4

Kernel Functions and Reproducing Kernel Hilbert Spaces

Whereas Chapter 3 gave a (moderately deep) introduction to the signal processing concepts to be used in this book, in this chapter we put together fundamental and advanced relevant concepts on Mercer's kernels and reproducing kernel Hilbert spaces (RKHSs). The fundamental building block of the kernel learning theory is the *kernel function*, which provides an elegant framework to compare complex and nontrivial objects. After its introduction, we review the concept of an RKHS, and state the representer theorem. Then we study the main properties on kernel functions and their construction, as well as the basic ideas to work with complex objects and reproducing spaces. The support vector regression (SVR) algorithm is also introduced in detail, as it will be widely used and modified in Part II for building many of the DSP algorithms with kernel methods therein. We end up the chapter with some synthetic examples illustrating the concepts and tools presented.

4.1 Introduction

Kernel methods build upon the notion of *kernel functions* and RKHSs. Roughly speaking, a Mercer kernel in a Hilbert space is a function that computes the inner product between two vectors embedded in that space. These vectors are maps of vectors in an Euclidean space, where the *mapping function* can be nonlinear. This informal definition will be formalized through Mercer's theorem, which gives the analytical power to kernel methods.

We will see that kernel functions are very often dot products in inaccessible Hilbert spaces. This means that vectors in that space are not explicitly defined; hence, only the dot products are accessible, but not the mapped vector coordinates therein. In spite of this, the expression of the dot product is the only tool needed to operate in such space. The RKHSs that are summarized in this chapter allow us to implicitly map a finite-dimension vector from an Euclidean space (typically known as *input space* $\mathcal{X}$) into a higher dimension Hilbert space (referred as *feature space* $\mathcal{H}$) by expressing the dot product inside this space as a function of the input space vectors solely. This property has allowed *kernel methods* to become a preferred tool in machine learning and pattern recognition for years, being an attractive alternative to traditional methods in statistics and signal processing. In the past decade, methods based on kernels have gained popularity in almost all applications of machine learning because of several

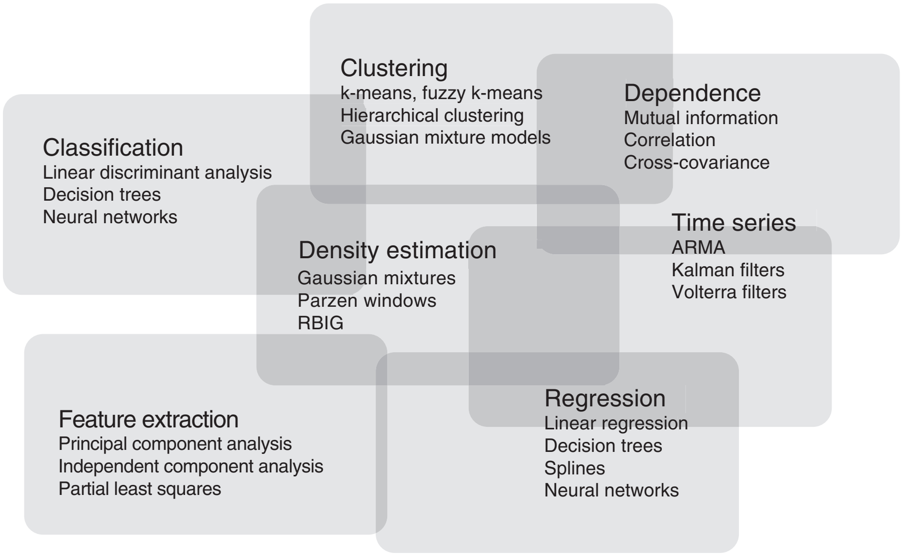

Figure 4.1 Different machine learning algorithms are available to tackle particular machine learning and signal-processing problems.

fundamental reasons. In particular, we can identify three main motivations for the use of kernel techniques in machine learning and signal processing.

The first one consists of the change of paradigm from classical nonlinear algorithms to kernel techniques that still rely on linear algebra. Linear methods used in statistics and machine learning typically resort to linear algebra operations, are very well established, and their properties and limits are mostly known. These facts give the user the tools needed to adapt linear algorithms to the application at hand and to obtain solutions very easily. Linear algorithms, however, are restricted to capturing linear relations between the features and dependent variables, and very often only capture second-order statistical relations. Such limitations call for extensions to nonlinear and higher order statistics algorithms. For every learning problem encountered in machine learning and signal processing, a full toolbox of nonlinear extensions is available (see Figure 4.1). Different approaches, however, include the nonlinearity in very different ways, as well as a particularly different set of parameters to tune and criteria to optimize their design. For example, the most well-known family of nonlinear algorithms is probably that of artificial NNs. In this case, a point-wise nonlinearity is included in every node/neuron in the network, and several parameters have to be tuned to control the model capacity (in this case, its architecture).

NNs dominated the field and applications in the 1980s and 1990s, and often produce improved performance with respect to linear algorithms. Nevertheless, a clear shortcoming was identified, which was that the knowledge about linear techniques cannot be easily translated into the design of the nonlinear ones. Actually, when using linear algorithms, a well-established theory and efficient methods are often available, but such advantages are lost when translating the linear neuron model into even a simple sigmoid-shaped neuron model. Kernel methods offer the opportunity to translate linear models into nonlinear models while still working with linear algebra. In this way, kernel methods will allow us to exploit all the intuitions and properties of linear algorithms. Essentially, kernel methods embed the data set S defined over the input or attribute feature space $\mathcal{X}$ ($S \subseteq \mathcal{X}$) into a higher (possibly infinite-dimensional) Hilbert space $\mathcal{H}$, also known as kernel *feature space*, and a linear algorithm is built on this last for yielding a nonlinear algorithm with respect to the input data space.

The second reason is related to the paradigm change introduced by kernel methods. They allow the user to take linear algorithms expressed in an Euclidean space and, by means of an often straightforward *kernelization* procedure, consisting of deriving a nonlinear counterpart of the algorithm, they provide nonlinear properties to the linear algorithms. The kernelization procedure consists of two basic steps. The first of them tries to find an expression of the linear algorithm as a function of dot products between data *only*. This representation is called the *dual* representation in contrast to the *primal* representation.[1] This is always possible for a linear algorithm by just taking into account that the parameters learned by the linear algorithm lie inside the subspace spanned by the data. A representation of these parameters as a linear combination of the mapped data straightforwardly leads to a dual representation, the combination parameters being the so-called *dual parameters*. The second step consists of the substitution of the Euclidean dot product by a dot product into an RKHS. From this point of view, kernel methods are methods that are linear in the parameters, or alternatively they are linear inside the RKHS. Notationally, the mapping function is denoted as $\boldsymbol{\phi} : \mathcal{X} \rightarrow \mathcal{H}$.

Whereas the nonlinear version of linear algorithms will exhibit increased flexibility, the computational load should increase, especially when we need to compute the sample new coordinates explicitly in that high-dimensional space. But this computation can be omitted by using the *kernel trick*; that is, when an algorithm can be expressed using dot products in the input space, then its (nonlinear) kernel version only needs the dot products between mapped samples. Kernel methods compute the similarity between examples (samples, objects, patterns, points) $S = \{\boldsymbol{x}_i\}_{i=1}^{N}$ using inner products between mapped examples, and thus the so-called kernel matrix $\boldsymbol{K}$, with entries $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$, contains all the necessary information to perform many classical linear algorithms in the feature space.

Figure 4.2 illustrates the concepts of kernel feature mapping, $\boldsymbol{\phi}$, and the exploitation of the kernel trick. The latter will allow us to measure similarities in the Hilbert space $\mathcal{H}$ using a reproducing kernel K that solely works with input examples from $\mathcal{X}$. In this example, the input space is not a proper representation space because the endowed Euclidean metric would wrongly tell that distant points along the manifold, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, are close. This is remedied by mapping the input examples to a Hilbert spaces $\mathcal{H}$ such that unfolds the original banana-shaped data distribution such that the linear dot product is an appropriate one. Computing such a dot product in $\mathcal{H}$ would be impossible unless the feature mapping is explicit. Fortunately, as we will see later, neither knowing the mapping nor the coordinates of the mapped data are strictly necessary, as we can

1 The dual representation is known in *multivariate statistics* as the Q-mode as opposed to the primal R-mode. Operating in Q-mode typically endows the method with numerical stability and computational efficiency in high-dimensional problems.

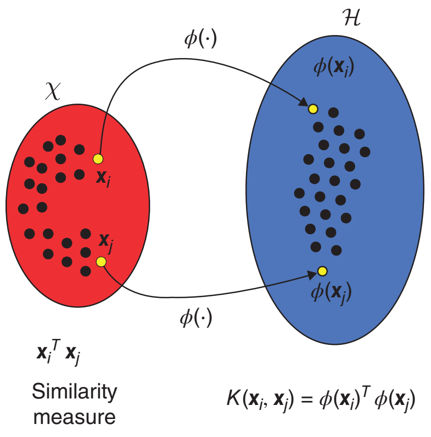

Figure 4.2 Illustration of the kernel feature mapping, ϕ, and the kernel trick by which we can measure similarities in $\mathcal{H}$ using a reproducing kernel that solely works with input examples from $\mathcal{X}$.

find a kernel function K that *reproduces* the similarity in $\mathcal{H}$ solely working with input examples from $\mathcal{X}$.

By comparison again with linear classical techniques, there is a third very important motivation for the use of kernel methods that is related to the *complexity of nonlinear machines*. If the structural complexity of a machine is too high, it is known that it may be unable to properly generalize. This is known as the problem of overfitting. In practice this problem boils down to models capable of adjusting the parameters so that the training data are fully explained, but then being unable to properly explain new, unseen, test data. Nonlinear learning machines typically introduce many parameters (on some occasions more than examples in the dataset) so they become extremely flexible and able to overfit the training set well. Limiting the capacity of the class of functions of interest is intimately related to the concept of *regularization*, already introduced in Chapter 3 for DSP signal models. We will see here how kernel methods include regularization in a very natural way, even working in inaccessible feature spaces. We will see also that kernel methods can generalize the concept of Tikhonov's regularization to infinite-dimensional feature spaces.

Finally, we should mention that kernel methods have been widely adopted in applied communities such as computer vision, time-series analysis and econometrics, physical sciences, as well as signal and image processing, because these methods typically lead to neat, simple, and intuitive algorithms in these fields. But, in addition, kernel methods allow the exploitation of a vast variety of kernel functions that can be designed and adapted to the application at hand. For example, combining and fusing heterogeneous signal sources (text, images, graphs) can be done easily via concepts of multiple kernel learning (MKL). Also, learning the kernel function directly from data is possible via generative models and exploitation of the wealth of information in the unlabeled data. Kernel design is an open research topic in machine learning and signal processing that does not stop giving theoretical results and designs for many different applications. In this chapter, we also review the main properties of kernel methods to construct new kernel functions.

4.2 Kernel Functions and Mappings

Kernel methods rely on the properties of kernel functions. As we will see next, they reduce to computing dot products of vectors mapped in Hilbert spaces through an implicit (not necessarily known) mapping function. A scalar product space (or pre-Hilbert space) is a space endowed with a scalar product. If the space is *complete* (i.e., every Cauchy sequence converges inside the space), then it is called a *Hilbert space*.

This section conveys a short introduction to kernel methods, functions approximation, and RKHSs. The tools reviewed here are of fundamental relevance for the next sections, and build upon solid branches of mathematics such as *linear algebra* (Golub and Van Loan, 1996) and *functional analysis* (Reed and Simon, 1981).

4.2.1 Measuring Similarity with Kernels

The dot (scalar or inner) product between two vectors is an algebraic operation that measures the similarity between them. Intuitively, a dot or scalar product measures how much of a vector is contained in the other, or alternatively how much the two vectors point in the same direction. The question raised here is whether one should naively rely on the dot product in the input space to measure the similarity of (possibly complex) objects or not. If the representation space of those objects is rich enough, a dot product in input space should suffice. Unfortunately, many times the input feature space is limited in resolution, and expressive power and a new richer representation is needed. Kernel methods rely on the notion of similarity between examples in a higher (possibly infinite-dimensional) Hilbert space. Consider the set of empirical data $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, in which $\boldsymbol{x}_i$ are the *inputs* taken from $\mathcal{X}$ and $y_i \in \mathcal{Y}$ are called the *outputs*. The learning from samples process consists of using these data pairs to predict a new set of test examples $\boldsymbol{x} \in \mathcal{X}$. In order to develop learning machines capable of generalizing well, kernel methods often support a good approximation to the structure of the data and incorporate regularization in a natural way.

In some of those cases when $\mathcal{X}$ does not stand for a good support of similarity, examples can be mapped to a (dot product) space $\mathcal{H}$ by using a mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}, \boldsymbol{x} \mapsto \boldsymbol{\phi}(\boldsymbol{x})$. The mapping function can be defined explicitly (if some prior knowledge about the problem is available) or implicitly, as in the case of kernel methods. The similarity between the elements in $\mathcal{H}$ can now be measured using its associated dot product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Here, we define a function that computes that similarity , $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that $(\boldsymbol{x}, \boldsymbol{x}') \mapsto K(\boldsymbol{x}, \boldsymbol{x}')$. This function, often called a *kernel*, is required to satisfy

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle_{\mathcal{H}}. \tag{4.1}$$

The mapping $\boldsymbol{\phi}$ is its *feature map*, the space $\mathcal{H}$ is the reproducing Hilbert *feature space*, and K is the reproducing kernel function since it *reproduces* dot products in $\mathcal{H}$ without even mapping data explicitly therein.

4.2.2 Positive-Definite Kernels

The types of kernels that can be written in the form of Equation 4.1 coincide with the class of *positive-definite kernels*. The property of positive definiteness is crucial in the field of kernel methods.

Definition 4.2.1 A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive-definite kernel *if and only if* there exists a Hilbert space $\mathcal{H}$ and a feature map $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}$ such that for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ we have $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle_{\mathcal{H}}$ for some inner-product space $\mathcal{H}$ such that $\forall \boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{\phi} \in \mathcal{H}$.

Definition 4.2.2 A kernel matrix (or Gram matrix) $\boldsymbol{K}$ is the matrix that results from applying the kernel function K to all pairs of points in the set $\{\boldsymbol{x}_i\}_{i=1}^{N}$; that is:

$$\boldsymbol{K} = \begin{pmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & K(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ K(\boldsymbol{x}_2, \boldsymbol{x}_1) & K(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_N, \boldsymbol{x}_1) & K(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}, \tag{4.2}$$

and whose entries are denoted as $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Kernel functions must be symmetric since inner products are always symmetric. In order to show that K is a valid kernel, it is not always sufficient to show that a mapping $\boldsymbol{\phi}$ exists; rather, this is a nontrivial theoretical task. In practice, a real symmetric $N \times N$ matrix $\boldsymbol{K}$, whose entries are $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ or simply $\boldsymbol{K}_{i,j}$, is called *positive-definite* if, for all $\alpha_1, \dots, \alpha_N \in \mathbb{R}$, $\sum_{i,j=1}^{N} \alpha_i \alpha_j \boldsymbol{K}_{i,j} \geq 0$. A positive-definite kernel produces a positive-definite Gram matrix in the RKHS.

Definition 4.2.3 Kernel matrices that are constructed from a kernel corresponding to a strict inner product space $\mathcal{H}$ are positive semidefinite.

This last property is easy to prove. Note that, by construction, we have $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle_{\mathcal{H}}$; thus, for any vector $\boldsymbol{\alpha} \in \mathbb{R}^N$:

$$\begin{aligned} \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\alpha} &= \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \boldsymbol{K}_{i,j} \alpha_j = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle_{\mathcal{H}} \alpha_j \\ &= \left\langle \sum_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i), \sum_j \alpha_j \boldsymbol{\phi}(\boldsymbol{x}_j) \right\rangle_{\mathcal{H}} = \left\| \sum_{i=1}^{N} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i) \right\|_{\mathcal{H}}^2 \geq 0. \end{aligned} \tag{4.3}$$

Accordingly, any algorithm which operates on the data in such a way that it can be mathematically described in terms of dot products can be used with any positive-definite kernel by simply replacing $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle_{\mathcal{H}}$ with kernel evaluations $K(\boldsymbol{x}, \boldsymbol{x}')$, a technique known as *kernelization* or a *kernel trick* (Schölkopf and Smola, 2002). Moreover, for a positive-definite kernel we do not need to know the explicit form of the feature map, but instead it is implicitly defined through the kernel calculation.

4.2.3 Reproducing Kernel in Hilbert Space and Reproducing Property

In this section we will define the notion of an RKHS through the reproducing property. Then, we will see that a kernel function is a positive-definite and symmetric function.

It is also relevant to observe that the kernel fully generates the space, and that for a given kernel there is a unique RKHS; conversely, every RKHS contains a single kernel.

Let us assume a Hilbert space $\mathcal{H}$ where its elements are functions, provided with a dot product $\langle \cdot, \cdot \rangle$. We will denote $f(\cdot)$ as one element of the space, and $f(\boldsymbol{x})$ as its value at a particular argument $\boldsymbol{x}$. We will assume that arguments belong to a real or complex Euclidean space; that is, $\boldsymbol{x} \in \mathbb{R}^N$ or $\boldsymbol{x} \in \mathbb{C}^N$ respectively.

Definition 4.2.4 RKHS (Aronszajn, 1950). A Hilbert space $\mathcal{H}$ is said to be an RKHS if: (1) the elements of $\mathcal{H}$ are complex- or real-valued functions $f(\cdot)$ defined on any set of elements $\boldsymbol{x}$; and (2) for every element $\boldsymbol{x}, f(\cdot)$ is bounded.

The name of these spaces comes from the so-called *reproducing property*. Indeed, in an RKHS $\mathcal{H}$, there exists a function $K(\cdot, \cdot)$ such that

$$f(\boldsymbol{x}) = \langle f(\cdot), K(\cdot, \boldsymbol{x}) \rangle, \quad f \in \mathcal{H} \tag{4.4}$$

by virtue of the Riesz representation theorem (Riesz and Nagy, 1955). This function is called a *kernel*.

Property 4 The kernel $K(\cdot, \cdot)$ is a positive-definite and symmetric function.

Assume that $f(\cdot) = K(\cdot, \boldsymbol{x})$ in Equation 4.4. Hence:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}') \rangle. \tag{4.5}$$

Applying the complex conjugate operator * in the kernel and the dot product leads to

$$K^*(\boldsymbol{x}, \boldsymbol{x}') = (\langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}') \rangle)^* = \langle K(\cdot, \boldsymbol{x}'), K(\cdot, \boldsymbol{x}) \rangle = K(\boldsymbol{x}', \boldsymbol{x}), \tag{4.6}$$

which proves that a kernel is symmetric. In addition, consider the series

$$\sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i), \tag{4.7}$$

where α_i is any finite set of complex numbers. The norm of this series can be computed as

$$\left\| \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i) \right\|^2 = \left\langle \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{N} \alpha_j, K(\cdot, \boldsymbol{x}_j) \right\rangle. \tag{4.8}$$

By virtue of the reproducing property, $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle K(\cdot, \boldsymbol{x}_i), K(\cdot, \boldsymbol{x}_j) \rangle$, and by the linearity of the dot product, Equation 4.8 can be written as

$$\left\| \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i) \right\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j, K(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0, \tag{4.9}$$

which proves that the kernel function is positive definite.

Property 5 The RKHS $\mathcal{H}$ with a kernel K is generated by $K(\cdot, \boldsymbol{x})$, where $\boldsymbol{x}$ belongs to any set.

It follows from the reproducing property that if $\langle f(\boldsymbol{x}), K(\cdot, \boldsymbol{x})\rangle = 0$, then, necessarily $f(\boldsymbol{x}) = 0$; hence, all elements in $\mathcal{H}$ are generated by the kernel.

Property 6 An RKHS contains a single reproducing kernel. Conversely, a reproducing kernel uniquely defines an RKHS.

If we consider the linear space generated by K, we can define

$$\langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}')\rangle = K(\boldsymbol{x}, \boldsymbol{x}'). \tag{4.10}$$

Let us now consider two sets of scalars μ_i and λ_j, $1 \leq i, j \leq N$; then:

$$\left\langle \sum_{i=1}^{N} \mu_i K(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{N} \lambda_j K(\cdot, \boldsymbol{x}_j) \right\rangle = \sum_{i=1}^{N} \sum_{j=1}^{N} \mu_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{4.11}$$

This expression satisfies the reproducing property and the requirements of dot product as follows:

$$\begin{array}{lcl} K(\boldsymbol{x}, \boldsymbol{x}') & = & K^*(\boldsymbol{x}', \boldsymbol{x}') \\ \langle K(\cdot, \boldsymbol{x}) + K(\cdot, \boldsymbol{x}'), K(\cdot, \boldsymbol{x}'')\rangle & = & \langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}')\rangle + \langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}'')\rangle \\ \langle \lambda K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}')\rangle & = & \lambda \langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x}')\rangle \\ \langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x})\rangle & = & 0 \quad \text{if and only if} \quad K(\cdot, x) = 0. \end{array}$$

The last requirement can be proven by using the Cauchy–Schwartz inequality:

$$\left\| \sum_{i=1}^{N} \lambda_i K(\cdot, \boldsymbol{x}_i), K(\cdot, \boldsymbol{x})\rangle \right\|^2 \leq \sum_{i=1}^{N} \lambda_i K(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{N} \lambda_j K(\cdot, \boldsymbol{x}_j)\rangle \langle K(\cdot, \boldsymbol{x}), K(\cdot, \boldsymbol{x})\rangle. \tag{4.12}$$

Hence:

$$\left\langle \sum_{i=1}^{N} \lambda_i K(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{N} \lambda_j K(\cdot, \boldsymbol{x}_j) \right\rangle = 0 \tag{4.13}$$

implies

$$\sum_{i=1} \lambda_i K(\boldsymbol{x}, \boldsymbol{x}_i) = 0 \tag{4.14}$$

for every $\boldsymbol{x}$. Therefore, K is unique in $\mathcal{H}$ and it generates a unique RKHS.

4.2.4 Mercer's Theorem

Mercer's theorem is one of the best known results of the mathematician James Mercer (January 15, 1883–February 21, 1932), and of fundamental importance in the context of kernel methods. Mercer's theorem is the key idea behind the so-called kernel trick, which allows one to solve a variety of nonlinear optimization problems through the construction of kernelized counterparts of linear algorithms. Mercer's theorem can be stated as follows.

Assume that $K(\cdot,\cdot)$ is a continuous kernel function satisfying the properties in Section 4.2.3. Assume further that the kernel belongs to the family of square integrable functions.

Theorem 4.2.5 (Mercer's theorem (Aizerman *et al.*, 1964)) Let $K(\boldsymbol{x},\boldsymbol{x}')$ be a bivariate function fulfilling the Mercer condition; that is, $\int_{\mathbb{R}^{N_r}\times\mathbb{R}^{N_r}} K(\boldsymbol{x},\boldsymbol{x}')f(\boldsymbol{x})f(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}' \geq 0$ for any square integrable function $f(\boldsymbol{x})$. Then, there exists a RKHS $\mathcal{H}$ and a mapping $\boldsymbol{\phi}(\cdot)$ such that $K(\boldsymbol{x},\boldsymbol{x}') = \langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{x}')\rangle$.

We will now consider a set D embedded in a space; typically, we will use only $\mathbb{R}^d$ or $\mathbb{C}^d$. From Mercer's theorem, it follows that a mapping function $\boldsymbol{\phi} : \mathcal{D} \rightarrow \mathcal{H}$ can be expressed as a (possibly infinite dimension) column vector:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \{\sqrt{\lambda_i}\boldsymbol{\phi}_i(\boldsymbol{x})\}_{i=1}^{\infty}. \tag{4.15}$$

The dot product between two of these maps $\boldsymbol{\phi}(\boldsymbol{x}_i)$ and $\boldsymbol{\phi}(\boldsymbol{x}_2)$ is defined then as the kernel function of vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ as

$$\langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{x}')\rangle = \boldsymbol{\phi}(\boldsymbol{x})^{\top}\boldsymbol{\phi}(\boldsymbol{x}') = \sum_{i=1}^{\infty}\lambda_i\boldsymbol{\phi}_i(\boldsymbol{x})\boldsymbol{\phi}_i(\boldsymbol{x}') = K(\boldsymbol{x},\boldsymbol{x}'). \tag{4.16}$$

Mercer's theorem shows that a mapping function into an RKHS and a dot product $K(\cdot,\cdot)$ exist if and only if $K(\cdot,\cdot)$ is a positive-definite function. Hence, if a given function $\mathcal{D}\times\mathcal{D}\rightarrow\mathbb{R}$ (or $\mathbb{C}$) is proven to be positive definite, then it is the kernel of a given RKHS.

Example 4.2.6 Kernel function of band-limited signals. Consider the interval $\omega \in [-W, W]$, and the subspace of square integrable functions generated by complex exponentials as

$$\boldsymbol{\phi}(t) = \mathrm{e}^{\mathbb{i}\omega t}. \tag{4.17}$$

Since the dot product of square integrable functions is the integral of their product, the kernel function inside this RKHS is

$$\langle h(t_1), h(t_2)\rangle = \frac{1}{2W}\int_{-W}^{W}\mathrm{e}^{\mathbb{i}\omega t_1}\,\mathrm{e}^{-\mathbb{i}\omega t_2}\,\mathrm{d}\omega = \frac{\sin(W(t_1-t_2))}{W(t_1-t_2)}. \tag{4.18}$$

This is then the kernel of the RKHS of signals that are band-limited to interval W. Moreover, the scalar dot product between functions is then

$$\langle f(\cdot), g(\cdot)\rangle = \int_{-W}^{W} F(\omega)G^*(\omega)\,\mathrm{d}\omega, \tag{4.19}$$

where $F(\omega)$ and $G(\omega)$ are the FT of $f(t)$ and $g(t)$ respectively. Indeed, if we denote the FT operator as $\mathcal{F}()$ and use this dot product, the reproducing property

$$f(t) = \langle f(\cdot), K(\cdot, t)\rangle = \int_{-W}^{W} F(\omega)\mathcal{F}^*(K(\cdot, t))\,\mathrm{d}\omega \tag{4.20}$$

holds. From Equation 4.18, it follows that $\mathcal{F}(K(\cdot, t)) = \mathrm{e}^{-\mathbb{i}\omega t}$. Then:

$$f(t) = \langle f(\cdot), K(\cdot, t)\rangle = \int_{-W}^{W} F(\omega)\,\mathrm{e}^{\mathbb{i}\omega t}\,\mathrm{d}\omega. \tag{4.21}$$

Example 4.2.7 Second-order polynomial kernel. Kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1+\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j)^2$ is a particular case of nth-order polynomial kernels. Assuming that $\boldsymbol{x}_i \in \mathbb{R}^2$, it is straightforward to find the mapping function $\boldsymbol{\phi} : \mathbb{R}^2 \rightarrow \mathcal{H}$. If $\boldsymbol{x}_i = \{x_i^{(1)}, x_i^{(2)}\}$, the dot product is

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j)^2 = \left(1 + x_i^{(1)}x_j^{(1)} + x_i^{(2)}x_j^{(2)}\right)^2, \tag{4.22}$$

which can be expanded as

$$\begin{aligned} K(\boldsymbol{x}_i, \boldsymbol{x}_j) = 1 &+ (x_i^{(1)}x_j^{(1)})^2 + (x_i^{(2)}x_j^{(2)})^2 + 2x_i^{(1)}x_j^{(1)} + 2x_i^{(2)}x_j^{(2)} \\ &+ 2x_i^{(1)}x_j^{(1)}x_i^{(2)}x_j^{(2)}. \end{aligned}$$

By visual inspection, the corresponding mapping can be found to be

$$\boldsymbol{\phi}(\boldsymbol{x}) = (1, (x^{(1)})^2, (x^{(2)})^2, \sqrt{2}x^{(1)}, \sqrt{2}x^{(2)}, \sqrt{2}x^{(1)}x^{(2)})^{\mathrm{T}}. \tag{4.23}$$

Each element of this vector is an eigenfunction of this space. The corresponding eigenvectors are respectively $\lambda_1 = \lambda_2 = \lambda_3 = 1$, and $\lambda_4 = \lambda_5 = \lambda_6 = \sqrt{2}$.

4.3 Kernel Properties

This section pays attention to important properties of kernel methods: the issue of regularization, the representer's theorem, and basic operations that can be implicitly done in RKHSs via kernels.

4.3.1 Tikhonov's Regularization

Regularization methods are those methods intended to turn an ill-posed problem into a well-posed one such that a stable solution exists. Well-posedness is a concept introduced by Hadamard in 1912 to determine the solvability of mathematical models of physical phenomena. He defined a problem as *well posed* if a solution of the problem exists and the solution is *unique*. Also, the solution must be *stable*; that is, when an initial condition slightly changes, the solution must not change abruptly. In particular, if a parameter estimation problem does not verify these three conditions, it is said to be *unstable*. A regularization method that assures well-posedness of problems is the Tikhonov minimization, which ensures stability of solutions in this sense (Tikhonov and Arsenin, 1977) (see Section 3.2.8).

Notationally, let us assume a set of N pairs of data examples $\{\boldsymbol{x}_i, y_i\}$; the problem consists of finding an estimation function f parameterized by a set of weights $\boldsymbol{a}$ such that we approximate observations as $y_i = f(\boldsymbol{x}_i, \boldsymbol{a}) + \epsilon_i$, where ϵ_i is the estimation error. The regularization procedure consists of constructing a functional (see Definition 3.2.11):

$$\mathcal{L} = \sum_{i=1}^{N} V(y_i, f(\boldsymbol{x}_i, \boldsymbol{a})) + \lambda\Omega(f), \tag{4.24}$$

where $V(\cdot)$ is a cost function over the empirical error or risk of the estimation procedure, and $\Omega(\cdot)$ plays the role of a regularizer over the parameters of the estimation function $f(\cdot)$. The idea behind the application of this functional is that, whereas the empirical risk is used in order to choose those parameters that produce the best prediction of y_i given $\boldsymbol{x}_i$, the regularizer is used to account for smoothness (flatness) of the solution. Let us exemplify the concept of smoothness through a simple example.

Example 4.3.1 Assume an arbitrary system expressed by

$$f(x) = -0.2x^3 - 0.5x + 0.5 \tag{4.25}$$

for $x \in \mathbb{R}$; this produces outputs y_i that are corrupted with zero mean white Gaussian noise of $\sigma_n = 0.03$. A 6th-order polynomial defined as

$$\hat{f}(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \tag{4.26}$$

is proposed for approximating the system's output. This model clearly has a complexity much higher than needed, and seven parameters including a bias term need to be adjusted, which are collectively grouped in $\boldsymbol{a} = [a_1, \dots, a_M]$, $M = 7$. A simple LS adjustment can be used to approximate the data. The polynomial shows a small error over the training sample, as is seen by the dotted line in Figure 4.3. Nevertheless, since the output data contain a given amount of noise, the test results in poor estimations with respect to the real function (continuous line). A regularized functional of the form

$$\mathcal{L} = \sum_{i=1}^{N} \|y_i - f(x_i, \boldsymbol{a})\|^2 + \lambda \sum_{j=0}^{M} a_j^2, \tag{4.27}$$

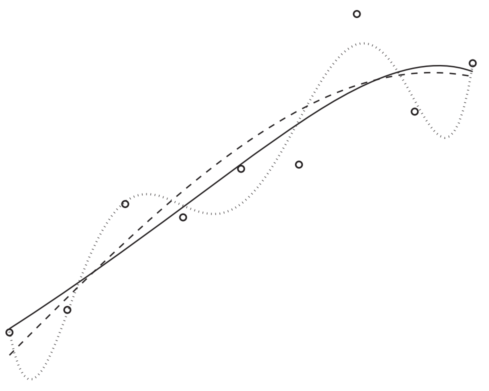

Figure 4.3 Nonlinear problem solved using two 6th-order polynomials. Circles represent points generated by the function (solid line) plus i.i.d. Gaussian noise. The dashed line corresponds to a regularized solution, while the dotted line corresponds to an unregularized solution ($\lambda = 0$).

which is known as ridge regression (Hoerl and Kennard, 1970) (see Exercise 4.9.3) can be applied in order to obtain a smoother function (dashed line) that better approximates the real function. The estimated parameters of the polynomial are as follows:

	a_0	a_1	a_2	a_3	a_4	a_5	a_6
LS	0.52	2.29	−22.6	91	−161.5	130.8	−39.73
RR	0.52	0.38	0.098	0.075	−0.117	−0.066	−0.035

which show how the range and variance of the obtained weights are constrained in the regularized solution, while the unregularized LS solution ($\lambda = 0$) yields big and uneven weights that produce unstable results (see Figure 4.3). Section 4.3.2 shows the extension of regularization to the field of RKHSs.

4.3.2 Representer Theorem and Regularization Properties

The representer theorem (Kimeldorf and Wahba, 1971) is a fundamental result in the field of kernel machines, since it establishes a generalization of the idea of regularization into an RKHS. The first motivation of the theorem is that, given a set of data, a linear expression of a solution based on a regularized functional exists in terms of a linear combination of the maps of the training data inside the RKHS. Moreover, recall that many kernels are expressed in infinite-dimension Hilbert spaces. This may completely preclude a good generalization, since the use of certain kernel functions may lead to a perfect fit of any data regardless of their statistical properties. Regularization becomes mandatory in all these cases. The second motivation is related to the fact that the implementation of smoothness is attached to the particular kernel chosen to solve the estimation problem. The theorem statement is as follows.

Theorem 4.3.2 (**Representer theorem**) (Kimeldorf and Wahba, 1971) Let $\Omega : [0, \infty) \to \mathbb{R}$ be a strictly monotonic increasing function; let $V : (\mathcal{X} \times \mathbb{R}^2)^N \to \mathbb{R} \cup \{\infty\}$ be an arbitrary loss function; and let $\mathcal{H}$ be an RKHS with reproducing kernel K. Then:

$$f^* = \min_{f \in \mathcal{H}} \{ V\left((f(\boldsymbol{x}_1), \boldsymbol{x}_1, y_1), \ldots, (f(\boldsymbol{x}_N), \boldsymbol{x}_N, y_N) \right) + \Omega(\|f\|_2^2) \} \tag{4.28}$$

admits a representation

$$f^*(\cdot) = \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i), \quad \alpha_i \in \mathbb{R}, \ \boldsymbol{\alpha} \in \mathbb{R}^{n \times 1} \tag{4.29}$$

That is, the function that minimizes the (regularized) optimization functional in Equation 4.28 is a linear function of dot products between data mapped into RKHS $\mathcal{H}$.

Since the mapped training data $\boldsymbol{\phi}(\boldsymbol{x}_i)$ span a subspace $\mathcal{H}_1$ inside the RKHS, a solution f may be expressed as a linear combination of these data plus a function g belonging to $\mathcal{H}_1 \perp \mathcal{H}_2$:

$$f = \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i) + g(\cdot) \tag{4.30}$$

for which, obviously,

$$\langle \boldsymbol{\phi}(\boldsymbol{x}_i), g \rangle = 0. \tag{4.31}$$

Then, on the one hand, by virtue of the reproducing property, for any training data point $\boldsymbol{x}_j$ the approximation function is

$$f(\boldsymbol{x}_j) = \sum_{i=1}^{N} \alpha_i K(\boldsymbol{x}_j, \boldsymbol{x}_i) + \langle \boldsymbol{x}_j, g(\cdot) \rangle \tag{4.32}$$

and, on the other hand, the regularization function Ω satisfies

$$\begin{aligned} \Omega\left(\left\| \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i) + g(\cdot) \right\| \right) &= \Omega\left(\sqrt{\left\| \sum_{i=1}^{N} \alpha_i K(\cdots, \boldsymbol{x}_i) \right\|^2} + \sqrt{\|g(\cdot)\|^2} \right) \\ &\geq \Omega\left(\| \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i) \| \right), \end{aligned} \tag{4.33}$$

which proves the theorem. We should stress that the representer theorem (Kimeldorf and Wahba, 1971) states that the solutions of a large class of optimization problems can be expressed by only a finite number of kernel functions. Hence its importance.

Example 4.3.3 Let us consider a linear prediction model $f(\boldsymbol{x}) = \langle f, K(\cdot, \boldsymbol{x}) \rangle$. A common regularizer in machine learning and statistics is the ℓ_2-norm $\|f\|_2^2$.

By virtue of the representer theorem, we can express the learning function f as a linear combination of kernel functions $f = \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i)$; then:

$$\begin{aligned} \|f\|^2 = \langle f, f\rangle &= \left\langle \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{N} \alpha_j K(\cdot, \boldsymbol{x}_j) \right\rangle \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j \left\langle K(\cdot, \boldsymbol{x}_i) K(\cdot, \boldsymbol{x}_j)\right\rangle \\ &\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\alpha}. \end{aligned} \tag{4.34}$$

Note that if we express $f(\boldsymbol{x}) = \langle \boldsymbol{w}, K(\cdot, \boldsymbol{x})\rangle = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x})$ then $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$ and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\alpha}$.

4.3.3 Basic Operations with Kernels

We now review some basic properties with kernels. The interest here is to introduce some basic operations that allow us to operate with objects embedded in high-dimensional, often inaccessible, feature spaces. Essentially, we will show that we can compute distances, norms, and angles, as well as perform projections and normalizations, in the feature space $\mathcal{H}$ via kernels in an implicit way.

Translation. A translation in feature space can be expressed as the modified feature map $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x}) + \Gamma$, $\Gamma \in \mathcal{H}$. Then, the translated dot product for $\langle \tilde{\boldsymbol{\phi}}(\boldsymbol{x}), \tilde{\boldsymbol{\phi}}(\boldsymbol{x}')\rangle_{\mathcal{H}}$ can be computed if we restrict Γ to lie in the span of the functions $\{\boldsymbol{\phi}(\boldsymbol{x}_1), \dots, \boldsymbol{\phi}(\boldsymbol{x}_N)\} \in \mathcal{H}$.

Centering. The previous translation allows us to center data $\{\boldsymbol{x}_i\}_{i=1}^{N} \in \mathcal{X}$ in the *feature space*. The mean of the data in $\mathcal{H}$ is $\boldsymbol{\phi}_\mu = c(1/N)\sum_{i=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_i)$, which is a linear combination of the span of functions, and hence it fulfills the requirement for Γ. One can center data in $\mathcal{H}$ by computing $\tilde{\boldsymbol{K}} = \boldsymbol{HKH}$, where entries of $\boldsymbol{H}$ are $\boldsymbol{H}_{ij} = \delta_{ij} - (1/N)$, and the Kronecker symbol is $\delta_{i,j} = 1$ if $i = j$ and zero otherwise.

Subspace projections. Given two points $\boldsymbol{\psi}$ and $\boldsymbol{\gamma}$ in the feature space, the projection of $\boldsymbol{\psi}$ onto the subspace spanned by $\boldsymbol{\gamma}$ is

$$\boldsymbol{\psi}' = \frac{\langle \boldsymbol{\gamma}, \boldsymbol{\psi}\rangle_{\mathcal{H}}}{\|\boldsymbol{\gamma}\|_{\mathcal{H}}^2} \boldsymbol{\gamma}.$$

Therefore, one can compute the projection $\boldsymbol{\psi}'$ expressed solely in terms of kernel evaluations. If one has access to vectors $\boldsymbol{\psi}$ and $\boldsymbol{\gamma}$, the projection can be computed explicitly. Otherwise, one can compute the dot products of the pre-image vectors using reproducing kernels. For example, assuming pre-images $\boldsymbol{\psi} = \boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\gamma} = \boldsymbol{\phi}(\boldsymbol{z})$, the norm of a vector in Hilbert spaces can be computed using a reproducing kernel; that is, $\|\boldsymbol{\gamma}\|_{\mathcal{H}}^2 = \langle \boldsymbol{\phi}(\boldsymbol{z}), \boldsymbol{\phi}(\boldsymbol{z})\rangle_{\mathcal{H}} = K_\Gamma(\boldsymbol{z}, \boldsymbol{z})$, and $\langle \boldsymbol{\gamma}, \boldsymbol{\psi}\rangle_{\mathcal{H}} = \langle \boldsymbol{\phi}(\boldsymbol{z}), \boldsymbol{\phi}(\boldsymbol{x})\rangle_{\mathcal{H}} = K_\Psi(\boldsymbol{z}, \boldsymbol{x})$.

Computing distances. Given that a kernel corresponds to a dot product in a Hilbert space $\mathcal{H}$, we can compute distances between mapped samples entirely in terms of kernel evaluations:

$$\|\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}')\|_{\mathcal{H}} = \sqrt{K(\boldsymbol{x},\boldsymbol{x}) + K(\boldsymbol{x}',\boldsymbol{x}') - 2K(\boldsymbol{x},\boldsymbol{x}')}. \tag{4.35}$$

Normalization. Exploiting the previous property, one can also normalize data in feature spaces implicitly:

$$K'(\boldsymbol{x},\boldsymbol{x}') = \left\langle \frac{\boldsymbol{\phi}(\boldsymbol{x})}{\|\boldsymbol{\phi}(\boldsymbol{x})\|}, \frac{\boldsymbol{\phi}(\boldsymbol{x}')}{\|\boldsymbol{\phi}(\boldsymbol{x}')\|} \right\rangle_{\mathcal{H}} = \frac{K(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{K(\boldsymbol{x},\boldsymbol{x})K(\boldsymbol{x}',\boldsymbol{x}')}}, \tag{4.36}$$

where the result is a new kernel function K'. Note that the operation is useless for some kernel functions. For example, the RBF kernel is given by $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/(2\sigma^2))$, so sample self-similarities are $K(\boldsymbol{x}_i,\boldsymbol{x}_i) = 1$ and normalization does not affect the resulting kernel. Note that the RBF kernel function implicitly maps examples to a hypersphere with unit norm, $\|\boldsymbol{\phi}(\boldsymbol{x})\| = 1$.

4.4 Constructing Kernel Functions

4.4.1 Standard Kernels

Any kernel method has its foundations in the definition of a kernel mapping function $\boldsymbol{\phi}$ that accurately measures the similarity among samples in some sense. But not all the kernel similarity functions can be used; instead, valid kernels are only those fulfilling Mercer's theorem. Roughly speaking, kernel functions yield positive-definite similarity matrices for a set of data measurements, and the most common ones in this setting are the linear $K(\boldsymbol{x},\boldsymbol{z}) = \langle \boldsymbol{x},\boldsymbol{z}\rangle$, the polynomial $K(\boldsymbol{x},\boldsymbol{z}) = (\langle \boldsymbol{x},\boldsymbol{z}\rangle + 1)^d$, $d \in \mathbb{Z}^+$, and the RBF $K(\boldsymbol{x},\boldsymbol{z}) = \exp(-\|\boldsymbol{x} - \boldsymbol{z}\|^2/2\sigma^2)$, $\sigma \in \mathbb{R}^+$. Note that, by Taylor series expansion, the RBF kernel is a polynomial kernel with infinite degree. Thus, the corresponding Hilbert space is infinite dimensional, which corresponds to a mapping into the space of functions $\mathcal{C}^\infty$. Table 4.1 summarizes the most useful kernels and their main characteristics.

The relation between *similarity* and *distance* has produced a high number of kernel functions that rely on standard distance measures, such as Mahalanobis kernels (Hofmann *et al.*, 2008). Most of the methods rely on the observation that the Gaussian kernel is of the form $K(\boldsymbol{x},\boldsymbol{x}') = \exp(-a\ d(\boldsymbol{x},\boldsymbol{x}'))$, where $d\ :\ \mathcal{X}\times\mathcal{X} \to \mathbb{R}_+$ represents a distance function and $a > 0$. Unfortunately, this instantiation in general is not true (Cortes *et al.*, 2003). If $\boldsymbol{K}$ is positive definite then $-\boldsymbol{K}$ is negative definite, but negative definite, but the converse is not true in general. For example $K(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^b$ is negative definite for $b \in [0,2)$. The RBF kernel is also of practical convenience (stability and only one parameter to be tuned), and it is the preferred kernel function in standard applications. Nevertheless, specific applications need particular kernel functions. Reviewing all the possible kernels is beyond the scope of this chapter. For a more general overview, including examples for other data structures such as graphs, trees, strings, and others, we refer the reader to Hofmann *et al.* (2008), Schölkopf and Smola (2002), Bakır *et al.* (2007), and Shawe-Taylor and Cristianini (2004).

Table 4.1 Main kernel functions used in the literature.

Kernel function	Expression
Linear kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + c$
Polynomial kernel	$K(\boldsymbol{x},\boldsymbol{y}) = (\alpha\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + c)^d$
RBF kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \exp\left(-\frac{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^2}{2\sigma^2}\right)$
Exponential kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \exp\left(-\frac{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert}{2\sigma^2}\right)$
Laplacian kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \exp\left(-\frac{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert}{\sigma}\right)$
Analysis of variance kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{k=1}^{d}\exp[-\sigma(\boldsymbol{x}^{(k)}-\boldsymbol{y}^{(k)})^2]^d$
Hyperbolic tangent (sigmoid) kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \tanh(\alpha\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + c)$
Rational quadratic kernel	$K(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^2}{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^2 + c}$
Multiquadric kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^2 + c^2}$
Inverse multiquadric kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{\sqrt{\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^2 + c^2}}$
Power kernel	$K(\boldsymbol{x},\boldsymbol{y}) = -\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^d$
Log kernel	$K(\boldsymbol{x},\boldsymbol{y}) = -\log(\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^d + 1)$
Cauchy kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{1 + (\Vert\boldsymbol{x}-\boldsymbol{y}\Vert^2/\sigma^2)}$
Chi-square kernel	$K(\boldsymbol{x},\boldsymbol{y}) = 1 - \sum_{k=1}^{d}\frac{(\boldsymbol{x}^{(k)}-\boldsymbol{y}^{(k)})^2}{\frac{1}{2}(\boldsymbol{x}^{(k)}+\boldsymbol{y}^{(k)})}$
Histogram (or min) intersection kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{k=1}^{d}\min(\boldsymbol{x}^{(k)},\boldsymbol{y}^{(k)})$
Generalized histogram intersection kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{k=1}^{m}\min(\vert\boldsymbol{x}^{(k)}\vert^{\alpha},\vert\boldsymbol{y}^{(k)}\vert^{\beta})$
Generalized T-Student kernel	$K(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{1 + \Vert\boldsymbol{x}-\boldsymbol{y}\Vert^d}$

4.4.2 Properties of Kernels

Taking advantage of some algebra and functional analysis properties (Golub and Van Loan 1996; Reed and Simon 1981), very useful properties of kernels can be derived. Let K_1 and K_2 be two positive-definite kernels on $\mathcal{X} \times \mathcal{X}$, $\mathcal{A}$ be a symmetric positive semidefinite matrix, $d(\cdot,\cdot)$ be a metric fulfilling distance properties, and $\mu > 0$. Then, the following kernels (Schölkopf and Smola, 2002) are valid:

$$K(\boldsymbol{x},\boldsymbol{x}') = K_1(\boldsymbol{x},\boldsymbol{x}') + K_2(\boldsymbol{x},\boldsymbol{x}') \tag{4.37}$$

$$K(\boldsymbol{x},\boldsymbol{x}') = \mu K_1(\boldsymbol{x},\boldsymbol{x}') \tag{4.38}$$

$$K(\boldsymbol{x},\boldsymbol{x}') = K_1(\boldsymbol{x},\boldsymbol{x}') \times K_2(\boldsymbol{x},\boldsymbol{x}') \tag{4.39}$$

$$K(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{x}^{\mathrm{T}}\mathcal{A}\boldsymbol{x}' \tag{4.40}$$

$$K(\boldsymbol{x},\boldsymbol{x}') = K(f(\boldsymbol{x}),f(\boldsymbol{x}')) \tag{4.41}$$

These basic properties make it easy to construct refined similarity measures better fitted to the data characteristics. One can sum dedicated kernels to different portions of the feature space, to different data representations, or even to different temporal or spatial scales through Equation 4.37. A scaling factor for each kernel can also be used (see Equation 4.38). Recent advances for kernel development also involve the following:

Convex combinations. By exploiting Equations 4.37 and 4.38, one can build new kernels by linear combinations of kernels:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{m=1}^{M} d_m K_m(\boldsymbol{x}, \boldsymbol{x}'), \tag{4.42}$$

where each kernel K_m could, for instance, work with particular feature subsets with eventually a different kernel function. This field of research is known as MKL, and many algorithms have been proposed to optimize jointly the weights and the kernel parameters (Rakotomamonjy *et al.*, 2008). Note that this composite kernel offers some insight into the problem as well, since relevant features receive higher values of d_m, and the corresponding kernel parameters θ_m yield information about similarity scales.

Deforming kernels. The field of semi-supervised kernel learning deals with techniques to modify the values of the training kernel, including the information from the whole data distribution. In this setting, the kernel K is either deformed with a graph distance matrix built with both labeled and unlabeled samples, or using kernels built from clustering solutions (Belkin *et al.* 2006; Sindhwani *et al.* 2005).

Generative kernels. Exploiting Equation 4.41, one can construct kernels from probability distributions by defining $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{p}, \boldsymbol{p}')$, where $\boldsymbol{p}$, $\boldsymbol{p}'$ are defined on the space $\mathcal{X}$ (Jaakkola and Haussler, 1999). This family of kernels is known as *probability product kernels between distributions* and defined as

$$K(\boldsymbol{p}, \boldsymbol{p}') = \langle \boldsymbol{p}, \boldsymbol{p}' \rangle = \int_{\mathcal{X}} \boldsymbol{p}(\boldsymbol{x})\boldsymbol{p}'(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}. \tag{4.43}$$

Joint input–output mappings. Kernels are typically built on a set of input samples. During recent years, the framework of *structured output learning* has dealt with the definition of joint input–output kernels, $K((\boldsymbol{x}, y), (\boldsymbol{x}', y'))$ (Bakır *et al.* 2007; Weston *et al.* 2003).

4.4.3 Engineering Signal Processing Kernels

The properties of kernel methods presented before have been widely used to develop new kernel functions suitable for tackling the peculiarities of the signals under analysis in a given DSP application. Let us review now some of the most relevant families of engineered kernels of special interest in signal processing.

Translation- or Shift-Invariant Kernels

A particular class of kernels is *translation-invariant kernels*, also known as shift-invariant kernels, which fulfill

$$K(\boldsymbol{u}, \boldsymbol{v}) = K(\boldsymbol{u} - \boldsymbol{v}). \tag{4.44}$$

A necessary and sufficient condition for a translation-invariant kernel to be Mercer's kernel (Zhang *et al.*, 2004) is that its FT must be real and nonnegative; that is:

$$\frac{1}{2\pi} \int_{\boldsymbol{v}=-\infty}^{+\infty} K(\boldsymbol{v})\, \mathrm{e}^{-\mathbb{i}2\pi\langle \boldsymbol{f}, \boldsymbol{v}\rangle} d\boldsymbol{v} \geq 0 \quad \forall \boldsymbol{f} \in \mathbb{R}^d \tag{4.45}$$

Bochner's theorem (Reed and Simon, 1981) states that a continuous shift-invariant kernel $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x} - \boldsymbol{x}')$ on $\mathbb{R}^d$ is positive definite if and only if the FT of K is nonnegative. If a shift-invariant kernel K is properly scaled, its FT $p(\boldsymbol{\omega})$ is a proper probability distribution.

As we will see in Chapter 10, this property has been recently used to approximate kernel functions and matrices with linear projections on a number of D randomly generated features as follows:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \int_{\mathbb{R}^d} p(\boldsymbol{\omega})\, \mathrm{e}^{-\mathbb{i}\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}')}\, \mathrm{d}\boldsymbol{\omega} \approx \sum\nolimits_{i=1}^{D} \tfrac{1}{D}\, \mathrm{e}^{-\mathbb{i}\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x}}\, \mathrm{e}^{\mathbb{i}\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x}'}, \tag{4.46}$$

where $p(\boldsymbol{\omega})$ is set to be the inverse FT of K, and $\boldsymbol{\omega}_i \in \mathbb{R}^d$ is randomly sampled from a data-independent distribution $p(\boldsymbol{\omega})$ (Rahimi and Recht, 2009). In this case, we define a D-dimensional feature map $\boldsymbol{z}(\boldsymbol{x}) \ :\ \mathbb{R}^d\mathrm{T}\mathbb{R}^D$, which can be *explicitly* constructed as $\boldsymbol{z}(\boldsymbol{x}) \ :=\ [\mathrm{e}^{(\mathbb{i}\boldsymbol{\omega}_1^{\mathrm{T}}\boldsymbol{x})}, \ldots, \exp(\mathbb{i}\boldsymbol{\omega}_D^{\mathrm{T}}\boldsymbol{x})]^{\mathrm{T}}$. In matrix notation, given N data points, the kernel matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$ can be approximated with the explicitly mapped data, $\boldsymbol{Z} = [\boldsymbol{z}_1 \cdots \boldsymbol{z}_n]^{\mathrm{T}} \in \mathbb{R}^{N\times D}$, and will be denoted as $\hat{\boldsymbol{K}} \approx \boldsymbol{Z}\boldsymbol{Z}^{\mathrm{T}}$. This property can be used to approximate any shift-invariant kernel. For instance, the familiar squared exponential (SE) Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/(2\sigma^2))$ can be approximated by using $\boldsymbol{\omega}_i \sim \mathcal{N}(0, \sigma^{-2}\boldsymbol{I}), 1 \leq i \leq D$.

Autocorrelation Kernels

A kernel related to the shift-invariant kernels family is the *autocorrelation-induced kernel*. Notationally, let $\{h_n\}$ be an $(N+1)$-samples limited-duration discrete-time real signal (i.e., $h_n = 0, \forall n \notin (0, N)$), and let $R_n^h = h_n * h_{-n}$ be its autocorrelation function. Then, the following shift-invariant kernel can be built:

$$K^h(n, m) = R_n^h(n - m), \tag{4.47}$$

which is called an *autocorrelation-induced kernel*, or simply an autocorrelation kernel. As R_n^h is an even signal, its spectrum is real and nonnegative, as expressed in Equation 4.45; hence, an autocorrelation kernel is always a Mercer kernel.

The previous two types of kernels are clear examples of the importance of including classic concepts of DSP in kernel-based algorithms. We will come back to this kernel in chapters in Part II.

Convolution Kernels

Convolution kernels, sometimes called *part kernels* or *bag of words kernels*, constitute an interesting development. The underlying idea for them is that the signal components (objects) are structured in any sense; thus, rather than measuring similarities between complete objects, one aggregates similarities of individual components (or feature subsets). A simple definition of a convolution kernel between $\boldsymbol{x}$ and $\boldsymbol{x}'$ is

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{\bar{\boldsymbol{x}}_i} \sum_{\bar{\boldsymbol{x}}'_i} \prod_{p=1}^{P} K_p(\boldsymbol{x}_p, \boldsymbol{x}'_p), \tag{4.48}$$

where $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{x}}'$ are the sets of parts ($p = 1, \dots, P$) of examples $\boldsymbol{x}$ and $\boldsymbol{x}'$ respectively, and $K_p(\boldsymbol{x}_p, \boldsymbol{x}'_p)$ is a kernel between the pth part of $\boldsymbol{x}$ and $\boldsymbol{x}'$.

It is easy to show that the RBF kernel is a convolution kernel by simply letting each of the P dimensions of $\boldsymbol{x}$ be a part, and using Gaussian kernels $K_p(\boldsymbol{x}_p, \boldsymbol{x}'_p) = \exp(-\|\boldsymbol{x}_p - \boldsymbol{x}'_p\|^2/(2\sigma^2))$. However, note also that the linear kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{p=1}^{P} \boldsymbol{x}_p \boldsymbol{x}'_p$ is not a convolution kernel, since we would need to sum products of more than one term.

Spatial Kernels

The latter convolution kernels have been extensively used in many fields of signal processing. In particular, the fields of image processing and computer vision have contributed with many kernel functions, especially designed to deal with the peculiarities of image features. Let us review two interesting spatial kernels to deal with *structured domains*, such as images.

A spatial kernel for image processing is inspired in the well-known concepts of *locality* and *receptive fields* of NNs. A kernel value is here computed by using not all features of an image, but only those which fall into a particular region inside a window (Schölkopf, 1997). Extensions of that kernel to deal with hierarchical organizations lead to the related *spatial pyramid match kernel* (Lazebnik *et al.*, 2006) and the *pyramid match kernel* (Grauman and Darrell, 2005). A standard approach in these methods is to divide the image into a grid of 1×1, 2×2, … equally spaced windows, resembling a pyramid, whose depth is referred as *level*. Then, for each level l, a histogram $\boldsymbol{h}_l$ is computed by concatenating the histograms of all subwindows within the level. Two images $\boldsymbol{x}, \boldsymbol{x}'$ are compared by combining the similarity of the individual levels:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{l=0}^{L-1} d_l K_l(\boldsymbol{h}_l, \boldsymbol{h}'_l), \tag{4.49}$$

where $d_l \in \mathbb{R}_+$ is an extra weighting parameter for each level, which must be optimized. As can be noted, the resulting kernel is just the result of applying the property of sum of kernels.

A second interesting kernel engineering was presented by Laparra *et al.* (2010) to deal with image denoising in the wavelet domain. Here, an SVR algorithm used a combination of kernels adapted to different scales, orientations, and frequencies in the *wavelet domain*. The specific signal relations were encoded in an anisotropic kernel obtained from mutual information measures computed on a representative image database. In particular, a set of Laplacian kernels was used to consider the intraband oriented relations within each wavelet subband:

$$K_\alpha(\boldsymbol{p}_i, \boldsymbol{p}_j) = \exp(-((\boldsymbol{p}_i - \boldsymbol{p}_j)^{\mathrm{T}} \boldsymbol{G}_\alpha^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{G}_\alpha (\boldsymbol{p}_i - \boldsymbol{p}_j))^{1/2}), \tag{4.50}$$

where $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2)$, σ_1, and σ_2 are the widths of the kernels, $\boldsymbol{p}_i \in \mathbb{R}^2$ denotes the spatial position of wavelet coefficient y_i within a subband, and $\boldsymbol{G}_\alpha$ is the 2D rotation matrix with rotation angle α, corresponding to the orientation of each subband.

Time–Frequency and Wavelet Kernels

Time–frequency signal decompositions in general, and wavelet analysis in particular, are at the core of signal and image processing. These signal representations have been widely studied and employed in practice. Inspired by wavelet theory, particular wavelet kernels have been proposed (Zhang *et al.*, 2004), which can be roughly approximated as

$$K(\boldsymbol{x}, \boldsymbol{x}') = \prod_{i=1}^{N} h\left(\frac{x_i - c}{a}\right) h\left(\frac{x'_i - c}{a}\right), \tag{4.51}$$

where a and c represent the wavelet dilation and translation coefficients respectively. A translation-invariant version of this kernel can be given by

$$K(\boldsymbol{x}, \boldsymbol{x}') = \prod_{i=1}^{N} h\left(\frac{x_i - x'_i}{a}\right), \tag{4.52}$$

where in both kernels the function $h(x)$ denotes a *mother wavelet function*, which is typically chosen to be $h(x) = \cos(1.75x)\exp(-x^2/2)$, as it yields to a valid admissible kernel function (Zhang *et al.*, 2004). Note that, in this case, the wavelet kernel is the result of applying the properties of direct sum and direct product of kernels.

4.5 Complex Reproducing Kernel in Hilbert Spaces

The complex representation of a magnitude is of important interest in signal processing, particularly in communications, since it provides a natural and compact expression that makes signal manipulation easier. The justification for the use of complex numbers in communications arises from the fact that any band-pass *real* signal centered around a given frequency admits a representation in terms of *in-phase* and *quadrature* components. Indeed, assume a real-valued signal $x(t)$ whose spectrum lies between two limits ω_{min} and ω_{max}. This function can the be expressed as

$$x(t) = A_{\mathrm{I}}(t)\cos\omega_0 t - A_{\mathrm{Q}}(t)\sin\omega_0 t, \tag{4.53}$$

with $A_{\mathrm{I}}(t)$ and $A_{\mathrm{Q}}(t)$ being the in-phase and quadrature components. This expression can be rewritten as

$$x(t) = \mathrm{Re}\{(A_{\mathrm{I}}(t) + \mathbb{i}A_{\mathrm{Q}}(t))\, e^{\omega_0 t}\} = \mathrm{Re}\{A(t)\, \mathrm{e}^{\omega_0 t}\} \tag{4.54}$$

for some arbitrary frequency ω_0 called central frequency or carrier, $A(t)$ being any arbitrary function called a complex envelope of $x(t)$. Since in-phase and quadrature components are orthogonal with respect to the dot product of L^2 functions, it is straightforward that $A(t)$ can be modulated as in Equation 4.53 without loss of information. The central frequency is usually known and removed during the signal processing, thus obtaining the complex envelope. This signal can either be processed as a pair of real-valued signals or as a complex signal, thus obtaining a more compact notation.

Though the concept of a complex-valued Mercer kernel is classic (Aronszajn, 1950), it was proposed by Martínez-Ramón *et al.* (2005); Martínez-Ramón *et al.* (2007) for its use in antenna array processing, and with a more formal treatment and rigorous justification by Bouboulis and coworkers (Bouboulis and Theodoridis, 2010, 2011; Bouboulis *et al.*, 2012) and Ogunfunmi and Paul (2011) for its use with the kernel LMS (KLMS) (Liu *et al.*, 2008, 2009) with the objective to extend this algorithm to a complex domain, as given in

$$\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x}) + \mathbb{i}\boldsymbol{\phi}(\boldsymbol{x}) = K((\boldsymbol{x}_{\mathbb{R}}, \boldsymbol{x}_{\mathbb{I}}), \cdot) + \mathbb{i}K((\boldsymbol{x}_{\mathbb{R}}, \boldsymbol{x}_{\mathbb{I}}), \cdot), \tag{4.55}$$

which is a transformation of the data $\boldsymbol{x} = \boldsymbol{x}_{\mathbb{R}} + \mathbb{i}\boldsymbol{x}_{\mathbb{I}} \in \mathbb{C}^d$ into a complex RKHS. This is known as the *complexification trick*, where the kernel is defined over real numbers.

Since the complex LMS algorithm involves the use of a complex gradient, the Wirtinger calculus must be introduced in the notation. The reason is that the cost function of the algorithm is real valued, and it is defined over a complex domain. Hence, it is non-holomorphic and complex derivatives cannot be used. A convenient way to compute such derivatives is to use Wirtinger derivatives. Assume a variable $x = x_{\mathbb{R}} + \mathbb{i}x_{\mathbb{I}} \in \mathbb{C}$ and a non-holomorfic function $f(x) = f_{\mathbb{R}}(x) + f_{\mathbb{I}}(x)$, then its Wirtinger derivatives with respect to x and x^* are

$$\begin{aligned}\frac{\partial f}{\partial x} &= \frac{1}{2}\left(\frac{\partial f_{\mathbb{R}}}{\partial x_{\mathbb{R}}} + \frac{\partial f_{\mathbb{I}}}{\partial x_{\mathbb{I}}}\right) + \frac{\mathbb{i}}{2}\left(\frac{\partial f_{\mathbb{I}}}{\partial x_{\mathbb{R}}} - \frac{\partial f_{\mathbb{R}}}{\partial x_{\mathbb{I}}}\right)\\ \frac{\partial f}{\partial x^*} &= \frac{1}{2}\left(\frac{\partial f_{\mathbb{R}}}{\partial x_{\mathbb{R}}} - \frac{\partial f_{\mathbb{I}}}{\partial x_{\mathbb{I}}}\right) + \frac{\mathbb{i}}{2}\left(\frac{\partial f_{\mathbb{I}}}{\partial x_{\mathbb{R}}} + \frac{\partial f_{\mathbb{R}}}{\partial x_{\mathbb{I}}}\right).\end{aligned} \tag{4.56}$$

This concept is restricted to complex-valued functions defined in $\mathbb{C}$. Authors generalize this concept to functions defined in n RKHS through the definition of Fréchet differentiability. The kernel function in this RKHS is defined, from Equation 4.55, as

$$\hat{K}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}^{\mathrm{H}}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}') \quad = (\phi(\boldsymbol{x}) - \mathbb{i}\phi(\boldsymbol{x}))(\phi^{\mathrm{T}}(\boldsymbol{x}) + \mathbb{i}\phi^{\mathrm{t}}(\boldsymbol{x})) = 2K(\boldsymbol{x}, \boldsymbol{x}'). \tag{4.57}$$

The representer theorem (see Theorem 4.3.2) can be rewritten here as follows:

$$f^*(\cdot) = \sum_{i=1}^{N} \alpha_i K(\cdot, \boldsymbol{x}_i) + \mathbb{i}\beta_i K(\cdot, \boldsymbol{x}_i), \tag{4.58}$$

where $\alpha_i, \beta_i \in \mathbb{R}$. Note, however, that Theorem 4.3.2 is already defined over complex numbers. Bouboulis *et al.* (2012) use pure complex kernels; that is, kernels defined over $\mathbb{C}$. In this case, the representer theorem can be simply written as

$$f^*(\cdot) = \sum_{i=1}^{N} \left(\alpha_i + \mathbb{i}\beta_i\right) K^*(\cdot, \boldsymbol{x}_i), \tag{4.59}$$

which resembles the standard expansion when working with real data except for the complex nature of the weights and the conjugate operation on the kernel.

4.6 Support Vector Machine Elements for Regression and Estimation

This section introduces the instantiation of a kernel method for regression and function approximation; namely, the SVR (Schölkopf and Smola 2002; Smola and Schölkopf 2004; Vapnik 1995). This method will accompany us in the following chapters, and it conveys all the key elements to work with a variety of estimation problems, including convexity of the optimization problem, regularized solution, sparsity, flexibility for nonlinear modeling, and adaptation to different sources of noise. Let us review two important definitions; namely, the SVR data model and the loss function used in the minimization problem. We will come back to these definitions later in order to define alternative signal models and cost functions, as well as to study model characteristics. Departing from the *primal problem*, we derive the SVR equations, and then we summarize the main properties of the SVR signal model.

4.6.1 Support Vector Regression Signal Model and Cost Function

Definition 4.6.1 (**Nonlinear SVR signal model**) Let a labeled training i.i.d. data set $\{(\boldsymbol{v}_i, y_i),\ i = 1, \dots, N\}$, where $\boldsymbol{v}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The SVR signal model first maps the observed explanatory vectors to a higher dimensional kernel feature space using a nonlinear mapping $\boldsymbol{\phi} : \mathbb{R}^N \to \mathcal{H}$, and then it calculates a linear regression model inside that feature space; that is:

$$\hat{y}_i = \langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{v}_i)\rangle + b, \tag{4.60}$$

where $\boldsymbol{w}$ is a weight vector in $\mathcal{H}$ and b is the regression bias term. Model residuals are given by $e_i = y_i - \hat{y}_i$.

In order to obtain the model coefficients, the SVR minimizes a cost function of the residuals, which is often regularized with the ℓ_2 norm of $\boldsymbol{w}$. That is, we minimize with regard $\boldsymbol{w}$ the following primal functional:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \mathcal{L}(e_i). \tag{4.61}$$

In the standard SVR formulation, the Vapnik ε-insensitive cost is often used (Vapnik, 1998).

Definition 4.6.2 (**Vapnik's ε-insensitive cost**) Given a set or residual errors e_i in an estimation problem, the ε-insensitive cost is given by

$$\mathcal{L}_\varepsilon(e_i) = C\max(|e_i| - \varepsilon, 0), \tag{4.62}$$

where C represents a trade-off between regularization and losses. Those residuals lower than ε are not penalized, whereas those larger ones have linear cost.

Here, we want to highlight an important issue regarding the loss. The Vapnik ε-insensitive cost function results in a suboptimal estimator in many applications when combined with a regularization term (Schölkopf and Smola 2002; Smola and Schölkopf 2004; Vapnik 1995). This is because a linear cost is not the most suitable one to deal with Gaussian noise, which will be a usual situation in a number of time-series analysis applications. This fact has been previously taken into account in the formulation of LS-SVM (Suykens *et al.*, 2002), also known as kernel ridge regression (KRR) (Shawe-Taylor and Cristianini, 2004), where a quadratic cost is used, but in this case, the property of sparsity is lost.

The ε-Huber cost was proposed by Rojo-Álvarez *et al.* (2004), combining both the quadratic and the ε-insensitive zones, and it was shown to be a more appropriate residual cost not only for time-series problems, but also for function approximation problems in general where the data can be fairly considered to be i.i.d. (Camps-Valls. *et al.*, 2007).

Definition 4.6.3 (**ε-Huber cost function**) The ε-Huber cost is given by

$$\mathcal{L}_{\varepsilon\mathrm{H}}(e_i) = \begin{cases} 0, & |e_i| \leq \varepsilon \\ \frac{1}{2\delta}(|e_i| - \varepsilon)^2, & \varepsilon \leq |e_i| \leq e_C \\ C(|e_i| - \varepsilon) - \frac{1}{2}\delta C^2, & |e_i| \geq e_C \end{cases} \tag{4.63}$$

where $e_C = \varepsilon + \delta C$; ε is the insensitive parameter, and δ and C control the trade-off between the regularization and the losses.

The three different regions in ε-Huber cost can work with different kinds of noise. First, the ε-insensitive zone neglects absolute residuals lower than ε. Second, the quadratic cost zone is appropriate for Gaussian noise. Third, the linear cost zone is efficient for limiting the impact of possibly present outliers on the model coefficients estimation. Note that Equation 4.63 represents the Vapnik ε-insensitive cost function when δ is small enough, the MMSE criterion for $\delta C \to \infty$ and $\varepsilon = 0$, and Huber cost function when $\varepsilon = 0$ (see Figure 4.4).

4.6.2 Minimizing Functional

In order to adjust the model parameters, an optimality criterion must be chosen. Typically in machine learning, one selects a risk function based on the *pdf* of data, which is unfortunately unknown. Hence, an induction principle is needed to best fit the real *pdf* based on the available data. A common choice is to minimize the so-called

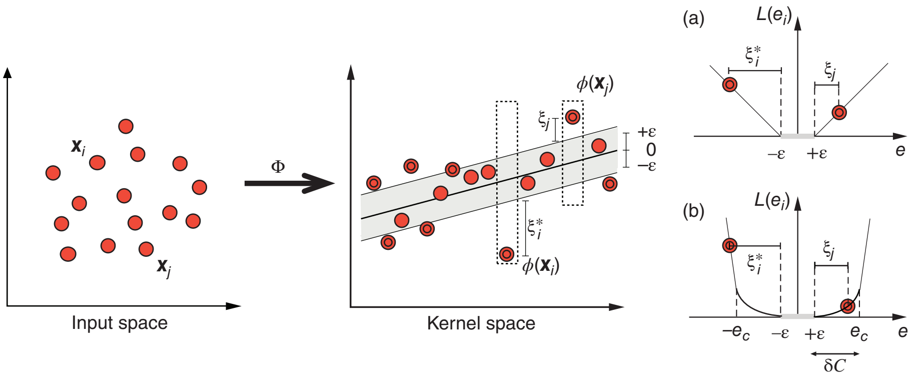

Figure 4.4 SVR signal model and cost functions. Samples in the input space are mapped onto an RKHS, and a linear regression is performed therein. All samples outside a fixed tube of size ε are penalized, and they are the support vectors (double circles). Penalization is given by (a) the Vapnik ε-insensitive or (b) the ε-Huber cost functions.

empirical risk; that is, the error in the training data set. However, to alleviate the problem of overfitting and control model complexity, regularization is usually adopted (Tikhonov and Arsenin, 1977), which is carried out in practice by minimizing the norm of the model parameters. This is intuitively equivalent to find the estimator which uses the minimum possible energy of the data to estimate the output. The resulting functional should take into account both this complexity term and an empirical error measurement term. The latter is defined according to an a-priori determined cost function of the committed errors.

Hence, the problem of estimating the coefficients can be stated as the minimization of the following functional:

$$F(\boldsymbol{w}, e_i) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \mathcal{L}_{\varepsilon\mathrm{H}}(e_i). \tag{4.64}$$

Introducing the previous loss in Equation 4.63 into Equation 4.64, we obtain the following functional:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2\delta}\sum_{i\in I_1}(\xi_i^2 + \xi_i^{*2}) + C\sum_{i\in I_2}(\xi_i + \xi_i^*) - \sum_{i\in I_2}\frac{\delta C^2}{2} \tag{4.65}$$

to be minimized with respect to $\boldsymbol{w}$ and $\{\xi_i^{(*)}\}$, and constrained to

$$y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{v}_i) - b \leq \varepsilon + \xi_i \tag{4.66}$$

$$-y_i + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{v}_i) + b \leq \varepsilon + \xi_i^* \tag{4.67}$$

$$\xi_i, \xi_i^* \geq 0 \tag{4.68}$$

for $i = 1, \cdots, N$. For notation simplicity, $\{\xi_i^{(*)}\}$ will denote both $\{\xi_i\}$ and $\{\xi_i^*\}$ hereafter. Here, $\{\xi_i^{(*)}\}$ are *slack variables* or *losses*, which are introduced to handle the residuals according to the robust cost function. I_1 and I_2 are the sets of samples for which losses are required to have a quadratic or a linear cost respectively, and these sets are not necessarily static during the optimization procedure.

The optimization problem involves two operations: a minimization and a set of constraints that cannot be violated. This a standard problem typically solved using Lagrange multipliers.[2] Essentially, one has to include linear constraints in Equations 4.66–4.68 into Equation 4.65, which gives us the dual form of the problem. Then, the primal–dual functional (sometimes referred to as the Lagrange functional) is given by

$$\begin{aligned} L_{\mathrm{PD}} = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2\delta}\sum_{i\in I_1}(\xi_i^2 + \xi_i^{*2}) + C\sum_{i\in I_2}(\xi_i + \xi_i^*) - \sum_{i\in I_2}\frac{\delta C^2}{2} \\ - \sum_i(\beta_i\xi_i + \beta_i^*\xi_i^*) + \sum_i(\alpha_i - \alpha_i^*)\left(y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{v}_i) - b - \varepsilon - \xi_i\right) \end{aligned} \tag{4.69}$$

constrained to $\alpha_i^{(*)}, \beta_i^{(*)}, \xi_i^{(*)} \geq 0$. By making zero the gradient of L_{PD} with respect to the primal variables (Rojo-Álvarez *et al.*, 2004), the following conditions are obtained:

$$\alpha_i^{(*)} = \frac{1}{\delta}\xi_i^{(*)} \quad (i \in I_1) \tag{4.70}$$

$$\alpha_i^{(*)} = C - \beta_i^{(*)} \quad (i \in I_2), \tag{4.71}$$

as well as the following expression relating the primal and dual model weights:

$$\boldsymbol{w} = \sum_n(\alpha_i - \alpha_i^*)\boldsymbol{\phi}(\boldsymbol{v}_i). \tag{4.72}$$

Constraints in Equations 4.70, 4.71, and 4.72 are then included into the Lagrange functional in Equation 4.69 in order to remove the primal variables. Note that the dual problem can be obtained and expressed in matrix form, and it corresponds to the maximization of

$$-\frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^{\mathrm{T}}[\boldsymbol{K} + \delta\boldsymbol{I}](\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^{\mathrm{T}}\boldsymbol{y} - \varepsilon\mathbf{1}^{\mathrm{T}}(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) \tag{4.73}$$

constrained to

$$C \geq \alpha_i^{(*)} \geq 0, \tag{4.74}$$

2 Giuseppe Lodovico (Luigi) Lagrangia (1736–1813) was the father of a famous methodology for optimization of functions of several variables subject to equality and inequality constraints. The method of Lagrange multipliers essentially solves $\max_{x,y} f(x, y)$ s.t. $g(x, y) = c$. He proposed to introduce a new variable (called the "Lagrange multiplier" α) and optimize the new function: $\max_{x,y,\lambda}\{\Lambda(x, y, \alpha)\}$ s.t. $f(x, y) - \alpha(g(x, y) - c)$, which is equivalent to solving the dual problem $\max_{x,y,\lambda}\{f(x, y) - \alpha(g(x, y) - c)\}$ s.t. $\alpha \geq 0$. Interestingly enough, the method is applicable to any number of variables and constraints, $\min\{f_0(x)\}$ s.t. $f_i(x) \leq 0$ and $h_j(x) = 0$. Then, the dual problem reduces to minimizing $\Lambda(x, \alpha, \mu) = f_0(x) - \sum_i \alpha_i f_i(x) - \sum_j \mu_j h_i(x)$. The proposed procedure to solve these types of problems is very simple: first, one has to derive this primal–dual problem and equate to zero, $\nabla_\lambda \Lambda(x, y, \alpha) = 0$, then the constraints obtained are stationary points of the solution, which are eventually included in the problem again. These give rise to a linear or quadratic problem, for which there are very efficient solvers nowadays.

where $\boldsymbol{\alpha}^{(*)} = [\alpha_1^{(*)}, \cdots, \alpha_N^{(*)}]^{\mathrm{T}}$. Then, after obtaining Lagrange multipliers $\boldsymbol{\alpha}^{(*)}$, the time-series model for a new sample at time instant m can be readily expressed as

$$y_j = f(\boldsymbol{v}_j) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \boldsymbol{\phi}(\boldsymbol{v}_i)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{v}_j) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) K(\boldsymbol{v}_i, \boldsymbol{v}_j), \tag{4.75}$$

where the dot product $\boldsymbol{\phi}(\boldsymbol{v}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{v}_j)$ has been finally replaced by the kernel function working solely on input samples $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$; that is: $K(\boldsymbol{v}_i, \boldsymbol{v}_j) = \boldsymbol{\phi}(\boldsymbol{v}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{v}_j)$, which is a function of weights in the input space associated with nonzero Lagrange multipliers. More details on the derivation of these equations can be found in the literature for SVR (Smola *et al.*, 1998) and for linear SVM–ARMA (Rojo-Álvarez *et al.*, 2004).

By including the ε-Huber residual cost into Equation 4.64, the SVR coefficients can be estimated by solving a quadratic programming (QP) problem (Rojo-Álvarez *et al.* 2004; Smola and Schölkopf 2004). Several relevant properties can be highlighted, which can be shown by stating the Lagrange functional, setting the Karush–Khun–Tucker (KKT) conditions, and then obtaining the dual functional. These properties are summarized next.

Property 7 (SVR sparse solution and support vectors) The weight vector in $\mathcal{H}$ can be expanded in a linear combination of the transformed input data:

$$\boldsymbol{w} = \sum_{i=1}^{N} \eta_i \boldsymbol{\phi}(\boldsymbol{v}_i), \tag{4.76}$$

where $\eta_i = (\alpha_i - \alpha_i^*)$ are the model weights, and $\alpha_i^{(*)}$ are the Lagrange multipliers corresponding to the positive and negative residuals in the nth observation. Observations with nonzero associated coefficients are called *s*upport vectors, and the solution is expressed as a function of them solely.

This property describes the sparse nature of the SVM solution for estimation problems.

Property 8 (Robust expansion coefficients) A nonlinear relationship between the residuals and the model coefficients for the ε-Huber cost is given by

$$\eta_i = \frac{\partial L_{\varepsilon\mathrm{H}}(e)}{\partial e}\bigg|_{e=e_i} = \begin{cases} 0, & |e_i| \le \varepsilon \\ \frac{1}{\delta} \cdot \mathrm{sgn}(e_i)(|e_i| - \varepsilon), & \varepsilon < |e_i| \le \varepsilon + \gamma C \\ C \cdot \mathrm{sgn}(e_i), & |e_i| > \varepsilon + \gamma C. \end{cases} \tag{4.77}$$

Therefore, the impact of a large residual e_i on the coefficients is limited by the value of C in the cost function, which yields estimates of the model coefficients that are robust in the presence of outliers.

The kernel trick in SVM consists of stating a data-processing algorithm in terms of dot products in the RKHS, and then substituting those products by Mercer kernels. The kernel expression is actually used in any kernel machine, but neither the mapping function $\boldsymbol{\phi}(\cdot)$ nor the RKHS need to be known explicitly. The Lagrangian of Equation 4.61 is used to obtain the dual problem, which in turn yields the Lagrange multipliers used as model coefficients.

Property 9 (Regularization in the dual) The dual problem of Equation 4.64 for the ε-Huber cost corresponds to the maximization of

$$-\frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^{\mathrm{T}}(\boldsymbol{K}+\delta\boldsymbol{I})(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)+(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^{\mathrm{T}}\boldsymbol{y}-\varepsilon\mathbf{1}^{\mathrm{T}}(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*) \tag{4.78}$$

constrained to $0 \leq \alpha_i^{(*)} \leq C$. Here, $\boldsymbol{\alpha}^{(*)} = [\alpha_1^{(*)}, \cdots, \alpha_n^{(*)}]^{\mathrm{T}}$, $\boldsymbol{y} = [y_1, \cdots, y_{N_r}]^{\mathrm{T}}$, $\boldsymbol{K}$ represents the kernel matrix, given by $\boldsymbol{K}_{i,j} = K(\boldsymbol{v}_i, \boldsymbol{v}_j) = \langle\boldsymbol{\phi}(\boldsymbol{v}_i), \boldsymbol{\phi}(\boldsymbol{v}_j)\rangle$, $\mathbf{1}$ is an all-ones column vector, and $\boldsymbol{I}$ is the identity matrix.

The use of the quadratic zone in the ε-Huber cost function gives rise to a numerical regularization. The effect of δ in the solution was analyzed by Rojo-Álvarez *et al.* (2004).

Property 10 (Estimator as an expansion of kernels) The estimator is given by a linear regression in the RKHS, and it can be expressed only in terms of the Lagrange multipliers and Mercer kernels as

$$\hat{y}(\boldsymbol{v}) = \langle\boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{v})\rangle + b = \sum_{i=1}^{N}\eta_i K(\boldsymbol{v}_i, \boldsymbol{v}) + b, \tag{4.79}$$

where only the support vectors (i.e., training examples whose corresponding Lagrange multipliers are nonzero) contribute to the solution.

4.7 Tutorials and Application Examples

In this section, we first present some simple examples of the concepts developed with kernels so far. Then, we present a set of real examples with several data bases focusing on SVR performance and the impact of the cost function.

4.7.1 Kernel Calculations and Kernel Matrices

In these first examples we will see how to compute some of the kernel matrices in Table 4.1. We will assume we have a matrix of vector samples for training, $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, where n is the number of vectors and d its dimensionality (or number of features), and a matrix of vector samples for testing, $\boldsymbol{X}_* \in \mathbb{R}^{m\times d}$, where m is the number of samples in the test set. The first two kernels in the table can be straightforwardly computed in Listing 4.1.

```
% Assuming we have Xt (training) and Xv (test)
Kt = Xt * Xt'; % Linear kernel for training
Kv = Xv * Xt'; % Linear kernel for validation/test
% Polynomial kernel
```

```
Kt = (Xt * Xt' + c).^d; % c: bias, d: polynomial degree
Kv = (Xv * Xt' + c).^d;
```

Listing 4.1 Linear and polynomial kernels (kernels1.m).

Kernel matrices must be positive definite to be valid. In MATLAB, there are several ways to test if a matrix is positive definite. In a positive-definite matrix all eigenvalues are positive; therefore, we can use `eig` to obtain all eigenvalues and check if all of them are positive (see Listing 4.2).

```
% Eigenvectors of Kt
e = eig(Kt);
if any(e < 0),
    error('Matrix is not P.D.')
end
```

Listing 4.2 Checking positive definiteness via eigenvalues (kernels2.m).

However, the preferred way to test for positive definiteness is using `chol`, as in Listing 4.3.

```
% Test is matris is P.D. using chol
[R,p] = chol(Kt);
if p > 0,
    error('Matrix is not P.D.')
end
```

Listing 4.3 Checking positive definiteness using `chol` (kernels3.m).

Note that sometimes this test can fail due to numerical precision. In those cases, the eigenvalues may have a very small negative value, but different from zero; therefore, the test is not passed. If you have the theoretical guarantee that your kernel function is positive definite, a way to work around numerical issues is to add a small value to the diagonal; for instance, like `Kt + 2*eps*eye(size(Kt))`.

In many cases it is very useful to inspect the kernel matrix to visualize structures or groups. Kernels measure the similarity between vectors in the feature space, and a visual representation of the kernel should reveal some structure. In the following example, in Listing 4.4, we will generate three random 2D sets, compute a basic linear kernel, and see its structure. Figure 4.5 shows the result of the example. It is clear that the kernel reveals the relationship between different groups.

```
np   = 200;
dist = 2.5;
m1   = [-dist ; dist];
m2   = [0 ; 0];
m3   = [2*dist ; 2*dist];
% Generate samples
X1 = randn(2,np) + repmat(m1,1,np);
X2 = randn(2,np) + repmat(m2,1,np);
X3 = randn(2,np) + repmat(m3,1,np);
% All samples matrix
X = [X1 X2 X3]';
% Normalization
Xn = zscore(X);
% Look at the generated examples
```

```
figure(1)
plot(Xn(1:np,1), Xn(1:np,2), 'xb', ...
      Xn((1:np)+1*np,1), Xn((1:np)+1*np,2), 'xr', ...
      Xn((1:np)+2*np,1), Xn((1:np)+2*np,2), 'xk')
% Linear kernel
K = Xn * Xn';
% Inspecting the kernel
figure(2), imagesc(K), axis square off
```

Listing 4.4 Generating random sets and inspecting the linear kernel (kernels4.m).

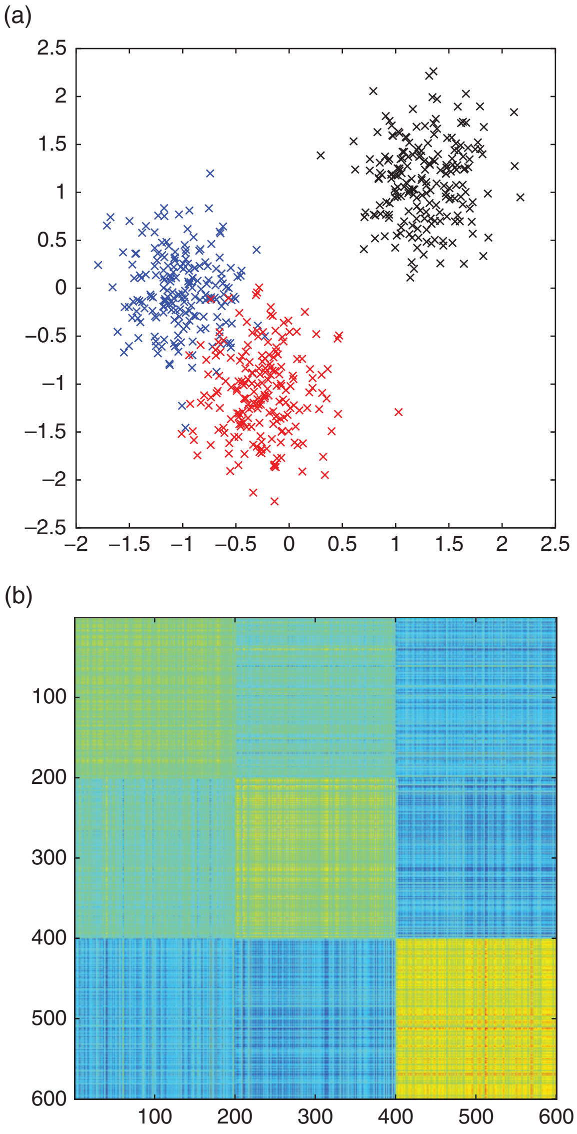

Figure 4.5 Example of kernel matrix: (a) the three sets; (b) generated linear kernel.

One the of most used kernels is the RBF or Gaussian kernel, also known as the squared exponential kernel, due to its simplicity (it only has one free parameter to adjust), and because it is a universal characteristic kernel that includes other kernels as particular cases. By exploiting the decomposition of the Euclidean distance we can efficiently compute this kernel, as seen in Listing 4.5.

```
% For the train kernel we have
nt = size(Xt,2);
ntsq = sum(Xt.^2,1);
Dt = 2 * (ntsq * ones(nt,1) - Xt' * Xt);
Kt = exp(-Dt / (2*sigma^2)); % sigma is the width of the RBF
% The test kernel can be computed as
nv = size(Xv,2);
nvsq = sum(Xv.^2,1);
Dv = ntsq' * ones(1,nv) + ones(nt,1) * nvsq - 2 * Xt' * Xv;
Kv = exp(-Dv / (2*sigma^2));
```

Listing 4.5 Computation of the RBF kernel (kernels5.m).

The exponential and Laplacian kernels can be obtained as in the RBF kernel just by taking the root of the `Dt` or `Dv` matrices in Listing 4.5. Similarly, it is easy to obtain the rational, multi-quadratic, power, log, Cauchy, and generalized T-Student kernels from these matrices. On the other hand, the chi-square and histogram intersection kernels are usually employed in computer vision, and they are particularly suitable for data in the form of normalized histograms. In general, given two vectors $\boldsymbol{x}$ and $\boldsymbol{z}$ of dimension d, we need to compute d kernels, one kernel per feature, and sum them all. For the chi-square kernel (other kernels are obtained similarly) a naive MATLAB implementation would be as given in Listing 4.6.

```
% Chi-Square kernel
d = size(X1,2); n1 = size(X1,1); n2 = size(X2,1);
K = 0;
for i = 1:d
    num = 2 * X1(:,i) * X2(:,i)';
    den = X1(:,i) * ones(1,n2) + ones(n1,1) * X2(:,i)';
    K = K + num./den;
end
```

Listing 4.6 Chi-square kernel (kernels6.m).

In the example in Listing 4.6, as the reader may notice, we have not computed the chi-square kernel as defined in Table 4.1. The reason, as pointed out by Vedaldi and Zisserman (2010), is that this kernel is only conditionally positive definite. It can be obtained as

$$K(\boldsymbol{x}, \boldsymbol{z}) = \sum_{i=1}^{d} \frac{2(x_i - z_i)^2}{(x_i + z_i)},$$

which is the kernel we calculate with Listing 4.6.

4.7.2 Basic Operations with Kernels

Let us start this subsection with a fundamental property in many kernel machines, which is the data centering in Hilbert space. This will need the property of translation of vectors. The example in Listing 4.7 shows how to center data, although it can easily modified to move the kernel to any other place in the Hilbert space.

```
% Assuming a pre-computed kernel matrix in Kt, centering is done as:
[Ni,Nj] = size(Kt);
Kt = Kt - ( mean(Kt,2)*ones(1,Nj) - ones(Ni,1)*mean(Kt,1) + mean(Kt(:)) );
% Sum columns and divide by the number of rows
S = sum(Kt) / Ni;
% Centered test kernel w.r.t. a train kernel
Kv = Kv - ( S' * ones(1,Nj) - ones(Ni,1) / Ni * sum(Kv) + sum(S) / Ni );
```

Listing 4.7 Centering kernels (kernels7.m).

Another important property of kernel methods is that one can estimate linear projections onto subspaces of the Hilbert feature space $\mathcal{H}$ explicitly. We are given a data matrix $\boldsymbol{X} \in \mathbb{R}^{N\times d} \subset \mathcal{X}$ which is mapped to an RKHS, $\boldsymbol{\phi} \in \mathbb{R}^{N\times d_{\mathcal{H}}} \subset \mathcal{H}$. Now we want to project the data onto a D-dimensional subspace of $\mathcal{H}$, where $D < d_{\mathcal{H}}$ given by a projection matrix $\boldsymbol{V} \in \mathbb{R}^{N\times N}$.

In order to map new input data $\boldsymbol{X}' \in \mathbb{R}^{m\times d}$ into a subspace of dimensionality D in $\mathcal{H}$, one has to first map the data to the Hilbert space $\boldsymbol{\Phi}'$, and then express the projection matrix $\boldsymbol{V}$ as a linear combination of the n examples therein, $\boldsymbol{V} = \boldsymbol{\Phi}^{\mathrm{T}}\mathcal{A}$:

$$\mathcal{P}_D(\boldsymbol{X}') = \boldsymbol{\Phi}\boldsymbol{V} = \boldsymbol{\Phi}'\boldsymbol{\Phi}^{\mathrm{T}}\mathcal{A} = \boldsymbol{K}\mathcal{A}, \tag{4.80}$$

where $\mathcal{A}, \boldsymbol{K} \in \mathbb{R}^{N\times N}$, $\boldsymbol{K}_{ij} := \langle \boldsymbol{\phi}(\boldsymbol{x}'_i), \boldsymbol{\phi}(\boldsymbol{x}_j)\rangle$, and hence projecting in a finite dimension D can be easily done by retaining (truncating) a number D of columns of $\mathcal{A}$.

This can be illustrated in Listing 4.8. Let us for a moment assume that the eigenvectors of the kernel matrix are the ones spanning the subspace we are interested in. The operation essentially reduces to an eigendecomposition of an $N \times N$ matrix, truncation of the eigenvectors matrix, and linear projection. This property and the latter application are widely used in kernel feature extraction methods, such as the KPCA, which will be revised thoroughly in Chapter 12.

```
K  = kernelmatrix('rbf',X,X,sigma);  % compute kernel matrix n x n
K2 = kernelmatrix('rbf',X2,X,sigma); % compute kernel matrix n x m
n = size(K,2);       % n: number of samples used in the kernel matrix
D = 10;              % subspace dimensionality (D<n)
[A L] = eigs(K,n);   % extract the top D eigenvectors of the kernel matrix
P_X2 = K2*A;         % projection of X2 onto the subspace of size m x n
P_X2 = K2*A(:,1:D);  % projection of X2 onto the subspace of size m x D
```

Listing 4.8 Projections with kernels (kernels8.m).

Very often we aim to estimate distances in the Hilbert space explicitly. Given two data matrices **X** and **Y**, one can compute the distances between the data points implicitly via kernels. Essentially, one is interested in mapping the data to a Hilbert space $\mathcal{H}$ which yields $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_y$, and estimating the (squared) Euclidean distance therein:

$$d^2_{xy} := \|\boldsymbol{\Phi}_x - \boldsymbol{\Phi}_y\|^2_{\mathcal{H}} = \boldsymbol{K}_x + \boldsymbol{K}_y - 2\boldsymbol{K}_{xy}, \tag{4.81}$$

which can be done in MATLAB with the following code: `D2_H = Kx+Ky-2*Kxy;`. It is worth noting that if we use an RBF kernel function, self-similarities become one, $K(\boldsymbol{x}_i, \boldsymbol{x}_i) = 1$, and then all points are mapped to a hypersphere and hypersphere and computing distances reduces to simply `D2_H = 2*(1-Kxy)`.

Another useful operation when working with kernels is about computing distances to the empirical center of mass. As an extension of the previous exercise, the reader may consider computing the distance from the given data points to its (empirical) center of mass in feature spaces. This would require computing the empirical mean in Hilbert space, but we can do this implicitly with kernels, as in Listing 4.9. Related to this example, one may think of normalizing the energy of the data points in Hilbert spaces. This operation of normalization is also possible via kernel functions only. Given a kernel matrix $\boldsymbol{K}$, it is trivial to compute its normalized version as in Listing 4.10.

```
n = size(K,1);
D = sum(K) / ell;
E = sum(D) / ell;
D2 = diag(K) - 2 * D' + E * ones(n,1);
```

Listing 4.9 Computing distances to the empirical center of mass (kernels9.m).

```
D = diag(1./sqrt(diag(K)));
Kn = D * K * D;
```

Listing 4.10 Kernel normalization (kernels10.m).

Note that in all these operations, only linear algebra operations, such as matrix multiplications or SVDs, are typically involved.

A quite useful operation with kernels is to measure how well *aligned* two kernel matrices are, either to discard redundant kernel parameters or to optimize the (in)dependence among them. Imagine that you are given a supervised problem like the "two moons" shown in Figure 4.6 in which you need to classify samples belonging to one of the two classes colored in blue or red. Building a proper kernel matrix requires

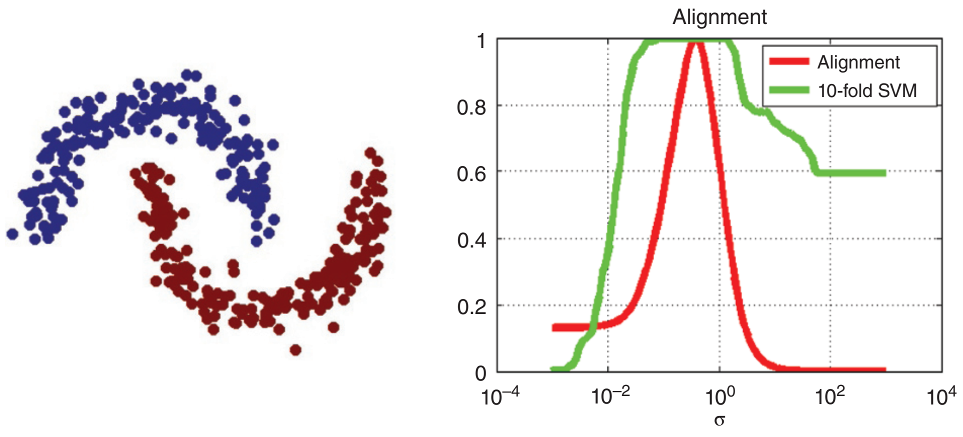

Figure 4.6 Illustration of the operation of alignment for the selection of hyperparameters.

typically tuning parameters, which is commonly done via cross-validation. A first guess of the proper parameter can be estimated via *kernel alignment* in a very cheap way without training any classifier, just by estimating the kernel hyperparameter that aligns best the mapped samples with the kernel matrix $\boldsymbol{K}_x(\boldsymbol{\theta})$ and the labels $\boldsymbol{y}$ (through the so-called *ideal kernel* $\boldsymbol{K}_{\text{ideal}} = \boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}$) in feature space. The problem then boils down to estimating the alignment measure

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{K} - \boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}\|_{\mathrm{F}}^2, \tag{4.82}$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. The (normalized) alignment measure for a particular hyperparameter can be easily computed as in Listing 4.11. Figure 4.6 shows the results obtained when using an RBF kernel parametrized with the length-scale σ parameter, comparing the results with an expensive training of an SVM by grid search of the hyperparameters.

```
temp = F_norm(K,Y);
A = temp/sqrt(F_norm(K,K)*F_norm(Y,Y));

function F = F_norm(A,B)
F = trace(A'*B);
```

Listing 4.11 Normalized alignment measure (kernels11.m).

4.7.3 Constructing Kernels

In this section we illustrate two examples of kernel construction, as a follow-up of Section 4.4.1. In particular, let us discuss the standard convex kernel combinations. Commonly used kernels families include the SE, periodic (Per), linear (Lin), and rational quadratic (RQ); see Table 4.1. We illustrate these basic kernels and some functions drawings in Figure 8.9. These basic kernels can be actually combined by following simple operations, such as summation, multiplication, or convolution. This way, one may build sophisticated kernels from simpler ones. The code in Listing 4.12 illustrates the construction of such kernels and several convex combinations.

```
%% Data
X_o = [-1.5 -1 -0.75 -0.4 -0.3 0]';
Y_o = [-1.6 -1.3 -0.5 0 0.3 0.6]';
%% Define a SE kernel
nu = 1.15; % scale of the kernel
s  = 0.30; % lengthscale sigma parameter of the kernel
kf = @(x,x2) nu^2 * exp( (x-x2).^2  /(-1*s^2) );
sn = 0.01; % known noise on observed data
ef = @(x,x2) sn^2 * (x==x2); % iid noise
% Let's sum the signal and noise kernel functions:
k = @(x,x2) kf(x,x2) + ef(x,x2);
% Let's plot the kernel function of domain x
np = 100; x = linspace(-1,1,np)';
y = kf(zeros(length(x),1),x);
figure(1), plot(x,y,'k','LineWidth',8), axis([-1 1 0 1.5]), axis off
% Now let's draw two random functions out of the kernel
K = zeros(length(x));
```

```
for i = 1:length(x)
    K(i,:) = kf(x(i) * ones(length(x),1), x);
end
% Due to numerical precision, we need to add a small factor to the ...
    diagonal
% to ensure positive definiteness
sphi = chol(K + 1e-12 * eye(size(K)))';
rf = sphi * randn(length(x),2);
figure(2), plot(x,rf(:,1)+1,x,rf(:,2)-1,'LineWidth',8), axis off

%% Done!

% As a proposed exercise, try to do the same with the
% rational-quadratic kernel, the periodic kernel and combinations.
% We give you some hint material:

% 1) Rational-quadratic kernel,
c = 0.05;
krq = @(x1,x2) sigma_f^2 * (1 - (x1-x2).^2 ./ ((x1-x2).^2 + c)) + ...
    ef(x1,x2);
% 2) Periodic kernel
p = 0.4; % periode
s = 1;   % SE kernel lengthscale
kper = @(x1,x2) sigma_f^2 * exp(-sin(pi*(x1-x2)/p).^2/l^2) + ef(x1,x2);
% 3) Linear kernel
offset = 0;
kl = @(x1,x2) sigma_f^2 * (x1 - offset) .* (x2 - offset) + 1e-5*ef(x1,x2);
```

Listing 4.12 Examples of kernel construction (kernels12.m).

In Figure 8.9 from Chapter 8, all the base kernels are 1D. Nevertheless, kernels over multidimensional inputs can actually be constructed by adding and multiplying kernels over individual dimensions; namely, (a) linear, (b) SEl (or RBF), (c) rational quadratic, and (d) periodic. See Table 4.1 for the explicit functional form of each kernel. Some simple kernel combinations are represented in the two last panels of Figure 8.9. For instance, a linear plus periodic covariances may capture structures that are periodic with trend (e), while a linear plus SE covariances can accommodate structures with increasing variation (f). By summing kernels, we can model the data as a superposition of independent functions, possibly representing different structures in the data. For example, in multitemporal image analysis, one could, for instance, dedicate a kernel for the time domain (perhaps trying to capture trends and seasonal effects) and another kernel function for the spatial domain (equivalently capturing spatial patterns and autocorrelations).

In time-series models, sums of kernels can express superposition of different processes, possibly operating at different scales. Very often, changes in geophysical variables through time occur at different temporal resolutions (hours, days, etc.), and this can be incorporated in the prior covariance with those simple operations. In multiple dimensions, summing kernels gives additive structure over different dimensions, similar to generalized additive models (Hastie *et al.*, 2009). Alternatively, multiplying kernels allows us to account for interactions between different input dimensions or different notions of similarity. In the following subsections, we also show how to design kernels that incorporate particular time resolutions, trends, and periodicities.

4.7.4 Complex Kernels

This example shows a nonlinear complex channel equalization with a linear filter and a kernel filter. A QPSK data burst $d_n = d_{r,n} + \mathbb{i} d_{i,n}$, where $d_{r,n}, d_{i,n} \in \{-1, 1\}$ is sent over a channel with a linear dispersive section of impulse response $h_n = \delta_n + (0.5 - 0.5\mathbb{i})\delta_{n-1} + (0.1 + 0.1\mathbb{i})\delta_{n-2}$, and a nonlinear section at its output with the function $x = f(u) = (1/3)u + u^{-1/3}$, where u is the linear channel output. The nonlinear channel output has a complex additive white Gaussian noise of standard deviation $\sigma_n = 0.05$. The linear transversal equalizer has the expression

$$y = \boldsymbol{w}\boldsymbol{x}_n^{\mathrm{H}}, \tag{4.83}$$

where $\boldsymbol{x}_n = (x_n \cdots x_{n-L+1})$, H is the complex transpose (Hermitian) operator, and L is the filter length.

The filter weights are adjusted using a ridge regression procedure, where regularization parameter λ is adjusted to the channel noise (i.e., $\lambda = \sigma_n^2$), and thus the filter is intended to minimize the cost function

$$L = \mathbb{E}[\|d_n - \boldsymbol{w}\boldsymbol{x}_n^{\mathrm{H}}\|] + \sigma_n^2\|\boldsymbol{w}\|^2. \tag{4.84}$$

The optimization is performed simply by nulling the gradient of the cost function with respect to the parameter vector, using the Wirtinger derivative. The expression of the optimal filter is simply

$$\boldsymbol{w} = \boldsymbol{d}\boldsymbol{X}(\boldsymbol{X}^{\mathrm{H}}\boldsymbol{X} + \sigma_n^2\boldsymbol{I})^{-1}, \tag{4.85}$$

where $\boldsymbol{X}$ is the matrix containing all training vectors $\boldsymbol{x}_n$, and $\boldsymbol{d}$ is a row vector containing the transmitted signals, assumed to be known by the receiver for training purposes.

The channel output data can be transformed using a nonlinear transformation into an RKHS, and then the expression for the weight vector is

$$\boldsymbol{w} = \boldsymbol{d}\boldsymbol{\Phi}(\boldsymbol{\Phi}^{\mathrm{H}}\boldsymbol{\Phi} + \sigma_n^2 I)^{-1}, \tag{4.86}$$

where $\boldsymbol{\Phi}$ is a matrix containing all row vectors $\boldsymbol{\phi}(\boldsymbol{x}_n)$, and $\boldsymbol{\phi}(\cdot)$ is a mapping into a given RKHS.

A kernelized version of this filter can be constructed using the complex version of the representer theorem (Equation 4.58), given by

$$\boldsymbol{w} = \sum_{i=1}^{N}(\alpha_i + \mathbb{i}\beta_i)K(\cdot, \boldsymbol{x}_i) = \sum_{i=1}^{N}(\alpha_i + \mathbb{i}\beta_i)\boldsymbol{\phi}(\boldsymbol{x}_i) = (\boldsymbol{\alpha} + \mathbb{i}\boldsymbol{\beta})\boldsymbol{\Phi}, \tag{4.87}$$

where $\boldsymbol{\alpha} + \mathbb{i}\boldsymbol{\beta}$ is a row vector containing all complex coefficients $\alpha_i + \mathbb{i}\beta_i$.

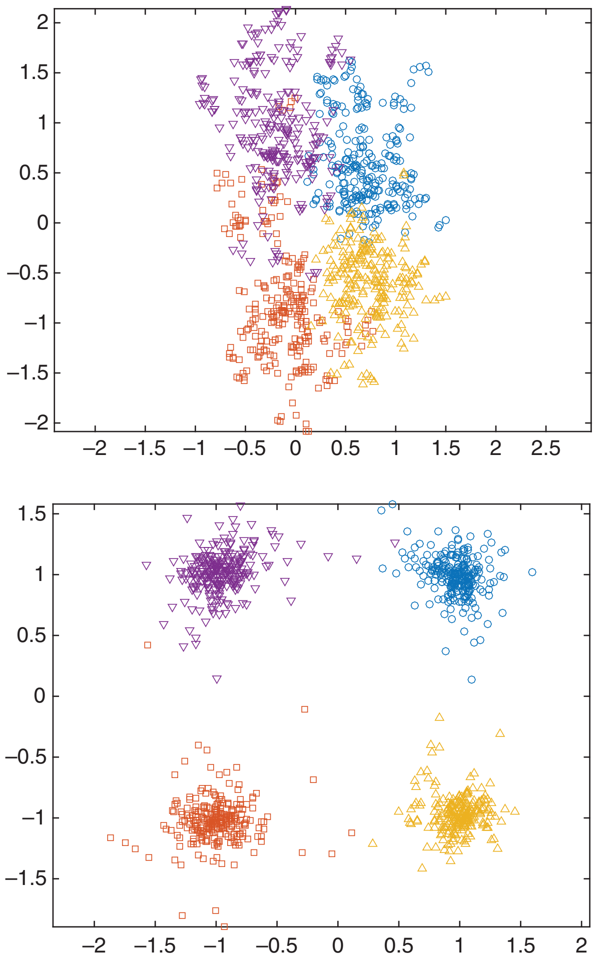

Figure 4.7 Estimated symbols from a burst of 1000 training data. Upper panel: the linear equalizer estimation; lower panel: the complex kernel equalizer estimation.

The combination of Equations 4.86 and 4.87 gives the result

$$\boldsymbol{\alpha} + \mathbb{i}\boldsymbol{\beta} = \boldsymbol{d}(\boldsymbol{\Phi}\boldsymbol{\Phi}^H + \sigma_n^2\boldsymbol{I})^{-1}. \tag{4.88}$$

The complex kernel estimator is then expressed as

$$y_n = \sum_{i=1}^{N}(\alpha_i + \mathbb{i}\beta_i)K(\boldsymbol{x}_i, \boldsymbol{x}_n). \tag{4.89}$$

In our example, a Gaussian kernel is used. The upper panel of Figure 4.7 shows the representation of the estimated complex symbols of a test burst after training a linear equalizer with Equation 4.85, and the lower panel the symbols estimated with the equalizer in Equation 4.89 using the optimization in Equation 4.88. As can be shown, the linear equalizer produces a poor estimation due to the nonlinear behavior of the channel. Nevertheless, the complex kernel estimator can model and invert this nonlinear behavior. Listing 4.13 has been used to obtain these results.

```
function kernels13
sn = 0.05;      % Noise std
L = 4;          % Filter length
p = sqrt(2)*2; % Kernel parameter
N = 1000;
h = [1 0.5-1j*0.5 .1+1j*.1]; % Linear channel impulse response

% Data
d = sign(randn(1,N)) + 1j*sign(randn(1,N));
d_test = sign(randn(1,N)) + 1j*sign(randn(1,N));

% Channel
x = channel(d,h,sn);
x_test = channel(d_test,h,sn);

% Data matrix
X = buffer(x,L,L-1)';
X_test = buffer(x_test,L,L-1)';

% Equalization with Linear MMSE
R = X' * X;
R = R + sn^2*eye(size(R));
w = d * X / R;
y = w * X_test';
figure(1), plotresults(y,d_test), title('Linear MMSE eq.')

% Equalization with Kernel MMSE
K = kernel_matrix(X,X,p);
alpha = d * pinv(K + sn^2*eye(size(K)));
K_test = kernel_matrix(X_test,X,p);
y = alpha * K_test;
figure(2), plotresults(y,d_test), title('Kernel MMSE eq.')

function x = channel(d,h,s)
% Channel: Linear section
xl = filter(h,1,d);
% Channel: Nonlinear section
xnl = xl.^(1/3) + xl / 3;
% Noise
x = xnl + s*(randn(size(xl)) + 1j*randn(size(xl)));

function K = kernel_matrix(X1,X2,p)
N1 = size(X1,1);
N2 = size(X2,1);
aux = kron(X1,ones(N2,1)) - kron(ones(1,N1),X2')';
D = buffer(sum(aux.*conj(aux),2),N2,0);
K = exp(-D/(2*p));

function plotresults(y,d)
plot(real(y(d == 1+1j)),  imag(y(d == 1+1j)),  'bo'), hold on
plot(real(y(d == -1-1j)), imag(y(d == -1-1j)), 'rs')
plot(real(y(d == 1-1j)),  imag(y(d == 1-1j)),  'm^')
plot(real(y(d == -1+1j)), imag(y(d == -1+1j)), 'kv'), hold off
axis equal, grid on
```

Listing 4.13 Complex kernel (kernels13.m).

4.7.5 Application Example for Support Vector Regression Elements

In this subsection, three types of loss-based SVRs are benchmarked: (1) the standard ε-insensitive SVR (ε-SVR); (2) the squared-loss SVR (also known as kernel ridge regression or LS-SVM); and (3) the ε-Huber-SVR. We have chosen a real regression example that involves different noise sources for the sake of illustration purposes. Results are obtained with NNs and with classical empirical models. These models are compared in terms of accuracy and bias of the estimations, and also their robustness is scrutinized when a low number of training samples are available in two different data sets related to the estimation of oceanic chlorophyll concentration from satellite data. More details can be found in Camps-Valls *et al.* (2006c).

Data Description

We use here two different datasets for illustration purposes. The first dataset simulates the data acquired by the medium-resolution imaging spectrometer (MERIS) on-board at the Envisat satellite (MERIS dataset), and in particular the spectral behavior of chlorophyll concentration C in the subsurface waters (Cipollini *et al.*, 2001). The data were generated according to a fixed and noise-free model, and the total number of samples (pairs of in-situ concentrations and acquired radiances) available for our experiments was 5000. These samples were randomly divided into three sets: a training set (500 samples, model building), a validation set (500 samples, free parameters tuning), and a test set (4000 samples).

The second dataset (SeaBAM dataset, (O'Reilly and Maritorena, 1997; O'Reilly *et al.*, 1998)) consists of a compilation of 919 in-situ measurements of chlorophyll concentration around the USA and Europe, all of them related to five different multispectral remote-sensing reflectance corresponding to the SeaWiFS wavelengths. The available data were randomly split into three sets: 230 samples for training, 230 samples for validation, and the remaining 459 samples for testing the performance. In both cases, we transformed the concentration data logarithmically, $y = \log(C)$, according to Cipollini *et al.* (2001). Hereafter, units of all accuracy and bias measurements are referred to y [$\log(\text{mg}/\text{m}^3)$] instead of C [mg/m^3].

Model Accuracy and Comparison

Table 4.2 presents results in the test set for all the SVR models and datasets. Results are compared with a feedforward NN trained with backpropagation (NN-BP) for both datasets. In the case of the SeaBAM dataset, we also include results with the models Morel-1, Morel-3, and CalCOFI 2-band (cubic and linear), as they performed best among a set of 15 *empirical* estimation models of the chlorophyll-a concentration in O'Reilly *et al.* (1998), and results from the ε-SVR obtained in Zhan *et al.* (2003). Several prediction error measures are considered, including mean error (ME), root mean-square error (RMSE), and mean absolute error (ABSE), as well as the correlation

Table 4.2 Results in the test set in the two datasets.

	ME	RMSE	ABSE	r
MERIS dataset				
NN-BP, 4 hidden nodes	-7.68×10^{-4}	0.024	0.016	0.998
ε-SVR	-2.36×10^{-4}	0.015	0.061	0.998
L_2-loss SVR	-9.96×10^{-4}	0.031	0.018	0.998
Proposed ε-Huber-SVR	-3.26×10^{-6}	0.011	0.004	0.999
SeaBAM dataset				
NN-BP, 4 hidden nodes	−0.046	0.143	0.111	0.971
ε-SVR in Zhan *et al.* (2003)	—	0.138	—	0.973
ε-SVR	−0.070	0.139	0.105	0.971
L_2-loss SVR	−0.034	0.140	0.107	0.971
ε-Huber-SVR	−0.020	0.137	0.104	0.972

coefficient between the desired output and the output offered by the models as a measure of fit. Source code for running the experiments can be found in this book's repository.

Table 4.2 shows that the kernel technique is more accurate (RMSE, ABSE) and unbiased (ME). For the MERIS data, significant numerical (see Table 4.2) and statistical differences between ε-Huber-SVR and the rest of the models were observed for both bias ($F = 2.46$, $p < 0.05$) and accuracy ($F = 767.8$, $p < 0.001$). For the SeaBAM dataset, the kernel method improved the performance, but no statistical differences were observed in bias ($F = 0.08$, $p = 0.92$) or accuracy ($F = 0.04$, $p = 9.86$). These results are similar to those previously reported in Zhan *et al.* (2003), but we were able to use half of the training samples, a particularly interesting situation when working with few in-situ measures usually available in biophysical problems.

Model Flexibility and Uncertainty

In order to model the accuracy demonstrated in the previous section, the ε-Huber cost function provides some relevant information about the reliability of the available data, which is extremely important when dealing with uncertainty. Figure 4.8 shows the histograms for residuals obtained with the ε-Huber-SVR model. If no uncertainty is present in data (MERIS data, synthetic data generated according to a physical model), we can see that the δC parameter covers a wide range of residuals and mainly penalizes them with a squared loss function (assuming Gaussian noise), and that the residual tails are linearly penalized (Cipollini *et al.*, 2001).

Model Robustness to Reduced Datasets

The number of the available in-situ measurements available is often scarce. In this experiment, we test the capabilities of the models to deal with low-sized datasets.

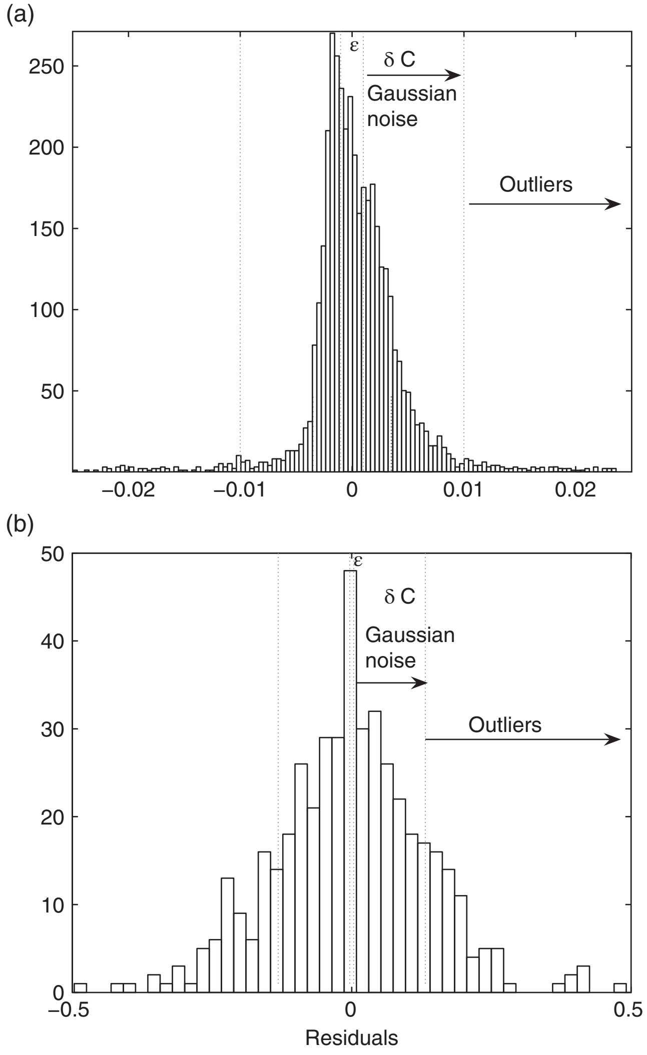

Figure 4.8 Histograms of residuals obtained with the ε-Huber-SVR model for (a) MERIS and (b) SeaBAM datasets.

Figure 4.9 shows the behavior of the RMSE versus the number of training samples in the SeaBAM data. We show the average value for 100 random different trials. The gain of the ε-SVR algorithm is noticeable, obtaining an average improvement of 8.5–13% in RMSE with respect to the other models. This effect is especially significant when working with very reduced training sets, as seen in the average gain of 16.67% in RMSE compared with ε-SVR when using only 10 training samples. We can conclude that, when the number of available samples is limited (low to moderate) and samples are affected by uncertainty factors, the versatility to accommodate different noise models to the available data makes ε-Huber-SVR an efficient and robust model.

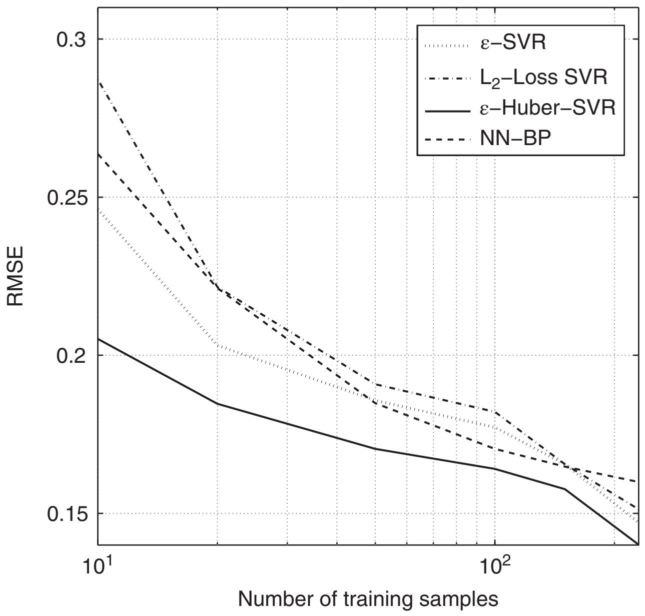

Figure 4.9 RMSE in the test SeaBAM dataset as a function of the number of training samples.

4.8 Concluding Remarks

In this chapter we introduced the most basic concepts of Mercer's kernels. The concepts of kernel function, feature mapping, positive definiteness and reproducing kernel Hilbert spaces were introduced as the core concepts that will accompany us throughout the remainder of the book.

The main idea in kernel methods is that positive definite kernels provide a measure of similarity between possibly complex objects. With the regularized risk framework one can implement the search over rich classes of functions and still obtain functions that can be expressed in finitely many terms of kernel evaluations. Another benefit is that this search can be made convex and thus yield problems which can be solved efficiently. Kernel methods generalize linear algorithms and allows us to develop nonlinear counterparts easily and still requiring only linear algebra. This is obviously of great practical convenience since linear algebra is perhaps one of the most solid and well-established part of mathematics.

The use of kernel methods in machine learning has expanded in the last decades, and they now constitute a preferred toolbox for practitioners in many disciplines, and in signal processing in particular. In the following chapters we will build upon the concepts provided here, and will subsequently introduce more advanced concepts to develop new kernel methods that tackle specific signal processing problems.

4.9 Questions and Problems

Exercise 4.9.1 Reproduce Example 4.2.6 for the case of a discrete finite FT.

Exercise 4.9.2 Consider a nonnegative function $F(\omega)$. Prove that the inverse FT of this function defines a shift-invariant kernel in a given RKHS of the form $K(\delta t)$ with $\delta t = t - t'$.

Exercise 4.9.3 The technique that optimizes parameters $\boldsymbol{w}$ of a linear estimator $\hat{y} = f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{a}$ by minimizing the regularized functional

$$\mathcal{L} = \sum_{i=1}^{N} \|y_i - f(\boldsymbol{x}_i)\|^2 + \lambda \sum_{j=0}^{M} a_j^2 \tag{4.90}$$

is known as ridge regression (Hoerl and Kennard, 1970). Derive the particular ridge regression algorithm for the example in Figure 4.3. In order to derive the algorithm, its parameters must be identified as the coefficients of the model; that is, $\boldsymbol{x} := [1, x, x^2, \dots, x^6]^{\mathrm{T}}$.

Exercise 4.9.4 Define $\mathcal{A}^2$ as the set of all complex-valued functions that are analytic inside the unit disk $D = \{\boldsymbol{x} : \|\boldsymbol{z}\| < 1, \boldsymbol{x} \in \mathbb{C}\}$ of the complex plane, and square integrable. If an orthonormal basis of $\mathcal{A}^2$ is

$$\boldsymbol{\phi}(\boldsymbol{x}) = \sqrt{\frac{n+1}{\pi}}\boldsymbol{x}^n,$$

then prove that the kernel of $\mathcal{A}^2$ (Bergman kernel) (Halmos, 1974) is

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{1}{\pi}\frac{1}{(1 - \boldsymbol{x}_1\boldsymbol{x}_2^*)}.$$

Exercise 4.9.5 Express the polynomial model in the example in Figure 4.3 as a function of Mercer kernels. Identify the kernel and its corresponding nonlinear mapping according to Mercer's theorem. Rewrite the ridge regression algorithm in Exercise 4.9.3 in terms of kernel dot products.

Exercise 4.9.6 Show that the following is a kernel function:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}(\|\boldsymbol{x} + \boldsymbol{y}\|_K^2 + \|\boldsymbol{x} - \boldsymbol{y}\|_K^2),$$

for the norm defined by any kernel function K.

Exercise 4.9.7 Let us compute the difference in feature space between two mapped vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; that is, $\boldsymbol{\phi} = \boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{y})$. Compute the reproducing kernel function that acts solely on examples in input space. Show that this is a valid kernel function.

Exercise 4.9.8 Let us compute the pointwise division in feature space between two mapped vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; that is, $\boldsymbol{\phi} = \boldsymbol{\phi}(\boldsymbol{x})/\boldsymbol{\phi}(\boldsymbol{y})$. Compute the reproducing kernel function that acts solely on examples in input space. Show that this is a valid kernel function.

Exercise 4.9.9 Demonstrate the properties in Section 4.4.2.

Exercise 4.9.10 Demonstrate that a continuous kernel $K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{x} - \boldsymbol{y})$ on $\mathbb{R}^d$ is positive definite if and only if $K(\Delta)$ is the FT of a nonnegative measure. Using this theorem, also demonstrate that if a shift-invariant kernel $K(\Delta)$ is properly scaled, its FT $p(\boldsymbol{\omega})$ is a proper probability distribution.

Exercise 4.9.11 Build a simple script for performing SVR on a sinc function in terms of an RBF kernel for a nonregularly sampled time grid. Search for the optimal width of the RBF kernel. What can you conclude by observing the spectra of the observation signal, of the interpolated signal, and of the tuned RBF kernel? Plot the residuals in terms of the coefficients and discuss the related properties.