad (11.28)$$

shown in Fig. 11.10. Note that for $\epsilon = 0$, it coincides with the absolute value loss function, and it is close to the Huber loss for small values of $\epsilon < 1$. Another version is the *quadratic* ϵ -insensitive, defined as

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})|^2 - \epsilon, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases} \quad (11.29)$$

which coincides with the squared error loss for $\epsilon = 0$. The corresponding graph is given in Fig. 11.10. Observe that the two previously discussed ϵ -insensitive loss functions retain their convex nature; however, they are no more differentiable at all points.

11.8.1 THE LINEAR ϵ -INSENSITIVE OPTIMAL REGRESSION

Let us now adopt (11.28) as the loss function to quantify model misfit. We will treat the regression task in (11.23), employing a linear model for f , that is,

$$f(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0.$$

Once we obtain the solution expressed in inner product operations, the more general solution for the case where f lies in an RKHS will be obtained via the kernel trick; that is, inner products will be replaced by kernel evaluations.

Let us now introduce two sets of *auxiliary* variables. If

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \geq \epsilon,$$

define $\tilde{\xi}_n \geq 0$, such that

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n.$$

Note that ideally, we would like to select $\boldsymbol{\theta}, \theta_0$, so that $\tilde{\xi}_n = 0$, because this would make the contribution of the respective term in the loss function equal to zero. Also, if

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq -\epsilon,$$

define $\xi_n \geq 0$, such that

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n.$$

Once more, we would like to select our unknown set of parameters so that ξ_n is zero.

We are now ready to formulate the minimizing task around the corresponding empirical cost, regularized by the norm of θ , which is cast in terms of the auxiliary variables as¹²

$$\text{minimize} \quad J(\theta, \theta_0, \xi, \tilde{\xi}) = \frac{1}{2} \|\theta\|^2 + C \left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n \right), \quad (11.30)$$

$$\text{subject to} \quad y_n - \theta^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n, \quad n = 1, 2, \dots, N, \quad (11.31)$$

$$\theta^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n, \quad n = 1, 2, \dots, N, \quad (11.32)$$

$$\tilde{\xi}_n \geq 0, \quad \xi_n \geq 0, \quad n = 1, 2, \dots, N. \quad (11.33)$$

Before we proceed further some explanations are in order.

- The auxiliary variables $\tilde{\xi}_n$ and ξ_n , $n = 1, 2, \dots, N$, which measure the excess error with regard to ϵ , are known as *slack variables*. Note that according to the ϵ -insensitive rationale, any contribution to the cost function of an error with absolute value less than or equal to ϵ is zero. The previous optimization task attempts to estimate θ , θ_0 so that the contribution of error values larger than ϵ and smaller than $-\epsilon$ is minimized. Thus, the optimization task in (11.30)–(11.33) is equivalent to minimizing the empirical loss function

$$\frac{1}{2} \|\theta\|^2 + C \sum_{n=1}^N \mathcal{L}(y_n, \theta^T \mathbf{x}_n + \theta_0),$$

where the loss function is the linear ϵ -insensitive one. Note that any other method for minimizing (nondifferentiable) convex functions could be used (for example, Chapter 8). However, the constrained optimization involving the slack variables has a historical value and it was the path that paved the way in employing the kernel trick, as we will see soon.

The Solution

The solution of the optimization task is obtained by introducing Lagrange multipliers and forming the corresponding Lagrangian (see below for the detailed derivation). Having obtained the Lagrange multipliers, the solution turns out to be given in a simple and rather elegant form,

$$\hat{\theta} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n,$$

where $\tilde{\lambda}_n, \lambda_n$, $n = 1, 2, \dots, N$, are the Lagrange multipliers associated with each one of the constraints in (11.31) and (11.32). It turns out that the Lagrange multipliers are nonzero *only* for those points \mathbf{x}_n that correspond to error values *either equal to or larger than* ϵ . These are known as *support vectors*. Points that score error values *less* than ϵ correspond to zero Lagrange multipliers and do not participate

¹² It is common in the literature to formulate the regularized cost via the parameter C multiplying the loss term and not $\|\theta\|^2$. In any case, they are both equivalent.

in the formation of the solution. The bias term can be obtained by anyone from the set of equations

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 = \epsilon, \quad (11.34)$$

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n = \epsilon, \quad (11.35)$$

where n above runs over the points that are associated with $\tilde{\lambda}_n > 0$ ($\lambda_n > 0$) and $\tilde{\xi}_n = 0$ ($\xi_n = 0$) (note that these points form a subset of the support vectors). In practice, $\hat{\theta}_0$ is obtained as the average from all the previous equations.

For the more general setting of solving a linear task in an RKHS, the solution is of the same form as the one obtained before. All one needs to do is to replace \mathbf{x}_n with the respective images, $\kappa(\cdot, \mathbf{x}_n)$, and the vector $\hat{\theta}$ by a function, $\hat{\theta}$, i.e.,

$$\hat{\theta}(\cdot) = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \kappa(\cdot, \mathbf{x}_n).$$

Once $\hat{\theta}, \hat{\theta}_0$ have been obtained, we are ready to perform prediction. Given a value \mathbf{x} , we first perform the (implicit) mapping using the feature map

$$\mathbf{x} \mapsto \kappa(\cdot, \mathbf{x}),$$

and we get

$$\hat{y}(\mathbf{x}) = \langle \hat{\theta}, \kappa(\cdot, \mathbf{x}) \rangle + \hat{\theta}_0,$$

or

$$\boxed{\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n) \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0 : \quad \text{SVR prediction},} \quad (11.36)$$

where $N_s \leq N$ is the number of nonzero Lagrange multipliers. Observe that (11.36) is an expansion in terms of nonlinear (kernel) functions. Moreover, as only a fraction of the points is involved (N_s), the use of the ϵ -insensitive loss function achieves a form of *sparsification* on the general expansion dictated by the representer theorem in (11.17).

Solving the Optimization Task

The reader who is not interested in proofs can bypass this part in a first reading.

The task in (11.30)–(11.33) is a *convex programming* minimization, with a set of linear inequality constraints. As discussed in Appendix C, a minimizer has to satisfy the following *Karush–Kuhn–Tucker* (KKT) conditions:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0}, \quad \frac{\partial L}{\partial \theta_0} = 0, \quad \frac{\partial L}{\partial \tilde{\xi}_n} = 0, \quad \frac{\partial L}{\partial \xi_n} = 0, \quad (11.37)$$

$$\tilde{\lambda}_n (y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) = 0, \quad n = 1, 2, \dots, N, \quad (11.38)$$

$$\lambda_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n - \epsilon - \xi_n) = 0, \quad n = 1, 2, \dots, N, \quad (11.39)$$

$$\tilde{\mu}_n \tilde{\xi}_n = 0, \quad \mu_n \xi_n = 0, \quad n = 1, 2, \dots, N, \quad (11.40)$$

$$\tilde{\lambda}_n \geq 0, \quad \lambda_n \geq 0, \quad \tilde{\mu}_n \geq 0, \quad \mu_n \geq 0, \quad n = 1, 2, \dots, N, \quad (11.41)$$

where L is the respective Lagrangian,

$$\begin{aligned} L(\boldsymbol{\theta}, \theta_0, \tilde{\boldsymbol{\xi}}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n \right) \\ &\quad + \sum_{n=1}^N \tilde{\lambda}_n (y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) \\ &\quad + \sum_{n=1}^N \lambda_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n - \epsilon - \xi_n) \\ &\quad - \sum_{n=1}^N \tilde{\mu}_n \tilde{\xi}_n - \sum_{n=1}^N \mu_n \xi_n, \end{aligned} \quad (11.42)$$

where $\tilde{\lambda}_n, \lambda_n, \tilde{\mu}_n, \mu_n$ are the corresponding Lagrange multipliers. A close observation of (11.38) and (11.39) reveals that (why?)

$$\tilde{\xi}_n \xi_n = 0, \quad \tilde{\lambda}_n \lambda_n = 0, \quad n = 1, 2, \dots, N. \quad (11.43)$$

Taking the derivatives of the Lagrangian in (11.37) and equating to zero results in

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n, \quad (11.44)$$

$$\frac{\partial L}{\partial \theta_0} = 0 \longrightarrow \sum_{n=1}^N \tilde{\lambda}_n = \sum_{n=1}^N \lambda_n, \quad (11.45)$$

$$\frac{\partial L}{\partial \tilde{\xi}_n} = 0 \longrightarrow C - \tilde{\lambda}_n - \tilde{\mu}_n = 0, \quad (11.46)$$

$$\frac{\partial L}{\partial \xi_n} = 0 \longrightarrow C - \lambda_n - \mu_n = 0. \quad (11.47)$$

Note that all one needs in order to obtain $\hat{\boldsymbol{\theta}}$ are the values of the Lagrange multipliers. As discussed in Appendix C, these can be obtained by writing the problem in its dual representation form, that is,

$$\begin{aligned} \text{maximize with respect to } \boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}} & \quad \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon (\tilde{\lambda}_n + \lambda_n) \\ & \quad - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \mathbf{x}_n^T \mathbf{x}_m, \end{aligned} \quad (11.48)$$

$$\text{subject to} \quad 0 \leq \tilde{\lambda}_n \leq C, \quad 0 \leq \lambda_n \leq C, \quad n = 1, 2, \dots, N, \quad (11.49)$$

$$\sum_{n=1}^N \tilde{\lambda}_n = \sum_{n=1}^N \lambda_n. \quad (11.50)$$

Concerning the maximization task in (11.48)–(11.50), the following comments are in order.

- Plugging into the Lagrangian the estimate obtained in (11.44) and following the steps as required by the dual representation form (Problem 11.10), (11.48) results.
- From (11.46) and (11.47), taking into account that $\mu_n \geq 0$, $\tilde{\mu}_n \geq 0$, (11.49) results.
- The beauty of the dual representation form is that it involves the observation vectors in the form of inner product operations. Thus, when the task is solved in an RKHS, (11.48) becomes

$$\begin{aligned} \text{maximize with respect to } \boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}} & \quad \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon (\tilde{\lambda}_n + \lambda_n) \\ & - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \kappa(\mathbf{x}_n, \mathbf{x}_m). \end{aligned}$$

- The KKT conditions convey important information. The Lagrange multipliers, $\tilde{\lambda}_n, \lambda_n$, for points that score error less than ϵ , that is,

$$|\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n| < \epsilon,$$

are zero. This is a direct consequence of (11.38) and (11.39) and the fact that $\tilde{\xi}_n, \xi_n \geq 0$. Thus, the Lagrange multipliers are *nonzero* only for points which score error either equal to ϵ ($\tilde{\xi}_n, \xi_n = 0$) or larger values ($\tilde{\xi}_n, \xi_n > 0$). In other words, only the points with nonzero Lagrange multipliers (support vectors) enter in (11.44), which leads to a *sparsification* of the expansion in (11.44).

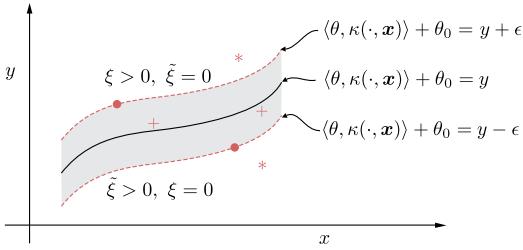
- Due to (11.43), either $\tilde{\xi}_n$ or ξ_n can be nonzero, but not both of them. This also applies to the corresponding Lagrange multipliers.
- Note that if $\tilde{\xi}_n > 0$ (or $\xi_n > 0$), then from (11.40), (11.46), and (11.47) we obtain

$$\tilde{\lambda}_n = C \text{ or } \lambda_n = C.$$

That is, the respective Lagrange multipliers get their maximum value. In other words, they have a “big say” in the expansion in (11.44). When $\tilde{\xi}_n$ and/or ξ_n are zero, then

$$0 \leq \tilde{\lambda}_n \leq C, \quad 0 \leq \lambda_n \leq C.$$

- Recall what we have said before concerning the estimation of θ_0 . Select any point corresponding to $0 < \tilde{\lambda}_n < C$ ($0 < \lambda_n < C$) which we know correspond to $\tilde{\xi}_n = 0$ ($\xi_n = 0$). Then $\hat{\theta}_0$ is computed from (11.38) and (11.39). In practice, one selects all such points and computes θ_0 as the respective mean.
- Fig. 11.11 illustrates $\hat{y}(\mathbf{x})$ for a choice of $\kappa(\cdot, \cdot)$. Observe that the value of ϵ forms a “tube” around the respective graph. Points lying outside the tube correspond to values of the slack variables larger than zero.

**FIGURE 11.11**

The tube around the nonlinear regression curve. Points outside the tube (denoted by stars) have either $\tilde{\xi} > 0$ and $\xi = 0$ or $\xi > 0$ and $\tilde{\xi} = 0$. The rest of the points have $\tilde{\xi} = \xi = 0$. Points that are inside the tube correspond to zero Lagrange multipliers.

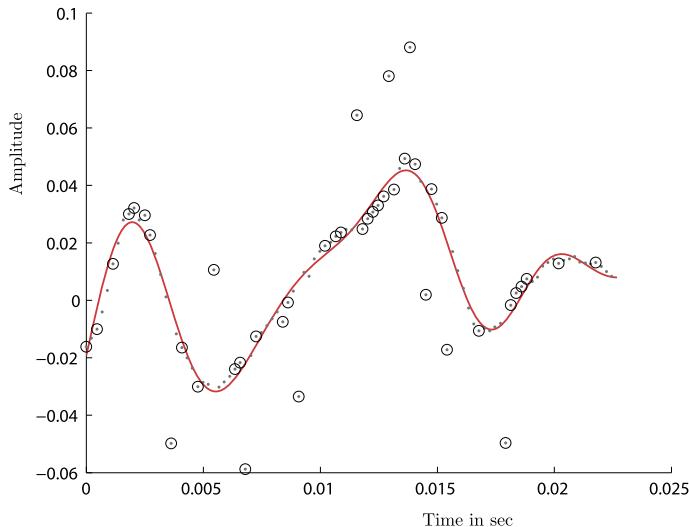
Remarks 11.3.

- Besides the linear ϵ -insensitive loss, similar analysis is valid for the quadratic ϵ -insensitive and Huber loss functions (for example, [28]). It turns out that use of the Huber loss function results in a larger number of support vectors. Note that a large number of support vectors increases complexity, as more kernel evaluations are involved.
- *Sparsity and ϵ -insensitive loss function:* Note that (11.36) has exactly the same form as (11.18). However, in the former case, the expansion is a sparse one using $N_s < N$, and in practice often $N_s \ll N$. The obvious question that is now raised is whether there is a “hidden” connection between the ϵ -insensitive loss function and the sparsity promoting methods, discussed in Chapter 9. Interestingly enough, the answer is in the affirmative [47]. Assuming the unknown function, g , in (11.23) to reside in an RKHS, and exploiting the representer theorem, it is approximated by an expansion in an RKHS and the unknown parameters are estimated by minimizing

$$L(\boldsymbol{\theta}) = \frac{1}{2} \left\| y(\cdot) - \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) \right\|_{\mathbb{H}}^2 + \epsilon \sum_{n=1}^N |\theta_n|.$$

This is similar to what we did for the kernel ridge regression with the notable exception that the ℓ_1 norm of the parameters is involved for regularization. The norm $\|\cdot\|_{\mathbb{H}}$ denotes the norm associated with the RKHS. Elaborating on the norm, it can be shown that for the noiseless case, the minimization task becomes identical with the SVR one.

Example 11.3. Consider the same time series used for the nonlinear prediction task in Example 11.2. This time, the SVR method was used optimized around the linear ϵ -insensitive loss function, with $\epsilon = 0.003$. The same Gaussian kernel with $\sigma = 0.004$ was employed as in the kernel ridge regression case. Fig. 11.12 shows the resulting prediction curve, $\hat{y}(x)$ as a function of x given in (11.36). The encircled points are the support vectors. Even without the use of any quantitative measure, the resulting curve fits the data samples much better compared to the kernel ridge regression, exhibiting the enhanced robustness of the SVR method relative to the kernel ridge regression, in the presence of outliers.

**FIGURE 11.12**

The resulting prediction curve for the same data points as those used for Example 11.2. The improved performance compared to the kernel ridge regression used for Fig. 11.9 is readily observed. The encircled points are the support vectors resulting from the optimization, using the ϵ -insensitive loss function.

Remarks 11.4.

- A more recent trend to deal with outliers is via their explicit modeling. The noise is split into two components, the inlier and the outlier. The outlier part has to be sparse; otherwise, it would not be called outlier. Then, sparsity-related arguments are mobilized to solve an optimization task that estimates both the parameters as well as the outliers (see, for example, [15, 71, 80, 85, 86]).

11.9 KERNEL RIDGE REGRESSION REVISITED

The kernel ridge regression was introduced in Section 11.7. Here, it will be restated via its dual representation form. The ridge regression in its primal representation can be cast as

$$\begin{aligned} \text{minimize with respect to } \boldsymbol{\theta}, \boldsymbol{\xi} & \quad J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \sum_{n=1}^N \xi_n^2 + C \|\boldsymbol{\theta}\|^2, \\ \text{subject to} & \quad y_n - \boldsymbol{\theta}^T \mathbf{x}_n = \xi_n, \quad n = 1, 2, \dots, N, \end{aligned} \tag{11.51}$$

which leads to the following Lagrangian:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\lambda}) = \sum_{n=1}^N \xi_n^2 + C \|\boldsymbol{\theta}\|^2 + \sum_{n=1}^N \lambda_n (y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \xi_n), \quad n = 1, 2, \dots, N. \tag{11.52}$$

Differentiating with respect to $\boldsymbol{\theta}$ and ξ_n , $n = 1, 2, \dots, N$, and equating to zero, we obtain

$$\boldsymbol{\theta} = \frac{1}{2C} \sum_{n=1}^N \lambda_n \mathbf{x}_n \quad (11.53)$$

and

$$\xi_n = \frac{\lambda_n}{2}, \quad n = 1, 2, \dots, N. \quad (11.54)$$

To obtain the Lagrange multipliers, (11.53) and (11.54) are substituted in (11.52), which results in the dual formulation of the problem, that is,

$$\begin{aligned} \text{maximize with respect to } \boldsymbol{\lambda} \quad & \sum_{n=1}^N \lambda_n y_n - \frac{1}{4C} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \\ & - \frac{1}{4} \sum_{n=1}^N \lambda_n^2, \end{aligned} \quad (11.55)$$

where we have replaced $\mathbf{x}_n^T \mathbf{x}_m$ with the kernel operation according to the kernel trick. It is a matter of straightforward algebra to obtain (see [98] and Problem 11.9)

$$\boldsymbol{\lambda} = 2C(\mathcal{K} + CI)^{-1} \mathbf{y}, \quad (11.56)$$

which combined with (11.53) and involving the kernel trick we obtain the prediction rule for the kernel ridge regression, that is,

$$\hat{y}(\mathbf{x}) = \mathbf{y}^T (\mathcal{K} + CI)^{-1} \boldsymbol{\kappa}(\mathbf{x}), \quad (11.57)$$

which is the same as (11.27); however, via this path one needs not to assume invertibility of \mathcal{K} . An efficient scheme for solving the kernel ridge regression has been developed in [118, 119].

11.10 OPTIMAL MARGIN CLASSIFICATION: SUPPORT VECTOR MACHINES

The optimal classifier, in the sense of minimizing the classification error, is the Bayesian classifier as discussed in Chapter 7. The method, being a member of the generative learning family, requires the knowledge of the underlying probability distributions. If these are not known, an alternative path is to resort to discriminative learning techniques and adopt a discriminant function f that realizes the corresponding classifier and try to optimize it so as to minimize the respective empirical loss, that is,

$$J(f) = \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)),$$

where

$$y_n = \begin{cases} +1, & \text{if } \mathbf{x}_n \in \omega_1, \\ -1, & \text{if } \mathbf{x}_n \in \omega_2. \end{cases}$$

BAYESIAN LEARNING: INFERENCE AND THE EM ALGORITHM

12

CONTENTS

12.1	Introduction	595
12.2	Regression: a Bayesian Perspective	596
12.2.1	The Maximum Likelihood Estimator	597
12.2.2	The MAP Estimator	598
12.2.3	The Bayesian Approach	599
12.3	The Evidence Function and Occam's Razor Rule.....	605
	Laplacian Approximation and the Evidence Function	607
12.4	Latent Variables and the EM Algorithm	611
12.4.1	The Expectation-Maximization Algorithm	611
12.5	Linear Regression and the EM Algorithm.....	613
12.6	Gaussian Mixture Models	616
12.6.1	Gaussian Mixture Modeling and Clustering	620
12.7	The EM Algorithm: a Lower Bound Maximization View	623
12.8	Exponential Family of Probability Distributions	627
12.8.1	The Exponential Family and the Maximum Entropy Method	633
12.9	Combining Learning Models: a Probabilistic Point of View	634
12.9.1	Mixing Linear Regression Models	634
	<i>Mixture of Experts</i>	638
	<i>Hierarchical Mixture of Experts</i>	638
12.9.2	Mixing Logistic Regression Models.....	639
Problems	641	
MATLAB® Exercises	643	
References	645	

12.1 INTRODUCTION

The Bayesian approach to parameter inference was introduced in Chapter 3. Compared to other methods for parameter estimation we have covered, the Bayesian method adopts a radically different viewpoint. The unknown set of parameters are treated as random variables instead of as a set of fixed (yet unknown) values. This was a revolutionary idea at the time it was introduced by Bayes and later on by Laplace, as pointed out in Chapter 3. Even now, after more than two centuries, it may seem strange to assume that a physical phenomenon/mechanism is controlled by a set of random parameters. However, there is a subtle point here. Treating the underlying set of parameters as random variables, θ , we do not really imply a random nature for them. The associated randomness, in terms of the prior distribution

$p(\boldsymbol{\theta})$, encapsulates our *uncertainty* about their values, prior to receiving any measurements/observations. Stated differently, the prior distribution represents our *belief* about the different possible values, although only one of them is actually true. From this perspective, probabilities are viewed in a more open-minded way, that is, as measures of uncertainty, as discussed in the beginning of Chapter 2.

Recall that parameter learning from data is an inverse problem. Basically, all we do is deduce the “causes” (parameters) from the “effects” (observations). The Bayes theorem can be seen as an inversion procedure expressed in a probabilistic context. Indeed, given the set of observations, say, \mathcal{X} , which are controlled by the unknown set of parameters, we write

$$p(\boldsymbol{\theta}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{X})}.$$

All that is needed for the above inversion is to have a guess about $p(\boldsymbol{\theta})$. This term has brought a lot of controversy in the statistical community for a number of years. However, once a reasonable guess of the prior is available, a number of advantages associated with the Bayesian approach emerge, compared to the alternative route. The latter embraces methods that view the parameters deterministically as constants of unknown values, and they are also referred to as *frequentist* techniques. The term comes from the more classical view of probabilities as frequencies of occurrence of repeatable events. A typical example of this family of methods is the maximum likelihood approach, which estimates the values of the parameters by maximizing $p(\mathcal{X}|\boldsymbol{\theta})$; the value of the latter conditional probability density function (PDF) is solely controlled by the obtained observations in a sequence of experiments.

This is the first of two chapters dedicated to Bayesian learning. We present the main concepts and philosophy behind Bayesian inference. We introduce the expectation-maximization (EM) algorithm and apply it in some typical machine learning parametric modeling tasks, such as regression, mixture modeling, and mixture of experts. Finally, the exponential family of distributions is introduced and the notion of conjugate priors is discussed.

12.2 REGRESSION: A BAYESIAN PERSPECTIVE

The Bayesian inference treatment of the linear regression task was introduced in Chapter 3. In the current chapter, we go beyond the basic definitions and reveal and exploit various possibilities that the Bayesian philosophy offers to the study of this important machine learning task. Let us first summarize the findings of Chapter 3 and then start building upon them.¹

Recall the (generalized) linear regression task, as it was introduced in previous chapters, that is,

$$\mathbf{y} = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) + \eta = \theta_0 + \sum_{k=1}^{K-1} \theta_k \phi_k(\mathbf{x}) + \eta, \quad (12.1)$$

where $\mathbf{y} \in \mathbb{R}$ is the output random variable, $\mathbf{x} \in \mathbb{R}^l$ is the input random vector, $\eta \in \mathbb{R}$ is the noise disturbance, $\boldsymbol{\theta} \in \mathbb{R}^K$ is the unknown parameter vector, and

¹ Recall our adopted notation: random variables and vectors are denoted with roman and their respected measured values/observations with Times Roman fonts.

$$\boldsymbol{\phi}(\mathbf{x}) := [\phi_1(\mathbf{x}), \dots, \phi_{K-1}(\mathbf{x}), 1]^T,$$

where ϕ_k , $k = 1, \dots, K - 1$, are some (fixed) basis functions. As we already know, typical examples of such functions can be the Gaussian function, splines, monomials, and others. We are given a set of N output–input training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$. In our current setting, we assume that the respective (unobserved) noise values η_n , $n = 1, 2, \dots, N$, are samples of jointly Gaussian distributed random variables with covariance matrix Σ_η , that is,

$$p(\boldsymbol{\eta}) = \frac{1}{(2\pi)^{N/2} |\Sigma_\eta|^{1/2}} \exp\left(-\frac{1}{2} \boldsymbol{\eta}^T \Sigma_\eta^{-1} \boldsymbol{\eta}\right), \quad (12.2)$$

where $\boldsymbol{\eta} = [\eta_1, \eta_2, \dots, \eta_N]^T$.

12.2.1 THE MAXIMUM LIKELIHOOD ESTIMATOR

The maximum likelihood (ML) method was introduced in Chapter 3. According to the method, the unknown set of parameters are treated as a deterministic vector variable, $\boldsymbol{\theta}$, which parameterizes the PDF that describes the output vector of observations

$$\mathbf{y} = \Phi \boldsymbol{\theta} + \boldsymbol{\eta}, \quad (12.3)$$

where

$$\Phi = \begin{bmatrix} \boldsymbol{\phi}^T(\mathbf{x}_1) \\ \boldsymbol{\phi}^T(\mathbf{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\mathbf{x}_N) \end{bmatrix} \quad (12.4)$$

and

$$\mathbf{y} = [y_1, y_2, \dots, y_N]^T.$$

A simple replacement of X with Φ in (3.61) changes the ML estimate to

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = (\Phi^T \Sigma_\eta^{-1} \Phi)^{-1} \Phi^T \Sigma_\eta^{-1} \mathbf{y}. \quad (12.5)$$

For the simple case of uncorrelated noise samples of equal variance σ_η^2 ($\Sigma_\eta = \sigma_\eta^2 I$), Eq. (12.5) becomes identical to the least-squares (LS) solution

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} = \hat{\boldsymbol{\theta}}_{\text{LS}}. \quad (12.6)$$

A major drawback of the ML approach is that it is vulnerable to overfitting, because no care is taken of complex models that try to “learn” the specificities of the particular training set, as already discussed in Chapter 3.

12.2.2 THE MAP ESTIMATOR

According to the maximum a posteriori probability (MAP) method, the unknown set of parameters is treated as a random vector θ and its posterior, for a given set of output observations, y , is expressed as

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}, \quad (12.7)$$

where $p(\theta)$ is the associated prior PDF. We have eliminated from the notation the dependence on \mathcal{X} , to make it look simpler. We emphasize that the input set, $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, is considered fixed, so all the randomness associated with y is due to the noise source. Assuming both the prior and the conditional PDFs to be Gaussians,² that is,

$$p(\theta) = \mathcal{N}(\theta|\theta_0, \Sigma_\theta) \quad (12.8)$$

and

$$p(y|\theta) = \mathcal{N}(y|\Phi\theta, \Sigma_\eta), \quad (12.9)$$

where (12.2) and (12.3) have been used, the posterior $p(\theta|y)$ turns out also to be Gaussian with mean vector

$$\mu_{\theta|y} := \mathbb{E}[\theta|y] = \theta_0 + \left(\Sigma_\theta^{-1} + \Phi^T \Sigma_\eta^{-1} \Phi \right)^{-1} \Phi^T \Sigma_\eta^{-1} (y - \Phi\theta_0). \quad (12.10)$$

Because the maximum of a Gaussian coincides with its mean, we have

$$\hat{\theta}_{\text{MAP}} = \mathbb{E}[\theta|y]. \quad (12.11)$$

In the present chapter's appendix, an analytical proof of (12.10) is provided.³ It suffices to replace in (12.143) $t \rightarrow y$, $z \rightarrow \theta$, $A \rightarrow \Phi$, $\Sigma_{t|z} \rightarrow \Sigma_\eta$, and $\Sigma_z \rightarrow \Sigma_\theta$. Note that the MAP estimate is a regularized version of $\hat{\theta}_{\text{ML}}$. Regularization is achieved via θ_0 and Σ_θ , which are imposed by the prior $p(\theta)$. If one assumes $\Sigma_\theta = \sigma_\theta^2 I$, $\Sigma_\eta = \sigma_\eta^2 I$, and $\theta_0 = \mathbf{0}$, then (12.10) coincides with the solution of the regularized LS (ridge) regression,⁴

$$\hat{\theta}_{\text{MAP}} = \left(\lambda I + \Phi^T \Phi \right)^{-1} \Phi^T y, \quad (12.12)$$

where we have set $\lambda := \frac{\sigma_\eta^2}{\sigma_\theta^2}$. We already know from Chapter 3 that the value of λ is critical to the performance of the estimator with respect to the mean-square error (MSE) performance. The main issue now becomes how to choose a good value for λ , or equivalently for Σ_θ , Σ_η in the more general case. In practice, the cross-validation method (Chapter 3) is employed; different values of λ are tested and

² Because in this chapter many random variables will be involved, we explicitly state the name of the variable to which we refer in $\mathcal{N}(\cdot|\cdot, \cdot)$.

³ Because the appendix serves the needs of various parts of the book, each time involving different variables, one has to make the necessary notational substitutions. Note that the appendix can be downloaded from the book's site.

⁴ Recall from Section 3.8 that this is valid if either the data have been centered or the intercept (bias) is involved in the regularizing norm term.

the one that leads to the best MSE (or some other criterion) is selected. However, this is a computationally costly procedure, especially for complex models, where a large number of parameters is involved. Moreover, such a procedure forces us to use only a fraction of the available data for training, to reserve the rest for testing. The reader may wonder why we do not use the training data to optimize with respect to both the unknown parameter vector θ and the regularization parameter. Let us consider as an example the simpler case of ridge regression, for centered data ($\theta_0 = 0$). The cost function comprises two terms, one that is data dependent and measures the misfit, and one that depends only on the unknown parameters,

$$J(\theta, \lambda) = \|\mathbf{y} - \Phi\theta\|^2 + \lambda\|\theta\|^2. \quad (12.13)$$

It is obvious that the only value of λ that leads to the minimum squared error fit *over the training data set* (empirical loss) is $\lambda = 0$. Any other value of λ would result in an estimate of θ which scores larger values of the squared error term; this is natural, because for $\lambda \neq 0$ the optimization has to take care of the extra regularizing term, too. It is only when *test data sets* are employed, where values of $\lambda \neq 0$ lead to an overall decrease of the MSE (not the empirical one).

12.2.3 THE BAYESIAN APPROACH

The Bayesian approach to regression attempts to overcome the previously reported drawbacks, which are associated with the overfitting. All the involved parameters can be estimated on the training set. In this vein, the parameters will be treated as random variables. At the same time, because the main task now becomes that of inferring the PDF that describes the unknown set of parameters, instead of obtaining a single vector estimate, one has more information at her/his disposal. Having said that, it does not mean that Bayesian techniques are necessarily free from cross-validation; this will be needed to assess their overall performance. We will comment further on this in the Remarks at the end of Section 12.3.

As we know, the starting point is the same as that for MAP, and in particular (12.7). However, instead of taking just the maximum of the numerator in (12.7), we will make use of $p(\theta|y)$ as a whole. Most of the secrets here lie in the denominator $p(y)$, which is basically the normalizing constant,

$$p(y) = \int p(y|\theta)p(\theta)d\theta. \quad (12.14)$$

As we will soon see, there is much more information hidden in $p(y)$ that goes beyond the need of just computing $p(\theta|y)$. The difficulty with (12.14) is that, in general, the evaluation of the integral cannot be performed analytically. In such cases, one has to resort to approximate techniques to obtain the required information. To this end, a number of approaches are available, and a large part of this book is dedicated to their study. More specifically, the following methods have been proposed and will be considered:

- the Laplacian approximation method, presented in Section 12.3;
- the variational approximation method, presented in Section 13.2;
- the variational bound approximation method, presented in Section 13.8;
- Monte Carlo techniques for the evaluation of the integral, which are discussed in Chapter 14;
- message passing algorithms, to be discussed in Chapter 15.

For the case under study in this section, where $p(y|\theta)$ and $p(\theta)$ are both assumed to be Gaussians, $p(y)$ can be evaluated analytically; it turns out that the joint distribution $p(y, \theta)$ is also Gaussian and hence the marginal $p(y)$ is Gaussian as well. All these are shown in detail in the appendix of this chapter. Indeed, if we set in (12.146) and (12.151) of the appendix $z \rightarrow \theta$, $t \rightarrow y$, and $A \rightarrow \Phi$, it turns out that for the regression model of (12.3) and the prior PDF in (12.8) as well as the noise model of (12.2), we obtain

$$p(y) = \mathcal{N}(y | \Phi\theta_0, \Sigma_\eta + \Phi\Sigma_\theta\Phi^T). \quad (12.15)$$

Moreover, the posterior $p(\theta|y)$ is also Gaussian,

$$p(\theta|y) = \mathcal{N}(\theta | \mu_{\theta|y}, \Sigma_{\theta|y}), \quad (12.16)$$

where $\mu_{\theta|y}$ is given by (12.10) and the covariance matrix results from (12.147), after the appropriate notational substitutions, that is,

$$\Sigma_{\theta|y} = (\Sigma_\theta^{-1} + \Phi^T \Sigma_\eta^{-1} \Phi)^{-1}. \quad (12.17)$$

The posterior PDF in (12.16) encapsulates our knowledge about θ , after the observations y have been obtained. Hence, our uncertainty about θ has been reduced, which is the main reason that (12.16) is different from the prior PDF in (12.8); the latter represents only our initial guess. The covariance matrix in (12.17) provides the information about our uncertainty with respect to θ . If the Gaussian in (12.16) is very broad around its mean $\mu_{\theta|y}$, it indicates that in spite of the reception of the observations still much uncertainty about θ remains. This can be due (a) to the nature of the problem, for example, high noise variance, as this is conveyed by Σ_η , (b) to the number of observations, N , which may not be enough, and/or (c) to modeling inaccuracies, as this is conveyed by Φ in (12.17). The opposite comments are in order if the posterior PDF is sharply peaked around its mean.

As we have already stated in Chapter 3, the Bayesian philosophy provides the means for a direct inference of the output variable, which in many applications is the quantity of interest; given the input vector, the task is to predict the output. In such cases, estimating a value for the unknown θ is only the means to an end. To formulate the prediction task directly, without involving θ , one has to integrate the contribution of θ . Having learned the posterior $p(\theta|y)$, given a new input vector x , for the regression model in (12.1), the conditional PDF of the output variable, y , given the set of observations, is written as

$$p(y|x, y) = \int p(y|x, \theta) p(\theta|y) d\theta. \quad (12.18)$$

Note that we have used $p(y|x, y, \theta) = p(y|x, \theta)$ because y is conditionally independent of y given the value of θ . As already stated, strictly speaking, the posterior should have been denoted as $p(\theta|y; \mathcal{X})$ to indicate the dependence on the input training samples. However, the dependence on \mathcal{X} has been suppressed to unclutter notation.

In the sequel, and in order to simplify algebra and focus on the concepts, we assume that the noise model in (12.2) is such that $\Sigma_\eta = \sigma_\eta^2 I$ and also $\Sigma_\theta = \sigma_\theta^2 I$ for the prior PDF in (12.8). Then we have

$$p(y|x, \theta) = \mathcal{N}(y | \theta^T \phi(x), \sigma_\eta^2),$$

and (12.17) and (12.10) for the posterior covariance matrix and mean, respectively, become

$$\Sigma_{\theta|y} = \left(\frac{1}{\sigma_\theta^2} I + \frac{1}{\sigma_\eta^2} \Phi^T \Phi \right)^{-1}, \quad (12.19)$$

$$\mu_{\theta|y} = \theta_0 + \frac{1}{\sigma_\eta^2} \left(\frac{1}{\sigma_\theta^2} I + \frac{1}{\sigma_\eta^2} \Phi^T \Phi \right)^{-1} \Phi^T (y - \Phi \theta_0). \quad (12.20)$$

The integration in (12.18) can now be carried out analytically as in (12.136) and (12.137) and using (12.150) and (12.151) in the appendix, with $z \rightarrow \theta$, $t \rightarrow y$, $A \rightarrow \phi^T$, $\mu_z \rightarrow \mu_{\theta|y}$, $\Sigma_z \rightarrow \Sigma_{\theta|y}$, $\Sigma_{t|z} \rightarrow \sigma_\eta^2$, and we obtain

$p(y|\mathbf{x}, \mathbf{y}) = \mathcal{N}(y|\mu_y, \sigma_y^2) : \text{ predictive distribution,}$

(12.21)

where

$$\mu_y = \phi^T(\mathbf{x}) \mu_{\theta|y}, \quad (12.22)$$

$$\begin{aligned} \sigma_y^2 &= \sigma_\eta^2 + \phi^T(\mathbf{x}) \Sigma_{\theta|y} \phi(\mathbf{x}) \\ &= \sigma_\eta^2 + \phi^T(\mathbf{x}) \left(\frac{1}{\sigma_\theta^2} I + \frac{1}{\sigma_\eta^2} \Phi^T \Phi \right)^{-1} \phi(\mathbf{x}) \\ &= \sigma_\eta^2 + \sigma_\eta^2 \sigma_\theta^2 \phi^T(\mathbf{x}) \left(\sigma_\eta^2 I + \sigma_\theta^2 \Phi^T \Phi \right)^{-1} \phi(\mathbf{x}). \end{aligned} \quad (12.23)$$

Hence, given \mathbf{x} one can predict the respective value of y using the most probable value, that is, μ_y in (12.22). Note that the same prediction value would result via the MAP estimate in (12.10) (or (12.12), if $\theta_0 = \mathbf{0}$, also obtained via the ridge regression task). Have we then gained anything extra by adopting the Bayesian approach? The answer is in the affirmative. *More information concerning the predicted value is now available, because (12.23) quantifies the associated uncertainty.*

To investigate (12.23) further, let us simplify it by adopting the following approximation:

$$R_\phi := \mathbb{E}[\phi(\mathbf{x}) \phi^T(\mathbf{x})] \simeq \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi^T(\mathbf{x}_n) = \frac{1}{N} \Phi^T \Phi,$$

or

$$\Phi^T \Phi \simeq N R_\phi, \quad (12.24)$$

where R_ϕ is the autocorrelation matrix of the random vector $\phi(\mathbf{x})$. Employing (12.24) into (12.23) leads to

$$\sigma_y^2 \simeq \sigma_\eta^2 \left(1 + \sigma_\theta^2 \phi^T(\mathbf{x}) \left(\sigma_\eta^2 I + N \sigma_\theta^2 R_\phi \right)^{-1} \phi(\mathbf{x}) \right), \quad (12.25)$$

which for large enough N becomes

$$\sigma_y^2 \simeq \sigma_\eta^2 \left(1 + \frac{1}{N} \boldsymbol{\phi}^T(\mathbf{x}) R_\phi^{-1} \boldsymbol{\phi}(\mathbf{x}) \right).$$

Thus, for a large number of observations, $\sigma_y^2 \rightarrow \sigma_\eta^2$, and our uncertainty is contributed by the noise source, which cannot be reduced anymore. For smaller values of N , there is extra uncertainty associated with the parameter $\boldsymbol{\theta}$, measured by σ_θ^2 in (12.25).

So far in this section, we dealt with Gaussians, which led to tractable and analytically computed integrals. Are there ways to attack more general cases? Moreover, even in the case of Gaussian PDFs, we have assumed the covariance matrices Σ_θ , Σ_η to be known. In practice, they are not. Even if one assumes that Σ_η can be experimentally measured, there still remains Σ_θ . Can one select the related parameters via an optimization process? If the answer is yes, can this optimization be carried out on the training set, or one would necessarily run into problems similar to the ones we faced with the regularization approach? We will indulge in all these challenges in the sections to follow.

Remarks 12.1.

- The MAP estimator is sometimes referred to as *Type I* estimator, to be distinguished from the Type II estimation method, which will be discussed in Remarks 12.2, in the next section.
- The posterior mean in (12.10) can be met in different variants, which are obtained via the application of the matrix inversion lemmas given in Appendix A.1. In the chapter's appendix, it is shown that (Eq. (12.152))

$$\boldsymbol{\mu}_{\theta|y} = \left(\Sigma_\theta^{-1} + \Phi^T \Sigma_\eta^{-1} \Phi \right)^{-1} \left(\Phi^T \Sigma_\eta^{-1} \mathbf{y} + \Sigma_\theta^{-1} \boldsymbol{\theta}_0 \right) \quad (12.26)$$

or (Eq. (12.148))

$$\boldsymbol{\mu}_{\theta|y} = \boldsymbol{\theta}_0 + \Sigma_\theta \Phi^T \left(\Sigma_\eta + \Phi \Sigma_\theta \Phi^T \right)^{-1} (\mathbf{y} - \Phi \boldsymbol{\theta}_0). \quad (12.27)$$

Also, using Woodbury's identity from Appendix A.1, we can readily see that

$$\Sigma_{\theta|y} = \Sigma_\theta - \Sigma_\theta \Phi^T \left(\Sigma_\eta + \Phi \Sigma_\theta \Phi^T \right)^{-1} \Phi \Sigma_\theta. \quad (12.28)$$

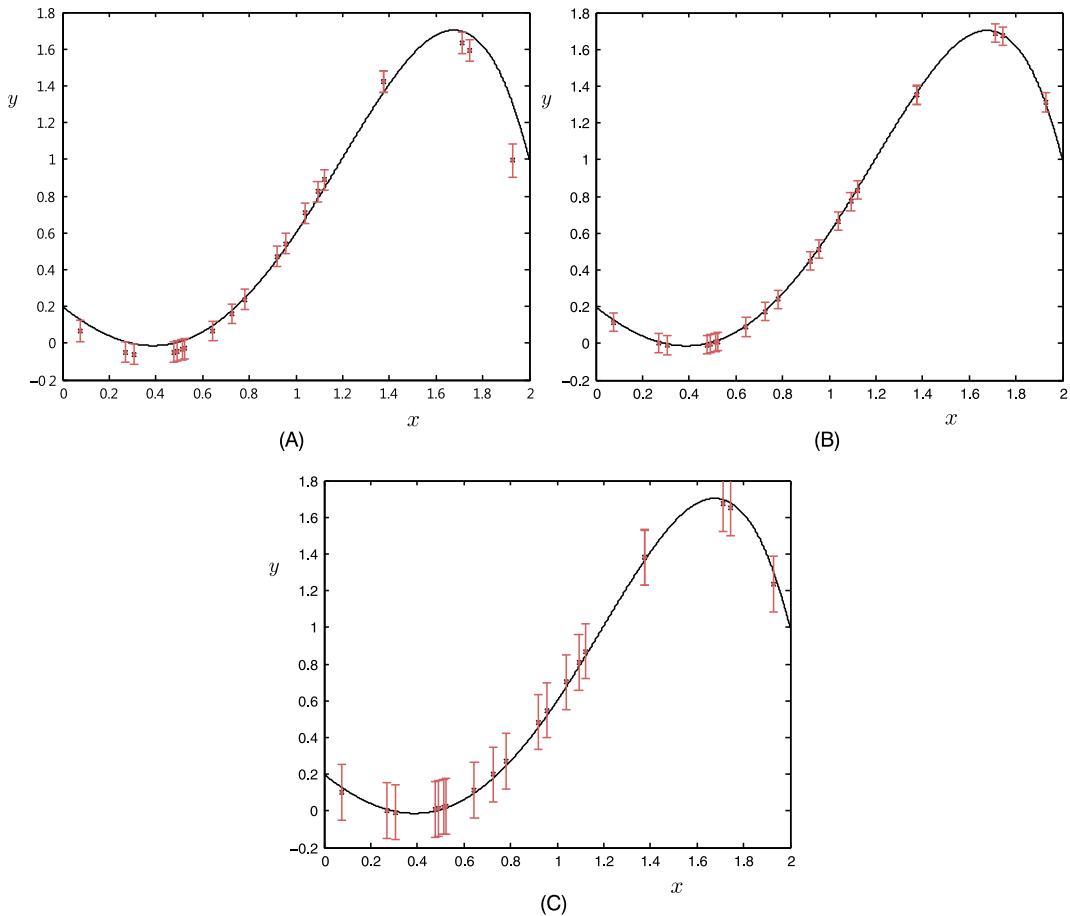
In practice, one uses the most computationally convenient form, depending on the dimensionality of the involved matrices to invert the one of lower dimension.

Example 12.1. This example demonstrates the prediction task summarized in (12.22) and (12.23). Data are generated based on the following nonlinear model:

$$y_n = \theta_0 + \theta_1 x_n + \theta_2 x_n^2 + \theta_3 x_n^3 + \theta_5 x_n^5 + \eta_n, \quad n = 1, 2, \dots, N,$$

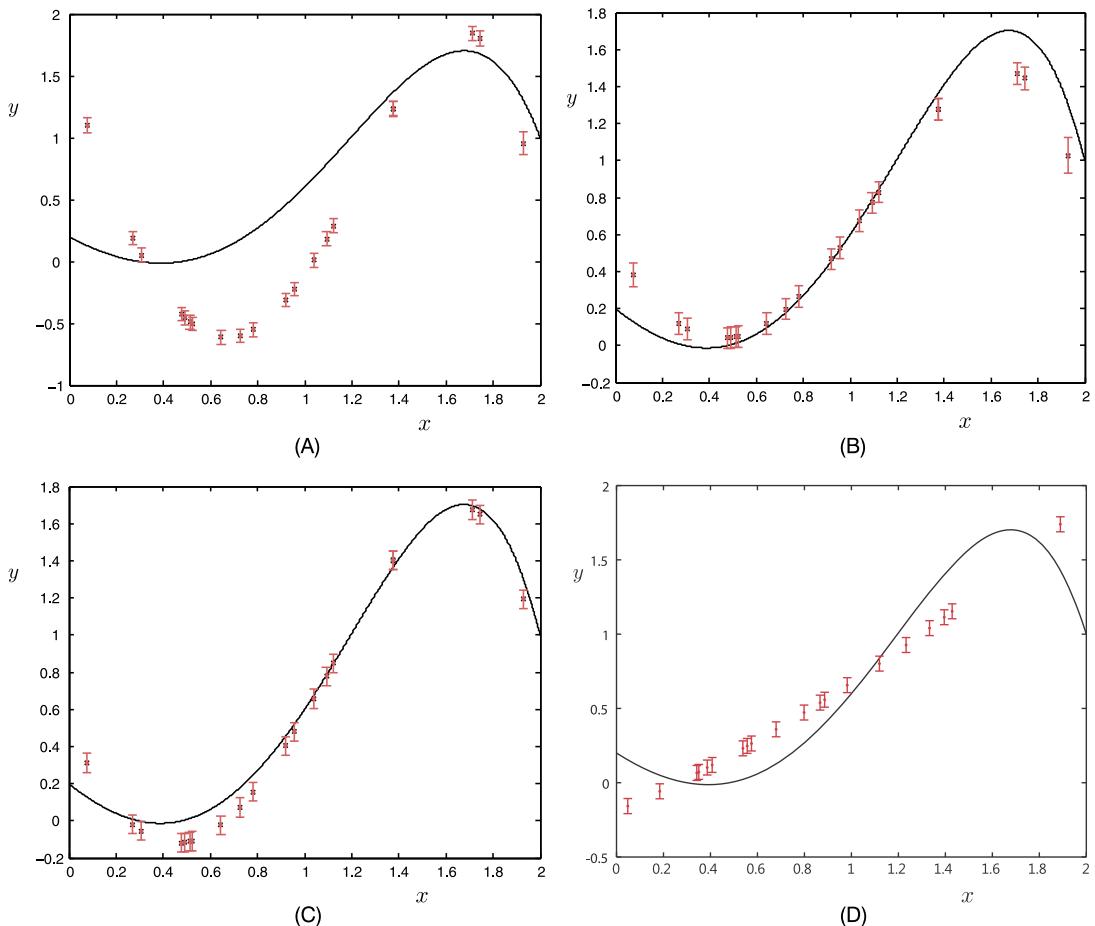
where η_n are i.i.d. noise samples drawn from a zero mean Gaussian with variance σ_η^2 . Samples x_n are equidistant points in the interval $[0, 2]$. The goal of the task is to predict the value y given a measured value x , using (12.22). The parameter values used to generate the data were equal to

$$\theta_0 = 0.2, \theta_1 = -1, \theta_2 = 0.9, \theta_3 = 0.7, \theta_5 = -0.2.$$

**FIGURE 12.1**

Each one of the red points (y, x) indicates the prediction (y) corresponding to the input value (x) . The error bars are dictated by the computed variance, σ_y^2 . The mean values used in the Gaussian prior are equal to the true values of the unknown model. (A) $\sigma_\eta^2 = 0.05$, $N = 20$, $\sigma_\theta^2 = 0.1$. (B) $\sigma_\eta^2 = 0.05$, $N = 500$, $\sigma_\theta^2 = 0.1$. (C) $\sigma_\eta^2 = 0.15$, $N = 500$, $\sigma_\theta^2 = 0.1$. Observe that the larger the data set is, the better the predictions are and the larger the noise variance is, the larger the error bars become.

- (a)** In the first set of experiments, a Gaussian prior for the unknown $\boldsymbol{\theta}$ was used with mean $\boldsymbol{\theta}_0$ equal to the previous true set of parameters and $\Sigma_\theta = 0.1I$. Also, the true model structure was used to construct the matrix Φ . Fig. 12.1A shows the points (y, x) in red together with the error bars, as measured by the computed σ_y^2 , for the case of $N = 20$ training points and $\sigma_\eta^2 = 0.05$. Fig. 12.1B demonstrates the obtained improvement when the training points are increased to $N = 500$, while keeping the values of the other two parameters unchanged. Fig. 12.1C corresponds to the latter case, where the noise variance is increased to $\sigma_\eta^2 = 0.15$.

**FIGURE 12.2**

In this set of figures the mean values of the prior are different from that of the true model. (A) $\sigma_\eta^2 = 0.05$, $N = 20$, $\sigma_\theta^2 = 0.1$. (B) $\sigma_\eta^2 = 0.05$, $N = 20$, $\sigma_\theta^2 = 2$; observe the effect of using larger variance for the prior. (C) $\sigma_\eta^2 = 0.05$, $N = 500$, $\sigma_\theta^2 = 0.1$; observe the effect of the larger training data set. (D) The points correspond to a wrong model.

- (b) In the second set of experiments, we kept the correct model, but the mean of the prior was given a different value to that of the true model, namely,

$$\boldsymbol{\theta}_0 = [-10.54, 0.465, 0.0087, -0.093, -0.004]^T.$$

Fig. 12.2A corresponds to the case of $\sigma_\eta^2 = 0.05$, $N = 20$, and $\sigma_\theta^2 = 0.1$. Note the improvement that is obtained when increasing $\sigma_\theta^2 = 2$, shown in Fig. 12.2B, while N and σ_η^2 remain the same as before; this is because the model takes into consideration our uncertainty about the prior mean

being away from the true value. Fig. 12.2C corresponds to $\sigma_\eta^2 = 0.05$, $N = 500$, and $\sigma_\theta^2 = 0.1$ and shows the advantage of using a large number of training points.

- (c) Fig. 12.2D corresponds to the case where the adopted model for prediction is the wrong one, that is,

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \eta.$$

The used values were $\sigma_\eta^2 = 0.05$, $N = 500$, and $\sigma_\theta^2 = 2$. Observe that once a wrong model has been adopted, one must not have “high expectations” for good prediction performance.

12.3 THE EVIDENCE FUNCTION AND OCCAM'S RAZOR RULE

In the previous section, we made a comment about the importance of the marginal PDF $p(\mathbf{y})$. This section is fully dedicated to this quantity. In the notation used in (12.14), we did tacitly suppress the dependence on the adopted model. For example, the Gaussian assumption for the prior in (12.8) and for the conditional in (12.9) should have been reflected in the marginal as $p(\mathbf{y}; \boldsymbol{\phi}, \Sigma_\eta, \Sigma_\theta)$, because different Gaussians, different basis functions, and different orders K of the model can be used. Furthermore, non-Gaussian PDFs can also be adopted. In a more general setting, let us make the dependence on the model explicit as $p(\mathbf{y}|\mathcal{M}_i)$. Assuming the choice of a model to be random, mobilizing the Bayes theorem once more, we have

$$P(\mathcal{M}_i|\mathbf{y}) = \frac{P(\mathcal{M}_i)p(\mathbf{y}|\mathcal{M}_i)}{p(\mathbf{y})}, \quad (12.29)$$

where

$$p(\mathbf{y}) = \sum_i P(\mathcal{M}_i)p(\mathbf{y}|\mathcal{M}_i), \quad (12.30)$$

and $P(\mathcal{M}_i)$ is the prior probability of \mathcal{M}_i ; $P(\mathcal{M}_i)$ provides a measure of the subjective prior over all possible models, which expresses our guess on how plausible a model is with respect to alternative ones, prior to the data arrival. Because the denominator in (12.29) is independent of the model, one can obtain the most probable model, after observing \mathbf{y} , by maximizing the numerator. If one assigns to all possible models equal probabilities, then detecting the most probable model under the given set of observations becomes a task of maximizing $p(\mathbf{y}|\mathcal{M}_i)$ with respect to the model, \mathcal{M}_i . This is the reason that this PDF is known as the *evidence function* for the model or simply as the *evidence*. In practice, we content ourselves with using the most probable model, although an orthodox Bayesian would suggest to average all obtained quantities over all possible models, as in (12.30). In an ideal Bayesian setting, one does not choose among models; predictions are performed by summing over all possible models, each one weighted by the respective probability. However, in many practical problems we may have reasons to suggest that the evidence function is strongly peaked around a specific model; after all, such an assumption may simplify the task considerably.

From a mathematical formulation's point of view, each model is expressed in terms of a set of nonrandom (deterministic) parameters. To get the model that maximizes the evidence is equivalent to maximizing with respect to these parameters. For example, let us assume that in the regression task we adopt a Gaussian distribution for the noise and a Gaussian prior for the regression task random

The BIC is obtained as a large N approximation to (12.38), assuming a broad enough Gaussian prior, and manipulating a bit the determinant involved in the last term. For a discussion including other related criteria, see [3,41].

- The Bayesian framework is also closely related to the minimum description length (MDL) methods. The log-evidence is associated with the number of bits in the shortest message that encodes the data via model \mathcal{M}_i (for example, [45]).
- *Type II maximum likelihood*: Note that the evidence is the marginal likelihood function after integrating out the parameters θ . To distinguish it from the MAP method, when the evidence function is maximized with respect to a set of unknown parameters, it is usually referred to as *generalized maximum likelihood* or *Type II maximum likelihood* and sometimes as *empirical Bayes*. Recall from Remarks 12.1 that the MAP is named Type I estimator.

12.4 LATENT VARIABLES AND THE EM ALGORITHM

At the end of Section 12.3, it was pointed out that if we assume that $p(\mathbf{y}|\theta)$ and $p(\theta)$ are Gaussians of the form given in (12.39), then the evidence function associated with the regression task in Eq. (12.3) is also Gaussian parameterized via the (hyper)parameters $\sigma_\eta^2, \sigma_\theta^2$. Let us denote this set of unknown non-random parameters as $\xi = [\sigma_\eta^2, \sigma_\theta^2]^T$, and we can write $p(\mathbf{y}; \xi)$. Maximizing the evidence with respect to ξ becomes a typical maximum likelihood one. However, in general, such closed form expressions for the evidence function are not possible, and the integration in (12.14) is intractable. The main source of difficulty is the fact that our regression model is described via two sets of random variables, that is, \mathbf{y} and θ , yet only one of them, \mathbf{y} , can be directly observed. The other one, θ , cannot be observed, and this is the reason that the Bayesian philosophy tries to integrate it out of the joint PDF $p(\mathbf{y}, \theta)$. If θ could be observed, then the unknown set of parameters ξ could be obtained by maximizing the likelihood $p(\mathbf{y}, \theta; \xi)$, given a set of (joint) observations of (\mathbf{y}, θ) . Because they cannot be observed, the random variables in the vector θ are known as *hidden* variables.

Although we introduced the notion of hidden variables via our familiar regression task, unobserved variables (besides the noise) occur very often in a number of problems in probability and statistics. In a number of cases, from a larger set of jointly distributed random variables, only some can be observed and the rest remain hidden. Moreover, it is often useful to *build* hidden variables into a model by design. These variables are meant to represent latent causes that influence the observed variables and their introduction may facilitate the analysis. Often, such models associate one extra variable for *each one* of the observations. We will refer to such unobserved variables as *latent* variables. Their difference with the hidden ones is that their number is equal to that of the observations and grows accordingly as more observations get available. In contrast, unobserved random variables that are associated with the model and not with each one of the observations, individually, will be referred to as hidden variables.

12.4.1 THE EXPECTATION-MAXIMIZATION ALGORITHM

The *expectation-maximization* (EM) algorithm is an elegant algorithmic tool to maximize the likelihood (evidence) function for problems with latent/hidden variables. We will state the problem in a general formulation, and then we will apply it to different tasks, including regression.

Let \mathbf{x} be a random vector and let \mathcal{X} be the respective set of observations. Let $\mathcal{X}^l := \{\mathbf{x}_1^l, \dots, \mathbf{x}_N^l\}$ be the corresponding set of latent variables; these can be either of a discrete or of a continuous nature. Each observation in \mathcal{X} is associated with a latent vector \mathbf{x}^l in \mathcal{X}^l . These latent variables are also known as *local* ones and each one expresses the hidden structure associated with the corresponding observation. We will refer to the set $\{\mathcal{X}, \mathcal{X}^l\}$ as the *complete* data set and to the set of observations, \mathcal{X} , as the *incomplete* one. Hidden random parameters, $\boldsymbol{\theta}$, can also be dealt with as latent variables; however, their number remains fixed (independent of N) and they are also known as *global* variables. In such cases, the complete data set is $\{\mathcal{X}, \mathcal{X}^l, \boldsymbol{\theta}\}$. To unclutter notation, we will focus on the set \mathcal{X}^l ; yet, everything to be said also applies to hidden/global as well as to a combination of local/latent and global/hidden variables. Furthermore, let the corresponding joint distribution be parameterized in terms of a set of unknown nonrandom (hyper)parameters, $\boldsymbol{\xi}$. We further assume that, although \mathcal{X}^l cannot be observed, the posterior distribution $p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi})$ ($P(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi})$ for the discrete case) is fully specified, given the values in $\boldsymbol{\xi}$ and the observations in \mathcal{X} . This is a critical assumption for the EM algorithm. If the posterior PDF is not known, then one has to resort to variants of the EM which attempt to approximate it. We will come to such schemes in Section 13.2.

If the complete log-likelihood $p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi})$ were available, then the problem would be a typical maximum likelihood one. However, because no observations for the latent variables are available, the EM algorithm considers the *expectation* of the complete log-likelihood with respect to the latent variables associated with \mathcal{X}^l ; this operation is possible, because the posterior distribution $p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi})$ is assumed to be known, provided that $\boldsymbol{\xi}$ is known. It can be shown that maximizing this expectation is equivalent to maximizing the corresponding evidence function $p(\mathcal{X}; \boldsymbol{\xi})$ (see Problem 12.3 and Section 12.7). To this end, the EM algorithm builds on an iterative philosophy, initialized by an arbitrary value $\boldsymbol{\xi}^{(0)}$. Then it proceeds along the following steps.

The EM algorithm

- Expectation E-step: At the $(j + 1)$ th iteration, compute $p(\mathcal{X}^l | \mathcal{X}, \boldsymbol{\xi}^{(j)})$ and

$$\boxed{Q(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)}) = \mathbb{E} \left[\ln p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi}) \right]}, \quad (12.40)$$

where the expectation is taken with respect to $p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi}^{(j)})$.

- Maximization M-step: Determine $\boldsymbol{\xi}^{(j+1)}$ so that

$$\boxed{\boldsymbol{\xi}^{(j+1)} = \arg \max_{\boldsymbol{\xi}} Q(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)})}. \quad (12.41)$$

- Check for convergence according to a criterion. If it is not satisfied go back to step 1.

A possible convergence criterion is to check whether $\|\boldsymbol{\xi}^{(j+1)} - \boldsymbol{\xi}^{(j)}\| < \epsilon$, for some user-defined constant ϵ . The use of the EM algorithm presupposes that working with the joint PDF $p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi})$ is computationally tractable. This is, for example, the case when working within the exponential family of PDFs, where the E-step may require only the computation of a few statistics of the latent variables. The exponential family of distributions is a computationally convenient one and it is treated in more detail in Section 12.8.

Remarks 12.3.

- The EM algorithm was proposed and given its name in the seminal 1977 paper by Arthur Dempster, Nan Laird, and Donald Rubin [12]. The paper generalized previously published results, as for example [2,38], and had a significant impact as a powerful tool in statistics. The complete convergence proof was given in [47]. See, for example, [29] for a related discussion.
- It can be shown that the EM algorithm converges to an (in general, local) maximum of $p(\mathcal{X}; \boldsymbol{\xi})$, which was our original goal. The likelihood never decreases. The convergence is slower than the quadratic convergence of Newton-type searching techniques, although near an optimal point a speedup may be possible. However, the convergence of the algorithm is smooth and its complexity more attractive to Newton-type schemes, with no matrix inversions involved. The keen reader may obtain more information in, for example, [14,30,33,43].
- The EM algorithm can be modified to obtain the MAP estimate. To this end, the M-step is changed to (Problem 12.4)

$$\boldsymbol{\xi}^{(j+1)} = \arg \max_{\boldsymbol{\xi}} \left\{ \mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)}) + \ln p(\boldsymbol{\xi}) \right\}, \quad (12.42)$$

where $p(\boldsymbol{\xi})$ is the prior PDF associated with $\boldsymbol{\xi}$, if it is considered to be a random vector.

- The EM algorithm can be sensitive to the choice of the initial point $\boldsymbol{\xi}^{(0)}$. In practice, one can run the algorithm a number of times, starting from different initial points, and keep the best of the results. Other initialization procedures have also been used, depending on the application.
- *Missing data:* The EM algorithm can also be used to cope with cases where some of the values from the observed training data are missing. Missing values can be treated as hidden variables and maximization of the likelihood can be done by marginalizing over them. Such a procedure makes sense only if data are *missing at random*; that is, the cause of missing data is a random event and does not depend on the values of the unobserved samples.

12.5 LINEAR REGRESSION AND THE EM ALGORITHM

The Bayesian viewpoint to the regression task was considered in Section 12.2.3 via the Gaussian model assumption for $p(\mathbf{y}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$, given in (12.9) and (12.8), which subsequently led to a Gaussian posterior for $p(\boldsymbol{\theta}|\mathbf{y})$, given in (12.16). In the current section, and for the sake of presentation simplicity, we will adopt the special case of diagonal covariance matrices, that is, $\Sigma_\eta = \sigma_\eta^2 I$, $\Sigma_\theta = \sigma_\theta^2 I$, and $\boldsymbol{\theta}_0 = \mathbf{0}$.

Our goal now becomes to consider σ_η^2 and σ_θ^2 as (nonrandom) parameters and to obtain their values by maximizing the corresponding evidence function in (12.15). To this end, we will use the EM algorithm. Following the notation that we have adopted so far for the regression task, the observed variables are the outputs, \mathbf{y} , and the unobserved ones comprise the random parameters, $\boldsymbol{\theta}$, that define the regression model. Hence, in the current context, \mathbf{y} will replace \mathcal{X} and $\boldsymbol{\theta}$ will take the place of \mathcal{X}' in the general formulation of the EM algorithm in Section 12.4.1.

A prerequisite in order to apply the EM procedure is the knowledge of the posterior, which for this case is known, given the values of the parameters. We will work with the precision variables, and the

parameter vector becomes

$$\xi = [\alpha, \beta]^T, \quad \alpha = \frac{1}{\sigma_\theta^2} \quad \text{and} \quad \beta = \frac{1}{\sigma_\eta^2}.$$

The EM algorithm is initialized with some arbitrary positive values $\alpha^{(0)}$ and $\beta^{(0)}$. Then the algorithm at the $(j+1)$ th iteration step, where $\alpha^{(j)}$ and $\beta^{(j)}$ are assumed known, proceeds as follows:

- E-Step: Compute the posterior $p(\boldsymbol{\theta}|y; \xi^{(j)})$, which according to (12.16) and for $\boldsymbol{\theta}_0 = \mathbf{0}$ is fully specified if we compute its mean and covariance matrix, using (12.19) and (12.20), that is,

$$\Sigma_{\theta|y}^{(j)} = (\alpha^{(j)} I + \beta^{(j)} \Phi^T \Phi)^{-1}, \quad (12.43)$$

$$\boldsymbol{\mu}_{\theta|y}^{(j)} = \beta^{(j)} \Sigma_{\theta|y}^{(j)} \Phi^T \mathbf{y}. \quad (12.44)$$

Compute the expected value of the log-likelihood associated with the complete data set; this is given by

$$\ln p(\mathbf{y}, \boldsymbol{\theta}; \xi) := \ln p(\mathbf{y}, \boldsymbol{\theta}; \alpha, \beta) = \ln (p(\mathbf{y}|\boldsymbol{\theta}; \beta) p(\boldsymbol{\theta}; \alpha)),$$

or

$$\begin{aligned} \ln p(\mathbf{y}, \boldsymbol{\theta}; \alpha, \beta) &= \frac{N}{2} \ln \beta + \frac{K}{2} \ln \alpha - \frac{\beta}{2} \|\mathbf{y} - \Phi \boldsymbol{\theta}\|^2 - \frac{\alpha}{2} \boldsymbol{\theta}^T \boldsymbol{\theta} \\ &\quad - \left(\frac{N}{2} + \frac{K}{2} \right) \ln(2\pi). \end{aligned} \quad (12.45)$$

Treating the hidden parameters as random variables, the expected value of (12.45), with respect to $\boldsymbol{\theta}$, is carried out via the Gaussian posterior defined by (12.43) and (12.44). To this end, the following steps are adopted.

1. To compute $\mathbb{E}[\boldsymbol{\theta}^T \boldsymbol{\theta}]$, recall the definition of the respective covariance matrix,

$$\Sigma_{\theta|y}^{(j)} = \mathbb{E}[(\boldsymbol{\theta} - \boldsymbol{\mu}_{\theta|y}^{(j)})(\boldsymbol{\theta} - \boldsymbol{\mu}_{\theta|y}^{(j)})^T] \quad (12.46)$$

or

$$\mathbb{E}[\boldsymbol{\theta} \boldsymbol{\theta}^T] = \Sigma_{\theta|y}^{(j)} + \boldsymbol{\mu}_{\theta|y}^{(j)} \boldsymbol{\mu}_{\theta|y}^{(j)T}, \quad (12.47)$$

which results in

$$\begin{aligned} A := \mathbb{E}[\boldsymbol{\theta}^T \boldsymbol{\theta}] &= \mathbb{E}[\text{trace}\{\boldsymbol{\theta} \boldsymbol{\theta}^T\}] \\ &= \text{trace}\{\boldsymbol{\mu}_{\theta|y}^{(j)} \boldsymbol{\mu}_{\theta|y}^{(j)T} + \Sigma_{\theta|y}^{(j)}\} \\ &= \|\boldsymbol{\mu}_{\theta|y}^{(j)}\|^2 + \text{trace}\{\Sigma_{\theta|y}^{(j)}\}. \end{aligned} \quad (12.48)$$

2. To compute $\mathbb{E}[\|y - \Phi\theta\|^2]$, define $\psi := y - \Phi\theta$, and use the previous rationale to compute $\mathbb{E}[\psi^T \psi]$, which leads to (Problem 12.5)

$$B := \mathbb{E}[\|y - \Phi\theta\|^2] = \|y - \Phi\mu_{\theta|y}^{(j)}\|^2 + \text{trace}\left\{\Phi\Sigma_{\theta|y}^{(j)}\Phi^T\right\}. \quad (12.49)$$

Hence,

$$\mathcal{Q}(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}) = \frac{N}{2} \ln \beta + \frac{K}{2} \ln \alpha - \frac{\beta}{2} B - \left(\frac{N}{2} + \frac{K}{2}\right) \ln(2\pi). \quad (12.50)$$

- M-Step: Compute

$$\begin{aligned} \alpha^{(j+1)} : \frac{\partial}{\partial \alpha} \mathcal{Q}(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}) &= 0, \\ \beta^{(j+1)} : \frac{\partial}{\partial \beta} \mathcal{Q}(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}) &= 0, \end{aligned}$$

which trivially lead to

$$\alpha^{(j+1)} = \frac{K}{\|\mu_{\theta|y}^{(j)}\|^2 + \text{trace}\left\{\Sigma_{\theta|y}^{(j)}\right\}}, \quad (12.51)$$

$$\beta^{(j+1)} = \frac{N}{\|y - \Phi\mu_{\theta|y}^{(j)}\|^2 + \text{trace}\left\{\Phi\Sigma_{\theta|y}^{(j)}\Phi^T\right\}}. \quad (12.52)$$

Once the algorithm converges, the resulting values for α and β are used to completely specify the involved PDFs, which can be used either to obtain an estimate of $\hat{\theta}$, for example, $\hat{\theta} = \mathbb{E}[\theta|y]$, or make predictions via (12.21).

Example 12.2. In this example, the generalized linear regression model of Example 12.1 is reconsidered. The goal is to use the EM algorithm of Section 12.5, as summarized by the recursions (12.43), (12.44), (12.51), and (12.52). The variance of the Gaussian noise used in the model to generate the data was set equal to $\sigma_\eta^2 = 0.05$. The number of training points was $N = 500$. For the EM algorithm, both α and β were initialized to one. The correct dimensionality for the unknown parameter vector was used. The recovered values after the convergence of the EM were $\alpha = 1.32$ corresponding to $\sigma_\theta^2 = 0.756$ and $\beta = 19.96$ corresponding to $\sigma_\eta^2 = 0.0501$. Note that the latter is very close to the true variance of the noise. Then, predictions of the output variable y were performed at 20 points, using (12.22) and the value of $\mu_{\theta|y}$ recovered by the EM algorithm, via (12.44).

Fig. 12.5A shows the predictions together with the associated error bars, computed from (12.23) using the values of σ_η^2 and σ_θ^2 obtained via the EM algorithm. Fig. 12.5B shows the convergence curve for σ_η^2 as a function of the number of iterations of the EM algorithm.

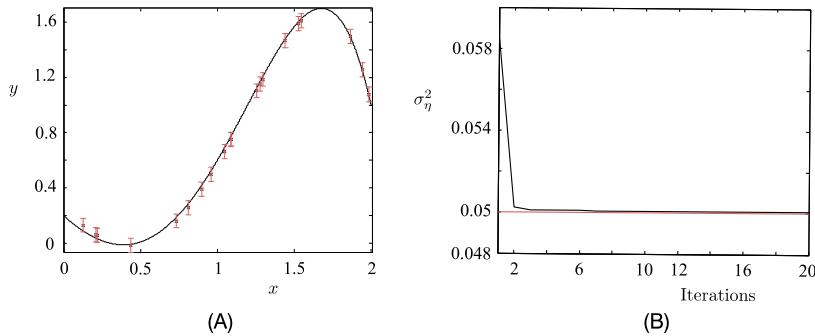


FIGURE 12.5

(A) The original graph from which the training points were sampled. In red, the respective predictions \hat{y} and associated error bars for 20 randomly chosen points are shown. (B) The convergence curve for σ_η^2 as a function of the iterations of the EM algorithm. The red line corresponds to the true value.

12.6 GAUSSIAN MIXTURE MODELS

So far, we have seen a number of PDFs that can be used to model the distribution of an unknown random vector $\mathbf{x} \in \mathbb{R}^l$. However, all these models restrict the PDF to a specific functional term. Mixture modeling provides the freedom to model the unknown PDF, $p(\mathbf{x})$, as a linear combination of different distributions, that is,

$$p(\mathbf{x}) = \sum_{k=1}^K P_k p(\mathbf{x}|k), \quad (12.53)$$

where P_k is the parameter weighting the specific contributing PDF, $p(\mathbf{x}|k)$. To guarantee that $p(\mathbf{x})$ is a PDF, the weighting parameters must be nonnegative and add to one ($\sum_{k=1}^K P_k = 1$). The physical interpretation of (12.53) is that we are given a set of K distributions, $p(\mathbf{x}|k)$, $k = 1, 2, \dots, K$. Each observation \mathbf{x}_n , $n = 1, 2, \dots, N$, is drawn from one of these K distributions, but we are not told from which one. All we know is a set of parameters, P_k , $1, 2, \dots, K$, each one providing the probability that a sample has been drawn from the corresponding PDF, $p(\mathbf{x}|k)$. It can be shown that for a large enough number of mixtures, K , and appropriate choice of the involved parameters, one can approximate arbitrarily close any continuous PDF.

Mixture modeling is a typical task involving latent variables; that is, the labels k of the PDF from which an obtained observation has originated. In practice, each $p(\mathbf{x}|k)$ is chosen from a known PDF family, parameterized via a set of parameters, and (12.53) can be rewritten as

$$p(\mathbf{x}) = \sum_{k=1}^K P_k p(\mathbf{x}|k; \xi_k), \quad (12.54)$$

and the task is to estimate (P_k, ξ_k) , $k = 1, 2, \dots, K$, based on a set of observations \mathbf{x}_n , $n = 1, 2, \dots, N$. The set of observations $\mathcal{X} = \{\mathbf{x}_n, n = 1, \dots, N\}$ forms the incomplete set while the complete set $\{\mathcal{X}, \mathcal{K}\}$ comprises the sample pairs (\mathbf{x}_n, k_n) , $n = 1, \dots, N$, with k_n being the label of the

PROBABILISTIC GRAPHICAL MODELS: PART I

15

CONTENTS

15.1 Introduction	771
15.2 The Need for Graphical Models	772
15.3 Bayesian Networks and the Markov Condition	774
15.3.1 Graphs: Basic Definitions	775
15.3.2 Some Hints on Causality	779
15.3.3 <i>d</i> -Separation	781
15.3.4 Sigmoidal Bayesian Networks	785
15.3.5 Linear Gaussian Models	786
15.3.6 Multiple-Cause Networks	786
15.3.7 I-Maps, Soundness, Faithfulness, and Completeness	787
15.4 Undirected Graphical Models	788
15.4.1 Independencies and I-Maps in Markov Random Fields	790
15.4.2 The Ising Model and Its Variants	791
15.4.3 Conditional Random Fields (CRFs)	794
15.5 Factor Graphs	795
15.5.1 Graphical Models for Error Correcting Codes	797
15.6 Moralization of Directed Graphs	798
15.7 Exact Inference Methods: Message Passing Algorithms	799
15.7.1 Exact Inference in Chains	799
15.7.2 Exact Inference in Trees	803
15.7.3 The Sum-Product Algorithm	804
15.7.4 The Max-Product and Max-Sum Algorithms	809
Problems	816
References	818

15.1 INTRODUCTION

In Fig. 13.2, we used a graphical description to indicate conditional dependencies among various parameters that control the “fusion” of the prior and conditional PDFs in a hierarchical manner. Our purpose there was more of a pedagogical nature; we could live without it. In this chapter, graphical models emerge out of necessity. In many everyday machine learning applications involving multivariate statistical modeling, even simple inference tasks can easily become computationally intractable. Typical applications involve bioinformatics, speech recognition, machine vision, and text mining, to name but a few.

Graph theory has proved a powerful and elegant tool that has extensively been used in optimization and computational theory. A graph encodes dependencies among interacting variables and can be used

to formalize the probabilistic structure that underlies our modeling assumptions. This can then be used to facilitate computations in a number of inference tasks, such as the calculation of marginals, modes, and conditional probabilities. Moreover, graphical models can be used as a vehicle to impose approximations onto the models when computational needs go beyond the available resources.

Early celebrated examples of the use of such models in learning tasks are the hidden Markov models, Kalman filtering, and error correcting coding, which have been popular since the early 1960s.

This is the first of two chapters dedicated to probabilistic graphical models. This chapter focuses on the basic definitions and concepts, and most of its material is a must for a first reading on the topic. A number of basic graphical models are discussed, such as Bayesian networks (BNs) and Markov random fields (MRFs). Exact inference is presented, and the elegant message passing algorithm for inference on chains and trees is introduced.

15.2 THE NEED FOR GRAPHICAL MODELS

Let us consider a simplified example of a learning system in the context of a medical application. Such a system comprises a set of m diseases that correspond to hidden variables and a set of n symptoms (findings). The diseases are treated as random variables, d_1, d_2, \dots, d_m , and each of them can be absent or present and thus can be encoded by a zero or a one, that is, $d_j \in \{0, 1\}$, $j = 1, 2, \dots, m$. The same applies to the symptoms f_i , which can either be absent or present; hence, $f_i \in \{0, 1\}$, $i = 1, 2, \dots, n$. The symptoms comprise the observed variables.¹

The goal of the system is to predict a disease hypothesis, that is, the presence of a number of diseases, given the presence of a set of symptoms which have been observed. During the training, which is based on experts' assessments, the system learns the prior probabilities $P(d_j)$ and the conditional probabilities $P_{ij} = P(f_i = 1|d_j = 1)$, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, m$. The latter comprise a table of nm entries. For a realistic system, these numbers can be very large. For example, in [41], m is of the order of 500–600 and n of the order of 4000. Let \mathbf{f} be the vector that corresponds to a specific set of observations for the findings, indicating the presence or absence of the respective symptoms. Assuming that symptoms are *conditionally independent*, given any disease hypothesis, \mathbf{d} , we can write

$$P(\mathbf{f}|\mathbf{d}) = \prod_{i=1}^n P(f_i|\mathbf{d}). \quad (15.1)$$

Ideally, one should be able to obtain the conditional probabilities $P(f_i|\mathbf{d})$ for each disease hypothesis. However, for all possible 2^m combinations of \mathbf{d} , this should require a huge amount of training data, which is impossible to collect for any practical system. This is bypassed by adopting the following model:

$$P(f_i = 0|\mathbf{d}) = \prod_{j=1}^m (1 - P_{ij})^{d_j}, \quad (15.2)$$

where the exponent is set to zero, $d_j = 0$, when the disease is not related to the symptom. This is known as the *noisy-OR* model. That is, it is assumed that for a negative finding the individual causes

¹ In a more realistic system, some of the findings may not be available, that is, they may be unobservable.

are independent [37]. Obviously,

$$P(f_i = 1|\mathbf{d}) = 1 - P(f_i = 0|\mathbf{d}).$$

Let us now assume that we observe a set of findings, \mathbf{f} , and we want to infer $P(d_j|\mathbf{f})$ for some j . Then

$$\begin{aligned} P(d_j = 1|\mathbf{f}) &= \frac{P(\mathbf{f}|d_j = 1)P(d_j = 1)}{P(\mathbf{f})} \\ &= \frac{\sum_{\mathbf{d}:d_j=1} P(\mathbf{f}|\mathbf{d})P(\mathbf{d})}{\sum_{\mathbf{d}} P(\mathbf{f}|\mathbf{d})P(\mathbf{d})}. \end{aligned} \quad (15.3)$$

The summation in the denominator involves 2^m terms. For $m \sim 500$, this is a formidable task that simply cannot be carried out in a realistic time.

The previous example indicates that once one gets involved with complex systems, even innocent looking tasks turn out to be computationally intractable. Thus, one has either to be more clever in exploiting possible independencies in the data, which can reduce the required number of computations, or make certain assumptions/approximations. In this chapter, we will study both alternatives.

Before we proceed further, it is interesting to point out another source of computational obstacles besides the calculation of Eq. (15.3). In practice, it may be more convenient to perform addition instead of implementing multiplication; multiplying a large number of variables of small values such as probabilities may cause arithmetic accuracy problems. One way to bypass products is either via logarithmic or exponential operations, which transform products into summations. For example, Eq. (15.2) can be rewritten as

$$P(f_i = 0|\mathbf{d}) = \exp\left(-\sum_{j=1}^m \theta_{ij} d_j\right), \quad (15.4)$$

where $\theta_{ij} := -\ln(1 - P_{ij})$ and

$$P(f_i = 1|\mathbf{d}) = 1 - \exp\left(-\sum_{j=1}^m \theta_{ij} d_j\right). \quad (15.5)$$

Observe that the presence in Eq. (15.1) of terms corresponding to negative findings contributes linearly to the complexity (product of exponentials correspond to summations). However, this is not the case with the terms associated with positive findings. Take, for example, the extreme case where all findings are negative. Then

$$\begin{aligned} P(\mathbf{f} = \mathbf{0}|\mathbf{d}) &= \prod_{i=1}^n \exp\left(-\sum_{j=1}^m \theta_{ij} d_j\right) \\ &= \exp\left(-\sum_{i=1}^n \left(\sum_{j=1}^m \theta_{ij} d_j\right)\right). \end{aligned} \quad (15.6)$$

Consider now $f_1 = 1$ and the rest to be $f_i = 0, i = 2, \dots, n$. Then

$$P(f|\mathbf{d}) = \left(1 - \exp\left(-\sum_{j=1}^m \theta_{1j} d_j\right)\right) \exp\left(-\sum_{i=2}^n \left(\sum_{j=1}^m \theta_{ij} d_j\right)\right), \quad (15.7)$$

where the number of exponents to be computed is two. It can easily be shown that the cross-product terms lead to an exponential computational growth [20] (Problem 15.1).

The path we will follow in order to derive efficient exact inference algorithms as well as to derive efficient approximation rules, when exact inference is not possible, will be via the use of graphical models.

15.3 BAYESIAN NETWORKS AND THE MARKOV CONDITION

Before we move on to definitions, let us first see how the existence of some structure in a joint distribution can simplify the task of marginalization. We will demonstrate it using discrete probabilities, where the use of counting can make things simpler.

Let us consider l discrete jointly distributed random variables. Applying the product rule of probability, we obtain

$$P(x_1, x_2, \dots, x_l) = P(x_l|x_{l-1}, x_{l-2}, \dots, x_1) P(x_{l-1}|x_{l-2}, \dots, x_1) \dots P(x_1). \quad (15.8)$$

Assume that each one of these variables takes values in the discrete set $\{1, 2, \dots, k\}$. In the general case, if we want to marginalize with respect to one of the variables, say, x_1 , we must sum over the others, that is,

$$P(x_1) = \sum_{x_2} \dots \sum_{x_l} P(x_1, x_2, \dots, x_l),$$

where each one of the summations is over k possible values, which is equivalent to $\mathcal{O}(k^l)$ summations; for large values of k and/or l , this is a formidable and sometimes impossible task. Let us consider now one extreme case, where all the involved variables are mutually independent. Then the product rule becomes

$$P(x_1, x_2, \dots, x_l) = \prod_{i=1}^l P(x_i),$$

and marginalization turns out to be the trivial identity

$$P(x_1) = \left(\sum_{x_l} P(x_l) \sum_{x_{l-1}} P(x_{l-1}) \dots \sum_{x_2} P(x_2) \right) P(x_1), \quad (15.9)$$

because each summation is carried out independently, and of course results to one. In other words, exploiting the product rule and the statistical independence can bypass the obstacle of the exponential growth of the computational load. As a matter of fact, the previous full-independence assumption gives birth to the naive Bayes classifier (Chapter 7).

In this chapter, we are going to study cases that lie between the previous two extremes. The general idea is to be able to express the joint probability distribution (probability density/mass function) in terms of *products* of factors, where each one of them depends on a *subset* of the involved variables. This can be expressed by writing the joint distribution as

$$p(x_1, x_2, \dots, x_l) = \prod_{i=1}^l p(x_i | \text{Pa}_i), \quad (15.10)$$

where Pa_i denotes the subset of variables associated with the random variable x_i . Take the following example:

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = p(x_6 | x_4) p(x_5 | x_3, x_4) p(x_4 | x_1, x_2) p(x_3 | x_1) p(x_2) p(x_1). \quad (15.11)$$

Then $\text{Pa}_6 = \{x_4\}$, $\text{Pa}_5 = \{x_3, x_4\}$, $\text{Pa}_4 = \{x_1, x_2\}$, $\text{Pa}_3 = \{x_1\}$, $\text{Pa}_2 = \emptyset$, $\text{Pa}_1 = \emptyset$. The variables in the set, Pa_i , are defined as the *parents* of the respective x_i , and from a statistical point of view this means that x_i is statistically independent of *all* the variables *given* the values of its parents. Every $p(x_i | \text{Pa}_i)$ expresses a *conditional independence* relationship and it imposes a *probabilistic structure* that underlies our multivariate set. It is such types of independencies that we will exploit in order to perform inference tasks at a lower computational cost.

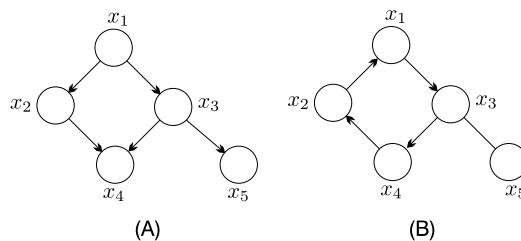
15.3.1 GRAPHS: BASIC DEFINITIONS

A graph $G = \{V, E\}$ is a collection of nodes/vertices $V = \{x_1, \dots, x_l\}$ and a collection of edges (arcs) $E \subset V \times V$. Each edge connects two vertices and it is denoted as a pair, $(x_i, x_j) \in E$. An edge can be either *directed*—then we write $(x_i \rightarrow x_j)$ to indicate the direction—or *undirected*—then we simply write (x_i, x_j) . Suppose we have a set of nodes x_1, x_2, \dots, x_k , $k \geq 2$, and a corresponding set of edges $(x_{i-1}, x_i) \in E$ or $(x_{i-1} \rightarrow x_i) \in E$, $2 \leq i \leq k$; that is, the edges connect pairs of nodes in *sequence* and they can be either directed or not. This sequence of edges is called a *path* from x_1 to x_k . If there is at least one directed edge, the path is called *directed*. A *cycle* is a path from a node to itself. A *chain* or a *trail* is a path that can be “run” either from x_1 to x_k or from x_k to x_1 ; that is, all directed edges are replaced by undirected ones.

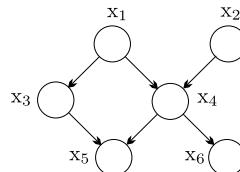
A *directed* graph comprises directed edges only, and it is called a *directed acyclic graph* (DAG) if it contains *no* cycles. Given a DAG, a node in it, x_i , is called *parent* of x_j if there is a (directed) edge from x_i to x_j , and we call x_j the *child* of x_i . A node x_j is called a *descendant* of x_i and x_i an *ancestor* of x_j if there is a path from x_i to x_j . A node x_j is called *nondescendant* of x_i if it is not a descendant of x_i . A graph is said to be *fully connected* or *complete* if there is an edge between every pair of nodes. Fig. 15.1 illustrates the previous definitions.

Definition 15.1. A *Bayesian network structure* is a DAG whose nodes represent random variables, x_1, \dots, x_l , and every variable (node), x_i , is *conditionally independent* of the set of *all* its nondescendants, given the set of all its *parents*. Sometimes this is also known as the *Markov condition*.

If we denote the set of the nondescendants of a node x_i as ND_i , the Markov condition can be written as [12] $x_i \perp ND_i | \text{Pa}_i$, $\forall i = 1, 2, \dots, l$. Sometimes, the conditional independencies are also known as *local independencies*. Stated differently, a BN graphical structure is a convenient way to encode

**FIGURE 15.1**

(A) This is a DAG because there are no cycles; x_1 is a parent of both x_2 and x_3 ; x_4 and x_5 are children of x_3 ; x_1 , x_2 , and x_3 are ancestors of x_4 , while x_4 and x_5 are descendants of x_1 ; x_5 is a nondescendant of x_2 and x_4 . (B) This is not a DAG and the sequence $(x_2, x_1, x_3, x_4, x_2)$ comprises a cycle. The edge (x_3, x_5) is undirected. The sequence of nodes (x_1, x_3, x_5) forms a directed path, and the sequence (x_1, x_2, x_4, x_3) forms a chain, once directed edges are replaced by undirected ones.

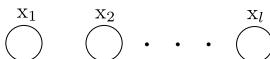
**FIGURE 15.2**

The BN structure corresponding to the PDF in Eq. (15.11). Observe that x_5 is conditionally independent of x_1 and x_2 given the values of x_3 and x_4 . Note that nodes in a BN structure correspond to random variables.

conditional independencies. Fig. 15.2 shows the DAG that expresses the conditional independencies used in Eq. (15.11), in order to express the joint distribution as a product of factors. Conditional independence among random variables, that is, $x \perp\!\!\!\perp y | z$, or, equivalently, $p(x|y, z) = p(x|z)$, means that once we know the value of z , observing the value of y gives no additional information about x (note that the previous makes sense only if $p(y, z) > 0$). For example, the probability of children having a good education depends on whether they grow up in a poor or a rich (low or high gross national product [GNP]) country. The probability of someone getting a high-paying job depends on her/his level of education. The probability of someone getting a high-paying job is independent of the country in which he or she was born and raised, given the level of her/his education.

Theorem 15.1. *Let G be a BN structure and let p be the joint probability distribution of the random variables associated with the graph. Then p is equal to the product of the conditional distributions of all the nodes given the values of their parents, and we say that p factorizes over G .*

The proof of the theorem is done by induction (Problem 15.2). Moreover, the reverse of this theorem is also true. The previous theorem assumed a distribution and built the BN based on the underlying conditional independencies. The next theorem deals with the reverse procedure. One builds a graph based on a set of conditional distributions—one for each node of the network.

**FIGURE 15.3**

Bayesian network structure for independent variables. No edges are present because every variable is independent of all the others and no parents can be identified.

Theorem 15.2. *Let G be a DAG and associate a conditional probability for each node, given the values of its parents. Then the product of these conditional probabilities yields a joint probability of the variables. Moreover, the Markov condition is satisfied.*

The proof of this theorem is given in Problem 15.4. Note that in this theorem, we used the term probability and not distribution. The reason is that the theorem is not true for every form of conditional densities (PDFs) [14]. However, it holds true for a number of widely used PDFs, such as the Gaussians. This theorem is very useful because, often in practice, this is the way we construct a probabilistic graphical model—building it hierarchically, using reasoning on the corresponding physical process that we want to model, and encoding conditional independencies in the graph.

Fig. 15.3 shows the BN structure describing a set of mutually independent variables (naive Bayes assumption).

Definition 15.2. A *Bayesian network* (BN) is a pair (G, p) , where the distribution p factorizes over the DAG G , in terms of a set of conditional probability distributions, associated with the nodes of G .

In other words, a BN is associated with a specific distribution. In contrast, a BN structure refers to any distribution that satisfies the Markov condition as expressed by the network structure.

Example 15.1. Consider the following simplified study relating the GNP of a country to the level of education and the type of a job an adult gets later in her/his professional life. Variable x_1 is binary with two values, HGP and LGP, corresponding to countries with high and low GNP, respectively. Variable x_2 gets three values, NE, LE, and HE, corresponding to no education, low-level, and high-level education, respectively. Finally, variable x_3 gets also three possible values, UN, LP, HP, corresponding to unemployed, low-paying, and high-paying jobs, respectively. Using a large enough sample of data, the following probabilities are learned:

1. Marginal probabilities:

$$P(x_1 = LGP) = 0.8, \quad P(x_1 = HGP) = 0.2.$$

2. Conditional probabilities:

$$\begin{aligned} P(x_2 = NE|x_1 = LGP) &= 0.1, \quad P(x_2 = LE|x_1 = LGP) = 0.7, \\ P(x_2 = HE|x_1 = LGP) &= 0.2, \\ P(x_2 = NE|x_1 = HGP) &= 0.05, \quad P(x_2 = LE|x_1 = HGP) = 0.2, \\ P(x_2 = HE|x_1 = HGP) &= 0.75, \\ P(x_3 = UN|x_2 = NE) &= 0.15, \quad P(x_3 = LP|x_2 = NE) = 0.8, \end{aligned}$$

$$\begin{aligned}
P(x_3 = HP|x_2 = NE) &= 0.05, \\
P(x_3 = UN|x_2 = LE) &= 0.10, \quad P(x_3 = LP|x_2 = LE) = 0.85, \\
P(x_3 = HP|x_2 = LE) &= 0.05, \\
P(x_3 = UN|x_2 = HE) &= 0.05, \quad P(x_3 = LP|x_2 = HE) = 0.15, \\
P(x_3 = HP|x_2 = HE) &= 0.8.
\end{aligned}$$

Note that these values are not the result of a specific experiment. However, they are in line with the general trend provided by more professional studies, which involve many more random variables. However, for pedagogical reasons we keep the example simple.

The first observation is that even for this simplistic example involving only three variables, one has to obtain 17 probability values. This verifies the high computational load that may be required for such tasks.

Fig. 15.4 shows the BN that captures the previously stated conditional probabilities. Note that the Markov condition renders x_3 independent of x_1 , given the value of x_2 . Indeed, the job that one finds is independent of the GNP of the country, given her/his education level. We will verify that by playing with the laws of probability for the previously defined values.

According to Theorem 15.2, the joint probability of an event is given by the product

$$P(x_1, x_2, x_3) = P(x_3|x_2)P(x_2|x_1)P(x_1). \quad (15.12)$$

In other words, the probability of someone coming from a rich country, having a good education, and getting a high-paying job will be equal to $(0.8)(0.75)(0.2) = 0.12$; similarly, the probability of somebody coming from a poor country, having a low-level education, and getting a low-paying job is 0.476.

As a next step, we will verify the Markov condition, implied by the BN structure, using the probability values given before. That is, we will verify that using conditional probabilities to build the network, these probabilities basically encode *conditional independencies*, as Theorem 15.2 suggests. Let us consider

$$\begin{aligned}
P(x_3 = HP|x_2 = HE, x_1 = HGP) &= \frac{P(x_3 = HP, x_2 = HE, x_1 = HGP)}{P(x_2 = HE, x_1 = HGP)} \\
&= \frac{0.12}{P(x_2 = HE, x_1 = HGP)}.
\end{aligned}$$

Also,

$$\begin{aligned}
P(x_2 = HE, x_1 = HGP) &= P(x_2 = HE|x_1 = HGP)P(x_1 = HGP) \\
&= 0.75 \times 0.2 = 0.15,
\end{aligned}$$

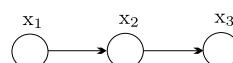


FIGURE 15.4

BN for Example 15.1. Note that $x_3 \perp\!\!\!\perp x_1 | x_2$.

which finally results to

$$\begin{aligned} P(x_3 = HP | x_2 = HE, x_1 = HGP) &= 0.8 \\ &= P(x_3 = HP | x_2 = HE), \end{aligned}$$

which verifies the claim. The reader can check that this is true for all possible combinations of values.

15.3.2 SOME HINTS ON CAUSALITY

The existence of directed links in a Bayesian network *does not* necessarily reflect a cause–effect relationship from a parent to a child node.² It is a well-known fact in statistics that correlation between two variables does not always establish a causal relationship between them. For example, their correlation may be due to the fact that they both relate to a latent (unknown) variable. A typical example is the discussion related to whether smoking causes cancer or they are both due to an unobserved genotype that causes cancer and at the same time a craving for nicotine; for many years, this argument was used as a defense line of the tobacco companies.

Let us return to Example 15.1. Although GNP and quality of education are correlated, one cannot say that GNP is a cause of the educational system. No doubt there is a multiplicity of reasons, such as the political system, the social structure, the economic system, historical reasons, and tradition, all of which need to be taken into consideration. As a matter of fact, the structure of the graph relating the three variables in the example could be reversed. We could collect data the other way around; obtain the probabilities $P(x_3 = UN)$, $P(x_3 = LP)$, $P(x_3 = HP)$, and then the conditional probabilities $P(x_2 | x_3)$ (e.g., $P(x_2 = HE | x_3 = UN)$) and finally $P(x_1 | x_2)$ (e.g., $P(x_1 = HGP | x_2 = HE)$). In principle, such data can also be collected from a sample of people. In such a case, the resulting BN would comprise again three nodes as in Fig. 15.4, but with the direction of the arrows reversed. This is also reasonable because the probability of someone coming from a rich or a poor country is independent of her/his job, given the level of education. Moreover, both models should result in the same joint probability distribution for any joint event. Thus, if the direction of the arrows were to indicate causality, then this time, it would be that the educational system has a cause–effect relationship on the GNP. This, for the same reasons stated before, cannot be justified. Having said all that, it does not necessarily mean that cause–effect relationships are either absent in a BN or it is not important to know them. On the contrary, in many cases, there is good reason to strive to unveil the underlying cause–effect relationships while building a BN.

Let us elaborate a bit more on this and see why exploiting any underlying cause–effect relationships can be to our benefit. Take, for example, the BN in Fig. 15.5 relating the presence or absence of a disease with the findings from two medical tests. Let x_1 indicate the presence or absence of a disease and x_2, x_3 the discrete outcomes that can result from the two tests.

The BN in Fig. 15.5A complies with our common sense reasoning that x_1 (disease) causes x_2 and x_3 (tests). However, this is not possible to deduce by *simply* looking at the available probabilities. This is because the probability laws are *symmetric*. Even if x_1 is the cause, we can still compute $P(x_1 | x_2)$

² This topic will not be pursued any further; its purpose is to make the reader aware of the issue. It can be bypassed in a first reading.

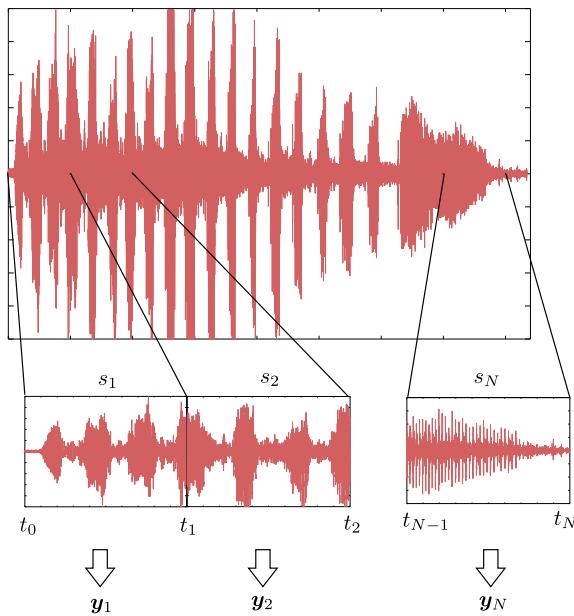
Remarks 16.3.

- Following the success of loopy belief propagation in turbo decoding, further research verified its performance potential in a number of tasks, such as low-density parity-check codes [15,47], network diagnostics [48], sensor network applications [24], and multiuser communications [70]. Furthermore, a number of modified versions of the basic scheme have been proposed. In [74], the so-called tree-reweighted belief propagation is proposed. In [26], arguments from information geometry are employed and in [78], projection arguments in the context of information geometry are used. More recently, the belief propagation algorithm and the mean field approximation were proposed to be optimally combined to exploit their respective advantages [65]. A related review can be found in [76]. In a nutshell, this old scheme is still alive and kicking!
- In Section 13.10, the expectation propagation algorithm was discussed in the context of parameter inference. The scheme can also be adopted in the more general framework of graphical models, if the place of parameters is taken by the hidden variables. Graphical models are particularly tailored for this approach because the joint PDF is factorized. It turns out that if the approximate PDF is completely factorized, corresponding to a partially disconnected network, the expectation propagation algorithm turns out to be the loopy belief propagation algorithm [50]. In [51], it is shown that a new family of message passing algorithms can be obtained by utilizing a generalization of the KL divergence as the optimizing cost. This family encompasses a number of previously developed schemes.
- Besides the approximation techniques that were previously presented, another popular pool of methods is the Markov chain Monte Carlo (MCMC) framework. Such techniques were discussed in Chapter 14 (see, for example, [25] and the references therein).

16.4 DYNAMIC GRAPHICAL MODELS

All the graphical models that have been discussed so far were developed to serve the needs of random variables whose statistical properties remained fixed over time. However, this is not always the case. As a matter of fact, the terms *time adaptivity* and *time variation* are central for most parts of this book. Our focus in this section is to deal with random variables whose statistical properties are not fixed but are allowed to undergo changes. A number of time series as well as sequentially obtained data fall under this setting with applications ranging from signal processing and robotics to finance and bioinformatics.

A key difference here, compared to what we have discussed in the previous sections of this chapter, is that now observations are sensed sequentially and the *specific sequence* in which they occur carries important information, which has to be respected and exploited in any subsequent inference task. For example, in speech recognition, the sequence in which the feature vectors result is very important. In a typical speech recognition task, the raw speech data are *sequentially* segmented in short (usually overlapping) time windows and from each window a feature vector is obtained (e.g., DFT of the samples in the respective time slot). This is illustrated in Fig. 16.14. These feature vectors constitute the observation sequence. Besides the information that resides in the specific values of these observation vectors, the sequence in which the observations appear discloses important information about the word that is spoken; our language and spoken words are highly structured human activities. Similar argu-

**FIGURE 16.14**

A speech segment and N time windows, each one of length equal to 500 ms. They correspond to time intervals $[0, 500]$, $[500, 1000]$, and $[3500, 4000]$, respectively. From each one of them, a feature vector, \mathbf{y} , is generated. In practice, an overlap between successive windows is allowed.

ments hold true for applications such as learning and reasoning concerning biological molecules, for example, DNA and proteins.

Although any type of graphical model has its dynamic counterpart, we will focus on the family of *dynamic Bayesian networks* and, in particular, a specific type known as *hidden Markov models*.

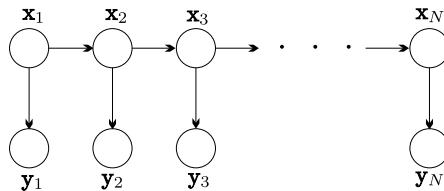
A very popular and effective framework to model sequential data is via the so-called *state-observation* or *state-space* models. Each set of random variables, $\mathbf{y}_n \in \mathbb{R}^l$, which are observed at time n , is associated with a corresponding hidden/latent random vector \mathbf{x}_n (not necessarily of the same dimensionality as that of the observations). The system dynamics are modeled via the latent variables and observations are considered to be the output of a measuring *noisy* sensing device. The so-called *latent Markov models* are built around the following two independence assumptions:

$$(1) \quad \mathbf{x}_{n+1} \perp (\mathbf{x}_1, \dots, \mathbf{x}_{n-1}) \mid \mathbf{x}_n, \quad (16.24)$$

$$(2) \quad \mathbf{y}_n \perp (\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{x}_{n+1}, \dots, \mathbf{x}_N) \mid \mathbf{x}_n, \quad (16.25)$$

where N is the total number of observations. The first condition defines the system dynamics via the transition model

$$p(\mathbf{x}_{n+1} \mid \mathbf{x}_1, \dots, \mathbf{x}_n) = p(\mathbf{x}_{n+1} \mid \mathbf{x}_n), \quad (16.26)$$

**FIGURE 16.15**

The Bayesian network corresponding to a latent Markov model. If latent variables are of a discrete nature, this corresponds to an HMM. If both observed and latent variables are continuous and follow a Gaussian distribution, this corresponds to a linear dynamic system (LDS). Note that the observed variables comprise the leaves of the graph.

and the second one via the observation model

$$p(y_n | x_1, \dots, x_N) = p(y_n | x_n). \quad (16.27)$$

In words, *the future is independent of the past given the present, and the observations are independent of the future and past given the present*.

The previously stated independencies are graphically represented via the graph of Fig. 16.15. If the hidden variables are of a discrete nature, the resulting model is known as a hidden Markov model. If, on the other hand, both hidden and observation variables are of a continuous nature, the resulting model gets rather involved to deal with. However, analytically tractable tools can be and have been developed for some special cases. In the so-called *linear dynamic systems* (LDSs), the system dynamics and the generation of the observations are modeled as

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \quad (16.28)$$

$$\mathbf{y}_n = H_n \mathbf{x}_n + \boldsymbol{v}_n, \quad (16.29)$$

where $\boldsymbol{\eta}_n$ and \boldsymbol{v}_n are zero mean, mutually independent noise disturbances modeled by Gaussian distributions. This is the celebrated Kalman filter, which we have already discussed in Chapter 4 and it will also be considered, from a probabilistic perspective, in Chapter 17. The probabilistic counterparts of Eqs. (16.28) and (16.29) are

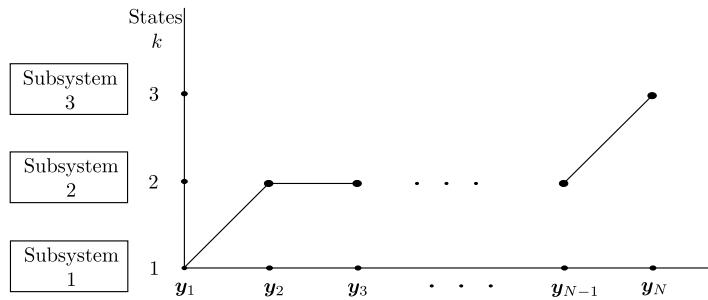
$$p(\mathbf{x}_n | \mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n | F_n \mathbf{x}_{n-1}, Q_n), \quad (16.30)$$

$$p(\mathbf{y}_n | \mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n | H_n \mathbf{x}_n, R_n), \quad (16.31)$$

where Q_n and R_n are the covariance matrices of $\boldsymbol{\eta}_n$ and \boldsymbol{v}_n , respectively.

16.5 HIDDEN MARKOV MODELS

Hidden Markov models are represented by the graphical model in Fig. 16.15 and Eqs. (16.26) and (16.27). The latent variables are discrete; hence, we write the *transition* probability as $P(x_n | x_{n-1})$ and this corresponds to a table of probabilities. Observation variables can either be discrete or continuous.

**FIGURE 16.16**

The unfolding in time of a trajectory that associates observations with states.

Basically, an HMM is used to model a *quasistationary* process that undergoes *sudden* changes among a number of, say, K subprocesses. Each one of these subprocesses is described by different statistical properties. One could alternatively view it as a combined system comprising a number of subsystems; each one of these subsystems generates data/observations according to a different statistical model; for example, one may follow a Gaussian and the other one a Student's t distribution. Observations are emitted by these subsystems; however, once an observation is received, we do not know which subsystem this was emitted from. This reminds us of the mixture modeling task of a PDF; however, in mixture modeling, we did not care about the sequence in which observations occur.

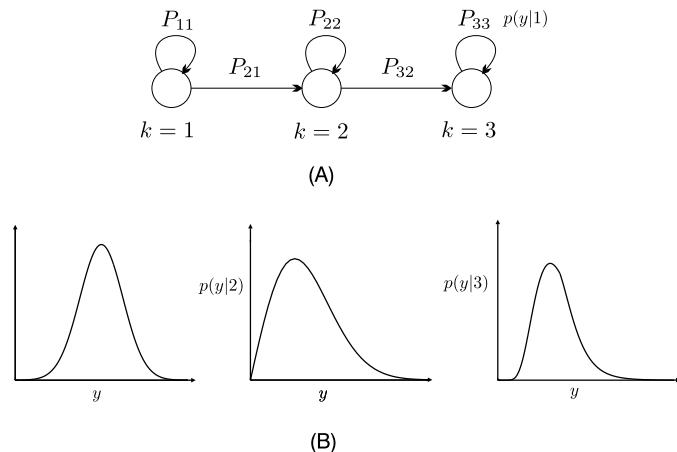
For modeling purposes, we associate with each observation y_n a hidden variable, $k_n = 1, 2, \dots, K$, which is the (random) index indicating the subsystem/subprocess that generated the respective observation vector. We will call it the *state*. Each k_n corresponds to \mathbf{x}_n of the general model. The sequence of the complete observation set (y_n, k_n) , $n = 1, 2, \dots, N$, forms a trajectory in a two-dimensional grid, having the states on one axis and the observations on the other. This is shown in Fig. 16.16 for $K = 3$. Such a path reveals the origin of each observation; y_1 was emitted from state $k_1 = 1$, y_2 from $k_2 = 2$, y_3 from $k_3 = 2$, and y_N from $k_N = 3$. Note that each trajectory is associated with a probability distribution, that is, the joint distribution of the complete set. Indeed, the probability that the trajectory of Fig. 16.16 will occur depends on the value of $P((y_1, k_1 = 1), (y_2, k_2 = 2), (y_3, k_3 = 2), \dots, (y_N, k_N = 3))$. We will soon see that some of the possible trajectories that can be drawn in the grid are not allowed in practice; this may be due to physical constraints concerning the data generation mechanism that underlies the corresponding system/process.

Transition probabilities. As already said, the dynamics of a latent Markov model are described in terms of the distribution $p(\mathbf{x}_n | \mathbf{x}_{n-1})$, which for an HMM becomes the set of probabilities

$$P(k_n | k_{n-1}), \quad k_n, k_{n-1} = 1, 2, \dots, K,$$

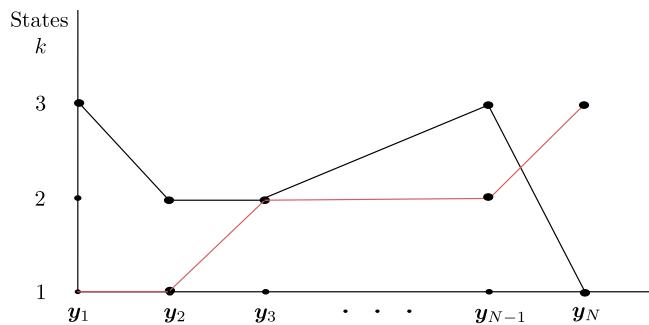
indicating the probability of the system to "jump" at time n to state k_n from state k_{n-1} , where it was at time $n - 1$. In general, this table of probabilities may be time-varying. In the standard form of HMMs, this is considered to be independent of time and we say that our model is *homogeneous*. Thus, we can write

$$P(k_n | k_{n-1}) = P(i | j) := P_{ij}, \quad i, j = 1, 2, \dots, K.$$

**FIGURE 16.17**

(A) A three-state left-to-right HMM model. (B) Each state is characterized by different statistical properties.

Note that some of these transition probabilities can be zero, depending on the modeling assumptions. Fig. 16.17A shows an example of a three-state system. The model is of the so-called *left-to-right* type, where two types of transitions are allowed: (a) self transitions and (b) transitions from a state of a lower index to a state of a higher index. The system, once it jumps into a state k , emits data according to a probability distribution $p(y|k)$, as illustrated in Fig. 16.17B. Besides the left-to-right models, other alternatives have also been proposed [8,63]. The states correspond to certain physical characteristics of the corresponding system. For example, in speech recognition, the number of states that are chosen to model a spoken word depends on the expected number of sound phenomena (phonemes) within the

**FIGURE 16.18**

The black trajectory is not allowed to occur under the HMM model of Fig. 16.17. Transitions from state $k = 3$ to state $k = 2$ and from $k = 3$ to $k = 1$ are not permitted. In contrast, the state unfolding in the red curve is in agreement with the model.

word. Typically, three to four states are used per phoneme. Another modeling path uses the average number of observations resulting from various versions of a spoken word as an indication of the number of states. Seen from the transition probabilities perspective, an HMM is basically a stochastic finite state automaton that generates an observation string. Note that the semantics of Fig. 16.17 is different from and must not be confused with the graphical structure given in Fig. 16.15. Fig. 16.17 is a graphical interpretation of the transition probabilities among the states; it says nothing about independencies among the involved random variables. Once a state transition model has been adopted, some trajectories in the trellis diagram of Fig. 16.16 will not be allowed. In Fig. 16.18, the red trajectory is not in line with the model of Fig. 16.17.

16.5.1 INFERENCE

As in any graphical modeling task, the ultimate goal is inference. Two types of inference are of particular interest in the context of classification/recognition. Let us discuss it in the framework of speech recognition; similar arguments hold true for other applications. We are given a set of (output variables) observations, $\mathbf{y}_1, \dots, \mathbf{y}_N$, and we have to decide to which spoken word these correspond. In the database, each spoken word is represented by an HMM model, which is the result of extensive training. An HMM model is fully described by the following set of parameters:

HMM model parameters

1. Number of states K .
 2. The probabilities for the initial state at $n = 1$ to be at state k , that is, P_k , $k = 1, 2, \dots, K$.
 3. The set of transition probabilities P_{ij} , $i, j = 1, 2, \dots, K$.
 4. The state emission distributions $p(\mathbf{y}|k)$, $k = 1, 2, \dots, K$, which can be either discrete or continuous.
- Often, these probability distributions may be parameterized, $p(\mathbf{y}|k; \theta_k)$, $k = 1, 2, \dots, K$.

Prior to inference, all the involved parameters are assumed to be known. Learning of the HMM parameters takes place in the training phase; we will come to it shortly.

For the recognition, a number of scores can be used. Here we will discuss two alternatives that come as a direct consequence of our graphical modeling approach. For a more detailed discussion, see, for example, [72].

In the first one, the joint distribution for the observed sequence is computed, after marginalizing out all hidden variables; this is done for each one of the models/words. Then the word that scores the larger value is selected. This method corresponds to the sum-product rule. The other path is to compute, for each model/word, the optimal trajectory in the trellis diagram; that is, the trajectory that scores the highest joint probability. In the sequel, we decide in favor of the model/word that corresponds to the largest optimal value. This method is an implementation of the max-sum rule.

The Sum-Product Algorithm: the HMM Case

The first step is to transform the directed graph of Fig. 16.15 to an undirected one; a factor graph or a junction tree graph. Note that this is trivial for this case, as the graph is already a tree. Let us work with the junction tree formulation. Also, in order to use the message passing formulas of (16.4) and (16.5) as well as (16.6) and (16.7) for computing the distribution values, we will first adopt a more compact way of representing the conditional probabilities. We will employ the technique that was used in Section 13.4 for the mixture modeling case. Let us denote each latent variable as a K -dimensional

vector, $\mathbf{x}_n \in \mathbb{R}^K$, $n = 1, 2, \dots, N$, whose elements are all zero except at the k th location, where k is the index of the (unknown) state from which y_n has been emitted, that is,

$$\mathbf{x}_n^T = [x_{n,1}, x_{n,2}, \dots, x_{n,K}] : \begin{cases} x_{n,i} = 0, & i \neq k \\ x_{n,k} = 1. \end{cases}$$

Then we can compactly write

$$P(\mathbf{x}_1) = \prod_{k=1}^K P_k^{x_{1,k}}, \quad (16.32)$$

and

$$P(\mathbf{x}_n | \mathbf{x}_{n-1}) = \prod_{i=1}^K \prod_{j=1}^K P_{ij}^{x_{n-1,j} x_{n,i}}. \quad (16.33)$$

Indeed, if the jump is from a specific state j at time $n-1$ to a specific state i at time n , then the only term that survives in the previous product is the corresponding factor, P_{ij} . The joint probability distribution of the complete set, as a direct consequence of the Bayesian network model of Fig. 16.15, is written as

$$p(Y, X) = P(\mathbf{x}_1) p(y_1 | \mathbf{x}_1) \prod_{n=2}^N P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(y_n | \mathbf{x}_n), \quad (16.34)$$

where

$$p(y_n | \mathbf{x}_n) = \prod_{k=1}^K (p(y_n | k; \boldsymbol{\theta}_k))^{x_{n,k}}. \quad (16.35)$$

The corresponding junction tree is trivially obtained from the graph in Fig. 16.15. Replacing directed links with undirected ones and considering cliques of size two, the graph in Fig. 16.19A results. However, as all the y_n variables are observed (*instantiated*) and no marginalization is required, their

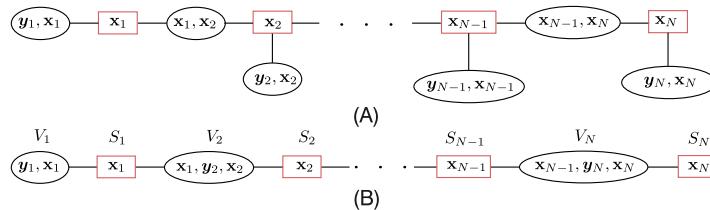


FIGURE 16.19

(A) The junction tree that results from the graph of Fig. 16.15. (B) Because y_n are observed, their effect is only of a multiplicative nature (no marginalization is involved) and its contribution can be trivially absorbed by the potential functions (distributions) associated with the latent variables.

multiplicative contribution can be absorbed by the respective conditional probabilities, which leads to the graph of Fig. 16.19B. Alternatively, this junction tree can be obtained if one considers the nodes $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{y}_n)$, $n = 2, 3, \dots, N$, to form cliques associated with the potential functions

$$\psi_1(\mathbf{x}_1, \mathbf{y}_1) = P(\mathbf{x}_1)p(\mathbf{y}_1|\mathbf{x}_1) \quad (16.36)$$

and

$$\psi_n(\mathbf{x}_{n-1}, \mathbf{y}_n, \mathbf{x}_n) = P(\mathbf{x}_n|\mathbf{x}_{n-1})p(\mathbf{y}_n|\mathbf{x}_n), \quad n = 2, \dots, N. \quad (16.37)$$

The junction tree of Fig. 16.19B results by eliminating nodes from the cliques starting from \mathbf{x}_1 . Note that the normalizing constant is equal to one, $Z = 1$.

To apply the sum-product rule for junction trees, Eq. (16.5) now becomes

$$\begin{aligned} \mu_{V_n \rightarrow S_n}(\mathbf{x}_n) &= \sum_{\mathbf{x}_{n-1}} \psi_n(\mathbf{x}_{n-1}, \mathbf{y}_n, \mathbf{x}_n) \mu_{S_{n-1} \rightarrow V_n}(\mathbf{x}_{n-1}) \\ &= \sum_{\mathbf{x}_{n-1}} \mu_{S_{n-1} \rightarrow V_n}(\mathbf{x}_{n-1}) P(\mathbf{x}_n|\mathbf{x}_{n-1}) p(\mathbf{y}_n|\mathbf{x}_n). \end{aligned}$$

Also,

$$\mu_{S_{n-1} \rightarrow V_n}(\mathbf{x}_{n-1}) = \mu_{V_{n-1} \rightarrow S_{n-1}}(\mathbf{x}_{n-1}). \quad (16.38)$$

Thus,

$$\mu_{V_n \rightarrow S_n}(\mathbf{x}_n) = \sum_{\mathbf{x}_{n-1}} \mu_{V_{n-1} \rightarrow S_{n-1}}(\mathbf{x}_{n-1}) P(\mathbf{x}_n|\mathbf{x}_{n-1}) p(\mathbf{y}_n|\mathbf{x}_n), \quad (16.39)$$

with

$$\mu_{V_1 \rightarrow S_1}(\mathbf{x}_1) = P(\mathbf{x}_1)p(\mathbf{y}_1|\mathbf{x}_1). \quad (16.40)$$

In the HMM literature, it is common to use the “alpha” symbol for the exchanged messages, that is,

$$\alpha(\mathbf{x}_n) := \mu_{V_n \rightarrow S_n}(\mathbf{x}_n). \quad (16.41)$$

If one considers that the message passing terminates at a node V_n , then based on (16.7), and taking into account that the variables $\mathbf{y}_1, \dots, \mathbf{y}_n$ are clumped to the observed values (recall the related comment following Eq. (15.44)), it is readily seen that

$$\alpha(\mathbf{x}_n) = p(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n, \mathbf{x}_n), \quad (16.42)$$

which can also be deduced by the respective definitions in (16.39) and (16.40); all hidden variables, except \mathbf{x}_n , have been marginalized out. This is a set of K probability values (one for each value of \mathbf{x}_n). For example, for $\mathbf{x}_n : \mathbf{x}_{n,k} = 1$, $\alpha(\mathbf{x}_n)$ is the probability of the trajectory to be at time n at state k and having obtained the specific observations up to and including time n . From (16.42), one can readily obtain the joint probability distribution (evidence) over the observation sequence, comprising N time

instants, that is,

$$p(Y) = \sum_{\mathbf{x}_N} p(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N, \mathbf{x}_N) = \sum_{\mathbf{x}_N} \alpha(\mathbf{x}_N) : \text{ evidence of observations,}$$

which, as said in the beginning of the section, is a quantity used for classification/recognition.

In the signal processing “jargon,” the computation of $\alpha(\mathbf{x}_n)$ is referred to as the *filtering recursion*. By the definition of $\alpha(\mathbf{x}_n)$, we have [2]

$$\alpha(\mathbf{x}_n) = \underbrace{p(\mathbf{y}_n | \mathbf{x}_n)}_{\text{corrector}} \cdot \underbrace{\sum_{\mathbf{x}_{n-1}} \alpha(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1})}_{\text{predictor}} : \text{ filtering recursion.} \quad (16.43)$$

As is the case with the Kalman filter, to be treated in Chapter 17, the only difference there will be that the summation is replaced by integration. Having adopted Gaussian distributions, these integrations translate into updates of the respective mean values and covariance matrices. The physical meaning of (16.43) is that the predictor provides a prediction on the state using all the past information prior to n . Then this information is corrected based on the observation \mathbf{y}_n , which is received at time n . Thus, the updated information, based on the entire observation sequence up to and including the current time n , is readily available by

$$P(\mathbf{x}_n | Y_{[1:n]}) = \frac{\alpha(\mathbf{x}_n)}{p(Y_{[1:n]}),}$$

where the denominator is given by $\sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$, and $Y_{[1:n]} := (\mathbf{y}_1, \dots, \mathbf{y}_n)$.

Let us now carry on with the second message passing phase, in the opposite direction than before, in order to obtain

$$\mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n) = \sum_{\mathbf{x}_{n+1}} \mu_{S_{n+1} \rightarrow V_{n+1}}(\mathbf{x}_{n+1}) P(\mathbf{x}_{n+1} | \mathbf{x}_n) p(\mathbf{y}_{n+1} | \mathbf{x}_{n+1}),$$

with

$$\mu_{S_{n+1} \rightarrow V_{n+1}}(\mathbf{x}_{n+1}) = \mu_{V_{n+2} \rightarrow S_{n+1}}(\mathbf{x}_{n+1}).$$

Hence,

$$\mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n) = \sum_{\mathbf{x}_{n+1}} \mu_{V_{n+2} \rightarrow S_{n+1}}(\mathbf{x}_{n+1}) P(\mathbf{x}_{n+1} | \mathbf{x}_n) p(\mathbf{y}_{n+1} | \mathbf{x}_{n+1}), \quad (16.44)$$

with

$$\mu_{V_{N+1} \rightarrow S_N}(\mathbf{x}_N) = 1. \quad (16.45)$$

Note that $\mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n)$ involves K values and for the computation of each one of them K summations are performed. So, the complexity scales as $\mathcal{O}(K^2)$ per time instant. In the HMM literature, the symbol “beta” is used,

$$\beta(\mathbf{x}_n) = \mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n). \quad (16.46)$$

From the recursive definition in (16.44) and (16.45), where $\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \dots, \mathbf{x}_N$ have been marginalized out, we can equivalently write

$$\beta(\mathbf{x}_n) = p(y_{n+1}, y_{n+2}, \dots, y_N | \mathbf{x}_n). \quad (16.47)$$

That is, conditioned on the values of \mathbf{x}_n , for example, $\mathbf{x}_n : x_{nk} = 1$, $\beta(\mathbf{x}_n)$ is the value of the joint distribution for the observed values, y_{n+1}, \dots, y_N , to be emitted when the system is at state k at time n .

We have now all the “ingredients” in order to compute marginals. From (16.7), we obtain (explain it based on the independence properties that underlie an HMM)

$$\begin{aligned} p(\mathbf{x}_n, \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N) &= \mu_{V_{n-1} \rightarrow S_n}(\mathbf{x}_n) \mu_{V_n \rightarrow S_n}(\mathbf{x}_n) \\ &= \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n), \end{aligned} \quad (16.48)$$

which in turns leads to

$$\gamma(\mathbf{x}_n) := P(\mathbf{x}_n | Y) = \frac{\alpha(\mathbf{x}_n) \beta(\mathbf{x}_n)}{p(Y)} : \text{ smoothing recursion.}$$

(16.49)

This part of the recursion is known as the *smoothing* recursion. Note that in this computation, both past (via $\alpha(\mathbf{x}_n)$) and future (via $\beta(\mathbf{x}_n)$) data are involved.

An alternative way to obtain $\gamma(\mathbf{x}_n)$ is via its own recursion together with $\alpha(\mathbf{x}_n)$, by avoiding $\beta(\mathbf{x}_n)$ (Problem 16.16). In such a scenario, both passing messages are related to densities with regard to \mathbf{x}_n , which has certain advantages for the case of linear dynamic systems.

Finally, from (16.6) and recalling (16.38), (16.41), and (16.46), we obtain

$$\begin{aligned} p(\mathbf{x}_{n-1}, \mathbf{x}_n, Y) &= P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(y_n | \mathbf{x}_n) \mu_{S_n \rightarrow V_n}(\mathbf{x}_n) \mu_{S_{n-1} \rightarrow V_{n-1}}(\mathbf{x}_{n-1}) \\ &= \alpha(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(y_n | \mathbf{x}_n) \beta(\mathbf{x}_n), \end{aligned} \quad (16.50)$$

or

$$\begin{aligned} p(\mathbf{x}_{n-1}, \mathbf{x}_n | Y) &= \frac{\alpha(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(y_n | \mathbf{x}_n) \beta(\mathbf{x}_n)}{p(Y)} \\ &:= \xi(\mathbf{x}_{n-1}, \mathbf{x}_n). \end{aligned} \quad (16.51)$$

Thus, $\xi(\cdot, \cdot)$ is a table of K^2 probability values. Let $\xi(x_{n-1,j}, x_{n,i})$ correspond to $x_{n-1,j} = x_{n,i} = 1$. Then $\xi(x_{n-1,j}, x_{n,i})$ is the probability of the system being at states j and i at times $n - 1$ and n , respectively, conditioned on the transmitted sequence of observations.

In Section 15.7.4 a message passing scheme was proposed for the efficient computation of the maximum of the joint distribution. This can also be applied in the junction tree associated with an HMM. The resulting algorithm is known as the *Viterbi* algorithm. The Viterbi algorithm results in a straightforward way from the general max-sum algorithm. The algorithm is similar to the one derived before; all one has to do is to replace summations with the maximum operations. As we have already commented, while discussing the max-product rule, computing the sequence of the complete set $(\mathbf{y}_n, \mathbf{x}_n)$,

$n = 1, 2, \dots, N$, that maximizes the joint probability, using back-tracking, equivalently defines the optimal trajectory in the two-dimensional grid.

Another inference task that is of interest in practice, besides recognition, is prediction, that is, given an HMM and the observation sequence $\mathbf{y}_n, n = 1, 2, \dots, N$, to optimally predict the value \mathbf{y}_{n+1} . This can also be performed efficiently by appropriate marginalization (Problem 16.17).

16.5.2 LEARNING THE PARAMETERS IN AN HMM

This is the second time we refer to the learning of graphical models. The first time was at the end of Section 16.3.2. The most natural way to obtain the unknown parameters is to maximize the likelihood/evidence of the joint probability distribution. Because our task involves both observed and latent variables, the EM algorithm is the first one that comes to mind. However, the underlying independencies in an HMM will be employed in order to come up with an efficient learning scheme. The set of the unknown parameters, Θ , involves (a) the initial state probabilities, $P_k, k = 1, \dots, K$, (b) the transition probabilities, $P_{ij}, i, j = 1, 2, \dots, K$, and (c) the parameters in the probability distributions associated with the observations, $\boldsymbol{\theta}_k, k = 1, 2, \dots, K$.

Expectation step: From the general scheme presented in Section 12.4.1 (with Y in place of \mathcal{X} , X in place of \mathcal{X}^l , and Θ in place of ξ) at the $(t + 1)$ th iteration, we have to compute

$$\mathcal{Q}(\Theta, \Theta^{(t)}) = \mathbb{E} [\ln p(Y, X; \Theta)],$$

where $\mathbb{E}[\cdot]$ is the expectation with respect to $P(X|Y; \Theta^{(t)})$. From (16.32)–(16.35) we obtain

$$\begin{aligned} \ln p(Y, X; \Theta) &= \sum_{k=1}^K (x_{1,k} \ln P_k + \ln p(y_1|k; \boldsymbol{\theta}_k)) \\ &\quad + \sum_{n=2}^N \sum_{i=1}^K \sum_{j=1}^K (x_{n-1,j} x_{n,i}) \ln P_{ij} \\ &\quad + \sum_{n=2}^N \sum_{k=1}^K x_{n,k} \ln p(y_n|k; \boldsymbol{\theta}_k), \end{aligned}$$

thus,

$$\begin{aligned} \mathcal{Q}(\Theta, \Theta^{(t)}) &= \sum_{k=1}^K \mathbb{E}[x_{1,k}] \ln P_k + \sum_{n=2}^N \sum_{i=1}^K \sum_{j=1}^K \mathbb{E}[x_{n-1,j} x_{n,i}] \ln P_{ij} \\ &\quad + \sum_{n=1}^N \sum_{k=1}^K \mathbb{E}[x_{n,k}] \ln p(y_n|k; \boldsymbol{\theta}_k). \end{aligned} \tag{16.52}$$

Let us now recall (16.49) to obtain

$$\mathbb{E}[x_{n,k}] = \sum_{\mathbf{x}_n} P(\mathbf{x}_n|Y; \Theta^{(t)}) x_{n,k} = \sum_{\mathbf{x}_n} \gamma(\mathbf{x}_n; \Theta^{(t)}) x_{n,k}.$$

Note that $x_{n,k}$ can either be zero or one; hence, its mean value will be equal to the probability that \mathbf{x}_n has the k th element $x_{n,k} = 1$ and we denote it as

$$\mathbb{E}[x_{n,k}] = \gamma(x_{n,k} = 1; \Theta^{(t)}). \quad (16.53)$$

Recall that given $\Theta^{(t)}$, $\gamma(\cdot; \Theta^{(t)})$ can be efficiently computed via the sum-product algorithm described before. In a similar spirit and mobilizing the definition in (16.51), we can write

$$\begin{aligned} \mathbb{E}[x_{n-1,j} x_{n,i}] &= \sum_{\mathbf{x}_n} \sum_{\mathbf{x}_{n-1}} P(\mathbf{x}_n, \mathbf{x}_{n-1} | Y; \Theta^{(t)}) x_{n-1,j} x_{n,i} \\ &= \sum_{\mathbf{x}_n} \sum_{\mathbf{x}_{n-1}} \xi(\mathbf{x}_n, \mathbf{x}_{n-1}; \Theta^{(t)}) x_{n-1,j} x_{n,i} \\ &= \xi(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(t)}). \end{aligned} \quad (16.54)$$

Note that $\xi(\cdot, \cdot; \Theta^{(t)})$ can also be efficiently computed as a by-product of the sum-product algorithm, given $\Theta^{(t)}$. Thus, we can summarize the E-step as

$$\begin{aligned} Q(\Theta, \Theta^{(t)}) &= \sum_{k=1}^K \gamma(x_{1,k} = 1; \Theta^{(t)}) \ln P_k \\ &\quad + \sum_{n=2}^N \sum_{i=1}^K \sum_{j=1}^K \xi(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(t)}) \ln P_{ij} \\ &\quad + \sum_{n=1}^N \sum_{k=1}^K \gamma(x_{n,k} = 1; \Theta^{(t)}) \ln p(\mathbf{y}_n | k; \boldsymbol{\theta}_k). \end{aligned} \quad (16.55)$$

Maximization step: In this step, it suffices to obtain the derivatives/gradients with regard to P_k , P_{ij} , and $\boldsymbol{\theta}_k$ and equate them to zero in order to obtain the new estimates, which will comprise $\Theta^{(t+1)}$. Note that P_k and P_{ij} are probabilities; hence, their maximization should be constrained so that

$$\sum_{k=1}^K P_k = 1 \quad \text{and} \quad \sum_{i=1}^K P_{ij} = 1, \quad j = 1, 2, \dots, K.$$

The resulting reestimation formulas are (Problem 16.18)

$$P_k^{(t+1)} = \frac{\gamma(x_{1,k} = 1; \Theta^{(t)})}{\sum_{i=1}^K \gamma(x_{1,i} = 1; \Theta^{(t)})}, \quad (16.56)$$

$$P_{ij}^{(t+1)} = \frac{\sum_{n=2}^N \xi(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(t)})}{\sum_{n=2}^N \sum_{k=1}^K \xi(x_{n-1,j} = 1, x_{n,k} = 1; \Theta^{(t)})}. \quad (16.57)$$

The reestimation of $\boldsymbol{\theta}_k$ depends on the form of the corresponding distribution $p(\mathbf{y}_n | k; \boldsymbol{\theta}_k)$. For example, in the Gaussian scenario, the parameters are the mean values and the elements of the covariance matrix.

In this case, we obtain exactly the same iterations as those resulting for the problem of Gaussian mixtures (see Eqs. (12.60) and (12.61)), if in place of the posterior we use γ .

In summary, training an HMM comprises the following steps:

1. Initialize the parameters in Θ .
2. Run the sum-product algorithm to obtain $\gamma(\cdot)$ and $\xi(\cdot, \cdot)$, using the current set of parameter estimates.
3. Update the parameters as in (16.56) and (16.57).

Iterations in steps 2 and 3 continue until a convergence criterion is met, such as in EM. This iterative scheme is also known as the *Baum–Welch* or *forward-backward* algorithm. Besides the forward-backward algorithm for training HMMs, the literature is rich in a number of alternatives with the goal of either simplifying computations or improving performance. For example, a simpler training algorithm can be derived tailored to the Viterbi scheme for computing the optimum path (e.g., [63, 72]). Also, to further simplify the training algorithm, we can assume that our state observation variables, y_n , are discretized (quantized) and can take values from a finite set of L possible ones, $\{1, 2, \dots, L\}$. This is often the case in practice. Furthermore, assume that the first state is also known. This is, for example, the case for left-to-right models like the one shown in Fig. 16.17. In such a case, we need not compute estimates of the initial probabilities. Thus, the unknown parameters to be estimated are the transition probabilities and the probabilities $P_y(r|i)$, $r = 1, 2, \dots, L$, $i = 1, 2, \dots, K$, that is, the probability of emitting symbol r from state i .

Viterbi reestimation: The goal of the algorithm is to obtain the best path and compute the associated cost, say, D , along the path. In the speech literature, the algorithm is also known as the *segmental k-means training* algorithm [63].

Definitions:

- $n_{i|j} :=$ number of transitions from state j to state i .
- $n_{\cdot|j} :=$ number of transitions originated from state j .
- $n_{i|\cdot} :=$ number of transitions terminated at state i .
- $n(r|i) :=$ number of times observation $r \in \{1, 2, \dots, L\}$ occurs jointly with state i .

Iterations:

- Initial conditions: Assume the initial estimates of the unknown parameters.
- Step 1: From the available best path, reestimate the new model parameters as

$$P^{(\text{new})}(i|j) = \frac{n_{i|j}}{n_{\cdot|j}},$$

$$P_x^{(\text{new})}(r|i) = \frac{n(r|i)}{n_{i|\cdot}}.$$

- Step 2: For the new model parameters, obtain the best path and compute the corresponding overall cost $D^{(\text{new})}$. Compare it with the cost D of the previous iteration. If $D^{(\text{new})} - D > \epsilon$, set $D = D^{(\text{new})}$ and go to step 1. Otherwise stop.

The Viterbi reestimation algorithm can be shown to converge to a proper characterization of the underlying observations [14].

Remarks 16.4.

- *Scaling:* The probabilities α and β , being less than one, as iterations progress can take very small values. In practice, the dynamic range of their computed values may exceed that of the computer. This phenomenon can be efficiently dealt with within an appropriate scaling. If this is done properly on both α and β , then the effect of scaling cancels out [63].
- *Insufficient training data set:* Generally, a large amount of training data is necessary to learn the HMM parameters. The observation sequence must be sufficiently long with respect to the number of states of the HMM model. This will guarantee that all state transitions will appear a sufficient number of times, so that the reestimation algorithm learns their respective parameters. If this is not the case, a number of techniques have been devised to cope with the issue. For a more detailed treatment, the reader may consult [8,63] and the references therein.

16.5.3 DISCRIMINATIVE LEARNING

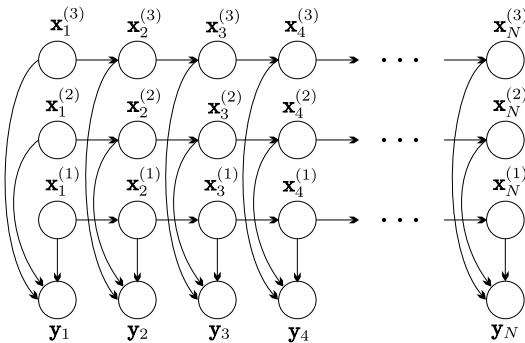
Discriminative learning is another path that has attracted a lot of attention. Note that the EM algorithm optimizes the likelihood with respect to the unknown parameters of a *single* HMM in “isolation”; that is, without considering the rest of the HMMs, which model the other words (in the case of speech recognition) or other templates/prototypes that are stored in the database. Such an approach is in line with what we defined as generative learning in Chapter 3. In contrast, the essence of discriminative learning is to optimize the set of parameters so that the models become optimally discriminated over the training sets (e.g., in terms of the error probability criterion). In other words, the parameters describing the different statistical models (HMMs) are optimized in a *combined* way, not individually. The goal is to make the different HMM models as distinct as possible, according to a criterion. This has been an intense line of research and a number of techniques have been developed around criteria that lead to either convex or nonconvex optimization methods (see, for example, [33] and the references therein).

Remarks 16.5.

- Besides the basic HMM scheme, which was described in this section, a number of variants have been proposed in order to overcome some of its shortcomings. For example, alternative modeling paths concern the first-order Markov property and propose models to extend correlations to longer times.

In the *autoregressive HMM* [11], links are added among the observation nodes of the basic HMM scheme in Fig. 16.15; for example, y_n is not only linked to x_n , but it shares direct links with, for example, y_{n-2} , y_{n-1} , y_{n+1} , and y_{n+2} , if the model extends correlations up to two time instants away. A different concept has been introduced in [56] in the context of *segment modeling*. According to this model, each state is allowed to emit, say, d successive observations, which comprise a segment. The length of the segment, d , is itself a random variable and it is associated with a probability $P(d|k)$, $k = 1, 2, \dots, K$. In this way, correlation is introduced via the joint distribution of the samples comprising the segment.

- *Variable duration HMM:* A serious shortcoming of the HMMs, which is often observed in practice, is associated with the self-transition probabilities, $P(k|k)$, which are among the model parameters associated with an HMM. Note that the probability of the model being at state k for d successive

**FIGURE 16.20**

A factorial HMM with three chains of hidden variables.

instants (initial transition to the state and $d - 1$ self-transitions) is given by

$$P_k(d) = (P(k|k))^{d-1} (1 - P(k|k)),$$

where $1 - P(k|k)$ is the probability of leaving the state. For many cases, this exponential state duration dependence is not realistic. In variable-duration HMMs, $P_k(d)$ is explicitly modeled. Different models for $P_k(d)$ can be employed (see, e.g., [46, 68, 72]).

- Hidden Markov modeling is among the most powerful tools in machine learning and has been widely used in a large number of applications besides speech recognition. Some sampled references are [9] in bioinformatics, [16, 36] in communications, [4, 73] in optical character recognition (OCR), and [40, 61, 62] in music analysis/recognition, to name but a few. For a further discussion on HMMs, see, for example, [8, 64, 72].

16.6 BEYOND HMMs: A DISCUSSION

In this section, some notable extensions of the hidden Markov models, which were previously discussed, are considered in order to meet requirements of applications where either the number of states is large or the homogeneity assumption is no more justified.

16.6.1 FACTORIAL HIDDEN MARKOV MODELS

In the HMMs considered before, the system dynamics is described via the hidden variables, whose graphical representation is a chain. However, such a model may turn out to be too simple for certain applications. A variant of the HMM involves M chains, instead of one chain, where each chain of hidden variables unfolds in time independently of the others. Thus at time n , M hidden variables are involved, denoted as $\mathbf{x}_n^{(m)}$, $m = 1, 2, \dots, M$ [17, 34, 81]. The observations occur as a combined emission where all hidden variables are involved. The respective graphical structure is shown in Fig. 16.20 for $M = 3$. Each one of the chains develops on its own, as the graphical model suggests. Such models are

PARTICLE FILTERING

17

CONTENTS

17.1 Introduction	871
17.2 Sequential Importance Sampling	871
17.2.1 Importance Sampling Revisited	872
17.2.2 Resampling	873
17.2.3 Sequential Sampling	875
17.3 Kalman and Particle Filtering	878
17.3.1 Kalman Filtering: a Bayesian Point of View	878
17.4 Particle Filtering.....	881
17.4.1 Degeneracy	885
17.4.2 Generic Particle Filtering	886
17.4.3 Auxiliary Particle Filtering	889
Problems	895
MATLAB® Exercises	898
References.....	899

17.1 INTRODUCTION

This chapter is a follow-up to Chapter 14, whose focus was on Monte Carlo methods. Our interest now turns to a special type of sampling techniques known as sequential-sampling methods. In contrast to the Monte Carlo methods, considered in Chapter 14, here we will assume that distributions from which we want to sample are time-varying, and that sampling will take place in a sequential fashion. The main emphasis of this chapter is on particle filtering techniques for inference in state-space dynamic models. In contrast to the classical form of Kalman filtering, here the model is allowed to be nonlinear and/or the distributions associated with the involved variables non-Gaussians.

17.2 SEQUENTIAL IMPORTANCE SAMPLING

Our interest in this section shifts toward tasks where data are sequentially arriving, and our goal becomes that of sampling from their joint distribution. In other words, we are receiving observations $\mathbf{x}_n \in \mathbb{R}^l$ of random vectors \mathbf{x}_n . At some time n , let $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ denote the set of the available samples and let the respective joint distribution be denoted as $p_n(\mathbf{x}_{1:n})$. No doubt, this new task is to be treated with special care. Not only is the dimensionality of the task (number of random variables, i.e., $\mathbf{x}_{1:n}$) now time-varying, but also, after some time has elapsed, the dimensionality will be very large and,

Remarks 17.1.

- Convergence results concerning sequential importance sampling can be found in, for example, [5–7]. It turns out that, in practice, the use of resampling leads to substantially smaller variances.
- From a practical point of view, sequential importance methods with resampling are expected to work reasonably well if the desired successive distributions at different time instants do not differ much and the choice of $q_n(\mathbf{x}_n | \mathbf{x}_{1:n-1})$ is *close to the optimal* one (see, e.g., [15]).

17.3 KALMAN AND PARTICLE FILTERING

Particle filtering is an instance of the sequential Monte Carlo methods. Particle filtering is a technique born in the 1990s and it was first introduced in [18] as an attempt to solve estimation tasks in the context of state-space modeling for the more general nonlinear and non-Gaussian scenarios. The term “particle filtering” was coined in [3], although the term “particle” had been used in [25].

Hidden Markov models (HMMs), which are treated in Section 16.4, and Kalman filters, treated in Chapter 4, are special types of state-space (state-observation) modeling. The former address the case of discrete state (latent) variables and the latter the continuous case, albeit in the very special case of linear and Gaussian scenario. In particle filtering, the interest shifts to models of the following form:

$$\mathbf{x}_n = \mathbf{f}_n(\mathbf{x}_{n-1}, \boldsymbol{\eta}_n) : \text{ state equation,} \quad (17.12)$$

$$\mathbf{y}_n = \mathbf{h}_n(\mathbf{x}_n, \mathbf{v}_n) : \text{ observations equation,} \quad (17.13)$$

where \mathbf{f}_n and \mathbf{h}_n are nonlinear, in general, (vector) functions, $\boldsymbol{\eta}_n$ and \mathbf{v}_n are noise sequences, and the dimensions of \mathbf{x}_n and \mathbf{y}_n can be different. The random vector \mathbf{x}_n is the (latent) state vector and \mathbf{y}_n corresponds to the observations. There are two inference tasks that are of interest in practice.

Filtering: Given the set of observations, $\mathbf{y}_{1:n}$, in the time interval $[1, n]$, compute

$$p(\mathbf{x}_n | \mathbf{y}_{1:n}).$$

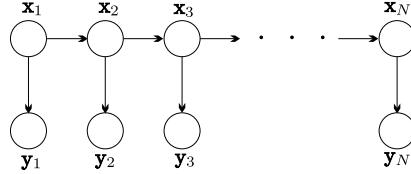
Smoothing: Given the set of observations $\mathbf{y}_{1:N}$ in a time interval $[1, N]$, compute

$$p(\mathbf{x}_n | \mathbf{y}_{1:N}), \quad 1 \leq n \leq N.$$

Before we proceed to our main goal, let us review the simpler case, that of Kalman filters, this time from a Bayesian viewpoint.

17.3.1 KALMAN FILTERING: A BAYESIAN POINT OF VIEW

Kalman filtering was discussed in Section 4.10 in the context of linear estimation methods and the mean-square error criterion. In the current section, the Kalman filtering algorithm will be rederived following concepts from the theory of graphical models and Bayesian networks, which are treated in Chapters 15 and 16. This probabilistic view will then be used for the subsequent nonlinear generalizations in the framework of particle filtering. For the linear case model, Eqs. (17.12) and (17.13)

**FIGURE 17.2**

Graphical model corresponding to the state-space modeling for Kalman and particle filters.

become

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \quad (17.14)$$

$$\mathbf{y}_n = H_n \mathbf{x}_n + \mathbf{v}_n, \quad (17.15)$$

where F_n and H_n are matrices of appropriate dimensions. We further assume that the two noise sequences are statistically independent and of a Gaussian nature, that is,

$$p(\boldsymbol{\eta}_n) = \mathcal{N}(\boldsymbol{\eta}_n | \mathbf{0}, Q_n), \quad (17.16)$$

$$p(\mathbf{v}_n) = \mathcal{N}(\mathbf{v}_n | \mathbf{0}, R_n). \quad (17.17)$$

The kick-off point for deriving the associated recursions is the Bayes rule,

$$\begin{aligned} p(\mathbf{x}_n | \mathbf{y}_{1:n}) &= \frac{p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{y}_{1:n-1}) p(\mathbf{x}_n | \mathbf{y}_{1:n-1})}{Z_n} \\ &= \frac{p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{y}_{1:n-1})}{Z_n}, \end{aligned} \quad (17.18)$$

where

$$\begin{aligned} Z_n &= \int p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{y}_{1:n-1}) d\mathbf{x}_n \\ &= p(\mathbf{y}_n | \mathbf{y}_{1:n-1}), \end{aligned} \quad (17.19)$$

and we have used the fact that $p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{y}_{1:n-1}) = p(\mathbf{y}_n | \mathbf{x}_n)$, which is a consequence of Eq. (17.15). For those who have already read Chapter 15, recall that Kalman filtering is a special case of a Bayesian network and corresponds to the graphical model given in Fig. 17.2. Hence, due to the Markov property, \mathbf{y}_n is independent of the past given the values in \mathbf{x}_n . Moreover, note that

$$\begin{aligned} p(\mathbf{x}_n | \mathbf{y}_{1:n-1}) &= \int p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{y}_{1:n-1}) p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1}) d\mathbf{x}_{n-1} \\ &= \int p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1}) d\mathbf{x}_{n-1}, \end{aligned} \quad (17.20)$$

where, once more, the Markov property (i.e., Eq. (17.14)) has been used.

Eqs. (17.18)–(17.20) comprise the set of recursions which lead to the update

$$p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}) \longrightarrow p(\mathbf{x}_n|\mathbf{y}_{1:n}),$$

starting from the initial (prior) $p(\mathbf{x}_0|\mathbf{y}_0) := p(\mathbf{x}_0)$. If $p(\mathbf{x}_0)$ is chosen to be Gaussian, then all the involved PDFs turn out to be Gaussian due to Eqs. (17.16) and (17.17) and the linearity of Eqs. (17.14) and (17.15); this makes the computation of the integrals a trivial task following the recipe rules in the Appendix of Chapter 12.

Before we proceed further, note that the recursions in Eqs. (17.18) and (17.20) are an instance of the sum-product algorithm for graphical models. Indeed, to put our current discussion in this context, let us compactly write the previous recursions as

$$p(\mathbf{x}_n|\mathbf{y}_{1:n}) = \underbrace{\frac{p(\mathbf{y}_n|\mathbf{x}_n)}{Z_n}}_{\text{corrector}} \underbrace{\int p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}) p(\mathbf{x}_n|\mathbf{x}_{n-1}) d\mathbf{x}_{n-1}}_{\text{predictor}} : \text{ filtering.}$$

(17.21)

Note that this is of exactly the same form, within the normalizing factor, as Eq. (16.43) of Chapter 16; just replace summation with integration. One can rederive Eq. (17.21) using the sum-product rule, following similar steps as for Eq. (16.43). The only difference is that the normalizing constant has to be involved in all respective definitions and we replace summations with integrations. Because all the involved PDFs are Gaussians, the computation of the involved normalizing constants is trivially done; moreover, it suffices to derive recursions only for the respective mean values and covariances.

In Eq. (17.20), we have

$$p(\mathbf{x}_n|\mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n|F_n \mathbf{x}_{n-1}, Q_n).$$

Let, also, $p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$ be Gaussian with mean and covariance matrix

$$\boldsymbol{\mu}_{n-1|n-1}, \quad P_{n-1|n-1},$$

respectively, where the notation is chosen for the derived recursions to comply with the algorithm given in Section 4.10. Then, according to the Appendix of Chapter 12, $p(\mathbf{x}_n|\mathbf{y}_{1:n-1})$ is a Gaussian marginal PDF with mean and covariance given by (see Eqs. (12.150) and (12.151))

$$\boldsymbol{\mu}_{n|n-1} = F_n \boldsymbol{\mu}_{n-1|n-1}, \quad (17.22)$$

$$P_{n|n-1} = Q_n + F_n P_{n-1|n-1} F_n^T. \quad (17.23)$$

Also, in Eq. (17.18) we have

$$p(\mathbf{y}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n|H_n \mathbf{x}_n, R_n).$$

From the Appendix of Chapter 12, and taking into account Eqs. (17.22) and (17.23), we find that $p(\mathbf{x}_n|\mathbf{y}_{1:n})$ is the posterior (Gaussian) with mean and covariance given by (see Eqs. (12.148) and (12.149))

$$\boldsymbol{\mu}_{n|n} = \boldsymbol{\mu}_{n|n-1} + K_n (\mathbf{y}_n - H_n \boldsymbol{\mu}_{n|n-1}), \quad (17.24)$$

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}, \quad (17.25)$$

where

$$K_n = P_{n|n-1} H_n^T S_n^{-1}, \quad (17.26)$$

and

$$S_n = R_n + H_n P_{n|n-1} H_n^T. \quad (17.27)$$

Note that these are exactly the same recursions that were derived in Section 4.10 for the state estimation; recall that under the Gaussian assumption, the posterior mean coincides with the least-squares estimate.

Here we have assumed that matrices F_n , H_n as well as the covariance matrices are known. This is most often the case. If not, these can be learned using similar arguments as those used in learning the HMM parameters, which are discussed in Section 16.5.2 (see, e.g., [2]).

17.4 PARTICLE FILTERING

In Section 4.10, extended Kalman filtering (EKF) was discussed as one possibility to generalize Kalman filtering to nonlinear models. Particle filtering, to be discussed next, is a powerful alternative technique to EKF. The involved PDFs are approximated by *discrete random measures*. The underlying theory is that of sequential importance sampling (SIS); as a matter of fact, particle filtering is an instance of SIS.

Let us now consider the state-space model of the general form in Eqs. (17.12) and (17.13). From the specific form of these equations (and by the Bayesian network nature of such models, for the more familiar reader) we can write

$$p(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) = p(\mathbf{x}_n | \mathbf{x}_{n-1}) \quad (17.28)$$

and

$$p(\mathbf{y}_n | \mathbf{x}_{1:n}, \mathbf{y}_{1:n-1}) = p(\mathbf{y}_n | \mathbf{x}_n). \quad (17.29)$$

Our starting point is the *sequential* estimation of $p(\mathbf{x}_{1:n} | \mathbf{y}_{1:n})$; the estimation of $p(\mathbf{x}_n | \mathbf{y}_{1:n})$, which comprises our main goal, will be obtained as a by-product. Note that [15]

$$\begin{aligned} p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) &= p(\mathbf{x}_n, \mathbf{x}_{1:n-1}, \mathbf{y}_n, \mathbf{y}_{1:n-1}) \\ &= p(\mathbf{x}_n, \mathbf{y}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) p(\mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) \\ &= p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}), \end{aligned} \quad (17.30)$$

where Eqs. (17.28) and (17.29) have been employed.

Our goal is to obtain an approximation, via the generation of particles, of the conditional PDF,

$$p(\mathbf{x}_{1:n} | \mathbf{y}_{1:n}) = \frac{p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n})}{\int p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\mathbf{x}_{1:n}} = \frac{p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n})}{Z_n}, \quad (17.31)$$

DIMENSIONALITY REDUCTION AND LATENT VARIABLE MODELING

19

CONTENTS

19.1	Introduction	1040
19.2	Intrinsic Dimensionality	1041
19.3	Principal Component Analysis	1041
	PCA, SVD, and Low Rank Matrix Factorization	1043
	Minimum Error Interpretation	1045
	PCA and Information Retrieval	1045
	Orthogonalizing Properties of PCA and Feature Generation	1046
	Latent Variables	1047
19.4	Canonical Correlation Analysis	1053
19.4.1	Relatives of CCA	1056
	Partial Least-Squares	1056
19.5	Independent Component Analysis	1058
19.5.1	ICA and Gaussianity	1058
19.5.2	ICA and Higher-Order Cumulants	1059
	ICA Ambiguities	1060
19.5.3	Non-Gaussianity and Independent Components	1061
19.5.4	ICA Based on Mutual Information	1062
19.5.5	Alternative Paths to ICA	1065
	The Cocktail Party Problem	1066
19.6	Dictionary Learning: the k-SVD Algorithm	1069
	Why the Name k -SVD?	1072
	Dictionary Learning and Dictionary Identifiability	1072
19.7	Nonnegative Matrix Factorization	1074
19.8	Learning Low-Dimensional Models: a Probabilistic Perspective	1076
19.8.1	Factor Analysis	1077
19.8.2	Probabilistic PCA	1078
19.8.3	Mixture of Factors Analyzers: a Bayesian View to Compressed Sensing	1082
19.9	Nonlinear Dimensionality Reduction	1085
19.9.1	Kernel PCA	1085
19.9.2	Graph-Based Methods	1087
	Laplacian Eigenmaps	1087
	Local Linear Embedding (LLE)	1091
	Isometric Mapping (ISOMAP)	1092
19.10	Low Rank Matrix Factorization: a Sparse Modeling Path	1096
19.10.1	Matrix Completion	1096

19.10.2 Robust PCA	1100
19.10.3 Applications of Matrix Completion and ROBUST PCA	1101
<i>Matrix Completion</i>	1101
<i>Robust PCA/PCA</i>	1102
19.11 A Case Study: FMRI Data Analysis	1103
Problems	1107
<i>MATLAB® Exercises</i>	1107
References	1108

19.1 INTRODUCTION

In many practical applications, although the data reside in a high-dimensional space, the true dimensionality, known as *intrinsic dimensionality*, can be of a much lower value. We have met such cases in the context of sparse modeling in Chapter 9. There, although the data lay in a high-dimensional space, a number of the components were known to be zero. The task was to learn the locations of the zeros; this is equivalent to learning the specific subspace, which is determined by the locations of the nonzero components. In this chapter, the goal is to treat the task in a more general setting and assume that the data can lie in any possible subspace (not only the ones formed by the removal of coordinate axes) or manifold. For example, in a three-dimensional space, the data may cluster around a straight line, or around the circumference of a circle or the graph of a parabola, arbitrarily placed in \mathbb{R}^3 . In all previous cases, the intrinsic dimensionality of the data is equal to one, as any of these curves can equivalently be described in terms of a single parameter. Fig. 19.1 illustrates the three cases. Learning the lower-dimensional structure associated with a given set of data is gaining in importance in the context of *big data* processing and analysis. Some typical examples are the disciplines of computer vision, robotics, medical imaging, and computational neuroscience.

The goal of this chapter is to introduce the reader to the main directions, which are followed in this topic, starting from more classical techniques, such as principal component analysis (PCA) and factor analysis, both in their standard as well as in their probabilistic formulations. Canonical correlation analysis (CCA), independent component analysis (ICA), nonnegative matrix factorization (NMF), and

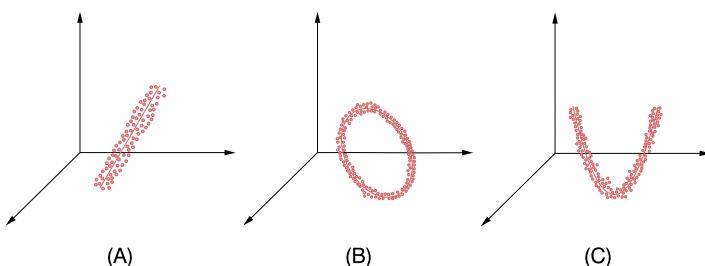


FIGURE 19.1

The data reside close to (A) a straight line, (B) the circumference of a circle, and (C) the graph of a parabola in the three-dimensional space. In all three cases, the intrinsic dimensionality of the data is equal to one. In (A) the data are clustered around a (translated/affine) linear subspace and in (B) and (C) around one-dimensional manifolds.

dictionary learning techniques are also discussed; in the latter case, data are represented via an expansion in terms of overcomplete dictionaries, and sparsity-related arguments are mobilized to detect the most relevant atoms in the dictionary. Finally, nonlinear techniques for learning (nonlinear) manifolds are presented such as the kernel PCA, the local linear embedding (LLE), and the isometric mapping (ISOMAP) techniques. At the end of the chapter, a case study in the context of fMRI data analysis is presented.

19.2 INTRINSIC DIMENSIONALITY

A data set $\mathcal{X} \subset \mathbb{R}^l$ is said to have *intrinsic dimensionality* $m \leq l$ if \mathcal{X} can be (approximately) described in terms of m free parameters. Take as an example the case where the vectors in \mathcal{X} are generated as functions in terms of m random variables, that is, $\mathbf{x} = \mathbf{g}(\mathbf{u}_1, \dots, \mathbf{u}_m)$, $\mathbf{u}_i \in \mathbb{R}$, $i = 1, \dots, m$. The corresponding geometric interpretation is that the respective observation vectors will lie along a manifold, whose form depends on the vector-valued function $\mathbf{g} : \mathbb{R}^m \mapsto \mathbb{R}^l$. Let us consider the case where

$$\mathbf{x} = [r \cos \theta, r \sin \theta]^T,$$

where r is a constant and the random variable $\theta \in [0, 2\pi]$. The data lie along the circumference of a circle of radius r and a single free parameter suffices to describe the data. If now a small amount of noise is added, then the data will be clustered close to the circumference, as for example in Fig. 19.1B, and the intrinsic dimensionality is equal to one. From a statistical point of view, it means that the components of the random vectors are highly correlated. Sometimes, we say that the “effective” dimensionality is lower than the apparent one of the “ambient” space, in which the lower-dimensional manifold lies.

In a more general setting, the data may lie in groups of manifolds or even in groups of clusters or they may follow a special spatial or temporal structure. For example, in the wavelet domain most of the coefficients of an image are close to zero and can be neglected, yet the larger (nonzero) ones have a particular structure that is characteristic of natural images. Such a structured sparsity has been exploited in the JPEG2000 coding scheme. Structured sparsity representations are often met in many big data applications (see, for example, [42]). In this chapter, we will only focus on identifying manifold structures, linear (subspaces/affine subspaces) in the beginning and nonlinear ones later on.

Learning the manifold in which a data set resides can be used to provide a compact low-dimensional encoding of a high-dimensional data set, which can subsequently be exploited for performing processing and learning tasks in a much more efficient way. Also, dimensionality reduction can be used for data visualization.

19.3 PRINCIPAL COMPONENT ANALYSIS

Principal component analysis (PCA) or *Karhunen–Loève transform* is among the oldest and most widely used methods for dimensionality reduction [105]. The assumption underlying PCA, as well as any dimensionality reduction technique, is that the observed data are generated by a system or process that is driven by a (relatively) small number of *latent* (not directly observed) variables. The goal is to learn this latent structure.

Given a set of observation vectors, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, of a random vector \mathbf{x} , which will be assumed to be of zero mean (otherwise the mean/sample mean is subtracted), PCA determines a *subspace* of dimension $m \leq l$, such that after projection on this subspace, the statistical variation of the data is optimally retained. This subspace is defined in terms of m *mutually orthogonal axes*, known as *principal axes* or *principal directions*, which are computed so that the variance of the data, after projection on the subspace, is maximized [95].

We will derive the principal axes in a step-wise fashion. First, assume that $m = 1$ and the goal is to find a single direction in \mathbb{R}^l so that the variance of the corresponding projections of the data points is maximized. Let \mathbf{u}_1 denote the principal axis. The variance of the projections (having assumed centered data) is given by

$$\begin{aligned} J(\mathbf{u}_1) &= \frac{1}{N} \sum_{n=1}^N (\mathbf{u}_1^T \mathbf{x}_n)^2 = \frac{1}{N} \sum_{n=1}^N (\mathbf{u}_1^T \mathbf{x}_n)(\mathbf{x}_n^T \mathbf{u}_1) \\ &= \mathbf{u}_1^T \hat{\Sigma} \mathbf{u}_1, \end{aligned}$$

where

$$\hat{\Sigma} := \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \quad (19.1)$$

is the sample covariance matrix of the data. For large values of N or if the statistics can be computed, the covariance (instead of the sample covariance) matrix can be used. The task now becomes that of maximizing the variance. However, because we are only interested in directions, the principal axis will be represented by the respective unit norm vector. Thus, the optimization task is cast as

$$\boxed{\mathbf{u}_1 = \arg \max_{\mathbf{u}} \mathbf{u}^T \hat{\Sigma} \mathbf{u},} \quad (19.2)$$

$$\boxed{\text{s.t. } \mathbf{u}^T \mathbf{u} = 1.} \quad (19.3)$$

This is a constrained optimization problem and the corresponding Lagrangian is given by

$$L(\mathbf{u}, \lambda) = \mathbf{u}^T \hat{\Sigma} \mathbf{u} - \lambda(\mathbf{u}^T \mathbf{u} - 1). \quad (19.4)$$

Taking the gradient and setting it equal to zero we get

$$\hat{\Sigma} \mathbf{u} = \lambda \mathbf{u}. \quad (19.5)$$

In other words, the principal direction is an eigenvector of the sample covariance matrix. Plugging Eq. (19.5) into Eq. (19.2) and taking into account (19.3), we obtain

$$\mathbf{u}^T \hat{\Sigma} \mathbf{u} = \lambda. \quad (19.6)$$

Hence, the variance is maximized if \mathbf{u}_1 is the eigenvector that corresponds to the maximum eigenvalue, λ_1 . Recall that because the (sample) covariance matrix is symmetric and positive semidefinite, all the eigenvalues are real and nonnegative. Assuming $\hat{\Sigma}$ to be invertible (hence, necessarily, $N > l$), the

eigenvalues are all positive, that is, $\lambda_1 > \lambda_2 > \dots > \lambda_l > 0$, and we also assume they are distinct, in order to simplify the discussion.

The second principal component is selected so that it (a) is orthogonal to \mathbf{u}_1 and (b) maximizes the variance after projecting the data onto this direction. Following similar arguments as before, a similar optimization task results with an extra constraint, $\mathbf{u}^T \mathbf{u}_1 = 0$. It can easily be shown (Problem 19.1) that the second principal axis is the eigenvector corresponding to the second-largest eigenvalue, λ_2 . The process continues until m principal axes have been obtained; they are the eigenvectors corresponding to the m largest eigenvalues.

PCA, SVD, AND LOW RANK MATRIX FACTORIZATION

The SVD decomposition of a matrix was discussed in Section 6.4. Given a matrix $X \in \mathbb{R}^{l \times N}$, we can write

$$X = UDV^T. \quad (19.7)$$

For a rank r matrix X , U is the $l \times r$ matrix having as columns the eigenvectors corresponding to the r nonzero eigenvalues of XX^T , and V is the $N \times r$ matrix having as columns the respective eigenvectors of $X^T X$; D is a square $r \times r$ diagonal matrix comprising the singular values¹ $\sigma_i := \sqrt{\lambda_i}$, $i = 1, 2, \dots, r$. If we construct X to have as columns the data vectors \mathbf{x}_n , $n = 1, 2, \dots, N$, then XX^T is a scaled version of the corresponding sample covariance matrix, $\hat{\Sigma}$; hence, the respective eigenvectors coincide and the corresponding eigenvalues are equal within a scaling factor (N). Without harming generality, we can assume XX^T to be full rank ($r = l < N$), and Eq. (19.7) becomes

$$X = \underbrace{[\mathbf{u}_1, \dots, \mathbf{u}_l]}_{l \times l} \begin{bmatrix} \sqrt{\lambda_1} \mathbf{v}_1^T \\ \vdots \\ \sqrt{\lambda_l} \mathbf{v}_l^T \end{bmatrix} = [\mathbf{u}_1, \dots, \mathbf{u}_l] \begin{bmatrix} \sqrt{\lambda_1} v_{11} & \dots & \sqrt{\lambda_1} v_{1n} & \dots & \sqrt{\lambda_1} v_{1N} \\ \vdots & & \vdots & & \vdots \\ \sqrt{\lambda_l} v_{l1} & \dots & \sqrt{\lambda_l} v_{ln} & \dots & \sqrt{\lambda_l} v_{lN} \end{bmatrix}. \quad (19.8)$$

Thus, the columns of X can be written in terms of the following expansion²:

$$\mathbf{x}_n = \sum_{i=1}^l z_{ni} \mathbf{u}_i = \sum_{i=1}^m z_{ni} \mathbf{u}_i + \sum_{i=m+1}^l z_{ni} \mathbf{u}_i, \quad (19.9)$$

where $\mathbf{z}_n^T := [z_{n1}, \dots, z_{nl}]$ is the n th column of the $l \times N$ matrix on the right-hand side in Eq. (19.8). That is, by definition, $z_{n1} = \sqrt{\lambda_1} v_{1n}$ and so on. The sum in Eq. (19.9) has been split into two terms,

¹ Because in some places we are going to involve the variance σ^2 , we will carry on working with the square root of the eigenvalues, to avoid possible confusion.

² Note that what we have defined in previous chapters as the data matrix is the transpose of X . This is because, for dimensionality reduction tasks, it is more common to work with the current notational convention. If the transpose of X is used, the expansion of the data vectors is in terms of the columns of V and the analysis carries on in a similar way.

where m can be any value $1 \leq m \leq l$. Note that, due to the orthonormality of the \mathbf{u}_i s,

$$z_{ni} = \mathbf{u}_i^T \mathbf{x}_n, \quad i = 1, 2, \dots, l, \quad n = 1, 2, \dots, N.$$

From Section 6.4, we know that the best, in the Frobenius sense, m rank matrix approximation of X is given by

$$\hat{X} = \underbrace{[\mathbf{u}_1, \dots, \mathbf{u}_m]}_{l \times m} \begin{bmatrix} \sqrt{\lambda_1} \mathbf{v}_1^T \\ \vdots \\ \sqrt{\lambda_m} \mathbf{v}_m^T \end{bmatrix}, \quad (19.10)$$

$$= \sum_{i=1}^m \sqrt{\lambda_i} \mathbf{u}_i \mathbf{v}_i^T. \quad (19.11)$$

Recalling the previous definition of z_{ni} , the n th column vector of \hat{X} can now be written as

$$\hat{\mathbf{x}}_n = \sum_{i=1}^m z_{ni} \mathbf{u}_i. \quad (19.12)$$

Comparing Eqs. (19.9) and (19.12) and taking into account the orthonormality of \mathbf{u}_i , $i = 1, 2, \dots, l$, we readily see that $\hat{\mathbf{x}}_n$ is the projection of the original observation vectors, \mathbf{x}_n , $n = 1, 2, \dots, N$, onto the subspace $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$ generated by the m principal axes of $XX^T (\hat{\Sigma})$ (Fig. 19.2).

The previous arguments establish a bridge between PCA and SVD. In other words, the principal axes can be obtained via the SVD decomposition of X . Moreover, the columns of the best m rank matrix approximation, \hat{X} , of X are the projections of the observation vectors \mathbf{x}_n on the (optimally) reduced in dimension subspace, spanned by the principal axes.

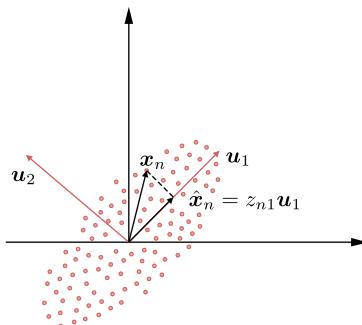


FIGURE 19.2

The projection of \mathbf{x}_n on the principal axis \mathbf{u}_1 is given by $\hat{\mathbf{x}}_n = z_{n1} \mathbf{u}_1$, where $z_{n1} = \mathbf{u}_1^T \mathbf{x}_n$.

Looking at Eq. (19.10), PCA can also be seen as a *low rank matrix factorization* method. Matrix factorization will be a recurrent theme in this chapter. Given a matrix X , there is not a unique way to factorize it, in terms of two matrices. PCA provides an m rank matrix factorization of X , by imposing *orthogonality* on the structure of the involved factors. Later on, we are going to discuss other approaches.

Finally, it is important to emphasize that the bridge between PCA and SVD establishes a connection between the low rank factorization of a matrix X and the intrinsic dimensionality of the subspace in which its column vectors reside, since this is the subspace where maximum variance of the data is guaranteed.

MINIMUM ERROR INTERPRETATION

Having established the bridge between PCA and SVD, another interpretation of the PCA method becomes readily available. Because \hat{X} is the best m rank matrix approximation of X in the Frobenius sense, the quantity

$$\|\hat{X} - X\|_F^2 := \sum_i \sum_j |\hat{X}(i, j) - X(i, j)|^2 = \sum_{n=1}^N \|\hat{x}_n - x_n\|^2$$

is minimum; that is, obtaining any other m -dimensional approximation (say, \tilde{x}_n) of x_n , by choosing to project onto another m -dimensional subspace, would result in higher squared error norm approximation, compared to that resulting from PCA. This is also a strong result that establishes a notable merit of the PCA method as a dimensionality reduction technique. This interpretation goes back to Pearson [146].

PCA AND INFORMATION RETRIEVAL

The previous minimum error interpretation paves the way to build around PCA an efficient searching procedure in identifying similar patterns in large databases. Assume that a number N of prototypes are represented in terms of l features, giving rise to feature vectors, $x_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, which are stored in a database. Given an unknown object, which is represented by a feature vector x , the task is to identify to which one among the prototypes this pattern is most similar. Similarity is measured in terms of the Euclidean distance $\|x - x_n\|^2$. If N and l are large, searching for the minimum Euclidean distance can be computationally very expensive. The idea is to keep in the database the components $z_n^{(m)} := [z_{n1}, \dots, z_{nm}]^T$ (see Eq. (19.12)) that describe the projections of the N prototypes in $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$, instead of the original l -dimensional feature vectors. Assuming that m is large enough to capture most of the variability of the original data (i.e., the intrinsic dimensionality of the data is m to a good approximation), then $z_n^{(m)}$ is a good feature vector description because we know that in this case $\hat{x}_n \approx x_n$. Given now an unknown pattern, x , we first project it onto $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$, resulting in

$$\hat{x} = \sum_{i=1}^m (\mathbf{u}_i^T x) \mathbf{u}_i := \sum_{i=1}^m z_i \mathbf{u}_i. \quad (19.13)$$

Then we have

$$\begin{aligned} \|\mathbf{x}_n - \mathbf{x}\|^2 &\approx \|\hat{\mathbf{x}}_n - \hat{\mathbf{x}}\|^2 = \left\| \sum_{i=1}^m z_{ni} \mathbf{u}_i - \sum_{i=1}^m z_i \mathbf{u}_i \right\|^2 \\ &= \|z_n^{(m)} - z\|^2, \end{aligned}$$

where $\mathbf{z} := [z_1, \dots, z_m]^T$. In other words, Euclidean distances are computed in the lower-dimensional subspace, which leads to substantial computational gains (see, for example, [22, 63, 160] and the references therein). This method is also known as *latent semantics indexing*.

ORTHOGONALIZING PROPERTIES OF PCA AND FEATURE GENERATION

We will now shed light on PCA from a different angle. We have just discussed, in the context of the information retrieval application, that PCA can also be seen as a feature generation method that generates a set of new feature vectors, \mathbf{z} , whose components describe a pattern in terms of the principal axes. Let us now assume (to make life easier) that N is large enough and the sample covariance matrix is a good approximation of the (full rank) covariance matrix $\Sigma = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$. We know that any vector $\mathbf{x} \in \mathbb{R}^l$ can be described in terms of $\mathbf{u}_1, \dots, \mathbf{u}_l$, that is,

$$\mathbf{x} = \sum_{i=1}^l z_i \mathbf{u}_i = \sum_{i=1}^l (\mathbf{u}_i^T \mathbf{x}) \mathbf{u}_i.$$

Our focus now turns to the covariance matrix of the random vectors, \mathbf{z} , as \mathbf{x} changes randomly. Taking into account that

$$z_i = \mathbf{u}_i^T \mathbf{x}, \quad (19.14)$$

and the definition of U in Eqs. (19.7) and (19.8), we can write $\mathbf{z} = U^T \mathbf{x}$, and hence

$$\mathbb{E}[\mathbf{z}\mathbf{z}^T] = \mathbb{E}[U^T \mathbf{x}\mathbf{x}^T U] = U^T \Sigma U.$$

However, we know from linear algebra (Appendix A.2) that U is the matrix that diagonalizes Σ ; hence,

$$\mathbb{E}[\mathbf{z}\mathbf{z}^T] = \text{diag}\{\lambda_1, \dots, \lambda_l\}. \quad (19.15)$$

In other words, the new features are *uncorrelated*, that is,

$$\mathbb{E}[z_i z_j] = 0, \quad i \neq j, \quad i, j = 1, 2, \dots, l. \quad (19.16)$$

Furthermore, note that the variances of z_i are equal to the eigenvalues λ_i , $i = 1, 2, \dots, l$, respectively. Hence, by selecting as features the ones that correspond to the dominant eigenvalues, one has maximally retained the total variance associated with the original features, x_i ; indeed, the corresponding total variance is given by the trace of the covariance matrix, which in turn is equal to the sum of the eigenvalues, as we know from linear algebra. In other words, the new set of features, z_i , $i = 1, 2, \dots, m$, represent the patterns in a more compact way, as they are *mutually uncorrelated*

and most of the variance is retained. It is common in practice, when the goal is that of feature generation, for each one of the z_i s to be normalized to unit variance.

Later on, we will see that a more recent method, known as ICA, imposes the constraint that after a linear transformation (a projection is a linear transformation, after all) the obtained latent variables (components) are statistically independent, which is a much stronger condition than being uncorrelated.

LATENT VARIABLES

The random components z_i , $i = 1, 2, \dots, m$, are known as *principal components*. Sometimes, their observed values, z_i , are known as *principal scores*. As a matter of fact, the principal components comprise the *latent* variables, which we mentioned at the beginning of this section.

According to the general (linear) latent variable modeling approach, we assume that our l variables comprising \mathbf{x} are modeled as

$$\mathbf{x} \approx A\mathbf{z}, \quad (19.17)$$

where A is an $l \times m$ matrix and $\mathbf{z} \in \mathbb{R}^m$ is the corresponding set of latent variables. Adopting the PCA model, we have shown that

$$A = [\mathbf{u}_1, \dots, \mathbf{u}_m] := U_m,$$

and the model implies that each one of the l components of \mathbf{x} is (approximately) generated in terms of these mutually uncorrelated m latent random variables, that is,

$$x_i \approx u_{i1}z_1 + \dots + u_{im}z_m. \quad (19.18)$$

Alternatively, in linear latent variable modeling, we can assume that the latent variables can also be recovered by a linear model from the original random variables, as for example,

$$\mathbf{z} = W\mathbf{x}. \quad (19.19)$$

In the case of the PCA approach, we have already seen that

$$W = U_m^T.$$

Eqs. (19.17) and (19.19) constitute the backbone of this chapter, and different methods provide different solutions for computing A or W .

Let us now collect all the principal score vectors, z_n , $n = 1, 2, \dots, N$, as the columns of the $m \times N$ score matrix Z , that is,

$$Z := [z_1, \dots, z_N]. \quad (19.20)$$

Then (19.10) can be rewritten in terms of the score matrix

$$X \approx U_m Z. \quad (19.21)$$

Moreover, taking into account the definition of the principal components in Eq. (19.14), we can also write

$$Z = U_m^T X. \quad (19.22)$$

Remarks 19.1.

- A major issue in practice is to select the m dominant eigenvalues. One way is to rank them in descending order, and determine m so that the gap between λ_m and λ_{m+1} is “large.” The interested reader can obtain more on this issue in [55, 104].
- The treatment so far involved centered quantities. In case we want to approximate the original observation vectors by taking into consideration the respective mean value of the data set, Eq. (19.13) is rephrased as

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \sum_{i=1}^m \mathbf{u}_i^T (\mathbf{x} - \bar{\mathbf{x}}) \mathbf{u}_i, \quad (19.23)$$

where $\bar{\mathbf{x}}$ is the sample mean (mean if it is known)

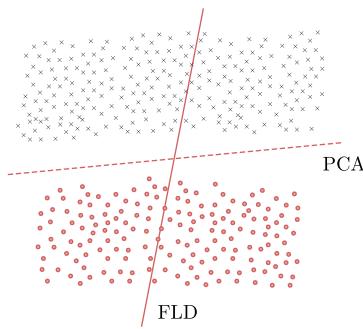
$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$$

and \mathbf{x} denotes the original (not centered) vector.

- PCA builds upon *global* information spread over *all* the data observations in the set \mathcal{X} . Indeed, the main source of information is the sample covariance matrix ($\mathbf{X}\mathbf{X}^T$). Thus, PCA is effective if the covariance matrix provides a sufficiently rich description of the data at hand. For example, this is the case for Gaussian-like distributions. In [41], modifications of the standard approach are suggested in order to deal with data having a clustered nature. Soon, we are going to discuss techniques alternative to PCA in order to overcome this drawback.
- Computing the SVD of large matrices can be computationally costly and a number of efficient techniques have been proposed (see, e.g., [1, 83, 194]). In a number of cases in practice, it turns out that $l > N$. Of course, in this case, the sample covariance is not invertible and some of the eigenvalues are zero. In such scenarios, it is preferable to work with an $\mathbf{X}^T\mathbf{X}$ ($N \times N$) instead of an $\mathbf{X}\mathbf{X}^T$ ($l \times l$) matrix. To this end, the relationships given in Section 6.4, in order to obtain \mathbf{u}_i from \mathbf{v}_i , can be employed.
- The treatment of PCA bears a similarity with the Fisher linear discriminant (FLD) method (Chapter 7). They both rely on the eigenstructure of matrices that, in one way or another, encode (co)variance information. However, note that PCA is an *unsupervised* method, in contrast to FLD, which is a *supervised* one. As a consequence, PCA performs dimensionality reduction so as to preserve data variability (variance) while FLD class separability. Fig. 19.3 demonstrates the difference in the resulting (hyper)planes.
- *Multidimensional scaling* (MDS) is another linear technique used to project in a lower-dimensional space, while respecting certain constraints. Given the set $\mathcal{X} \subset \mathbb{R}^l$, the goal is to project onto a lower-dimensional space, so that inner products are optimally preserved; that is, the cost

$$E = \sum_i \sum_j \left(\mathbf{x}_i^T \mathbf{x}_j - \mathbf{z}_i^T \mathbf{z}_j \right)^2$$

is minimized, where \mathbf{z}_i is the image of \mathbf{x}_i and the sum runs over all the training points in \mathcal{X} . The problem is similar to PCA and it can be shown that the solution is given by the eigendecomposition

**FIGURE 19.3**

The case of a two-class task in the two-dimensional space. PCA computes the direction along which the variance is maximally retained after the projections of the data on it. In contrast, FLD computes the line so that the class separability is maximized.

of the Gram matrix,³ $\mathcal{K} := X^T X$. Another side of the same coin is to require the Euclidean distances, instead of the inner products, to be optimally preserved. A Gram matrix, consistent with the squared Euclidean distances, can then be formed, leading to the same solution as before. It turns out that the solutions obtained by PCA and MDS are equivalent. This can readily be understood as $X^T X$ and XX^T share the same (nonzero) eigenvalues. The corresponding eigenvectors are different, yet they are related, as we have seen while introducing SVD in Section 6.4.

More on these issues can be found in [29,60]. As we will soon see in Section 19.9, the main idea behind MDS of preserving the distances is used, in one way or another, in a number of more recently developed nonlinear dimensionality reduction techniques.

- In a variant of the basic PCA, known as *supervised PCA* [16,195], the output variables in regression or in classification (depending on the problem at hand) are used together with the input ones, in order to determine the principal directions.

Example 19.1. This example demonstrates the power of PCA as a method to represent data in a lower-dimensional space. Each pattern in a database, described in terms of a feature vector, $x_n \in \mathbb{R}^l$, will be represented by a corresponding vector of a reduced dimensionality, $z_n^{(m)} \in \mathbb{R}^m$, $n = 1, 2, \dots, N$. In this example, each feature vector comprises the pixels of a 168×168 face image. These face images are members of the software-based aligned version [191] of the *Labeled Faces in the Wild* (LFW) database [102]. In particular, among the over 13,000 face images of this database, $N = 1924$ have been selected with criteria such as the quality of the image and the face angle (portraits were of preference). Moreover, the images are zoomed in order to omit most of the background. Examples of the face images used are depicted in Fig. 19.4 and the full collection of all the 1924 images can be found in the companion site of this book.

³ In order to avoid confusion, recall that here X has been defined as the transpose of what we called a data matrix in previous chapters.

**FIGURE 19.4**

Indicative examples of the face images used.

**FIGURE 19.5**

Examples of eigenfaces.

The images are first vectorized (in \mathbb{R}^l , $l = 168 \times 168 = 28,224$) and in the sequel are concatenated in the columns of the $28,224 \times 1924$ matrix X . Moreover, the mean value across each one of the rows is computed and then subtracted from the corresponding element of each column.

In this case, where $l > N$, it is convenient to compute the eigenvectors of $X^T X$, denoted by \mathbf{v}_i , $i = 1, \dots, N$, and then the principal axis directions, that is, the eigenvectors of XX^T are computed by $\mathbf{u}_i \propto X\mathbf{v}_i$ (Chapter 6, Eq. (6.18)). These eigenvectors can be rearranged in a matrix form to give 168×168 images, known as *eigenimages*, which in the particular case of face images are referred to as *eigenfaces*. Fig. 19.5 shows examples of eigenfaces resulting from the PCA of matrix X and specifically those corresponding, from top left to bottom right, to the 1st, 2nd, 6th, 7th, 8th, 10th, 11th, and 17th largest eigenvalues.

Next, the quality of reconstruction of an original image, in terms of its lower-dimensional representation, is examined according to Eq. (19.13) for different values of m . As an example, the images depicting Marilyn Monroe and Andy Warhol, shown in Fig. 19.4, are chosen. The results are illustrated in Fig. 19.6. It is observed that for $m = 100$, or even better, for $m = 600$, the resulting approximation is very close to the original images. Note that exact reconstruction will be achieved when the full set of the 1924 eigenfaces is used.

To put our previous findings in an information retrieval context, assume that one has available an image and wants to know what person is depicted in it. Assuming that the image of this person is in the database, the procedure would be (a) to vectorize the image, (b) to project it onto the subspace

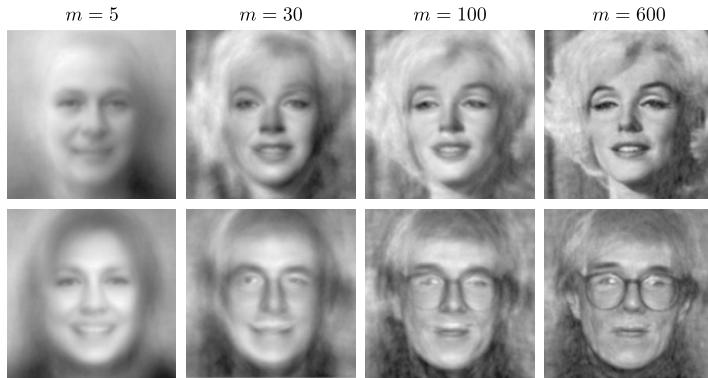
**FIGURE 19.6**

Image compression and reconstruction based on the first m eigenvectors.

spanned by the, say, $m = 100$ eigenfaces, and (c) to search in this lower-dimensional space to identify the vectorized image in the database that is closest in the Euclidean norm sense. Usually, it is preferable to identify the, say, five or ten most similar images and rank them according to the Euclidean distance (or any other distance) similarity. Then, through the database, he/she can have the name and all the associated information that is kept in the database.

In information retrieval, each one of the images in the database could be stored in terms of the corresponding vector of the principal scores.

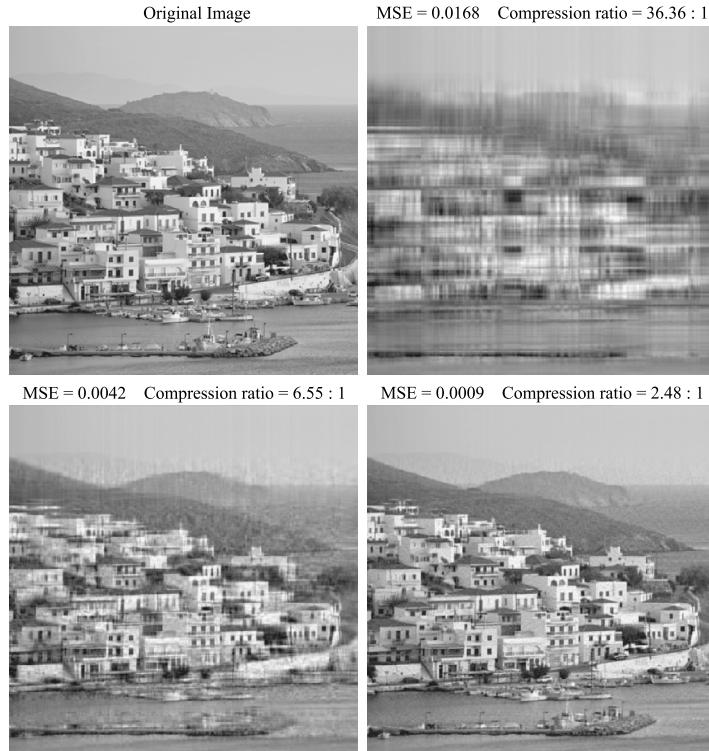
Example 19.2. In this example, the use of PCA for image compression is demonstrated. In the previous example, PCA was performed across the different images of a database. Here, the focus will be on a single image.

The pixel values of the image are stored in an $l \times N$ matrix X and the columns of this matrix are considered to be the observation vectors $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$. Note that X needs to be zero mean along the rows so the mean vector, $\bar{\mathbf{x}}$, is computed and subtracted from each column. Then the eigenvectors corresponding to the m , $1 \leq m < l$, largest eigenvalues are obtained either via the sample covariance matrix or directly through SVD. Exploiting the matrix factorization formulation of PCA in Eq. (19.22) a compressed representation of X , comprising m instead of l rows, is given by

$$Z^{(m)} = \underbrace{[\mathbf{u}_1, \dots, \mathbf{u}_m]}_{m \times l}^T X, \quad (19.24)$$

where the dimensionality m has been explicitly brought into the notation. Thus, only $Z^{(m)}$ and $\mathbf{u}_1, \dots, \mathbf{u}_m$ are needed to get an estimate of the, mean-subtracted, X via Eq. (19.21). Finally, in order to reconstruct the image, the mean vector $\bar{\mathbf{x}}$ needs to be added back to each column (see Eq. (19.23)).

The effectiveness of the PCA-based image compression will be demonstrated with the aid of the top-left image depicted in Fig. 19.7. This image is square, having $l = N = 400$. For any m chosen, the compression ratio is easily computed considering that instead of 400×400 values of the original image, after compression the storage of $2 \times m \times 400$ values for the matrix $Z^{(m)}$ and the eigenvectors,

**FIGURE 19.7**

PCA-based image compression. The image is from the Greek island Andros.

$\mathbf{u}_1, \dots, \mathbf{u}_m$, plus 400 values for the mean vector $\bar{\mathbf{x}}$ is needed. This amounts to a compression ratio of $400 : (2m + 1)$. The reconstructed images together with the corresponding MSE between the original and the reconstructed image for different compression rates are shown in Fig. 19.7.

Remarks 19.2.

- *Subspace tracking:* Online subspace tracking is another old area with a revived interest recently. A well-known algorithm of relative low complexity, for tracking the signal subspace, is the so-called projection approximation subspace tracking (PAST), proposed in [197]. In PAST, the recursive least-squares (RLS) technique is employed for subspace estimation. Alternative algorithms in this line of philosophy have been presented in, for example, [69,115,170,179].

More recently, the work in [25,50,137] tackles the problem of subspace tracking with missing/unobserved data. The methodology presented in [25] is based on gradient descent iterations on the Grassmannian manifold. Furthermore, the algorithms of [50,137] attempt to estimate the unknown subspace by minimizing properly constructed loss functions.

Finally, [51,52,92,132,162] attack the subspace tracking problem in environments where observations are contaminated by outlier noise.

19.5 INDEPENDENT COMPONENT ANALYSIS

The latent variable interpretation of PCA was summarized in Eqs. (19.17)–(19.19), where each one of the observed random variables, x_i , is (approximately) written as a linear combination of the latent variables (principal components in this case), z_i , which are in turn obtained via Eq. (19.19), imposing the uncorrelatedness constraint.

The kick-off point for ICA is to assume that the following latent model is true:

$$\mathbf{x} = A\mathbf{s}, \quad (19.44)$$

where the (unknown) latent variables of \mathbf{s} are assumed to be mutually statistically *independent* and we refer to them as the *independent components* (ICs). The task then comprises obtaining estimates of both the matrix A and the independent components. We will focus on the case where A is an $l \times l$ square matrix. Extensions to fat and tall matrices, corresponding to scenarios where the number of latent variables, m , is smaller or larger than the number of the observed random variables, l , have also been considered and developed (see, e.g., [100]).

Matrix A is known as the *mixing matrix* and its elements, a_{ij} , as the *mixing coefficients*. The resulting estimates of the latent variables will be denoted as z_i , $i = 1, 2, \dots, l$, and we will also refer to them as independent components. The observed random variables, x_i , $i = 1, 2, \dots, l$, are sometimes called the *mixture variables* or simply mixtures.

To obtain the estimates of the latent variables, we adopt the model

$$\hat{\mathbf{s}} := \mathbf{z} = W\mathbf{x}, \quad (19.45)$$

where W is also known as the *unmixing* or *separating* matrix. Note that

$$\mathbf{z} = W A \mathbf{s},$$

and we have to estimate the unknown parameters, so that \mathbf{z} is as close to \mathbf{s} as possible. For square matrices, $A = W^{-1}$, assuming invertibility.

19.5.1 ICA AND GAUSSIANITY

Although in general in statistics adopting the Gaussian assumption for a PDF seems to be rather a “blessing,” in the case of ICA this is not true anymore. This can easily be understood if we look at the consequences of adopting the Gaussian assumption. If the independent components follow Gaussian distributions, their joint PDF is given by

$$p(\mathbf{s}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{||\mathbf{s}||^2}{2}\right), \quad (19.46)$$

where for simplicity we have assumed that all the variables are normalized to unit variance. Let the mixing matrix, A , be an orthogonal one, that is, $A^{-1} = A^T$. Then the joint PDF of the mixtures is readily obtained as (see Eq. (2.45))

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{||A^T \mathbf{x}||^2}{2}\right) |\det(A^T)|. \quad (19.47)$$

However due to the orthogonality of A , we have $\|A^T \mathbf{x}\|^2 = \|\mathbf{x}\|^2$ and $|\det(A^T)| = 1$, which makes $p(\mathbf{s})$ indistinguishable from $p(\mathbf{x})$. That is, no conclusion about A can be drawn by observing \mathbf{x} , as all related information has been lost. Seen from another point of view, the mixtures \mathbf{x}_i are mutually uncorrelated, as $\Sigma_{\mathbf{x}} = I$, and ICA can provide no further information. This is a direct consequence of the fact that uncorrelatedness for jointly Gaussian variables is equivalent to independence (see Section 2.3.2). In other words, if the latent variables are Gaussians, ICA cannot take us any further than PCA, because the latter provides uncorrelated components. That is, the mixing matrix, A , is *not identifiable* for Gaussian independent components. In a more general setting, in a case where some of the components are Gaussians and some are not, ICA can identify the non-Gaussian ones. Thus, for a matrix A to be identifiable, *at most one* of the independent components can be Gaussian.

From a mathematical point of view, the ICA task is ill-posed for Gaussian variables. Indeed, assume that a set of independent Gaussian components, \mathbf{z} , have been obtained; then any linear transformation on \mathbf{z} by a unitary matrix will also be a solution (as shown previously). Note that this problem is bypassed in PCA, because the latter imposes a specific structure on the transformation matrix.

In order to deal with independence one has to involve, in one way or another, higher-order statistical information. Second-order statistical information suffices for imposing uncorrelatedness, as is the case with PCA, but it is not enough for ICA. To this end, a large number of techniques and algorithms have been developed over the years and reviewing all these techniques is far beyond the limits imposed on a book section. The goal here is to provide the reader with the essence behind these techniques and emphasize the need to bring higher-order statistics into the game. The interested reader can delve deeper in this field from [55, 58, 81, 100, 120].

19.5.2 ICA AND HIGHER-ORDER CUMULANTS

Imposing the constraint on the components of \mathbf{z} to be independent is equivalent to demanding all higher-order *cross-cumulants* (Appendix B.3) to be zero. One possibility to achieve this is to restrict ourselves up to the fourth-order cumulants [57]. As stated in Appendix B.3, the first three cumulants for zero mean variables are equal to the corresponding moments, that is,

$$\begin{aligned}\kappa_1(z_i) &= \mathbb{E}[z_i] = 0, \\ \kappa_2(z_i, z_j) &= \mathbb{E}[z_i z_j], \\ \kappa_3(z_i, z_j, z_k) &= \mathbb{E}[z_i z_j z_k],\end{aligned}$$

and the fourth-order cumulants are given by

$$\begin{aligned}\kappa_4(z_i, z_j, z_k, z_r) &= \mathbb{E}[z_i z_j z_k z_r] - \mathbb{E}[z_i z_j] \mathbb{E}[z_k z_r] \\ &\quad - \mathbb{E}[z_i z_k] \mathbb{E}[z_j z_r] - \mathbb{E}[z_i z_r] \mathbb{E}[z_j z_k].\end{aligned}$$

An assumption that is employed is that the involved PDFs are symmetric, which renders odd-order cumulants to zero. Thus, we are left only with the second- and fourth-order cumulants. Under the previous assumptions, our goal is to estimate the unmixing matrix, W , so that (a) the second-order and (b) the fourth-order cumulants become zero. This is achieved in two steps.

Step 1: Compute

$$\hat{\mathbf{z}} = U^T \mathbf{x}, \tag{19.48}$$

where U is the unitary $l \times l$ matrix associated with PCA. This transformation guarantees that the components of $\hat{\mathbf{z}}$ are uncorrelated, that is,

$$\mathbb{E}[\hat{z}_i \hat{z}_j] = 0, \quad i \neq j, \quad i, j = 1, 2, \dots, l.$$

Step 2: Compute an orthogonal matrix, \hat{U} , such that the fourth-order cross-cumulants of the components of the transformed random vector

$$\mathbf{z} = \hat{U}^T \hat{\mathbf{z}} \tag{19.49}$$

are zero. In order to achieve this, the following maximization task is solved:

$$\max_{\hat{U} \hat{U}^T = I} \sum_{i=1}^l \kappa_4^2(\mathbf{z}_i). \tag{19.50}$$

Step 2 is justified as follows. It can be shown [57] that the sum of the squares of the fourth-order cumulants is invariant under a linear transformation by an orthogonal matrix. Therefore, as the sum of the squares of the fourth-order cumulants is fixed for \mathbf{z} , maximizing the sum of the squares of the autocumulants of \mathbf{z} will force the corresponding cross-cumulants to zero. Observe that this is basically a diagonalization problem of the fourth-order cumulant multidimensional array. In practice, this can be achieved by generalizing the method of Givens rotations, used for matrix diagonalization [57]. Note that the sum that is maximized is a function of (a) the elements of the unknown matrix \hat{U} , (b) the elements of the known (for this step) matrix U , and (c) the cumulants of the random components of the mixtures \mathbf{x} , which have to be estimated prior to the application of the method. In practice, it usually turns out that setting the cross-cumulants to zero is only approximately achieved. This is because the model in Eq. (19.44) may not be exact, for example, due to the existence of noise. Also, the cumulants of the mixtures are only approximately known, because they are estimated by the available observations.

Once U and \hat{U} have been computed, the unmixing matrix is readily available and we can write

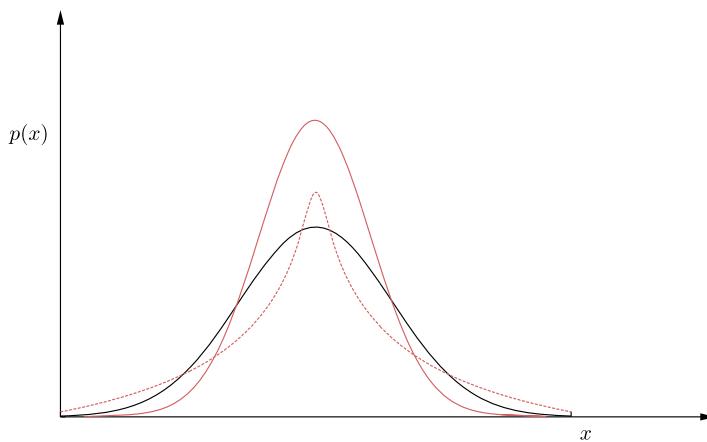
$$\mathbf{z} = W \mathbf{x} = (U \hat{U})^T \mathbf{x},$$

and the mixing matrix is given as $A = W^{-1}$.

A number of algorithms have been developed around the idea of higher-order cumulants, which are also known as *tensorial methods*. Tensors are generalizations of matrices and cumulant tensors are generalizations of the covariance matrix. Moreover, note that as the eigenanalysis of the covariance matrix leads to uncorrelated (principal) components, the eigenanalysis of the cumulant tensor leads to independent components. The interested reader can obtain a more detailed account of such techniques from [39, 57, 119].

ICA Ambiguities

Any ICA method can (approximately) recover the independent components within the following two indeterminacies.

**FIGURE 19.8**

A Gaussian (full-gray line) a super-Gaussian (dotted red line) and a sub-Gaussian (full-red line).

- Independent components (ICs) are recovered to within a constant factor. Indeed, if A and \mathbf{z} are the recovered quantities by an ICA algorithm, then $(1/a)A$ and $a\mathbf{z}$ is also a solution, as is readily seen from Eq. (19.44). Thus, usually the recovered latent variables (ICs) are normalized to unit variance.
- We cannot determine the order of the ICs. Indeed, if A and \mathbf{z} have been recovered and P is a permutation matrix, then AP^{-1} and $P\mathbf{z}$ is also a solution, because the components of $P\mathbf{z}$ are the same as those of \mathbf{z} in a different order (with the same statistical properties).

19.5.3 NON-GAUSSIANITY AND INDEPENDENT COMPONENTS

The fourth-order (auto)cumulant, of a random variable, z ,

$$\kappa_4(z) = \mathbb{E}[z^4] - 3(\mathbb{E}[z^2])^2,$$

is known as the *kurtosis* of the variable and it is a measure of *non-Gaussianity*. Variables following the Gaussian distribution have zero kurtosis. Sub-Gaussian variables (variables whose PDF falls at a slower rate than the Gaussian for the same variance) have negative kurtosis. Super-Gaussian variables (corresponding to PDFs that fall at a faster rate than the Gaussian) have positive kurtosis. Thus, if we keep the variance fixed (e.g., for variables normalized to unit variance), maximizing the sum of squared kurtosis, it results in maximizing the non-Gaussianity of the recovered ICs. Usually, the absolute value of the kurtosis of the recovered ICs is used as a measure of ranking them. This is important if ICA is used as a feature generation technique. Fig. 19.8 shows some typical examples of a sub-Gaussian and a super-Gaussian together with the corresponding Gaussian distribution. Also, another typical example of a sub-Gaussian distribution is the uniform one.

Recall from Chapter 12 (Section 12.8.1) that the Gaussian distribution is the one that maximizes the entropy under the variance and mean constraints. In other words, it is the most random one, under these

constraints, and from this point of view the least informative with respect to the underlying structure of the data. In contrast, distributions that have the least resemblance to the Gaussian are more interesting as they are able to better unveil the structure associated with the data. This observation is at the heart of *projection pursuit*, which is closely related to the ICA family of techniques. The essence of these techniques is to search for directions in the feature space where the data projections are described in terms of non-Gaussian distributions [97,106].

19.5.4 ICA BASED ON MUTUAL INFORMATION

The approach based on zeroing the second- and fourth-order cross-cumulants is not the only one. An alternative path is to estimate W by minimizing the *mutual information* among the latent variables. The notion of mutual information was introduced in Section 2.5. Elaborating a bit on Eq. (2.158) and performing the integrations on the right-hand side (for the case of more than two variables), it is readily shown that

$$I(\mathbf{z}) = -H(\mathbf{z}) + \sum_{i=1}^l H(z_i), \quad (19.51)$$

where $H(z_i)$ is the associated entropy of z_i , defined in Eq. (2.157). In Section 2.5 it has been shown that $I(\mathbf{z})$ is equal to the Kullback–Leibler (KL) divergence between the joint PDF $p(\mathbf{z})$ and the product of the respective marginal probability densities, $\prod_{i=1}^l p_i(z_i)$. The KL divergence (and, hence, the associated mutual information $I(\mathbf{z})$) is a nonnegative quantity and it becomes zero if the components z_i are statistically independent. This is because only in this case the joint PDF becomes equal to the product of the corresponding marginal PDFs, leading the KL divergence to zero. Hence, the idea now becomes to compute W so as to force $I(\mathbf{z})$ to be minimum, as this will make the components of \mathbf{z} as *independent as possible*. Plugging Eq. (19.45) into Eq. (19.51) and taking into account the formula that relates the two PDFs associated with \mathbf{x} and \mathbf{z} (Eq. (2.45)), we end up with

$$I(\mathbf{z}) = -H(\mathbf{x}) - \ln |\det(W)| - \sum_{i=1}^l \int p_i(z_i) \ln p_i(z_i) dz_i. \quad (19.52)$$

The elements of the unknown matrix, W , are also hidden in the marginal PDFs of the latent variables, z_i . However, it is not easy to express this dependence explicitly. One possibility is to expand each one of the marginal densities around the Gaussian PDF, denoted here as $g(z)$, following Edgeworth's expansion (Appendix B), and truncate the series to a reasonable approximation. For example, keeping the first two terms in the Edgeworth expansion we have

$$p_i(z_i) = g(z_i) \left(1 + \frac{1}{3!} \kappa_3(z_i) H_3(z_i) + \frac{1}{4!} \kappa_4(z_i) H_4(z_i) \right), \quad (19.53)$$

where $H_k(z_i)$ is the Hermite polynomial of order k (Appendix B). To obtain an approximate expression for $I(\mathbf{z})$, in terms of cumulants of z_i and W , we can (a) insert in Eq. (19.52) the PDF approximation in Eq. (19.53), (b) adopt the approximation $\ln(1+y) \simeq y - y^2$, and (c) perform the integrations. This

is no doubt a rather painful task! For the case of Eq. (19.53) and constraining W to be orthogonal, the following is obtained (e.g., [100]):

$$I(\mathbf{z}) \approx C - \sum_{i=1}^l \left(\frac{1}{12} \kappa_3^2(z_i) + \frac{1}{48} \kappa_4^2(z_i) + \frac{7}{48} \kappa_4^4(z_i) - \frac{1}{8} \kappa_3^2(z_i) \kappa_4(z_i) \right), \quad (19.54)$$

where C is a quantity independent of W . Under the assumption that the PDFs are symmetric (thus, third-order cumulants are zero), it can be shown that minimizing the approximate expression of the mutual information in Eq. (19.54) is equivalent to maximizing the sum of the squares of the fourth-order cumulants. Note that the orthogonal W constraint is not necessary, and if it is not adopted other approximate expressions for $I(\mathbf{z})$ result (for example, [91]).

Minimization of $I(\mathbf{z})$ in Eq. (19.54) can be carried out by a gradient descent technique (Chapter 5), where the involved expectations (associated with the cumulants) are replaced by the respective instantaneous values. Although we will not treat the derivation of algorithmic schemes in detail, in order to get a flavor of the involved tricks, let us go back to Eq. (19.52), before we apply the approximations. Because $H(\mathbf{x})$ does not depend on W , minimizing $I(\mathbf{z})$ is equivalent to the maximization of

$$J(W) = \ln |\det(W)| + \mathbb{E} \left[\sum_{i=1}^l \ln p_i(z_i) \right]. \quad (19.55)$$

Taking the gradient of the cost function with respect to W results in

$$\frac{\partial J(W)}{\partial W} = W^{-T} - \mathbb{E}[\boldsymbol{\phi}(\mathbf{z})\mathbf{x}^T], \quad (19.56)$$

where

$$\boldsymbol{\phi}(\mathbf{z}) := \left[-\frac{p'_1(z_1)}{p_1(z_1)}, \dots, -\frac{p'_l(z_l)}{p_l(z_l)} \right]^T, \quad (19.57)$$

and

$$p'_i(z_i) := \frac{dp_i(z_i)}{dz_i}, \quad (19.58)$$

and we used the formula

$$\frac{\partial \det(W)}{\partial W} = W^{-T} \det(W).$$

Obviously, the derivatives of the marginal probability densities depend on the type of approximation adopted in each case. The general gradient ascent scheme at the i th iteration step can now be written as

$$W^{(i)} = W^{(i-1)} + \mu_i \left((W^{(i-1)})^{-T} - \mathbb{E}[\boldsymbol{\phi}(\mathbf{z})\mathbf{x}^T] \right),$$

or

$$W^{(i)} = W^{(i-1)} + \mu_i \left(I - \mathbb{E}[\boldsymbol{\phi}(\mathbf{z})\mathbf{z}^T] \right) (W^{(i-1)})^{-T}. \quad (19.59)$$

In practice, the expectation operator is neglected and random variables are replaced by respective observations, in the spirit of the stochastic approximation rationale (Chapter 5).

The update equation in Eq. (19.59) involves the inversion of the transpose of the current estimate of W . Besides the computational complexity issues, there is no guarantee of the invertibility in the process of adaptation. The use of the so-called *natural gradient* [68], instead of the gradient in Eq. (19.56), results in

$$W^{(i)} = W^{(i-1)} + \mu_i \left(I - \mathbb{E}[\phi(\mathbf{z})\mathbf{z}^T] \right) W^{(i-1)}, \quad (19.60)$$

which does not involve matrix inversion and at the same time improves convergence. A more detailed treatment of this issue is beyond the scope of this book. Just to give an incentive to the mathematically inclined reader for indulging more deeply this field, it suffices to say that our familiar gradient, that is, Eq. (19.56), points to the steepest ascent direction if the space is Euclidean. However, in our case the parameter space consists of all the nonsingular $l \times l$ matrices, which is a multiplicative group. The space is Riemannian and it turns out that the natural gradient, pointing to the steepest ascent direction, results if we multiply the gradient in Eq. (19.56) by $W^T W$, which is the corresponding Riemannian metric tensor [68].

Remarks 19.4.

- From the gradient in Eq. (19.56), it is easy to see that at a stationary point the following is true:

$$\frac{\partial J(W)}{\partial W} W^T = \mathbb{E}[I - \phi(\mathbf{z})\mathbf{z}^T] = O. \quad (19.61)$$

In other words, what we achieve with ICA is a *nonlinear generalization* of PCA. Recall that for the latter, the uncorrelatedness condition can be written as

$$\mathbb{E}[I - \mathbf{z}\mathbf{z}^T] = O. \quad (19.62)$$

The presence of the *nonlinear* function ϕ takes us beyond simple uncorrelatedness, and brings the cumulants into the scene. As a matter of fact, Eq. (19.61) was the one that inspired the early pioneering work on ICA, as a direct nonlinear generalization of PCA [93,107].

- The origins of ICA are traced back to the seminal paper [93]. For a number of years, it remained an activity pretty much within the French signal processing and statistics communities. Two papers were catalytic for its widespread use and popularity, namely, [18] in the mid-1990s and the development of the FastICA⁴ [99], which allowed for efficient implementations (see [108] for a related review).
- In machine learning, the use of ICA as a feature generation technique is justified by the following argument. In [17], it is suggested that the outcome of the early processing performed by the visual cortical feature detectors might be the result of a *redundancy reduction* process. Thus, searching for independent features, conditioned on the input data, is in line with such a claim (see, for example, [75,114] and the references therein).

⁴ <http://research.ics.aalto.fi/ica/fastica/index.shtml>.

- Although we have focused on the noiseless case, extensions of ICA to noisy tasks have also been proposed (see, e.g., [100]). For an extension of ICA in the complex-valued case, see [2]. Nonlinear extensions have also been considered, including kernelized ICA versions (for example, [13]).
- In [3], the treatment of ICA also involves random processes and a wider class of signals, including Gaussians, can be identified.
- In [7], the multiset ICA framework of *independent vector analysis* (IVA) is discussed. It is shown that it generalizes the multiset CCA if higher-order, besides second-order, statistics are taken into account.

19.5.5 ALTERNATIVE PATHS TO ICA

Besides the previously discussed two paths to ICA, a number of alternatives have been suggested, shedding light on different aspects of the problem. Some notable directions are the following.

- *Infomax principle*: This method assumes that the latent variables are the outputs of a nonlinear system (neural network, Chapter 18) of the form

$$z_i = \phi_i(\mathbf{w}_i^T \mathbf{x}) + \eta, \quad i = 1, 2, \dots, l,$$

where ϕ_i are nonlinear functions and η is additive Gaussian noise. The weight vectors \mathbf{w}_i are computed so as to maximize the entropy of the outputs; the reasoning is based on some information theoretic arguments concerning the information flow in the network [18].

- *Maximum likelihood*: Starting from Eq. (19.44), the PDF of the observed variables is expressed in terms of the PDFs of the independent components

$$p(\mathbf{x}) = |\det(W)| \prod_{i=1}^l p_i(\mathbf{w}_i^T \mathbf{x}_i),$$

where we used

$$W := A^{-1}.$$

Assuming that we have N observations, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, and taking the logarithm of the joint $p(\mathbf{x}_1, \dots, \mathbf{x}_N)$, one can maximize the log-likelihood with respect to W . It is straightforward to derive the log-likelihood function and to observe that it is very similar to $J(W)$ given in Eq. (19.55). The p_i s are chosen so as to belong to families of non-Gaussians (for example, [100]). A connection between the infomax approach and the maximum likelihood one has been established in [37,38].

- *Negentropy*: According to this method, the starting point is to maximize the non-Gaussianity, which is now measured in terms of the *negentropy*, defined as

$$J(\mathbf{z}) := H(\mathbf{z}_{\text{Gauss}}) - H(\mathbf{z}),$$

where $\mathbf{z}_{\text{Gauss}}$ corresponds to Gaussian distributed variables of the same covariance matrix, which we know corresponds to the maximum entropy, H . Thus, maximizing the negentropy, which is a nonnegative function, is equivalent to making the latent variables as less Gaussian as possible. Usually, approximations of the negentropy are employed, which are expressed in terms of higher-order cumulants, or by matching the nonlinearity to source distribution [100,141].

- If the unmixing matrix is constrained to be orthogonal, the negentropy and the maximum likelihood approaches become equivalent [2].

THE COCKTAIL PARTY PROBLEM

A classical application that demonstrates the power of the ICA is the so-called *cocktail party problem*. In a party, there are various people speaking; in our case, we are going to consider music as well. Let us say that there are people (a female and a male) and there is also monophonic music, making three sources of sound in total. Then, three microphones (as many as the sources) are placed in different places in the room and the mixed speech signals are recorded. We denote the inputs to the three microphones as $x_1(t)$, $x_2(t)$, and $x_3(t)$, respectively. In the simplest of the models, the three recorded signals can be considered as linear combinations of the individual source signals. Delays are not considered. The goal is to use ICA and recover the original speech and music from the recorded mixed signals.

To this end and in order to bring the task in the formulation we have previously adopted, we consider the values of the three signals at different time instants as different observations of the corresponding random variables, x_1 , x_2 , and x_3 , which are put together to form the random vector \mathbf{x} . We further adopt the very reasonable assumption that the original source signals, denoted as $s_1(t)$, $s_2(t)$, and $s_3(t)$, are independent and (similarly as before) the values at different time instants correspond to the values of three latent variables, denoted together as a random vector \mathbf{s} .

We are ready now to apply ICA to compute the unmixing matrix W , from which we can obtain the estimates of the ICs corresponding to the observations received by the three microphones,

$$\mathbf{z}(t) = [z_1(t), z_2(t), z_3(t)]^T = W[x_1(t), x_2(t), x_3(t)]^T.$$

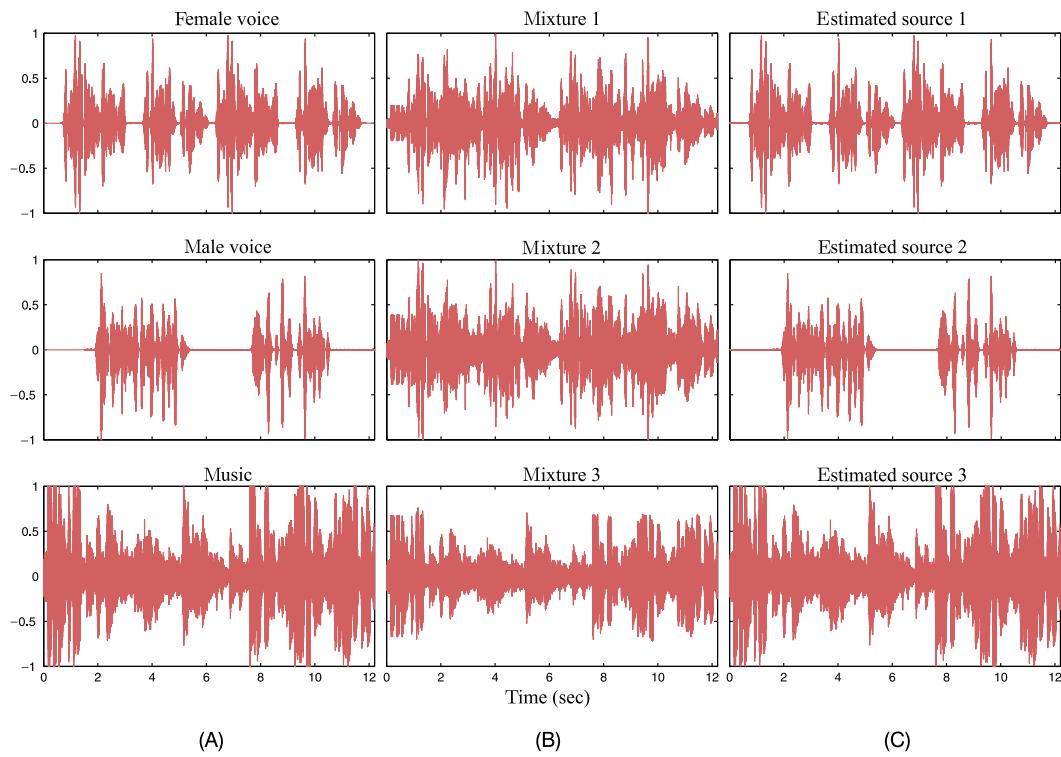
Fig. 19.9A shows the three different signals, which are linearly combined (by a set of mixing coefficients defining a mixing matrix A) to form the three “microphone signals.” Fig. 19.9B shows the resulting signals, which are then used as described before for the ICA analysis. Fig. 19.9C shows the recovered original signals, as the corresponding ICs. The FastICA algorithm was employed.⁵ Fig. 19.10 is the result when PCA is used and the original signals are obtained via the (three) principal components.

One can observe that ICA manages to separate the signals with very good accuracy, whereas PCA fails. The reader can also listen to the signals by downloading the corresponding “.wav” files from the site of this book.

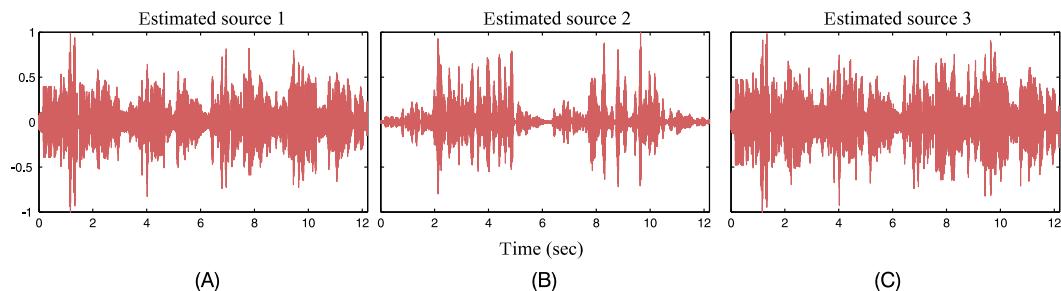
Note that the cocktail party problem is representative of a large class of tasks where a number of recorder signals result as linear combinations of other independent signals; the goal is the recovery of the latter. A notable application of this kind is found in electroencephalography (EEG). EEG data consist of electrical potentials recorded at different locations on the scalp (or more recently, in the ear [112]), which are generated by the combination of different underlying components of brain and muscle activity. The task is to use ICA to recover the components, which in turn can unveil useful information about the brain activity (for example, [158]).

The cocktail party problem is a typical example of a more general class of tasks known as *blind source separation* (BSS). The goal in these tasks is to estimate the “causes” (sources, original signals)

⁵ <http://research.ics.aalto.fi/ica/fastica/>.

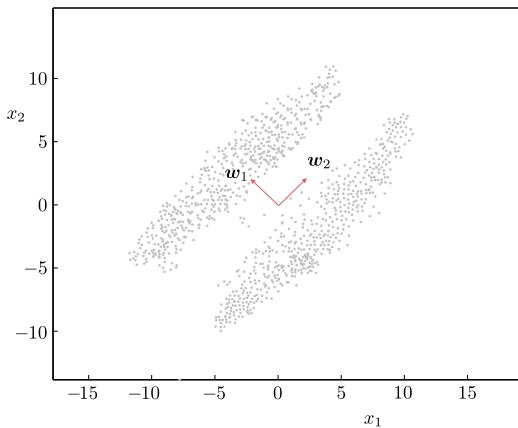
**FIGURE 19.9**

ICA source separation in the cocktail party setting.

**FIGURE 19.10**

PCA source separation in the cocktail party setting.

based only on information residing in the observations, without any other extra information, and this is the reason that the word “blind” is used. Viewed in another way, BSS is an example of unsupervised learning. ICA is one among the most widely used techniques for such problems.

**FIGURE 19.11**

The setup for the ICA simulation example. The two vectors point to the projection directions resulting from the analysis. The optimal direction for projection, resulting from the ICA analysis, is that of w_2 .

Example 19.4. The goal of this example is to demonstrate the power of ICA as a feature generation technique, where the most informative of the generated features are to be kept.

The example is a realization of the case shown in Fig. 19.11. A number of 1024 samples of a two-dimensional normal distribution were generated.

The mean and covariance matrix of the normal PDF were

$$\boldsymbol{\mu} = [-2.6042, 2.5]^T, \quad \Sigma = \begin{bmatrix} 10.5246 & 9.6313 \\ 9.6313 & 11.3203 \end{bmatrix}.$$

Similarly, 1024 samples from a second normal PDF were generated with the same covariance matrix and mean $-\boldsymbol{\mu}$. For the ICA, the method based on the second- and fourth-order cumulants, presented in this section, was used. The resulting transformation matrix W is

$$W = \begin{bmatrix} -0.7088 & 0.7054 \\ 0.7054 & 0.7088 \end{bmatrix} := \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix}.$$

The vectors \mathbf{w}_2 and \mathbf{w}_1 point in the principal and minor axis directions, respectively, obtained from the PCA analysis. According to PCA, the most informative direction is along the principal axis \mathbf{w}_2 , which is the one with maximum variance. However, the most interesting direction for projection, according to the ICA analysis, is that of \mathbf{w}_1 . Indeed, the kurtosis of the obtained ICs z_1, z_2 along these directions are

$$\begin{aligned} \kappa_4(z_1) &= -1.7, \\ \kappa_4(z_2) &= 0.1, \end{aligned}$$

respectively. Thus, projection in the principal (PCA) axis direction results in a variable with a PDF close to a Gaussian. The projection on the minor axis direction results in a variable with a PDF that

deviates from the Gaussian (it is bimodal) and it is the more interesting one from the classification point of view. This can be easily verified by looking at the figure; projecting on the direction w_2 leads to class overlapping.

19.6 DICTIONARY LEARNING: THE k -SVD ALGORITHM

The concept of *overcomplete dictionaries* and their importance in modeling real-world signals have been introduced in Chapter 9. We return to this topic, this time in a more general setting. There, the dictionary was assumed known with preselected atoms. In this section, the blind version of this task is considered; that is, the atoms of the dictionary are unknown and have to be estimated from the observed data. Recall that this was the case with ICA; however, instead of the independence concept, used for ICA, sparsity arguments will be mobilized here. Giving the freedom to the dictionary to adapt to the needs of the specific, each time, input can lead to enhanced performance compared to dictionaries with preselected atoms.

Our starting point is that the observed l random variables are expressed in terms of $m > l$ latent ones according to the linear model

$$\mathbf{x} = A\mathbf{z}, \quad \mathbf{x} \in \mathbb{R}^l, \quad \mathbf{z} \in \mathbb{R}^m, \quad (19.63)$$

and A is an unknown $l \times m$ matrix. Usually, $m \gg l$. Even if A were known and fixed, it does not need special mathematical skills to see that this task has not a single solution and one has to embed constraints into the problem. To this end, we are going to adopt sparsity promoting constraints, as we have already discussed in various parts in this book.

Let \mathbf{x}_n , $n = 1, 2, \dots, N$, be the observations that will constitute the only available information. The task is to obtain the atoms (columns of A) of the dictionary as well as the latent variables that are assumed to be sparse; that is, we are going to establish a sparse representation of our input observations (vectors). No doubt, there are different paths to achieve the goal. We are going to focus on one of the most widely known and used methods, known as *k -SVD*, proposed in [4].

Let $X := [\mathbf{x}_1, \dots, \mathbf{x}_N]$, $A := \{\mathbf{a}_1, \dots, \mathbf{a}_m\}$, and $Z := [\mathbf{z}_1, \dots, \mathbf{z}_N]$, where \mathbf{z}_n is the latent vector corresponding to the input \mathbf{x}_n , $n = 1, 2, \dots, N$. The dictionary learning task is cast as the following optimization problem

$$\text{minimize with respect to } A, Z \quad \|X - AZ\|_F^2, \quad (19.64)$$

$$\text{subject to} \quad \|\mathbf{z}_n\|_0 \leq T_0, \quad n = 1, 2, \dots, N, \quad (19.65)$$

where T_0 is a threshold value and $\|\cdot\|_0$ denotes the ℓ_0 norm, as discussed in Chapter 9. This is a nonconvex optimization task, and it is performed iteratively; each iteration comprises two stages. In the first one, A is assumed to be fixed and optimization is carried out with respect to \mathbf{z}_n , $n = 1, 2, \dots, N$. In the second stage, the latent vectors are assumed fixed and optimization is carried out with respect to the columns of A .

In *k -SVD*, a slightly different rationale is adopted. While optimizing with respect to the columns of A , one at a time, an update of some of the elements of Z is also performed. This is a crucial difference

of k -SVD with the more standard optimization techniques; it appears to lead to improved performance in practice.

Stage 1: Assume A to be known and fixed to the value obtained from the previous iteration. Then the associated optimization task becomes

$$\begin{aligned} \min_Z \quad & ||X - AZ||_F^2, \\ \text{s.t.} \quad & ||z_n||_0 \leq T_0, \quad n = 1, 2, \dots, N, \end{aligned}$$

which, due to the definition of the Frobenius norm, is equivalent to solving N distinct optimization tasks,

$$\min_{z_n} \quad ||x_n - Az_n||^2, \quad (19.66)$$

$$\text{s.t.} \quad ||z_n||_0 \leq T_0, \quad n = 1, 2, \dots, N. \quad (19.67)$$

A similar objective is met if the following optimization tasks are considered instead:

$$\begin{aligned} \min_{z_n} \quad & ||z_n||_0, \\ \text{s.t.} \quad & ||x_n - Az_n||^2 < \epsilon, \quad n = 1, 2, \dots, N, \end{aligned}$$

where ϵ is a constant acting as an upper bound of the error.

The task in Eqs. (19.66) and (19.67) can be solved by any one of the ℓ_0 minimization solvers, which have been considered in Chapter 10, for example, the OMP. This stage is known as *sparse coding*.

Stage 2: This stage is known as the *codebook update*. Having obtained z_n , $n = 1, 2, \dots, N$ (for fixed A), from stage 1, the goal now is to optimize with respect to the columns of A . This is achieved on a *column-by-column* basis. Assume that we currently consider the update of a_k ; this is carried out so as to minimize the (squared) Frobenius norm, $||X - AZ||_F^2$. To this end, we can write the product AZ as a sum of rank one matrices, that is,

$$AZ = [a_1, \dots, a_m][z_1^r, \dots, z_m^r]^T = \sum_{i=1}^m a_i z_i^{rT}, \quad (19.68)$$

where z_i^{rT} , $i = 1, 2, \dots, m$, are the *rows* of Z . Note that in the above sum, the vectors for indices, $i = 1, 2, \dots, k-1$, are fixed to their recently updated values during this second stage of the current iteration step, while and vectors corresponding to $i = k+1, \dots, m$, are fixed to the values that are available from the previous iteration step. This strategy allows for the use of the most recent updated information. We will now minimize with respect to the rank one outer product matrix, $a_k z_k^{rT}$. Observe that this product, besides the k th column of A , also involves the k th row of Z ; both of them will be updated. The rank one matrix is estimated so as to minimize

$$||E_k - a_k z_k^{rT}||_F^2, \quad (19.69)$$

where

$$E_k := X - \sum_{i=1, i \neq k}^m a_i z_i^{rT}.$$

In other words, we seek to find the best, in the Frobenius sense, rank one approximation of E_k . Recall from Chapter 6 (Section 6.4) that the solution is given via the SVD of E_k . However, if we do that, there is no guarantee that whatever sparse structure has been embedded in z_k^r , from the update in stage 1, will be retained. According to the k -SVD, this is bypassed by focusing on the active set, that is, involving only the nonzero of its coefficients. Thus, we first search for the locations of the nonzero coefficients in z_k^r and let

$$\omega_k := \left\{ j_k, 1 \leq j_k \leq N : z_k^r(j_k) \neq 0 \right\}.$$

Then, we form the reduced vector $\tilde{z}_k^r \in \mathbb{R}^{|\omega_k|}$, where $|\omega_k|$ denotes the cardinality of ω_k , which contains only the nonzero elements of z_k^r . A little thought reveals that when writing $X = AZ$, the column of current interest, a_k , contributes (as part of the corresponding linear combination) only to the columns x_{j_k} , $j_k \in \omega_k$, of X . We then collect the corresponding columns of E_k to construct a reduced-order matrix, \tilde{E}_k , which comprises the columns that are associated with the locations of the nonzero elements of z_k^r , and select $a_k \tilde{z}_k^{rT}$ so as to minimize

$$\|\tilde{E}_k - a_k \tilde{z}_k^{rT}\|_F^2. \quad (19.70)$$

Performing SVD, $\tilde{E}_k = UDV^T$, a_k is set equal to u_1 corresponding to the largest of the singular values and $\tilde{z}_k^r = D(1, 1)v_1$. Thus, the atoms of the dictionary are obtained in *normalized* form (recall from the theory of SVD that $\|u_1\| = 1$). In the sequel, the updated values obtained for \tilde{z}_k^r are placed in the corresponding locations in z_k^r . The latter now has at least as many zeros as it had before, as some of the elements in v_1 may be zeros. Simple arguments (Problem 19.3) show that at each iteration the error decreases and the algorithm converges to a local minimum. The success of the algorithm depends on the ability of the greedy algorithm to provide a sparse solution during the first stage. As we know from Chapter 10, greedy algorithms work well for sparsity levels, T_0 , small enough compared to l .

In summary, each iteration step of the k -SVD algorithm comprises the following computation steps.

- Initialize $A^{(0)}$ with columns normalized to unit ℓ_2 norm.
- Set $i = 1$.
- *Stage 1:* Solve the optimization task in Eqs. (19.66) and (19.67) to obtain the sparse coding representation vectors, z_n , $n = 1, 2, \dots, N$; use any algorithm developed for this task.
- *Stage 2:* For any column, $k = 1, 2, \dots, m$, in $A^{(i-1)}$, update it according to the following:
 - Identify the locations of the nonzero elements in the k th row of the computed, from stage 1, matrix Z .
 - Select the columns in E_k , which correspond to the locations of the nonzero elements of the k th row of Z and form a reduced-order error matrix, \tilde{E}_k .
 - Perform SVD on \tilde{E}_k : $\tilde{E}_k = UDV^T$.
 - Update the k th column of $A^{(i)}$ to be the eigenvector corresponding to the largest singular value, $a_k^{(i)} = u_1$.
 - Update Z , by embedding in the nonzero locations of its k th row the values $D(1, 1)v_1^T$.
- Stop if a convergence criterion is met.
- If not, $i = i + 1$, and continue.

WHY THE NAME k -SVD?

The SVD part of the name is pretty obvious. However, the reader may wonder about the presence of “ k ” in front. As stated in [4], the algorithm can be considered a generalization of the k -means algorithm, introduced in Chapter 12 (Algorithm 12.1). There, we can consider the mean values, which represent each cluster, as the code words (atoms) of a dictionary. During the first stage of the k -means learning, given the representatives of each cluster, a sparse coding scheme is performed; that is, each input vector is assigned to a single cluster. Thus, we can think of the k -means clustering as a sparse coding scheme that associates a latent vector with each one of the observations. Note that each of the latent vectors has only one nonzero element, pointing to the cluster where the respective input vector is assigned, according to the smallest Euclidean distance from all cluster representatives. This is a major difference with the k -SVD dictionary learning, during which each observation vector can be associated with more than one atom; hence, the sparsity level of the corresponding latent vector can be larger than one. Furthermore, based on the assignment of the input vectors to the clusters, in the second stage of the k -means algorithm, an update of the cluster representatives is performed, and for each representative only the input vectors assigned to it are used. This is also similar in spirit to what happens in the second stage of k -SVD. The difference is that each input observation may be associated with more than one atom. As pointed out in [4], if one sets $T_0 = 1$, the k -means algorithm can result from k -SVD.

DICTIONARY LEARNING AND DICTIONARY IDENTIFIABILITY

Dictionary learning was introduced via the ℓ_0 norm in (19.64)–(19.65). In a more general setting, the dictionary learning task is cast as follows:

$$\text{minimize with respect to } A \in \mathcal{A} \text{ and } Z \quad \{ \|X - AZ\|_F^2 + \lambda g(Z) \}, \quad (19.71)$$

where $g(Z)$ is a sparsity promoting function of the elements of matrix Z . For example,

$$g(Z) = \sum_{i=1}^m \sum_{j=1}^N |Z(i, j)|, \quad (19.72)$$

and λ is a regularization parameter. The set $\mathcal{A} \subseteq \mathbb{R}^{l \times m}$ is a compact constraint set. For example, often we assume that the columns of A have unit norm. This guarantees *scale invariance*. Indeed, assuming A, Z is a solution, if we do not use any constraint, the set $A' = (cA)$, $Z' = (\frac{1}{c}Z)$ would also be a solution, for any value of c . Other constraints can also be used to account for extra a priori knowledge concerning the available data (see, e.g., [85]).

In practice, the solution involves two stages, as with k -SVD. Assuming that $A^{(i)}$ is known at the iteration step i , minimization with respect to the code vectors first takes place, i.e.,

Stage 1:

$$Z^{(i)} = \min_Z \{ \|X - A^{(i)}Z\|_F^2 + \lambda g(Z) \}. \quad (19.73)$$

Then, fixing $Z^{(i)}$, the code book update results as

Stage 2:

$$A^{(i+1)} = \min_{A \in \mathcal{A}} \|X - AZ^{(i)}\|_F^2. \quad (19.74)$$

In the last equation, the sparsity promoting term is not involved, since it is independent of A . Assuming $g(Z)$ is a convex function, at each stage a convex optimization task is solved. Such problems are known as biconvex, in the sense that by fixing one of the arguments in the cost function the optimization problem becomes a convex one with respect to the other. The solution of the optimization task can be obtained via different paths; for example, via majorize-minimize (MM) techniques or the ADMM scheme (Section 8.14); see, e.g., [80, 140, 196] for some application cases under various constraints.

Solving the dictionary learning task is a nonconvex optimization problem. An important related issue is that of the characterization of the associated local minima. This task is another face of what we call *identifiability*. In other words, assuming that there exists a dictionary that generates the data, can this be recovered? To know the answer to this question is of major importance in a number of applications. For example, in a source localization task, the atoms of the dictionary relate to the directions of the arrival of the involved signals. Also, in fMRI (see Section 19.11) the atoms of the dictionary provide information related to the time responses of the neurons associated with the activated regions in the brain. In [86], it is shown that, under certain assumptions related to sparsity, the minimized cost function in the dictionary learning task has a *guaranteed* local minimum around the dictionary that generates the data, with high probability. Moreover, the bound on the number of samples that are sufficient for the existence of such a minimum is also derived.

Remarks 19.5.

- Alternative paths to k -SVD to dictionary learning have also been suggested. For example, in [73] a dictionary learning technique referred to as method of optimal directions (MOD) was proposed, which differs from k -SVD in the dictionary update step. In particular, the full dictionary is updated via direct minimization of the Frobenius norm. In [126, 143] probabilistic arguments are employed, using a Laplacian prior to enforce sparsity. We know from Chapter 13 (Section 13.5) that in this case, the involved integrations are not analytically tractable and the different methods differ in the different approximations used to bypass this obstacle. In the former, the maximum value of the integrand is used and in the latter a Gaussian approximation of the posterior is adopted in order to handle the integration. In [82], variational bound techniques are mobilized (see Section 13.9).
- The method proposed in [125] bears some similarities to k -SVD, because it also revolves around SVD, but the dictionary is constrained to be a union of orthonormal bases. This can lead to some computational advantages; on the other hand, k -SVD puts no constraints on the atoms of the dictionary, which gives more freedom in modeling the input. Another difference lies in the column-by-column update introduced in k -SVD.
- A more detailed comparative study of k -SVD with other methods is given in [4].
- Dictionary learning is essentially a matrix factorization problem where a certain type of constraint is imposed on the right matrix factor. This approach can be considered to be just a manifestation of a wider class of constrained matrix factorization methods that allow several types of constraints to hold. Such techniques include the regularized PCA, where functional and/or sparsity constraints are imposed to the left and to the right factors [12, 190, 200], as well as the structured sparse matrix factorization in [14].

- Besides the previously reported algorithms, a number of *online* and *distributed* dictionary learning schemes have been proposed. In [127], a distributed version of the k -SVD algorithm is developed. A cloud-based version of k -SVD is proposed in [149] with an emphasis on big data applications. A dictionary learning algorithm employing arguments inspired by the EXTRA optimization scheme, which is discussed in Section 8.15, is described in [182]. In [49], the various agents/nodes learn different parts of the dictionary. Such an approach is also suited for big data applications. An online algorithm for dictionary learning has been presented in [139] and an online version for a distributed setting, with provable convergence to a stationary point, has been proposed in [53]. In [62], the decentralized case over time-varying digraphs is considered that involves a general set of constraints, where a number of matrix factorization schemes result as special cases.

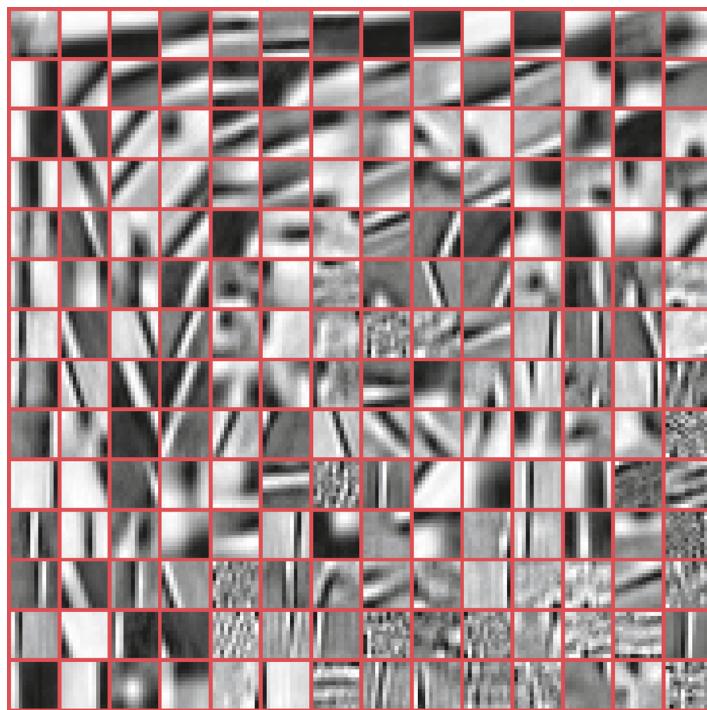
Example 19.5. The goal of this example is to show the performance of the dictionary learning technique in the context of the image denoising task. In the case study of Section 9.10, image denoising, based on a predetermined and fixed DCT dictionary, was considered. Here, k -SVD will be employed in order to learn the dictionary using information of the image *itself*. The two (256×256) images, without and with noise corresponding to $\text{PSNR} = 22$ are shown in Figs. 19.13A and B, respectively. The noisy image is divided in overlapping patches of size 12×12 (144), resulting in $(256 - 12 + 1)^2 = 60,025$ patches in total; these will constitute the training data set used for the learning of the dictionary. Specifically, the patches are sequentially extracted from the noisy image, vectorized in lexicographic order, and used as columns, one after the other, to define the $(144 \times 60,025)$ matrix X . Then, k -SVD is mobilized to train an overcomplete dictionary of size 144×196 . The resulting atoms, reshaped in order to form 12×12 pixel patches, are shown in Fig. 19.12. Compare the atoms of this dictionary with atoms of the fixed DCT dictionary of Fig. 9.14.

Next, we follow the same procedure as in Section 9.10, by replacing the DCT dictionary with the one obtained by the k -SVD method. The resulting denoised image is shown in Fig. 19.13C. Note that although the dictionary was trained based on the *noisy data*, it led to about 2 dB PSNR improvement over the fixed-dictionary case. As a matter of fact, because the number of patches is large and each one of them carries a different noise realization, the noise, during the dictionary learning stage, is averaged out leading to nearly noise-free dictionary atoms. More advanced use of dictionary learning techniques to further improve performance in tasks such as denoising and inpainting can be found in [71,72,138].

19.7 NONNEGATIVE MATRIX FACTORIZATION

The strong connection between dimensionality reduction and low rank matrix factorization has already been stressed while discussing PCA. ICA can also be considered as a low rank matrix factorization, if a smaller number, compared to the l observed random variables, of independent components is retained (e.g., selecting the $m < l$ least Gaussian ones).

An alternative to the previously discussed low rank matrix factorization schemes was suggested in [144,145], which guarantees the *nonnegativity* of the elements of the resulting matrix factors. Such a constraint is enforced in certain applications because negative elements contradict physical reality. For example, in image analysis, the intensity values of the pixels cannot be negative. Also, probability values cannot be negative. The resulting factorization is known as *nonnegative matrix factorization*

**FIGURE 19.12**

Dictionary resulting from k -SVD.

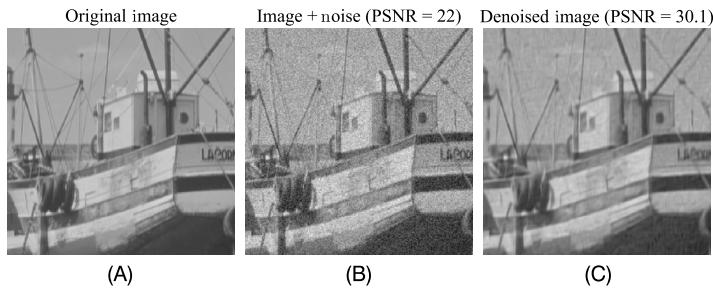
**FIGURE 19.13**

Image denoising based on dictionary learning.

(NMF) and it has been used successfully in a number of applications, including document clustering [192], molecular pattern discovery [28], image analysis [122], clustering [171], music transcription, music instrument classification [20,166], and face verification [198].

Given an $l \times N$ matrix X , the task of NMF consists of finding an approximate factorization of X , that is,

$$X \approx AZ, \quad (19.75)$$

where A and Z are $l \times m$ and $m \times N$ matrices, respectively, $m \leq \min(N, l)$, and all the matrix elements are nonnegative, that is, $A(i, k) \geq 0$, $Z(k, j) \geq 0$, $i = 1, 2, \dots, l$, $k = 1, 2, \dots, m$, $j = 1, 2, \dots, N$. Clearly, if matrices A and Z are of low rank, their product is also a low rank, at most m , approximation of X . The significance of the above is that every column vector in X is represented by the expansion

$$\mathbf{x}_i \approx \sum_{k=1}^m Z(k, i) \mathbf{a}_k, \quad i = 1, 2, \dots, N,$$

where \mathbf{a}_k , $k = 1, 2, \dots, m$, are the column vectors of A and constitute the basis of the expansion. The number of vectors in the basis is less than the dimensionality of the vector itself. Hence, NMF can also be seen as a method for *dimensionality reduction*.

To get a good approximation in Eq. (19.75) one can adopt different costs. The most common cost is the Frobenius norm of the error matrix. In such a setting, the NMF task is cast as follows:

$$\min_{A, Z} \|X - AZ\|_F^2 := \sum_{i=1}^l \sum_{j=1}^N (X(i, j) - [AZ](i, j))^2, \quad (19.76)$$

$$\text{s.t. } A(i, k) \geq 0, \quad Z(k, j) \geq 0, \quad (19.77)$$

where $[AZ](i, j)$ is the (i, j) element of matrix AZ , and i, j, k run over all possible values. Besides the Frobenius norm, other costs have also been suggested (see, e.g., [168]).

Once the problem has been formulated, the major issue rests at the solution of the optimization task. To this end, a number of algorithms have been proposed, for example, Newton-type or gradient descent-type algorithms. Such algorithmic issues, as well as a number of related theoretic ones, are beyond the scope of this book, and the interested reader may consult, for example, [54, 67, 178]. More recently, regularized versions, including sparsity promoting regularizers, have been proposed (see, for example, [55] for a more recent review on the topic).

19.8 LEARNING LOW-DIMENSIONAL MODELS: A PROBABILISTIC PERSPECTIVE

In this section, the emphasis is on looking at the dimensionality reduction task from a Bayesian perspective. Our focus will be more on presenting the main ideas and less on algorithmic procedures; the latter depend on the specific model and can be dug out from the palette of algorithms that have already been presented in Chapters 12 and 13. Our path to low-dimensional modeling traces its origin to the so-called *factor analysis*.

19.8.1 FACTOR ANALYSIS

Factor analysis was originally proposed in the work of Charles Spearman [169]. Charles Spearman (1863–1945) was an English psychologist who has made important contributions to statistics. Spearman was interested in human intelligence and developed the method in 1904, for analyzing multiple measures of cognitive performance. He argued that there exists a general intelligence factor (the so-called g-factor) that can be extracted by applying the factor analysis method on intelligence test data. However, this notion has been strongly disputed, as intelligence comprises a multiplicity of components (see, e.g., [84]).

Let $\mathbf{x} \in \mathbb{R}^l$. The factor analysis model assumes that there are $m < l$ underlying (latent) zero mean variables or *factors* $\mathbf{z} \in \mathbb{R}^m$ so that

$$\mathbf{x}_i - \boldsymbol{\mu}_i = \sum_{j=1}^m a_{ij} \mathbf{z}_j + \boldsymbol{\epsilon}_i, \quad i = 1, 2, \dots, l, \quad (19.78)$$

or

$$\mathbf{x} - \boldsymbol{\mu} = A\mathbf{z} + \boldsymbol{\epsilon}, \quad (19.79)$$

where $\boldsymbol{\mu}$ is the mean of \mathbf{x} and $A \in \mathbb{R}^{l \times m}$ is formed by the weights a_{ij} known as *factor loadings*. The variables \mathbf{z}_j , $j = 1, 2, \dots, m$, are sometimes called *common factors*, because they contribute to all observed variables, \mathbf{x}_i , and $\boldsymbol{\epsilon}_i$ are the *unique* or *specific factors*. As we have already done so far and without loss of generality, we will assume our data are centered, that is, $\boldsymbol{\mu} = \mathbf{0}$. In factor analysis, we assume $\boldsymbol{\epsilon}_i$ to be of zero mean and mutually uncorrelated, that is, $\Sigma_\epsilon = \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T] := \text{diag}\{\sigma_1^2, \sigma_2^2, \dots, \sigma_l^2\}$. We also assume that \mathbf{z} and $\boldsymbol{\epsilon}$ are independent. The m ($< l$) columns of A form a lower-dimensional subspace, and $\boldsymbol{\epsilon}$ is that part of \mathbf{x} not contained in this subspace. The first question that is now raised is whether the model in Eq. (19.79) is any different from our familiar regression task. The answer is in the affirmative. Note that here the matrix A is not known. All that we are given is the set of observations, \mathbf{x}_n , $n = 1, 2, \dots, N$, and we have to obtain the subspace described by A . It is basically the same linear model that we have considered so far in this chapter, with the difference that now we have introduced the noise term. Once A is known, \mathbf{z}_n can be obtained for each \mathbf{x}_n .

From Eq. (19.79), it is readily seen that

$$\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^T] = A\mathbb{E}[\mathbf{z}\mathbf{z}^T]A^T + \Sigma_\epsilon.$$

We will further assume that $\mathbb{E}[\mathbf{z}\mathbf{z}^T] = I$; hence, we can write

$$\Sigma_x = AA^T + \Sigma_\epsilon. \quad (19.80)$$

Hence, A results as a factor of $(\Sigma_x - \Sigma_\epsilon)$. However, such a factorization, if it exists, is not unique. This can be easily checked if we consider $\bar{A} = AU$, where U is an orthonormal matrix. Then $\bar{A}\bar{A}^T = AA^T$. This has brought a lot of controversy around the factor analysis method when it comes to interpreting individual factors (see, for example, [44] for a discussion). To remedy this drawback, a number of authors have suggested methods and criteria that deal with the rotation (orthogonal or oblique) in order to gain improved interpretation of the factors [157]. However, from our perspective, where the goal is to express our problem in a lower-dimensional space, this is not a problem. Any orthonormal matrix

imposes a rotation within the subspace spanned by the columns of A , but we do not care about the exact choice of the coordinates, that is, the common factors.

There are different methods to obtain A (see, e.g., [64]). A popular one is to assume $p(\mathbf{x})$ to be Gaussian and employ the maximum likelihood method to optimize with respect to the unknown parameters that define Σ_x in Eq. (19.80). Once A becomes available, one way to estimate the factors is to further assume that these can be expressed as linear combinations of the observations, that is,

$$\mathbf{z} = W\mathbf{x}.$$

Postmultiplying by \mathbf{x} , taking expectations, and recalling Eq. (19.79) and that $\mathbb{E}[\mathbf{z}\mathbf{z}^T] = I$, we get

$$\mathbb{E}[\mathbf{z}\mathbf{x}^T] = \mathbb{E}[\mathbf{z}\mathbf{z}^T A^T] + \mathbb{E}[\mathbf{z}\epsilon^T] = A^T. \quad (19.81)$$

Also,

$$\mathbb{E}[\mathbf{z}\mathbf{x}^T] = W\mathbb{E}[\mathbf{x}\mathbf{x}^T] = W\Sigma_x. \quad (19.82)$$

Hence,

$$W = A^T \Sigma_x^{-1}.$$

Thus, given a value \mathbf{x} , the values of the corresponding latent variables are obtained by

$$\mathbf{z} = A^T \Sigma_x^{-1} \mathbf{x}. \quad (19.83)$$

19.8.2 PROBABILISTIC PCA

New light on this old problem was shed via the Bayesian rationale in the late 1990s [154, 175, 176]; the task was treated for the special case $\Sigma_\epsilon = \sigma^2 I$ and it was named *probabilistic PCA* (PPCA). The latent variables, \mathbf{z} , are dressed with a Gaussian prior,

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} | \mathbf{0}, I),$$

which is in agreement with the earlier assumption $\mathbb{E}[\mathbf{z}\mathbf{z}^T] = I$, and the conditional PDF is chosen as

$$p(\mathbf{x} | \mathbf{z}) = \mathcal{N}(\mathbf{x} | Az, \sigma^2 I),$$

where, for simplicity, we assume $\mu = \mathbf{0}$ (otherwise the mean would be $Az + \mu$). We are by now pretty familiar with writing down

$$p(\mathbf{z} | \mathbf{x}) = \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_{z|x}, \Sigma_{z|x}), \quad (19.84)$$

and

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \mathbf{0}, \Sigma_x), \quad (19.85)$$

where (see Eqs. (12.17), (12.10), and (12.15), Chapter 12)

$$\Sigma_{z|x} = \left(I + \frac{1}{\sigma^2} A^T A \right)^{-1}, \quad (19.86)$$

Index

A

α -divergence, 690
Absolute value rectification, 940
Activation function, 907
Active chain, 783
Ad hoc networks, 230
AdaBoost scheme, 338, 342, 344
AdaDelta, 929
AdaGrad algorithm, 394, 927
Adam algorithm, 929
Adaptive algorithmic schemes, 180
Adaptive LASSO, 502
Adaptive projected subgradient method (APSM), 375, 376, 378–384, 421, 500, 505, 507, 580
Adaptive rejection sampling, 742
AdCoSaMP scheme, 504
Additive models, 582
Adversarial examples, 4, 15, 985–988, 1016
Adversarial training, 985, 987
Affine projection algorithm (APA), 206, 357
AlexNet, 972
Alternating optimization, 626
Alternating-direction method of multipliers (ADMM), 414
Analog-to-information conversion, 460
ANOVA, 583
ANOVA kernels, 584
Approximation error, 401
Assumed density filtering (ADF), 690
Attention mechanism, 982
Autocorrelation sequence, 44–47, 50, 51, 136, 137, 144, 577
Autocovariance, 42–45
Autoencoders, 994
Automatic relevance determination (ARD), 667, 669, 686
Autoregressive processes, 51
Autoregressive-moving average (ARMA) model, 53
Auxiliary particle filtering (APF), 889, 898
Average risk, 305

B

Backpropagation
algorithm, 913, 916
through time, 978
Backtracking, 811
Bag-of-words, 584
Bagging, 333
Base
classifier, 335, 338–340, 343, 344, 349
transition PDFs, 767
Basis pursuit, 442
Basis pursuit denoising (BPDN), 432
Batch learning, 388
Batch normalization, 932
Bayes classifier, 305, 315, 317
Bayes theorem, 23, 24, 102, 103, 303, 318, 596, 605, 606, 651, 714
Bayesian
classification rule, 302, 303, 306, 312, 317, 329, 348
classifier, 303–305, 307–309, 311, 562
learning, 13, 15, 92, 596, 627, 630, 648, 681, 686, 698, 715, 762, 1004
regression, 712, 714
Bayesian information criterion (BIC), 610, 864
Bayesian network (BN), 772, 777
structure, 775
Beamforming, 163
Bernoulli distribution, 29
Best linear unbiased estimator (BLUE), 161, 258
Beta distribution, 37
Bethe approximation, 840
Bias term, 72, 84, 90, 95, 98, 110, 549–551, 553, 557, 588, 904, 907, 912, 913, 917
Biased estimators, 81
Binary classification task, 536, 538, 563, 575
Binary classifier, 337
Binomial deviance function, 342
Binomial distribution, 30
Blind source separation (BSS), 1066
Block sparsity, 493, 670
Boltzmann machines, 794
Boosting, 337, 342, 349

- Bootstrap, 333
 Box–Muller method, 737
 Bregman divergence, 416
- C**
 Canonical correlation analysis (CCA), 16, 1040, 1053, 1082
 Capon beamforming, 166
 Capsule networks, 1007
 Categorical distribution, 31, 699
 Central limit theorem, 36
 Centralized networks, 229
 Change point detection, 762
 Character n -grams, 585
 Chi-squared distribution, 39
 Chinese restaurant process (CRP), 15, 648, 691, 697
 Classification
 error probability, 303, 304, 315
 task, 10, 77, 78, 99, 301, 302, 321, 345, 347, 348, 563, 564, 573, 589, 673, 675, 716, 904, 911, 985
 trees, 302, 329
 Classification and regression tree (CART), 329
 Classifier, 10
 combination, 335
 Cliques in MRFs, 789
 Clustering, 11
 Cocktail-party problem, 1066
 Coding schemes, 839
 Collaborative filtering, 1101
 Colored Gaussian noise, 102
 Combination schemes, 335
 Compatibility functions, 789
 Complete set, 661
 Composite mirror descent, 416
 Compressed sensing, 456, 459, 460, 479, 494, 512, 516, 520, 524, 1082, 1084, 1097, 1100, 1101
 Compressed sensing matching pursuit (CSMP), 479 scheme, 479, 504
 Compressive signals, 457
 Compressive sampling (CS), 456
 Concave functions, 355
 Concentration parameter, 691
 Conditional
 entropy, 58
 log-likelihood, 862
 probability, 22, 23
- Conditional random field (CRF), 794
 Conditional random Markov field, 794
 Conjugate function, 677
 Conjugate priors, 107
 Consensus matrix, 243
 Consensus-based algorithms, 240
 Consensus-based LMS, 240
 Consistent estimator, 100
 Constrained-based structure learning, 864
 Convergence
 linear, 274
 speeds, 246
 stochastic, 61
 Convergence in probability, 63
 Convergence in the mean-square sense, 62
 Convex
 cost functions, 681
 duality, 676
 functions, 354
 loss functions, 384, 509
 minimization problem, 1100
 optimization, 195, 207, 352, 451, 474, 483, 485, 487, 497, 503, 507, 513, 866
 programming minimization, 557
 1×1 convolution, 966
 Convolution matrix, 148
 Convolutional neural networks (CNN), 15, 956
 Coordinate and cyclic coordinate descent method, 281
 Coordinate descent algorithm, 283, 484
 Core vector machine (CVM), 576
 Correlated component analysis, 1055
 Correlation, 26
 Correlation matrix, 26
 Cosparsity, 513–516
 Covariance, 26, 31, 41, 170, 620, 880, 1025, 1053
 Covariance matrix, 26
 Cover’s theorem, 536
 Cramer–Rao bound, 83
 Cross-entropy, 936
 Cross-entropy loss error, 319
 Cross-validation, 111
 Cumulant generating function, 841
 Cumulative distribution function (CDF), 24, 736
 Cumulative loss, 204, 396
 Curse of dimensionality, 108
 Cyclic coordinate descent (CCD), 283
 Cyclic path, 230

D

d-connected nodes, 783
d-separation, 783
Dantzig selector, 497
Decentralized learning, 227
networks, 229
optimization, 417
schemes, 229
Decision feedback equalizer (DFE), 221
Decision surfaces, 307
Decision tree (DT), 348
Deconvolution, 143
Deep belief networks, 992, 993
Deep generative models, 988
Deep learning, 15, 78, 992, 1015, 1017
Deep networks over graphs, 1015
Deep neural networks, 3, 181, 913, 985, 1013, 1016, 1017
Degeneracy in particle filtering, 885
Denoised image, 469, 1074, 1108
Denoising autoencoder, 995
DenseNet, 975
Detailed balanced condition, 747
Dictionary atoms, 469, 1074, 1108
identifiability, 1072
learning, 16, 439, 1069, 1072–1074, 1106
techniques, 1041, 1074
matrix, 456
Differentiable cost function, 195
Differential entropy, 60
Diffusion LMS, 231, 235
Dimensionality reduction, 12, 16, 254, 260, 261, 267, 327, 328, 458, 1041, 1045, 1048, 1049, 1053, 1057, 1074, 1076, 1082, 1085
Directed acyclic graph (DAG), 775
Dirichlet distribution, 39
Dirichlet process (DP), 648, 692
mixture modeling, 698
Discrete gradient transform, 497–499
random measure, 872, 873, 881, 885, 897
Discrete cosine transform (DCT), 438
dictionary, 1074
Discrete wavelet transform (DWT), 438
Discriminant functions, 309
Discriminative learning, 78, 79, 562, 855, 862

Distributed learning, 227

Distributed learning in RKHSs, 579
Distributed optimization, 417
Dropout method, 943

E

Early stopping, 942
Earth moving distance (EMD), 1001
Echo path, 436, 492
Echo state network (ESN), 984
Efficient estimators, 84
Eigenimages, 1050
Elastic net, 497
Elimination sequence, 824, 830
Empirical Bayes, 611
Energy costs, 1017
Entropy, 58
Entropy component analysis (ECA), 1087
Epigraph of a function, 356
Equalization, 143
Ergodic Markov chains, 748
Erlang distribution, 39
Error correcting codes, 575
covariance matrix, 168, 174
learning curves, 247
probability cost function, 305
propagation, 271
resilience, 575
Errors-in-variables models, 287
Estimation error, 401
Excess error, 227, 556
Excess mean-square error, 202, 226
Expectation propagation, 686
Expectation-maximization (EM) algorithm, 609, 611–615, 617–620, 623, 648, 664, 665, 668, 670, 682, 683, 685, 722, 726, 732, 745, 821, 852, 855, 860, 1079–1081
online, 627
variational, 623, 665, 720, 837, 1080
Expected loss, 112
Expected loss-expect risk, 195
Explaining away, 782
Exponential distribution, 37
Exponential loss function, 338, 341
Exponentially weighted isometry property, 505
Extended Kalman filtering (EKF), 170, 881

F

Factor analysis, 1040, 1076
 Factor graphs, 795
 Factorization theorem, 87
 Factors, 789
 Fast Fourier transform (FFT), 216, 486
 Fast iterative shrinkage-thresholding algorithm (FISTA), 483
 Fast Newton transversal filter (FNTF), 281
 Feasibility set, 374
 Feature learning, 988
 Feature map, 540
 Feature space, 540
 Feature vector, 8, 9, 302–304, 315, 672, 702, 724, 983, 994, 1045, 1046, 1049
 Federated learning, 1017
 Fill-ins, 824
 Filtering, 48, 134
 Finite impulse response (FIR), 135
 Finite mixture modeling, 697
 Fisher linear discriminant (FLD), 1048
 Fisher’s discriminant, 322
 Fisher’s discriminant ratio, 325
 Fixed point set, 364
 Forward-backward algorithm, 854
 Forward-backward splitting algorithms, 412
 Fréchet inception distance, 1003
 Frequentist techniques, 596
 Fully connected networks, 912
 Functional brain networks (FBN), 1103
 Functional magnetic resonance imaging (fMRI), 1053, 1103–1105

G

Gabor dictionary, 517, 520–522
 Gamma distribution, 38
 Gaussian
 approximation, 886, 1073
 distribution, 32, 33, 36, 104, 107, 108, 118, 201, 246, 285, 296, 302, 308, 309, 312–314, 317, 342, 602, 605, 621, 644, 655, 666, 676, 685, 690, 701, 709, 714, 724, 726, 733, 786, 844, 850, 857, 895, 1061, 1062, 1108
 function, 597, 609, 683, 725
 independent components, 1059
 kernel, 675, 712, 716

mixture modeling, 15, 616, 617, 619–621, 635, 636, 644, 661, 662, 698, 701, 722, 726, 854

process, 41, 47, 648, 710–716
 white noise, 89

Generalization

error, 96, 110
 error performance, 14, 110, 204, 333, 334, 342, 400, 401, 549, 568, 574, 610

Generalized forward-backward algorithm, 809

Generalized maximum likelihood, 611

Generalized thresholding (GT), 508

Generative adversarial networks (GAN), 12, 995, 996

Generative learning, 79, 562, 855

Generators, pseudorandom, 734

Gibbs distribution, 789

Gibbs sampling, 758

Gibbs scheme, 759

GoogleNet, 972

Gradient, 72

algorithm, 412, 414
 ascent scheme, 990, 997, 1002, 1063
 averaging, 404
 descent, 180, 181, 183, 184, 189, 190, 194, 196, 233–235, 238, 246, 253, 270, 272, 273, 283, 295, 342, 352, 388, 391, 402, 404, 417, 481, 485, 913–917, 998, 1002, 1027, 1052, 1063

backpropagation, 1024

backpropagation scheme, 916, 1024

optimization algorithm, 914

optimization scheme, 232

optimizer, 1027

 with momentum, 925

discrete, 497, 499

proximal, 412–414

stochastic, 240, 275

Graph, 775

Graph embedding, 1094

Graphical models, 494, 655, 687, 731, 760, 771, 772,

774, 786, 794, 797, 799, 821, 830, 842,

859, 878, 880

learning, 852, 859

probabilistic, 690, 772, 821

Greedy algorithms, 474

Group LASSO, 492

H

Hamiltonian Monte Carlo methods, 761

Hammersley–Clifford theorem, 791

- Hammerstein models, 534
 Hebbian learning rule, 990
 Hidden Markov model (HMM), 15, 821, 843, 854, 878
 factorial, 857
 Hidden random parameters, 612, 648
 Hidden variables, 611–613, 636, 649, 650, 654, 656, 698, 745, 772, 786, 838, 842, 844, 847, 849, 856, 857, 981, 982, 988, 989, 993
 Hierarchical mixture of experts, 638
 Hierarchical priors, 610
 Hinge loss function, 373, 395, 420, 563, 576, 1011
 Homogeneous Markov chains, 746
 Homotopy methods, 478
 Huber loss function, 383, 554, 560, 588
 Hyperparameters, 606, 609, 610, 628, 629
 Hyperplane classifier, 570
 Hyperslabs, 208, 371, 376–379, 505, 507
 parameter, 510
 Hyperspectral image unmixing, 717
 Hyperspectral remote sensing, 717
 Hypersurface, 307–310, 634
- I**
- Identifiability of parameters, 619
 Ill-posed problems, 91
 Importance sampling (IS), 743
 Impulse response, 48–50, 64, 135, 139, 141, 142, 154, 162, 218, 223, 247, 285, 436
 Inception cost, 1003
 Inception network, 972
 Inception score (IS), 1003
 Incremental learning, 230, 1017
 Independent component analysis (ICA), 16, 1040
 Independent component (IC), 1058, 1061
 Independent Gaussian component, 1059
 Independent Gaussian variables, 610, 713
 Independent random variables, 108
 Independent vector analysis (IVA), 1065
 Indian buffet process (IBP), 15, 648, 701, 706
 Indicator function, 406
 Infinite impulse response (IIR), 136
 Infomax principle, 1065
 Information, 56
 Information projection, 686
 Intercausal reasoning, 782
 Interference cancelation, 140
 Intersymbol interference, 143
 Intrinsic dimensionality, 109, 1041
- Invariance, 46, 748, 1008
 property, 692
 Invariant distribution, 747
 Inverse problems, 91
 Ising model, 791
 Isometric mapping (ISOMAP), 1041, 1092–1094
 Isometry constant, 452
 Iterative hard thresholding (IHT), 490
 Iterative reweighted least-squares, 320
 Iterative shrinkage algorithms, 480
 Iterative soft thresholding (IST), 490
- J**
- Jacobian matrix, 28
 Jensen–Shannon divergence, 999
 Join tree, 824
 Joint
 optimization, 577
 PDF, 26–28, 42, 43
 probability, 22, 23
- K**
- Kalman filtering, 166, 167, 169–171, 174, 176, 180, 871, 878, 879, 881, 884, 895
 Bayesian point of view, 878
 Kalman gain, 269
 Karhunen–Loëve transform, 1041
 Kernel PCA, 1085
 Kernel ridge regression, 532, 551, 561
 Kernel trick, 540
 Kikuchi free energy, 841
 Kullback–Leibler divergence, 61, 625, 642, 649–651, 653, 686, 688, 689, 836, 837, 840, 999, 1001, 1062
 Kurtosis of a random variable, 1061
- L**
- Laplacian eigenmaps, 1087
 Laplacian matrix, 1088
 LARS, 478
 LARS-LASSO, 478
 LASSO, 431
 minimizer, 484
 Latent
 discrete variables, 617
 random variables, 835, 1047
 random vector, 661

- semantics indexing, 1046
- variables, 611, 612, 648, 652, 661, 685, 843, 844, 852, 859, 860, 1005–1007, 1047, 1058
- sparse vector, 1104
- Leaky ReLU, 940
- Learnability, 113
- Learning
 - algorithms, 180, 212
 - Bayesian, 13, 15, 92, 596, 627, 630, 648, 681, 686, 698, 715, 762, 1004
 - Bayesian networks, 994
 - curve, 189, 247
 - decentralized, 227
 - dictionary, 16, 439, 1069, 1072–1074, 1106
 - graphical models, 859
 - in RKHS, 579
 - machines, 903
 - methods, 3, 12, 13
 - nonlinear models, 532
 - online, 13, 14, 374, 384, 393, 396, 399, 402, 580, 908
 - parameter, 860
 - phase, 10, 982
 - probabilistic models, 992
 - rate, 1027, 1028
 - rule, 990
 - task, 113, 380, 393, 402, 403, 581, 985, 988, 992, 1014, 1041
 - theory, 396, 400, 903
- Least modulus loss function, 554
- Least-mean-squares (LMS)
 - performance, 204
 - scheme, 206, 231, 404
- Least-squares (LS)
 - classifiers, 78
 - cost, 268
 - estimator, 86, 89, 115, 116, 257–260, 293, 500
 - linear classifier, 345
 - overdetermined case, 265
 - solution, 89, 91, 102, 256, 258, 265, 279, 320, 327, 463
 - task, 254, 268, 283, 475, 476, 479, 504
 - underdetermined case, 265
- LeNet-5, 972
- Levenberg–Marquardt method, 402
- Levinson’s algorithm, 153
- Linear
 - classifier, 77, 109, 302, 310, 322, 323, 373, 390, 538, 564, 566, 567, 569
 - congruential generator, 734
 - convergence, 274
 - dichotomy, 536
 - regression, 15, 71, 72, 89, 91, 92, 101, 104, 114, 116, 582, 596, 613, 640, 655, 660, 670, 722
 - EM algorithm, 644
 - model, 84, 101, 106, 118, 119, 243, 246, 254, 286, 615, 634, 640, 644, 726
 - modeling, 258
 - subspace, 75, 440
 - systems, 48
 - time-invariant systems, 48
 - unbiased estimator, 161, 258
- Linear ϵ -insensitive loss function, 555
- Linear discriminant analysis (LDA), 312, 319
- Linear dynamic system (LDS), 844
- Linear programming (LP), 443
- Linearly constrained minimum variance beamforming, 166
- Linearly separable, 537, 538, 910
 - classes, 77, 373, 374, 390, 420, 537–539, 564, 572, 903, 1023
- Liquid state machines, 984
- Local linear embedding (LLE), 16, 1041, 1091
- Local minimizer, 357, 418
- Log-loss function, 342
- Log-odds ratio, 341
- Log-partition function, 841
- Logistic regression, 317, 319, 321, 348, 399, 583, 637, 639, 672, 673, 716, 725
- Logistic regression loss function, 302
- Logistic regression model, 317, 681
- Long short-term memory network (LSTM), 980
- Loss function, 70–72, 74, 75, 77, 78, 80–82, 89, 112, 122, 157, 180, 204, 253, 301, 302, 336, 338, 339, 342, 355, 371, 372, 374, 388, 390, 393, 394, 396–402, 405, 407, 420, 482, 505, 548, 551, 554–556, 563, 572, 581, 583, 913, 914, 1001, 1011, 1012, 1019, 1025
 - for regression, 344
- MSE, 244
- optimization, 548
- Loss matrix, 306

M

Machine learning, 2–4, 24, 45, 54, 70, 91, 109, 195, 204, 208, 260, 315, 334, 369, 373, 380, 384, 427, 428, 463, 492, 500, 574, 575, 596, 620, 653, 681, 758, 771, 791, 856, 903, 982, 985, 1064
 optimization tasks, 61
 tasks, 14, 68, 254, 436, 732, 784

Mahalanobis distance, 310

Manifold learning, 459, 574, 1084, 1094
 for dimensionality reduction, 1094
 methods, 16

Margin classification, 562

Margin error, 563, 564, 570

Marginal
 PDFs, 108, 1062
 probability, 23, 56, 57, 777, 803, 809, 828, 830, 865

Markov blanket, 784

Markov chain, 745

Markov chain Monte Carlo (MCMC), 842

Markov condition, 775

Markov networks, 788

Markov random field (MRF), 772, 788

Matching pursuit, 474

Matrix
 completion, 1097
 dictionary, 456
 error, 289
 noise, 296

Max output unit, 940

Max-product algorithm, 810

Max-sum algorithm, 811

Maximization
 step, 853
 view, 623

Maximum a posteriori (MAP), 303
 probability estimation, 107

Maximum entropy (ME) method, 61, 633

Maximum likelihood (ML), 68, 99, 303, 597
 Type II, 611

Maximum margin classifiers, 564

Maximum margin parameter estimation, 862

Maximum variance, 1045, 1057, 1068

Maximum variance unfolding, 1094

Mean field approximation, 649

Mean field equations, 837

Mean value, 25

Mean-square error (MSE)

cost, 158
 cost function, 244
 estimation, 94
 performance, 92
 prediction errors, 174

Measurement noise, 167, 1085

Memoryless nonlinear system, 534, 535

Memoryless nonlinearity, 534

Message passing schemes, 15, 484, 485, 772, 799, 802, 804, 807–811, 821, 822, 825, 827–829, 837–840, 842, 847, 849, 850

Metric distance function, 387

Metropolis–Hastings algorithm, 754

Minibatch schemes, 923

Minimization, 319, 321, 340, 346, 394, 395, 407, 415, 476, 486, 499, 515, 523, 524, 549, 551, 552, 833, 837, 1063, 1072, 1089
 problem in sparse modeling, 1101
 proximal, 409, 410
 solvers, 1070
 task, 89, 208, 275, 282, 411, 443, 452, 468, 477, 487, 497, 498, 513, 515, 524, 550, 560, 570, 575

Minimum

MSE performance, 116
 norm error, 255
 variance, 83, 86, 166
 estimator, 554
 unbiased linear estimator, 161

Minimum description length (MDL), 611, 864

Minimum enclosing ball (MEB), 576

Minimum variance distortionless response (MVDR), 166

Minimum variance unbiased estimator (MVUE), 82, 259

Minimum variance unbiased (MVU), 82, 427

Minmax optimization task, 998

Minorize-maximization methods, 625

Mirror descent algorithms (MDA), 415

Misadjustment, 202

Misclassification error, 303, 339, 562

Mixture modeling, 596, 616, 648, 664, 698, 699, 702, 703, 705, 845, 847

Gaussian, 15, 619–621, 635, 636, 661, 662, 698

Mixture of experts, 634

Mixture of factor analyzers, 1084

Mixture probabilities, 702, 704

- Model complexity, 96, 110, 111, 664, 864
 Modulated wideband converter (MWC), 461
 Moment matching, 688
 Moment projection, 687
 Momentum term, 925
 Moore–Penrose pseudoinverse, 256
 Moral graphs, 798
 Moralization, 798
 Moreau envelope, 405, 485
 Moreau–Yosida regularization, 405
 Moving average model, 54
 Multiclass classification schemes, 575
 Multiclass logistic regression, 346, 638
 Multidimensional scaling, 1048
 Multinomial distribution, 31
 Multinomial probability distribution, 661
 Multinulli distribution, 699
 Multipath channels, 436
 Multiple additive regression tree (MART), 344
 Multiple kernel learning (MKL), 580
 - learning task, 581
 Multiple measurement vector (MMV), 670
 Multiple-cause networks, 786
 Multitask learning, 15, 572, 1014, 1102
 Multivariate Gaussian distribution, 34, 36, 63, 102, 119, 630, 631, 643, 726
 Multivariate linear regression (MLR), 1057
 Mutual coherence of a matrix, 449
 Mutual information, 56
- N**
- Naive Bayes classifier, 315, 347, 582
 Natural gradient, 402, 1064
 Natural language processing (NLP), 7, 1017
 Near-to-Toeplitz matrices, 280
 Nearest neighbor classifier, 315
 Negative gradient, 182
 Negative gradient direction, 183
 Negentropy, 1065
 NESTA, 485
 Nesterov’s momentum algorithm, 927
 Network in network, 966
 Neural machine translation (NMT), 7, 15, 1017
 Neural network (NN), 3, 908
 - feed-forward, 908
 Neural Turing machine (NTM), 984
 Newton’s iterative minimization method, 271
 No free lunch theorem, 334
- Noise
 - cancelation, 141, 143, 175
 - canceler, 144, 145
 - covariance matrix, 166
 - disturbance, 596
 - field, 138
 - Gaussian, 86, 92, 175, 176, 259, 464, 615, 635, 645, 665, 670, 716, 719, 727
 - matrix, 296
 - outliers, 554
 - precision, 645
 - process, 143
 - realization, 1074
 - variance, 95, 119, 175, 208, 218, 224, 293, 421, 497, 600, 603, 637, 640, 643, 644, 660, 685, 718, 1084
 - white, 396
 Noisy observations, 242, 895
 Noisy-OR model, 772
 Nonconvex cost functions, 916
 Nonconvex optimization method, 855
 Nonconvex optimization task, 610, 999
 Nondifferentiable loss functions, 180, 481
 Nonexpansive operators, 363
 Nonlinear
 - classifier, 537
 - estimation, 196
 - estimator, 245
 - extensions, 395
 - function, 122, 532, 539, 551
 - mappings, 508
 - modeling tasks, 532, 533
 - regression, 710
 - regression task, 552
 - system, 533, 535
 - thresholding function, 480
 - vector functions, 170
 Nonlinearly separable classes, 77, 373
 Nonlinearly separable task, 910
 Nonnegative garrote thresholding rule, 435
 Nonnegative matrix factorization (NMF), 1040, 1075
 Nonnegatively truncated Gaussian prior, 719
 Nonparallel hyperplanes, 440
 Nonparametric classifier, 315
 Nonparametric estimation, 114
 Nonparametric modeling, 551
 Nonseparable classes, 569
 Nonsmooth regularization, 14

- Nonparse representations, 512
 Nonwhite Gaussian noise, 101, 106
 Normal distribution, 32
 Normal equations, 126
 Normal factor graph (NFG), 796
 Normalized LMS (NLMS), 211, 212, 218, 220, 247,
 283–285, 296, 378–380, 421
 set, 296, 421
 Nuclear norm, 1099
 Nuclear norm minimization, 1099–1101
 Numeric optimization schemes, 344
 Numerical errors, 170, 271
 Numerical performance, 271
 Nyström approximation of the kernel matrix, 579
- 0**
- Observations
 drawn, 231
 equation, 878
 sequence, 195, 377, 762, 894
 set, 455
 space, 453
 subspace, 459, 460
 transmitted sequence, 851
 vector, 255, 521
 Occam factor, 607
 One pixel camera, 458
 Online
 algorithms, 180, 199, 203, 205, 254, 268, 284, 393,
 395–397, 399, 401, 403–405, 414, 420,
 474, 499, 500, 503, 509, 510
 convergence properties, 396
 in RKHSs, 579
 performance, 394
 performance for convex optimization, 352
 EM algorithm, 627
 formulation, 394
 learning, 13, 14, 374, 384, 393, 396, 399, 580, 908
 for convex optimization, 393
 in RKHS, 579
 optimization rationale, 914
 performance, 227
 performance schemes, 399
 processing, 369, 374, 500
 processing rationale, 181
 rationale, 504
 sparsity promoting algorithms, 499
 stochastic gradient, 224, 404
- Optical character recognition (OCR), 9, 574, 856
 Optimal
 classifier, 316, 562
 MSE backward errors, 157
 proposal distributions, 888
 sparse solutions, 449, 477
 Optimal brain damage, 941
 Optimal brain surgeon, 942
 Optimization
 algorithms, 415, 625, 676, 840, 914, 1021
 convex, 352, 474, 483, 485, 503
 criteria, 308, 681
 decentralized, 417
 error, 402
 joint, 577
 loss function, 548
 parameter, 999
 path, 575
 problems, 637, 686, 794
 procedure, 502, 580
 process, 339, 341, 602, 831, 859
 rationale, 997
 schemes, 254, 394, 412, 862
 sparsity promoting, 498
 task, 89, 98, 158, 208, 254, 265, 275, 287, 289, 336,
 338, 342, 358, 391, 410, 441, 457, 478,
 486, 493, 556, 557, 561, 564, 606, 633,
 634, 793, 985, 1006, 1042, 1043, 1054,
 1070
 techniques, 13, 371, 655, 833
 theory, 482, 483
 Optimized performance, 384
 Optimized values, 662
 Optimizing cost, 842
 Optimizing nonsmooth convex cost functions, 384
 Oracle properties, 501
 Ordinary binary classification tree (OBCT), 329
 Orthogonal matching pursuit (OMP), 475
 Orthogonal projection matrix, 208
 Orthogonality condition, 124
 Overcomplete dictionary, 439, 441, 442, 450, 464,
 511, 512, 515, 1041, 1069, 1074, 1082,
 1108
 Overfitting, 68, 91, 92, 95, 108, 265, 268, 321, 336,
 342, 549, 597, 599, 606, 913, 1015
 performance, 539
 Overlapping classes, 78, 569, 572

- P**
- Pairwise MRFs, 793
 - Parameter
 - error vector convergence, 188
 - learning, 860
 - optimization, 999
 - regularization, 402, 435, 501, 502, 552
 - sparse vector, 445
 - vector, 74, 83, 87, 89–91, 104, 107, 196, 204, 222, 225, 237, 240, 397, 403, 407, 431, 446, 474, 475, 478, 479, 499, 523
 - Partial least-squares, 1056
 - Particle filtering, 878, 881
 - Partition function, 789
 - Path
 - history, 883
 - iterations, 186
 - optimization, 575
 - PDFs
 - conditional, 105, 303, 315, 317, 598, 767, 771, 866
 - marginal, 108, 1062
 - PEGASOS algorithm, 395
 - Perceptron algorithm, 391
 - Perceptron cost, 904
 - Perceptron cost function, 904
 - Perceptron learning rule, 903
 - Perfect elimination sequence, 824
 - Perfect maps, 787
 - Perfect periodic sequence (PPSEQ), 218
 - Performance
 - analysis, 205, 212, 237
 - bounds, 476, 480, 484, 567
 - convergence, 212, 219, 379
 - error, 222
 - gains, 580
 - index, 110, 204
 - loss, 109, 337, 401, 402
 - measures, 487, 500, 501
 - MSE, 92
 - online, 227
 - online algorithms, 394
 - overfitting, 539
 - properties, 198
 - Phase transition curve, 487, 489
 - Pitman–Yor IBP, 710
 - Point spread function (PSF), 175
 - Poisson process, 762
 - Pólya urn model, 695
 - Polygonal path, 478, 479
 - Polytrees, 804
 - Pooling, 963
 - Positive definite function, 46
 - Posterior
 - approximation, 1005
 - class probabilities, 862
 - densities, 883
 - distribution, 119, 612, 627, 698, 700
 - DP, 694, 695, 725
 - DP formulation, 695
 - error covariance matrices, 174
 - estimate, 652
 - estimator, 168
 - Gaussian, 106, 613, 614
 - Gaussian distribution, 643
 - Gaussian process, 716
 - joint, 700
 - mean, 602, 716, 727, 881
 - mean values, 670
 - prediction, 716
 - prediction variance, 716
 - probability, 79, 302, 316, 321, 335, 336, 617, 621, 622, 637, 673, 691, 694, 724, 797, 862
 - probability distributions, 648
 - Potential functions, 789
 - Potts model, 793
 - Power spectral density (PSD), 47, 577
 - Prediction error, 157
 - Prediction performance, 75, 333
 - Primal estimated subgradient solver, 395
 - Principal component, 1047
 - Principal component analysis (PCA), 16, 1040, 1041
 - Principal component pursuit (PCP), 1100
 - Probabilistic
 - graphical models, 690, 771, 772, 821
 - PCA, 1078
 - view, 878
 - Probability
 - discrete, 696, 774, 827
 - distribution, 54, 68, 99, 625, 627, 628, 653, 693, 699, 746, 747, 752, 762, 783, 787, 789, 791, 795, 816–818, 824, 825, 827, 831, 835, 841, 845, 988, 995–998, 1005
 - error, 77, 78, 111, 302, 304, 308, 332, 345, 347, 855, 862
 - for discrete variables, 26
 - function, 998

- function space, 999
 joint, 22, 23, 777, 778, 793, 797, 800, 813, 816,
 828, 831, 836, 840, 847, 852, 993
 laws, 779
 mass, 640, 667, 693, 694, 697, 1001
 posterior, 79, 302, 316, 321, 335, 336, 617, 621,
 622, 637, 673, 691, 694, 724, 797, 862
 sequence, 708, 709, 725
 space, 56
 theory, 20–22, 36, 54, 107, 609, 735
 transition matrix, 750
 Probability density function (PDF), 20, 24, 303, 554,
 693, 775
 Probability mass function (PMF), 22, 628
 Probit regression, 322
 Process noise, 167, 887
 Product rule of probability, 23
 Projected gradient method (PGM), 391
 Projected Landweber method, 391
 Projected subgradient method, 392
 Projection approximation subspace tracking (PAST),
 1052
 Projection matrix, 209
 Projections onto convex sets (POCS), 14, 357, 365
 schemes, 371
 Property sets, 374
 Proportionate NLMS, 212
 Proposal distribution, 741, 742, 754, 755, 872, 875,
 882, 884, 885
 Proximal
 gradient, 412–414
 algorithms, 412
 mapping, 485
 minimization, 409, 410
 operator, 405–414, 420, 485
 point algorithm, 410
 Proximal forward-backward splitting operator, 414
 Pruning scheme, 348
 Pseudocovariance matrix, 131
 Pseudorandom generator, 735
 Pseudorandom number generator, 733
- Q**
- Quadratic ϵ -insensitive loss function, 555
 Quadratic discriminant analysis (QDA), 312
 Quasistationary rationale, 859
 Quaternary classification, 576
- R**
- Random
 components, 1047
 distribution, 698
 draw, 695
 elements, 664
 events, 56
 experiment, 41
 Gaussian, 348, 716
 matrix, 451, 453, 454, 457, 463
 measure, 872, 893
 nature, 29, 595, 661
 number generation, 733
 parameters, 595, 606, 613
 process, 20, 41, 44, 62, 134–137, 141, 149, 157,
 163, 215, 226, 284, 693, 710–712, 716,
 1065
 projection, 459
 projection matrices, 459
 sampling, 735
 sequence, 62, 63, 247, 733, 745
 variables, 20, 22, 23, 68, 71, 74, 80, 94, 124, 125,
 131, 134, 162, 185, 199, 200, 225, 244,
 257, 258, 275, 449, 595–597, 599, 606,
 610, 648, 656, 664, 667, 690, 693, 699,
 700, 732, 734–736, 738, 740, 752, 772,
 774–776, 778, 781, 784, 842, 843, 847,
 860, 865, 871, 1041, 1047, 1053
 convergence, 62
 vector, 26–28, 39, 49, 95, 102, 103, 128, 144, 149,
 154, 160, 194, 236, 500, 501, 596, 612,
 613, 628, 642, 699, 710, 723, 732, 739,
 741, 759, 761, 871, 875, 1041, 1046, 1053
 variable, 743
 walk, 271, 751–753, 756, 767
 walk model, 884
- Random demodulator (RD), 458, 460
 Random fields, 138
 Random forests, 333
 Random Fourier feature (RFF), 532, 577
 rationale, 580
 Random Markov fields, 788
 Random sign ensemble (RSE), 491
 Random signals, 41
 Randomized
 algorithms, 459
 pruning, 873

- Randomness, 20, 59, 60, 62, 257, 456, 598, 627, 633, 734
 Randomness tests, 734
 Rao–Blackwell theorem, 88
 Rao–Blackwellization, 893
 Rayleigh’s ratio, 326
 Rectified linear unit, 939
 Recurrent neural network (RNN), 15, 976
 Recursive least-square (RLS)
 convergence, 379
 scheme, 254, 275
 Recursive least-squares algorithm (RLS), 268
 Recursive scheme, 151, 197, 281, 292
 Reduced convex hull (RCH), 572
 Regression, 11, 13, 68, 71, 77, 93, 94, 99, 117, 301, 317, 329, 344, 369, 374, 554, 575, 596, 598, 599, 606, 648, 672, 673, 712
 Bayesian, 712, 714
 linear, 15, 71, 72, 89, 91, 92, 101, 104, 114, 116, 582, 596, 613, 640, 655, 660, 670, 722
 model, 72, 86, 89, 96, 98, 115, 197, 200, 205, 231, 232, 239, 246, 283, 296, 369, 383, 420, 492, 505, 600, 611, 613, 666, 672, 682, 683
 nonlinear, 710
 task, 9, 11, 71, 77, 95, 111, 113, 301, 316, 333, 342, 382, 431, 434, 532, 536, 555, 589, 605, 611, 613, 640, 665, 714, 716
 task random parameters, 606
 trees, 329, 333, 344
 Regret analysis, 396
 Regularization, 13–15, 68, 89, 91–93, 260, 268, 380, 395, 401, 410, 427, 431, 474, 493, 539, 549–551, 560, 572, 940
 method, 430, 431
 parameter, 92, 402, 435, 501, 502, 552
 penalty, 497
 schemes, 342
 sparsity promoting, 509
 Regularized minimization, 405, 548
 Reinforcement learning (RL), 11, 12
 techniques, 984
 Reject option, 306
 Rejection probability, 755
 Rejection sampling, 739
 Rejection scheme, 742
 Relative entropy, 61, 936
 Relaxed projection operator, 364
 Relevance vector machine (RVM), 15
 Reparameterization trick, 1007
 Reproducing kernel Hilbert space (RKHS), 114, 532, 539, 540, 542, 545, 550, 551, 672, 710, 711, 714, 1085, 1087
 Reproducing property, 539
 Resampling methods, 873, 875
 Residual networks, 973
 Resolvent of the subdifferential mapping, 411
 Restricted Boltzmann machine (RBM), 988
 Restricted isometry property (RIP), 452
 Ridge regression, 13, 14, 89–92, 265, 267, 428, 432–435, 496, 497, 551–553, 560, 599, 601, 665, 714
 Ring networks, 230
 Risk classification, 306
 Risk minimization, 347, 574
 Robbins–Monro algorithm, 194
 Robustness against overfitting, 609
 Root square estimation consistence, 500
 Rotation invariance, 1092
 Running intersection property, 824
- S**
- Sample mean, 43
 SCAD thresholding rule, 435
 Scale invariance, 1072, 1092
 Scatter matrices, 324
 Schur algorithm, 157
 Scoring-based structure learning methods, 864
 Seismic signal processing, 143
 Semisupervised learning, 11, 12, 222, 992
 Sensing matrix, 446, 449, 453–458, 462, 463, 468, 474, 477, 479, 486, 488, 490, 505, 511–513, 523–525, 1085, 1096
 sparse, 524
 Separator, 826
 Sequential importance sampling (SIS), 881
 Sequential minimal optimization (SMO), 576
 Set membership algorithms, 379
 Sigmoid link function, 318
 Sigmoidal Bayesian networks, 785
 Sigmoidal networks, 993
 Signal
 processing, 2, 6, 13–16, 41, 45, 48, 134, 140, 180, 208, 352, 380, 384, 412, 414, 428, 454, 461, 463, 842, 850
 subspace, 1052

- Singleton, 385, 388, 411
- Singular value, 261
- Singular value decomposition (SVD), 254, 260
- Singular value thresholding (SVT), 1101
- Singular vectors, 261
- Slack variables, 556
- Smoothing, 893
- Smoothly clipped absolute deviation (SCAD), 435, 503
- Snapshot ensembling, 930
- Soft thresholding, 434
- Soundness, 787
- SpAPSM, 505, 510, 524
- Spark of a matrix, 447
- SpaRSA, 483
- Sparse, 380, 404, 428, 431, 436, 438, 439, 445, 446, 448, 453, 457, 458, 490, 494, 499, 501, 510, 513, 520, 560, 581, 1069, 1088, 1092, 1097, 1098
- analysis, 513, 515
- analysis representation, 511
- coding, 1070
- coding representation vectors, 1071
- coding scheme, 1072
- DCT transforms, 465
- DFT, 460
- factor analysis, 1082
- Gaussian processes, 715
- learning, 510
- linear regression modeling, 653
- matrix, 577, 1089, 1093, 1100, 1101, 1103
- MKL variants, 581
- modeling, 14, 428, 473, 510, 516, 1040, 1096
- multitone signals, 460
- parameter vector, 447, 475
- regression modeling, 717
- representation, 447, 456, 459, 463, 512, 1069
- sensing matrix, 524
- signal, 454, 474
- signal recovery, 455, 487
- signal representation, 436
- solution, 442, 451, 455, 468, 494, 497, 500, 502, 523, 581, 1071
- structure, 494, 1071
- synthesis, 513
- synthesis modeling, 516
- target vector, 479
- wavelet representation, 509, 510
- Sparse Bayesian learning (SBL), 15, 667, 669
- Sparsely represented signal, 458
- Sparsest solution, 440, 442, 448, 450–452, 477, 478, 515
- Sparsity, 436, 437, 442, 451, 461, 463, 489, 492, 497, 500, 513, 532, 560, 667, 669, 671, 672, 685, 703, 1069, 1073
- assumption, 510
- constraint, 440, 456, 480, 672, 719, 1073
- in multipath, 437
- level, 445, 449, 452–454, 476, 479, 480, 484, 487, 489, 507, 509, 510, 515, 1071, 1072, 1098
- promoting
 - algorithm, 381, 474, 509
 - online algorithms, 212
 - optimization, 498
 - priors, 1082
 - quantifying performance, 487
 - regularization, 509
 - regularizers, 1076
 - structure, 454
- Spectral unmixing (SU), 718
- Spike and slab prior, 671
- Split Levinson algorithm, 154
- Square root estimation consistency, 501
- Stable embeddings, 459
- Standard Gaussian, 696
- Standard regression model, 500
- State space models, 843
- Stationary covariance function, 712
- Stationary Gaussian process, 712
- Statistical independence, 23
- Steady state, 203
- Steering vector, 164
- Stick breaking construction, 697
- Stick breaking representation of a DP, 697
- Stochastic
 - approximation, 194
 - convergence, 61
 - gradient, 240, 275
 - descent, 196, 404
 - descent schemes, 214
 - descent!algorithm, 198
 - online versions, 404
 - rationale, 181, 241, 403
 - scheme, 197, 232, 404
 - matrix, 235

- Stochastic matrices
 left, right and doubly, 233
- Stochastic processes, 41
- Stochastic volatility models, 894
- String kernels, 548
- Strongly attracting mapping, 365
- Strongly convex function, 399
- Student's t distribution, 666
- Sub-Gaussian PDFs, 1061
- Sub-Nyquist sampling, 460
- Subclassifiers, 333
- Subdifferential, 385
- Subgradient, 352, 385, 386, 388–390, 393–395, 398, 406, 415, 416, 419, 905
 algorithm, 388, 390, 391, 393, 398, 399, 420
 method, 388, 390
- Subjective priors, 610
- Sublinear convergence rate, 482
- Suboptimal classifier, 315
- Subspace
 dimensionality, 440, 466
 estimation, 1052
 learning methods, 171
 linear, 75, 440
 observations, 459, 460
 signal, 1052
 tracking, 1052
 tracking problem, 1052
- Subspace pursuit (SP), 480
- Sufficient statistic, 87
- Sum rule for probabilities, 22
- Super-Gaussian PDFs, 1061
- Supervised classification, 113
- Supervised learning, 8, 9, 11, 12, 80, 988, 991, 992
- Supervised PCA, 1049
- Support consistency, 501
- Support vector, 556, 569
- Support vector machine (SVM), 395, 532, 672
 classification, 572
 classification task, 575
 classifiers, 576, 589
 kernel classifier, 587
- Support vector regression (SVR), 672
- Supporting hyperplane, 362
- Switching linear dynamic system (SLDS), 859
- System identification, 141
- T**
- Tangent hyperplane, 355
- TensorFlow machine learning, 1026
- Test error, 96
- Text classification, 584
- Thresholding schemes, 490
- Time-delay neural networks, 976
- Time-frequency analysis, 516
- Toeplitz matrices, 150
- Tokens-tokenization, 1018
- Total least-squares (TLS)
 estimator, 293
 method, 287
 solution, 292, 295
- Total variation (TV), 497
- Training error, 96
- Transfer learning, 1013
- Transition probabilities, 746, 747, 749–751, 844–847, 852, 854, 857
- Translation invariance, 1092
- Tree classifiers, 333
- Trees, 803
- Triangulated graphs, 822
- Two-stage thresholding (TST) scheme, 484
- Type I estimator, 602
- U**
- Unbiased estimator, 64, 81–83, 161, 162, 174, 258, 260, 313, 345, 744, 766, 872, 873, 897
 linear, 161, 258
- Unconstrained optimization, 485
- Uncorrelated noise, 597
- Uncorrelated random variables, 125
- Undirected graphical models, 788, 799, 840
- Uniform distribution, 32
- Uniform random projection (URP), 491
- Uniform spherical ensemble (USE), 491
- Univariate Gaussian, 643
- Universal approximation property, 947
- Unlearning contribution, 990
- Unmixing matrix, 1066
- Unobserved
 latent variables, 702
 random variables, 611
- Unregularized LS, 89, 434, 501, 502
- Unscented Kalman filters, 170
- Unsupervised learning, 11, 12, 988, 992, 1067

V

- Validation, 109, 946
- Value ratio (VR), 586
- Value similarity (VS), 585
- Variance, 25
 - minimum, 83, 86, 166
- Variational
 - approximation method, 681
 - autoencoders, 1004, 1005
 - EM algorithm, 623, 665, 720, 837, 1080
 - message passing, 839
 - parameters, 681
 - posterior estimates, 710
- Vector regression, 672
- Vector space model (VSM), 584
 - equivalent, 587
- Vertex component analysis (VCA), 720
- VGG-16, 972
- Viterbi algorithm, 851
- Viterbi reestimation, 854
- Volterra models, 534

W

- Wake-sleep algorithm, 994
- Wasserstein distance, 1001
- Wasserstein probability, 1001
- Weak classifier, 337, 344, 349
- Weak convergence, 367
- Weight optimization, 576
- Weighting scheme, 495
- Welch bound, 449
- Well-posed problems, 91
- White noise, 51
- Widely linear estimator/filtering, 130
- Wiener models, 534
- Wiener–Hammerstein models, 534
- Wiener–Hopf equations, 126
- Wireless sensor network (WSN), 228
- Wirtinger calculus, 132
- Wishart PDFs, 662, 699

Y

- Yule–Walker equations, 52