CHAPTER

LEARNING IN REPRODUCING KERNEL HILBERT SPACES

11

CHAPTER OUTLINE

11.1 INTRODUCTION

Our emphasis in this chapter will be on learning nonlinear models. The necessity of adopting nonlinear models has already been discussed in Chapter 3, in the context of the classification as well as the regression tasks. For example, recall that given two jointly distributed random vectors $(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^k \times \mathbb{R}^l$, then we know that the optimal estimate of $\mathbf{y}$ given $\mathbf{x} = \boldsymbol{x}$, in the mean-square error sense (MSE), is the corresponding conditional mean, that is, $\mathbb{E}[\mathbf{y}|\boldsymbol{x}]$, which in general is a nonlinear function of $\boldsymbol{x}$.

There are different ways of dealing with nonlinear modeling tasks. Our emphasis in this chapter will be on a path through the so-called reproducing kernel Hilbert spaces (RKHS). The technique consists of mapping the input variables to a new space, such that the originally nonlinear task is transformed into a linear one. From a practical point of view, the beauty behind these spaces is that their rich structure allows us to perform inner product operations in a very efficient way, with complexity independent of the dimensionality of the respective RKHS. Moreover, note that the dimension of such spaces can even be infinite.

We start the chapter by reviewing some more “traditional” techniques concerning Volterra series expansions, and then move slowly to explore the RKHS. Cover’s theorem, the basic properties of RKHS and their defining kernels are discussed. Kernel ridge regression and the support vector machine framework is presented. Then we move to online learning algorithms in RKHS and, finally, some more advanced concepts related to sparsity and multikernel representations are discussed. A case study in the context of text mining is presented at the end of the chapter.

11.2 GENERALIZED LINEAR MODELS

Given $(\mathrm{y}, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$, a generalized linear estimator $\hat{\mathrm{y}}$ of y has the form

$$\hat{\mathrm{y}} = f(\mathbf{x}) := \theta_0 + \sum_{k=1}^{K} \theta_k \phi_k(\mathbf{x}), \tag{11.1}$$

where $\phi_1(\cdot), \ldots, \phi_K(\cdot)$ are *preselected* (nonlinear) functions. A popular family of functions is the polynomial one, for example,

$$\hat{y} = \theta_0 + \sum_{i=1}^{l} \theta_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^{l} \theta_{im} x_i x_m + \sum_{i=1}^{l} \theta_{ii} x_i^2. \tag{11.2}$$

Assuming $l = 2$ ($\mathbf{x} = [x_1, x_2]^T$), then (11.2) can be brought into the form of (11.1) by setting $K = 5$ and $\phi_1(\mathbf{x}) = x_1$, $\phi_2(\mathbf{x}) = x_2$, $\phi_3(\mathbf{x}) = x_1 x_2$, $\phi_4(\mathbf{x}) = x_1^2$, $\phi_5(\mathbf{x}) = x_2^2$. The generalization of (11.2) to rth order polynomials is readily obtained and it will contain products of the form $x_1^{p_1} x_2^{p_2} \cdots x_l^{p_l}$, with $p_1 + p_2 + \cdots + p_l \leq r$. It turns out that the number of free parameters, K, for an rth order polynomial is equal to

$$K = \frac{(l+r)!}{r!l!}.$$

Just to get a feeling for $l = 10$ and $r = 3$, $K = 286$. The use of polynomial expansions is justified by the Weierstrass theorem, stating that every continuous function, defined on a compact (closed and bounded) subset $S \subset \mathbb{R}^l$, can be uniformly approximated as closely as desired, with an arbitrary small error, ϵ, by a polynomial function, for example, [97]. Of course, in order to achieve a good enough approximation, one may have to use a large value of r. Besides polynomial functions, other types of functions can also be used, such as splines and trigonometric functions.

A common characteristic of this type of models is that the basis functions in the expansion are *preselected* and they are fixed and independent of the data. The advantage of such a path is that the associated models are linear with respect to the unknown set of free parameters, and they can be estimated by following any one of the methods described for linear models, presented in Chapters 4–8. However, one has to pay a price for that. As it has been shown in [7], for an expansion involving K fixed functions, the squared approximation error *cannot* be made smaller than order $\left(\frac{1}{K}\right)^{\frac{2}{l}}$. In other words, for high-dimensional spaces and in order to get a small enough error, one has to use large values of K. This is another face of the curse of dimensionality problem. In contrast, one can get rid of the dependence of the approximation error on the input space dimensionality, l, if the expansion involves data-dependent functions, which are optimized with respect to the specific data set. This is, for example, the case for a class of neural networks, to be discussed in Chapter 18. In this case, the price one pays is that the dependence on the free parameters is now nonlinear, making the optimization with regard to the unknown parameters a harder task.

11.3 VOLTERRA, WIENER, AND HAMMERSTEIN MODELS

We now turn our focus to the case of modeling nonlinear systems, where the involved input-output entities are time series/discrete time signals denoted as (u_n, d_n), respectively. The counterpart of polynomial modeling in (11.2) is now known as the Volterra series expansion.

These types of models will not be pursued any more in this book, and they are briefly discussed here in order to put the nonlinear modeling task in a more general context as well as for historical reasons. *Thus, this section can be bypassed in a first reading.*

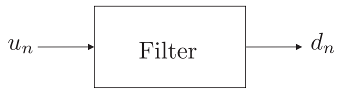

FIGURE 11.1

The nonlinear filter is excited by u_n and provides in its output d_n.

Volterra was an Italian mathematician (1860-1940) with major contributions also in physics and biology. One of his landmark theories is the development of Volterra series, which was used to solve integral and integro-differential equations. He was one of the Italian professors who refused to take an oath of loyalty to the fascist regime of Mussolini and he was obliged to resign from his university post.

Figure 11.1 shows an unknown nonlinear system/filter with the respective input-output signals. The output of a discrete time Volterra model can be written as

$$d_n = \sum_{k=1}^{r} \sum_{i_1=0}^{M} \sum_{i_2=0}^{M} \cdots \sum_{i_k=0}^{M} w_k(i_1, i_2, \ldots, i_k) \prod_{j=1}^{k} u_{n-i_j}, \tag{11.3}$$

where $w_k(\cdot, \ldots, \cdot)$ denotes the kth order *Volterra kernel*; in general, r can be infinite. For example, for $r = 2$ and $M = 1$, the input-output relation involves the linear combination of the terms

$$u_n, u_{n-1}, u_n^2, u_{n-1}^2, u_n u_{n-1}.$$

Special cases of the Volterra expansion are the *Wiener, Hammerstein, and Wiener-Hammerstein models*. These models are shown in Figure 11.2. The systems $h(\cdot)$ and $g(\cdot)$ are linear systems with memory, that is,

$$s_n = \sum_{i=0}^{M_1} h_n u_{n-i},$$

and

$$d_n = \sum_{i=0}^{M_2} g_n x_{n-i}.$$

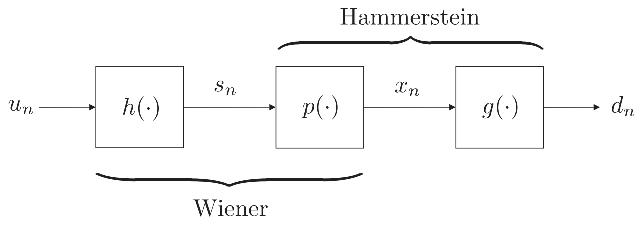

FIGURE 11.2

The Wiener model comprises a linear filter followed by a memoryless polynomial nonlinearity. The Hammerstein model consists of a memoryless nonlinearity followed by a linear filter. The Wiener-Hammerstein model is the combination of the two.

The central box corresponds to a memoryless nonlinear system, which can be approximated by a polynomial of degree r. Hence,

$$x_n = \sum_{k=1}^{r} c_k (s_n)^k.$$

In other words, a Wiener model is a linear time invariant system (LTI) followed by the memoryless nonlinearity and the Hammerstein model is the combination of a memoryless nonlinearity followed by an LTI system. The Wiener-Hammerstein model is the combination of the two. Note that each one of these models is nonlinear with respect to the involved free parameters. In contrast, the equivalent Volterra model is linear with regard to the involved parameters; however, the number of the resulting free parameters is significantly increased with the order of the polynomial and the filter memory taps (M_1 and M_2). The interesting feature is that the equivalent to a Hammerstein model Volterra expansion consists only of the diagonal elements of the associated Volterra kernels. In other words, the output is expressed in terms of $u_n, u_{n-1}, u_{n-2}, \dots$ and their powers; there are no cross-product terms [59].

Remarks 11.1.

- The Volterra series expansion was first introduced as a generalization of the Taylor series expansion. Following [102], assume a memoryless nonlinear system. Then, its input-output relationship is given by

  $$d(t) = f(u(t)),$$

  and adopting the Taylor expansion, for a particular time $t \in (-\infty, +\infty)$, we can write

  $$d(t) = \sum_{n=0}^{+\infty} c_n (u(t))^n, \tag{11.4}$$

  assuming that the series converges. The Volterra series is the extension of (11.4) to systems with memory and we can write

  $$\begin{aligned} d(t) = w_0 &+ \int_{-\infty}^{+\infty} w_1(\tau_1) u(t-\tau_1) d\tau_1 \\ &+ \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} w_2(\tau_1,\tau_2) u(t-\tau_1) u(t-\tau_2) d\tau_1 d\tau_2 \\ &+ \dots \end{aligned} \tag{11.5}$$

  In other words, the Volterra series is a power series with memory. The problem of convergence of the Volterra series is similar to that of the Taylor series. In analogy to the Weierstrass approximation theorem, it turns out that the output of a nonlinear system can be approximated arbitrarily close using a sufficient number of terms in the Volterra series expansion[1] [42].

  A major difficulty in the Volterra series is the computation of the Volterra kernels. Wiener was the first one to realize the potential of the Volterra series for nonlinear system modeling. In order to compute the involved Volterra kernels, he used the method of orthogonal functionals.
  The method resembles the method of using a set of orthogonal polynomials, when one tries to

[1] The proof involves the theory of continuous functionals. A functional is a mapping of a function to the real axis. Observe that each integral is a functional, for a particular t and kernel.

approximate a function via a polynomial expansion, [131]. More on the Volterra modeling and related models can be obtained in, for example, [56, 71, 103]. Volterra models have extensively been used in a number of applications, including communications (e.g., [11]), biomedical engineering (e.g., [73]), and automatic control (e.g., [31]).

11.4 COVER'S THEOREM: CAPACITY OF A SPACE IN LINEAR DICHOTOMIES

We have already justified the method of expanding an unknown nonlinear function in terms of a fixed set of nonlinear ones, by mobilizing arguments from the approximation theory. Although this framework fits perfectly to the regression task, where the output takes values in an interval in $\mathbb{R}$, such arguments are not well-suited for the classification. In the latter case, the output value is of a discrete nature. For example, in a binary classification task, $y \in \{1, -1\}$, and as long as the sign of the predicted value, $\hat{y}$, is correct, we do not care how close y and $\hat{y}$ are. In this section, we will present an elegant and powerful theorem that justifies the expansion of a classifier f in the form of (11.1). It suffices to look at (11.1) from a different angle.

Let us consider N points, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in \mathbb{R}^l$. We can say that these points are *in general position*, if *there is no* subset of $l+1$ of them lying on a $(l-1)$-dimensional hyperplane. For example, in the two-dimensional space, any three of these points are not permitted to lie on a straight line.

Theorem 11.1 (Cover's theorem)**.** *The number of groupings, denoted as $\mathcal{O}(N,l)$, that can be formed by $(l-1)$-dimensional hyperplanes to separate the N points in two classes, exploiting all possible combinations, is given by ([30], Problem 11.1),*

$$\mathcal{O}(N,l) = 2\sum_{i=0}^{l} \binom{N-1}{i},$$

where

$$\binom{N-1}{i} = \frac{(N-1)!}{(N-1-i)!i!}.$$

Each one of these groupings in two classes is also known as a (linear) *dichotomy*. Figure 11.3 illustrates the theorem for the case of $N=4$ points in the two-dimensional space. Observe that the possible groupings are [(ABCD)], [A,(BCD)], [B,(ACD)], [C,(ABD)], [D, (ABC)], [(AB), (CD)], and [(AC),(BD)]. Each grouping is counted twice, as it can belong to either ω_1 or ω_2 class. Hence, the total number of groupings is 14, which is equal to $\mathcal{O}(4,2)$. Note that the number of all possible combinations of N points in two groups is 2^N, which is 16 in our case. The grouping that is not counted in $\mathcal{O}(4,2)$, as it cannot be linearly separated, is [(BC),(AD)]. Note that if $N \le l+1$ then $\mathcal{O}(N,l) = 2^N$. That is, all possible combinations in groups of two are linearly separable; verify it for the case of $N=3$ in the two-dimensional space.

Based on the previous theorem, given N points in the l-dimensional space, the probability of grouping these points in two *linearly* separable classes is

$$\boxed{P_N^l = \frac{\mathcal{O}(N,l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^{l} \binom{N-1}{i}, & N > l+1, \\ 1, & N \le l+1. \end{cases}} \tag{11.6}$$

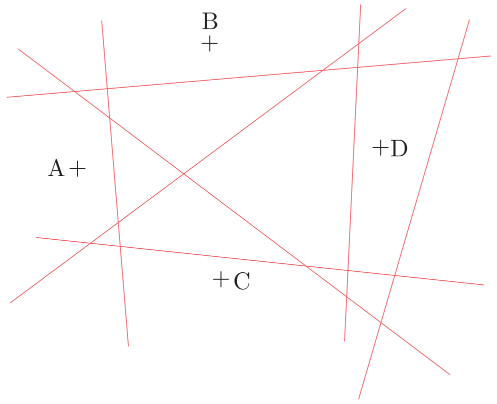

FIGURE 11.3

The possible number of linearly separable groupings of two, for four points in the two-dimensional space is $\mathcal{O}(4,2) = 14 = 2 \times 7$.

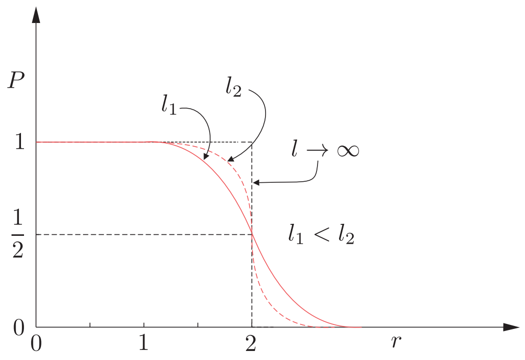

FIGURE 11.4

For $N > 2(l+1)$ the probability of linear separability becomes small. For large values of l, and provided $N < 2(l+1)$, the probability of any grouping of the data into two classes to be linearly separable tends to unity. Also, if $N \leq (l+1)$, all possible groupings in two classes are linearly separable.

To visualize this finding, let us write $N = r(l+1)$, and express the probability P_N^l in terms of r, for a fixed value of l. The resulting graph is shown in Figure 11.4. Observe that there are two distinct regions. One to the left of the point $r = 2$ and one to the right. At the point $r = 2$, that is, $N = 2(l+1)$, the probability is always $\frac{1}{2}$, because $\mathcal{O}(2l+2, l) = 2^{2l+1}$ (Problem 11.2). Note that the larger the value of l is the sharper the transition from one region to the other becomes. Thus, for large dimensional spaces and as long as $N < 2(l+1)$, the probability of any grouping of the points in two classes to be *linearly separable* tends to unity.

The way the previous theorem is exploited in practice is the following: Given N feature vectors $\boldsymbol{x}_n \in \mathbb{R}^l$, $n = 1, 2, \ldots, N$, a mapping

$$\boldsymbol{\phi} : \mathbb{R}^l \ni \boldsymbol{x}_n \longmapsto \boldsymbol{\phi}(\boldsymbol{x}_n) \in \mathbb{R}^K, \ K \gg l,$$

is performed. Then according to the theorem, the higher the value of K is the higher the probability becomes for the images of the mapping, $\boldsymbol{\phi}(\boldsymbol{x}_n) \in \mathbb{R}^K$, $n = 1, 2, \ldots, N$, to be linearly separable in the space $\mathbb{R}^K$. Note that the expansion of a nonlinear classifier (that predicts the label in a binary classification task) is equivalent with using a linear one on the images of the original points after the mapping. Indeed,

$$f(\boldsymbol{x}) = \sum_{k=1}^{K} \theta_k \phi_k(\boldsymbol{x}) + \theta_0 = \boldsymbol{\theta}^T \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}) \\ 1 \end{bmatrix}, \tag{11.7}$$

with

$$\boldsymbol{\phi}(\boldsymbol{x}) := [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_K(\boldsymbol{x})]^{\mathrm{T}}.$$

Provided that K is large enough, our task is *linearly separable* in the new space, $\mathbb{R}^K$ with high probability, which justifies the use of linear classifier, $\boldsymbol{\theta}$, in (11.7). The procedure is illustrated in Figure 11.5. The points in the two-dimensional space are not linearly separable. However, after the mapping in the three-dimensional space,

$$[x_1, x_2]^T \longmapsto \boldsymbol{\phi}(\boldsymbol{x}) = [x_1, x_2, f(x_1, x_2)]^T, \qquad f(x_1, x_2) = 4 \exp\big(-(x_1^2 + x_2^2)/3\big) + 5,$$

the points in the two classes become linearly separable. Note, however, that after the mapping, the points lie on the surface of a paraboloid. This surface is fully described in terms of two free variables. Loosely speaking, we can think of the two-dimensional plane, on which the data lie originally, to be folded/transformed to form the surface of the paraboloid. This is basically the idea behind the more

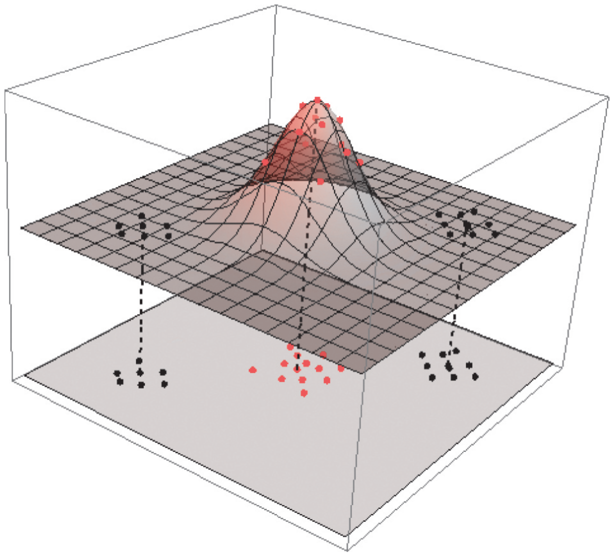

FIGURE 11.5

The points (red in one class and black for the other), that are not linearly separable in the original two-dimensional plane, become linearly separable after the nonlinear mapping in the three-dimensional space; one can draw a plane that separates the "black" from the "red" points.

general problem. After the mapping from the original l-dimensional space to the new K-dimensional one, the images of the points $\boldsymbol{\phi}(\boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$, lie on a l-dimensional surface (*manifold*) in $\mathbb{R}^K$ [17]. We cannot fool nature. Because l variables were originally chosen to describe each pattern (dimensionality, number of free parameters) the same number of free parameters will be required to describe the same objects after the mapping in $\mathbb{R}^K$. In other words, after the mapping, we embed an l-dimensional manifold in a K-dimensional space, in such a way, that the data in the two classes become linearly separable.

We have by now fully justified the need for mapping the task from the original low-dimensional space to a higher dimensional one, via a set of nonlinear functions. However, life is not easy to work in high-dimensional spaces. A large number of parameters are needed; this in turn poses computational complexity problems, and raises issues related to the generalization and overfitting performance of the designed predictors. In the sequel, we will address the former of the two problems by making a "careful" mapping to a higher dimensional space of a specific structure. The latter problem will be addressed via regularization, as it has already been discussed in various parts in previous chapters.

11.5 REPRODUCING KERNEL HILBERT SPACES

Consider a linear space $\mathbb{H}$, of real-valued functions defined on a set[2] $\mathcal{X} \subseteq \mathbb{R}^l$. Furthermore, suppose that $\mathbb{H}$ is a Hilbert space; that is, it is equipped with an inner product operation, $\langle \cdot, \cdot \rangle_{\mathbb{H}}$, that defines a corresponding norm $\| \cdot \|_{\mathbb{H}}$ and $\mathbb{H}$ is complete with respect to this norm.[3] From now on, and for notational simplicity, we drop out the subscript $\mathbb{H}$ from the inner product and norm notations and we are going to use them only if it is necessary to avoid confusion.

Definition 11.1. A Hilbert space $\mathbb{H}$ is called *reproducing kernel Hilbert space* (RKHS), if there exists a function

$$\kappa : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R},$$

with the following properties:

- For every $\boldsymbol{x} \in \mathcal{X}$, $\kappa(\cdot, \boldsymbol{x})$ belongs to $\mathbb{H}$.
- $\kappa(\cdot, \cdot)$ has the so-called *reproducing property*, that is,

$$\boxed{f(\boldsymbol{x}) = \langle f, \kappa(\cdot, \boldsymbol{x}) \rangle, \ \forall f \in \mathbb{H}, \ \forall \boldsymbol{x} \in \mathcal{X} : \quad \text{Reproducing Property.}} \tag{11.8}$$

A direct consequence of the reproducing property, if we set $f(\cdot) = \kappa(\cdot, \boldsymbol{y})$, $\boldsymbol{y} \in \mathcal{X}$, is that

$$\langle \kappa(\cdot, \boldsymbol{y}), \kappa(\cdot, \boldsymbol{x}) \rangle = \kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{y}, \boldsymbol{x}). \tag{11.9}$$

Definition 11.2. Let $\mathbb{H}$ be an RKHS, associated with a kernel function $\kappa(\cdot, \cdot)$, and $\mathcal{X}$ a set of elements. Then, the mapping

$$\boxed{\mathcal{X} \ni \boldsymbol{x} \longmapsto \phi(\boldsymbol{x}) := \kappa(\cdot, \boldsymbol{x}) \in \mathbb{H} : \quad \text{Feature Map,}}$$

is known as *feature map* and the space, $\mathbb{H}$, the *feature space*.

[2] Generalization to more general sets is also possible.

[3] For the unfamiliar reader, a Hilbert space is the generalization of Euclidean space allowing for infinite dimensions. More rigorous definitions and related properties are given in Section 8.15.

In other words, if $\mathcal{X}$ is the set of our observation vectors, the feature mapping maps each vector to the RKHS $\mathbb{H}$. Note that, in general, $\mathbb{H}$ can be of infinite dimension and its elements can be functions. That is, each training point is mapped to a function. In special cases, where $\mathbb{H}$ becomes a (finite dimensional) Euclidean space, $\mathbb{R}^K$, the image is a vector $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^K$. From now on, the general infinite dimensional case will be treated and the images will be denoted as functions, $\phi(\cdot)$. Let us now see what we have gained by choosing to perform the feature mapping from the original space to a high-dimensional RKHS one. Let $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} \subseteq \mathbb{R}^l$. Then, the inner product of the respective mapping images is written as

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle = \langle \kappa(\cdot, \boldsymbol{x}), \kappa(\cdot, \boldsymbol{y}) \rangle,$$

or

$$\boxed{\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle = \kappa(\boldsymbol{x}, \boldsymbol{y}) : \quad \text{Kernel Trick.}}$$

In other words, employing this type of mapping to our problem, we can perform inner product operations in $\mathbb{H}$ in a very efficient way; that is, via a function evaluation performed in the original low-dimensional space! This property is also known as the *kernel trick*, and it facilitates significantly the computations. As will become apparent soon, the way this property is exploited in practice involves the following steps:

1. Map (implicitly) the input training data to an RKHS

$$\boldsymbol{x}_n \longmapsto \phi(\boldsymbol{x}_n) \in \mathbb{H}, \quad n = 1, 2, \ldots, N.$$

2. Solve a *linear* estimation task in $\mathbb{H}$, involving the images $\phi(\boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$.
3. Cast the algorithm that solves for the unknown parameters in terms of inner product operations, in the form

$$\left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle, \quad i, j = 1, 2, \ldots, N.$$

4. Replace each inner product by a kernel evaluation, that is,

$$\left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

It is apparent that one does not need to perform any explicit mapping of the data. All is needed is to perform the kernel operations at the final step. Note that, the specific form of $\kappa(\cdot, \cdot)$ does not concern the analysis. Once the algorithm for the prediction, $\hat{y}$, has been derived, one can use different choices for $\kappa(\cdot, \cdot)$. As we will see, different choices for $\kappa(\cdot, \cdot)$ correspond to different types of nonlinearity. Figure 11.6 illustrates the rationale behind the procedure. In practice, the four steps listed above are equivalent to (a) work in the original (low-dimensional Euclidean space) and expressing all operations in terms of inner products and (b) at the final step substitute the inner products with kernel evaluations.

Example 11.1. Consider the case of the two-dimensional space and the mapping

$$\mathbb{R}^2 \ni \boldsymbol{x} \longmapsto \boldsymbol{\phi}(\boldsymbol{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \in \mathbb{R}^3.$$

Then, given two vectors $\boldsymbol{x} = [x_1, x_2]^T$ and $\boldsymbol{y} = [y_1, y_2]^T$, it is straightforward to see that

$$\boldsymbol{\phi}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})^2.$$

That is, the inner product in the new space is given in terms of a function of the variables in the original space,

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})^2.$$

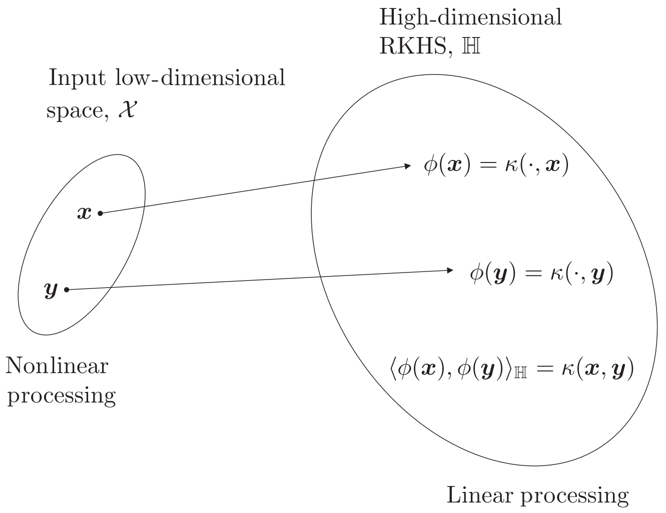
High-dimensional RKHS, $\mathbb{H}$
Input low-dimensional space, $\mathcal{X}$
$\phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x})$
$\boldsymbol{x}$
$\boldsymbol{y}$
$\phi(\boldsymbol{y}) = \kappa(\cdot, \boldsymbol{y})$
Nonlinear processing
$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathbb{H}} = \kappa(\boldsymbol{x}, \boldsymbol{y})$
Linear processing

FIGURE 11.6

The nonlinear task in the original low-dimensional space is mapped to a linear one in the high-dimensional RKHS $\mathbb{H}$. Using feature mapping, inner product operations are efficiently performed via kernel evaluations in the original low-dimensional spaces.

11.5.1 SOME PROPERTIES AND THEORETICAL HIGHLIGHTS

The reader who has no "mathematical anxieties" can bypass this subsection during a first reading.

Let $\mathcal{X}$ be a set of points. Typically $\mathcal{X}$ is a compact (closed and bounded) subset of $\mathbb{R}^l$. Let a function

$$\kappa : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R}.$$

Definition 11.3. The function κ is called a *positive definite kernel*, if

$$\boxed{\sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \geq 0 : \quad \text{Positive Definite Kernel},} \tag{11.10}$$

for any real numbers, a_n, a_m, any points $\boldsymbol{x}_n, \boldsymbol{x}_m \in \mathcal{X}$ and any $N \in \mathbb{N}$.

Note that (11.10) can be written in an equivalent form. Define the so-called *kernel matrix*, $\mathcal{K}$, of order, N,

$$\mathcal{K} := \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \vdots & \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}. \tag{11.11}$$

Then, (11.10) is written as

$$\boldsymbol{a}^{\mathrm{T}} \mathcal{K} \boldsymbol{a} \geq 0, \tag{11.12}$$

where

$$\boldsymbol{a} = [a_1, \ldots, a_N]^{\mathrm{T}}.$$

Because (11.10) is true for any $\boldsymbol{a} \in \mathbb{R}^N$, then (11.12) suggests that for a kernel to be positive definite, it suffices the corresponding kernel matrix to be positive semidefinite.[4]

Lemma 11.1. *The reproducing kernel, associated with an RKHS $\mathbb{H}$, is a positive definite kernel.*

The proof of the lemma is given in Problem 11.3. Note that the opposite is also true. It can be shown, [83, 107], that if $\kappa : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R}$ is a positive definite kernel, there exists an RKHS $\mathbb{H}$ of functions on $\mathcal{X}$, such that $\kappa(\cdot,\cdot)$ is a reproducing kernel of $\mathbb{H}$. This establishes the equivalence between reproducing and positive definite kernels. Historically, the theory of positive definite kernels was developed first in the context of integral equations by Mercer [77], and the connection to RKHS was developed later on, see, for example, [2].

Lemma 11.2. *Let $\mathbb{H}$ be an RKHS on the set $\mathcal{X}$ with reproducing kernel $\kappa(\cdot,\cdot)$. Then the linear span of the function $\kappa(\cdot,\boldsymbol{x})$, $\boldsymbol{x} \in \mathcal{X}$ is dense in $\mathbb{H}$, that is,*

$$\mathbb{H} = \overline{\mathrm{span}\{\kappa(\cdot,\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}\}}. \tag{11.13}$$

The proof of the lemma is given in Problem 11.4. The overbar denotes the closure of a set. In other words, $\mathbb{H}$ can be constructed by *all* possible linear combinations of the kernel function computed in $\mathcal{X}$, as well as the *limit points* of sequences of such combinations. Simply stated, $\mathbb{H}$ can be *fully generated from the knowledge* of $\kappa(\cdot,\cdot)$.

The interested reader can obtain more theoretical results concerning RKH spaces from, for example, [64, 84, 89, 104, 107, 109].

11.5.2 EXAMPLES OF KERNEL FUNCTIONS

In this subsection, we present some typical examples of kernel functions, which are commonly used in various applications.

- The *Gaussian* kernel is among the most popular ones and it is given by our familiar form

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

  with $\sigma > 0$ being a parameter. Figure 11.7a shows the Gaussian kernel as a function of $x, y \in \mathcal{X} = \mathbb{R}$ and $\sigma = 0.5$. Figure 11.7b shows $\phi(0) = \kappa(\cdot, 0)$ for various values of σ.

  The dimension of the RKHS generated by the Gaussian kernel is *infinite*. A proof that the Gaussian kernel satisfies the required properties can be obtained, for example, from [109].
- The *homogeneous polynomial* kernel has the form

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})^r,$$

  where r is a parameter.
- The *inhomogeneous* polynomial kernel is given by

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + c)^r,$$

[4] It may be slightly confusing that the definition of a positive definite kernel requires a positive semidefinite kernel matrix. However, this is what has been the accepted definition.

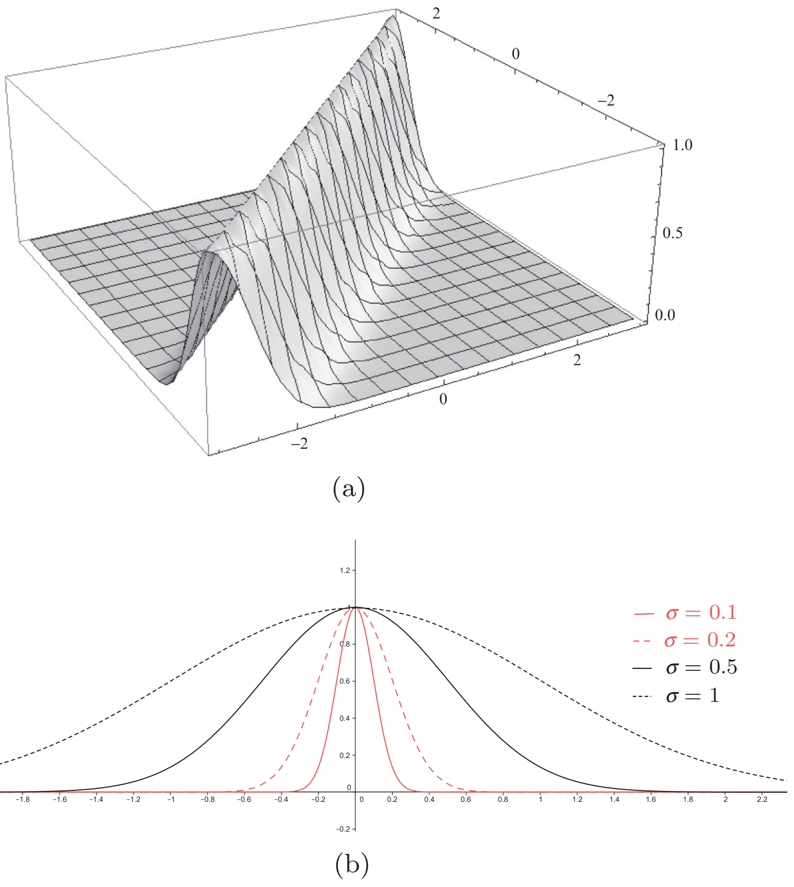

FIGURE 11.7

(a) The Gaussian kernel for $\mathcal{X} = \mathbb{R}$, $\sigma = 0.5$. (b) The element $\phi(0) = \kappa(\cdot, 0)$ for different values of σ.

where $c \geq 0$ and r parameters. The graph of the kernel is given in Figure 11.8a. In Figure 11.8b the elements $\phi(\cdot, x_0)$ are shown for different values of x_0. The dimensionality of the RKHS associated with polynomial kernels is finite.

- The *Laplacian* kernel is given by

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-t\|\boldsymbol{x} - \boldsymbol{y}\|\right),$$

where $t > 0$ is a parameter. The dimensionality of the RKHS associated with the Laplacian kernel is infinite.

- The *spline* kernels are defined as

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = B_{2p+1}(\|\boldsymbol{x} - \boldsymbol{y}\|^2),$$

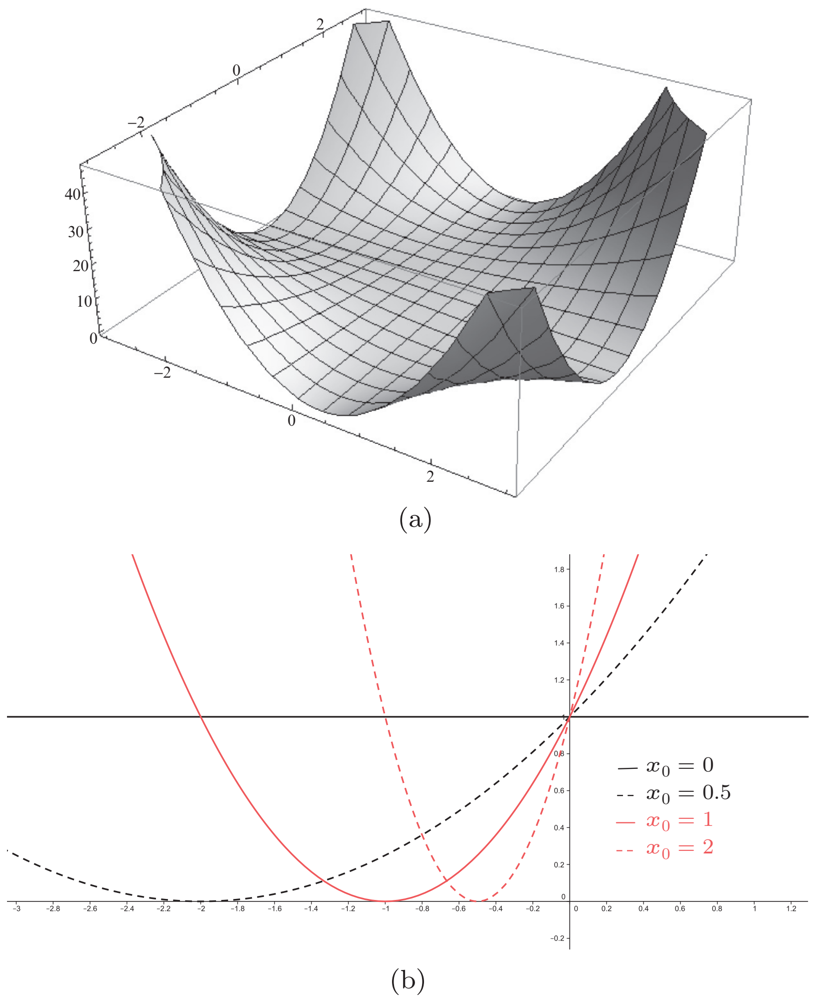

FIGURE 11.8

(a) The inhomogeneous polynomial kernel for $\mathcal{X} = \mathbb{R}$, $r = 2$. (b) The element $\phi(x) = \kappa(\cdot, x_0)$, for different values of x_0.

where the B_n spline is defined via the $n + 1$ convolutions of the unit interval $[-\frac{1}{2}, \frac{1}{2}]$, that is,

$$B_n(\cdot) := \bigotimes_{i=1}^{n+1} \chi_{[-\frac{1}{2},\frac{1}{2}]}(\cdot)$$

and $\chi_{[-\frac{1}{2},\frac{1}{2}]}(\cdot)$ is the characteristic function on the respective interval.[5]

[5] It is equal to one if the variable belongs to the interval and zero otherwise.

- The *sampling function* or *sinc* kernel is of particular interest from a signal processing point of view. This kernel function is defined as

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}.$$

Recall that we have met this function in Chapter 9 while discussing sub-Nyquist sampling. Let us now consider the set of all squared integrable functions, which are *band limited*, that is,

$$\mathcal{F}_B = \left\{ f : \int_{-\infty}^{+\infty} |f(x)|^2 dx < +\infty, \text{ and } |F(\omega)| = 0, \ |\omega| > \pi \right\},$$

where $F(\omega)$ is the respective Fourier transform

$$F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(x) e^{-j\omega x} dx.$$

It turns out that $\mathcal{F}_B$ is an RKHS whose reproducing kernel is the sinc function (e.g., [50]), that is,

$$\kappa(x, y) = \operatorname{sinc}(x - y).$$

This takes us back to the classical sampling theorem through the RKHS route. Without going into details, a by-product of this view is the Shannon's sampling theorem; any band limited function can be written as[6]

$$f(x) = \sum_n f(n) \operatorname{sinc}(x - n). \tag{11.14}$$

Constructing kernels

Besides the previous examples, one can construct more kernels by applying the following properties (Problem 11.6, [107]):

- If

$$\kappa_1(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R},$$
$$\kappa_2(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R},$$

are kernels, then

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa_1(\boldsymbol{x}, \boldsymbol{y}) + \kappa_2(\boldsymbol{x}, \boldsymbol{y}),$$

and

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \alpha \kappa_1(\boldsymbol{x}, \boldsymbol{y}), \ \alpha > 0,$$

and

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa_1(\boldsymbol{x}, \boldsymbol{y}) \kappa_2(\boldsymbol{x}, \boldsymbol{y}),$$

are also kernels.

- Let

$$f : \mathcal{X} \longmapsto \mathbb{R}.$$

[6] The key point behind the proof is that in $\mathcal{F}_B$, the kernel $\kappa(x, y)$ can be decomposed in terms of a set of orthogonal functions, that is, $\operatorname{sinc}(x - n), \ n = 0, \pm 1, \pm 2, \ldots$.

Then

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = f(\boldsymbol{x})f(\boldsymbol{y})$$

is a kernel.

- Let a function

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

and a kernel function

$$\kappa_1(\cdot,\cdot) : \mathbb{R}^l \times \mathbb{R}^l \longmapsto \mathbb{R}.$$

Then

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = \kappa_1\left(g(\boldsymbol{x}), g(\boldsymbol{y})\right)$$

is also a kernel.

- Let A be a positive definite $l \times l$ matrix. Then

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}} A \boldsymbol{y}$$

is a kernel.

- If

$$\kappa_1(\boldsymbol{x},\boldsymbol{y}) : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R},$$

then

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = \exp\left(\kappa_1(\boldsymbol{x},\boldsymbol{y})\right)$$

is also a kernel, and if $p(\cdot)$ is a polynomial with nonnegative coefficients,

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = p\left(\kappa_1(\boldsymbol{x},\boldsymbol{y})\right)$$

is also a kernel.

The interested reader will find more information concerning kernels and their construction in, for example, [51, 107, 109].

String kernels

So far, our discussion has been focused on input data that were vectors in a Euclidean space. However, as we have already pointed out, the input data need not necessarily be vectors, and they can be elements of more general sets.

Let us denote by $\mathcal{S}$ an alphabet set; that is, a set with a finite number of elements, which we call *symbols*. For example, this can be the set of all capital letters in the Latin alphabet. Bypassing the path of formal definitions, a *string* is a finite sequence, of any length, of symbols from $\mathcal{S}$. For example, two cases of strings are

$$T_1 = \text{“MYNAMEISSERGIOS”}, \quad T_2 = \text{“HERNAMEISDESPOINA”}.$$

In a number of applications, such as in text mining, spam filtering, text summarization, and bioinformatics, it is important to quantify how “similar” two strings are. However, kernels, by their definition, are similarity measures; they are constructed so as to express inner products in the high-dimensional feature space. An inner product is a similarity measure. Two vectors are most similar if they point to the same direction. Starting from this observation, there has been a lot of activity on defining kernels that measure similarity between strings. Without going into details, let us give such an example.

Let us denote by $\mathcal{S}^*$ the set of all possible strings that can be constructed using symbols from $\mathcal{S}$. Also, a string, s, is said to be a *substring* of x if $x = bsa$, where a and b are other strings (possibly empty) from the symbols of $\mathcal{S}$. Given two strings $x, y \in \mathcal{S}^*$, define

$$\kappa(x, y) := \sum_{s \in \mathcal{S}*} w_s \phi_s(x) \phi_s(y), \tag{11.15}$$

where, $w_s \geq 0$, and $\phi_s(x)$ is the number of times substring s appears in x. It turns out that this is indeed a kernel, in the sense that it complies with (11.10); such kernels constructed from strings are known as *string kernels*.

Obviously, a number of different variants of this kernel are available. The so-called *k-spectrum* kernel, considers common substrings only of length k. For example, for the two strings given before, the value of the 6-spectrum string kernel in (11.15) is equal to one (one common substring of length 6 is identified and appears once in each one of the two strings: “NAMEIS”). More on this topic, interested reader can obtain more on this topic from, for example, [107]. We will use the notion of the string kernel, in the case study in Section 11.15.

11.6 REPRESENTER THEOREM

The theorem to be stated in this section is of major importance from a practical point of view. It allows us to perform *empirical* loss function optimization, based on a finite set of training points, in a very efficient way even if the function to be estimated belongs to a very high (even infinite) dimensional space, $\mathbb{H}$.

Theorem 11.2. *Let*

$$\Omega : [0, +\infty) \longmapsto \mathbb{R}$$

be an arbitrary strictly monotonic increasing function. Let also

$$\mathcal{L} : \mathbb{R}^2 \longmapsto \mathbb{R} \cup \{\infty\}$$

be an arbitrary loss function. Then each minimizer, $f \in \mathbb{H}$, of the regularized minimization task,

$$\min_{f \in \mathbb{H}} J(f) := \sum_{n=1}^{N} \mathcal{L}\big(y_n, f(\boldsymbol{x}_n)\big) + \lambda \Omega\big(\|f\|^2\big) \tag{11.16}$$

admits a representation of the form,[7]

$$\boxed{f(\cdot) = \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n),} \tag{11.17}$$

where $\theta_n \in \mathbb{R},\ n = 1, 2, \ldots, N$.

[7] The property holds also for regularization of the form $\Omega(\|f\|)$, since the quadratic function is strictly monotonic on $[0, \infty)$, and the proof follows a similar line.

Proof. The linear span, $A := \text{span}\{\kappa(\cdot, \boldsymbol{x}_1), \ldots, \kappa(\cdot, \boldsymbol{x}_N)\}$, forms a closed subspace. Then, each $f \in \mathbb{H}$ can be decomposed into two parts (see, (8.20)), that is,

$$f(\cdot) = \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n) + f_\perp,$$

where $f_\perp$ is the part of f that is orthogonal to A. From the reproducing property, we obtain

$$\begin{aligned} f(\boldsymbol{x}_m) = \langle f, \kappa(\cdot, \boldsymbol{x}_m)\rangle &= \left\langle \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n), \kappa(\cdot, \boldsymbol{x}_m) \right\rangle \\ &= \sum_{n=1}^{N} \theta_n \kappa(\boldsymbol{x}_m, \boldsymbol{x}_n), \end{aligned}$$

where we used the fact that $\langle f_\perp, \kappa(\cdot, \boldsymbol{x}_n)\rangle = 0,\ n = 1, 2, \ldots, N$. In other words, the expansion in (11.17) guarantees that at the training points, the value of f does not depend on $f_\perp$. Hence, the first term in (11.16), corresponding to the empirical loss, does *not* depend on $f_\perp$. Moreover, for all $f_\perp$ we have

$$\begin{aligned} \Omega(\|f\|^2) &= \Omega\left(\left\| \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n) \right\|^2 + \|f_\perp\|^2 \right) \\ &\geq \Omega\left(\left\| \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n) \right\|^2 \right). \end{aligned}$$

Thus, for *any* choice of θ_n, $n = 1, 2, \ldots, N$, the cost function in (11.16) is minimized for $f_\perp = 0$. Thus, the claim is proved. □

The theorem was first shown in [60]. In [1], the conditions under which the theorem exists were investigated and related sufficient and necessary conditions were derived. The importance of this theorem is that in order to optimize (11.16) with respect to f, one uses the expansion in (11.17) and minimization is carried out with respect to the *finite* set of parameters, θ_n, $n = 1, 2, \ldots, N$.

Note that when working in high (even infinite) dimensional spaces, the presence of a regularizer can hardly be avoided; otherwise, the obtained solution will suffer from overfitting, as only a finite number of data samples are used for training. The effect of regularization on the generalization performance and stability of the associated solution has been studied in a number of classical papers, for example, [16, 37, 80, 92].

Usually, a bias term is often added and it is assumed that the minimizing function admits the following representation,

$$\tilde{f} = f + b, \tag{11.18}$$

$$f(\cdot) = \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n). \tag{11.19}$$

In practice, the use of a bias term (which does not enter in the regularization) turns out to improve performance. First, it enlarges the class of functions in which the solution is searched and potentially leads to better performance. Moreover, due to the penalization imposed by the regularizing term,

$\Omega(\|f\|^2)$, the minimizer pushes the values, which the function takes at the training points, to smaller values. The existence of b tries to "absorb" some of this action; see, for example, [109].

Remarks 11.2.

- We will use the expansion in (11.17) in a number of cases. However, it is interesting to apply this expansion to the RKHS of the band limited functions and see what comes out. Assume that the available samples from a function f are $f(n),\ n = 1, 2, \ldots, N$ (assuming the case of normalized sampling period $x_s = 1$). Then according to the representer theorem, we can write the following approximation.

$$f(x) \approx \sum_{n=1}^{N} \theta_n \operatorname{sinc}(x - n). \tag{11.20}$$

Taking into account the orthonormality of the $\operatorname{sinc}(\cdot - n)$ functions, we get $\theta_n = f(n),\ n = 1, 2, \ldots, N$. However, note that in contrast to (11.14), which is exact, (11.20) is only an approximation. On the other hand, (11.20) can be used even if the obtained samples are contaminated by noise.

11.6.1 SEMIPARAMETRIC REPRESENTER THEOREM

The use of the bias term is also theoretically justified by the generalization of the representer theorem [104]. The essence of this theorem is to expand the solution into two parts. One that lies in an RKHS, $\mathbb{H}$, and another one that is given as a linear combination of a set of preselected functions.

Theorem 11.3. *Let us assume that in addition to the assumptions adopted in Theorem 11.2, we are given the set of real-valued functions*

$$\psi_m : \mathcal{X} \longmapsto \mathbb{R}, \quad m = 1, 2, \ldots, M,$$

with the property that the $N \times M$ matrix with elements $\psi_m(\boldsymbol{x}_n)$, $n = 1, 2, \ldots, N,\ m = 1, 2, \ldots, M$, has rank M. Then, any

$$\tilde{f} = f + h,\ f \in \mathbb{H}, \quad h \in \operatorname{span}\{\psi_m, m = 1, 2, \ldots, M\},$$

solving the minimization task

$$\min_{\tilde{f}} J(\tilde{f}) := \sum_{n=1}^{N} \mathcal{L}\big(y_n, \tilde{f}(\boldsymbol{x}_n)\big) + \Omega(\|f\|^2), \tag{11.21}$$

admits the following representation:

$$\boxed{\tilde{f}(\cdot) = \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n) + \sum_{m=1}^{M} b_m \psi_m(\cdot).} \tag{11.22}$$

Obviously, the use of a bias term is a special case of the expansion above. An example of successful application of this theorem was demonstrated in [13] in the context of image de-noising. A set of nonlinear functions in place of ψ_m were used to account for the edges (nonsmooth jumps) in an image. The part of f lying in the RKHS accounted for the smooth parts in the image.

11.6.2 NONPARAMETRIC MODELING: A DISCUSSION

Note that searching a model function in an RKHS space is a typical task of *nonparametric* modeling. In contrast to the parametric modeling in Eq. (11.1), where the unknown function is *parameterized* in terms of a set of basis functions, the minimization in (11.16) or (11.21) is performed with regard to functions that are *constrained to belong in a specific space*. In the more general case, minimization could be performed with regard to any (continuous) function, for example,

$$\min_f \sum_{n=1}^{N} \mathcal{L}\big(y_n, f(\boldsymbol{x}_n)\big) + \lambda \phi(f),$$

where $\mathcal{L}(\cdot,\cdot)$ can be any loss function and ϕ an appropriately chosen regularizing functional. Note, however, that in this case, the presence of the regularization is crucial. If there is no regularization, then any function that *interpolates* the data is a solution; such techniques have also been used in interpolation theory, for example, [79, 93]. The regularization term, $\phi(f)$, helps to smooth out the function to be recovered. To this end, functions of derivatives have been employed. For example, if the minimization cost is chosen as

$$\sum_{n=1}^{N} (y_n - f(x_n))^2 + \lambda \int (f''(x))^2 \, \mathrm{d}x,$$

then the solution is a cubic spline; that is, a piecewise cubic function with knots the points x_n, $n = 1, 2, \ldots, N$ and it is continuously differentiable to the second order. The choice of λ controls the degree of smoothness of the approximating function; the larger its value the smoother the minimizer becomes.

If on the other hand, f is constrained to lie in an RKHS and the minimizing task is as in (11.16), then the resulting function is of the form given in (11.17), where a kernel function is placed at each input training point. It must be pointed out that the parametric form that now results was not in our original intentions. It came out as a by-product of the theory. However, it should be stressed that, in contrast to the parametric methods, now the number of parameters to be estimated is not fixed but it depends on the number of the training points. Recall that this is an important difference and it was carefully pointed out when parametric methods were introduced and defined in Chapter 3.

11.7 KERNEL RIDGE REGRESSION

Ridge regression was introduced in Chapter 3 and it has also been treated in more detail in Chapter 6. Here, we will state the task in a general RKHS. The path to be followed is the typical one used to extend techniques, which have been developed for linear models, to the more general RKH spaces.

We assume that the generation mechanism of the data, represented by the training set $(y_n, \boldsymbol{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, is modeled via a nonlinear regression task

$$y_n = g(\boldsymbol{x}_n) + \eta_n, \; n = 1, 2, \ldots, N. \tag{11.23}$$

Let us denote by f the estimate of the unknown g. Sometimes, f is called the *hypothesis* and the space $\mathbb{H}$ in which f is searched is known as the *hypothesis space*. We will further assume that f lies in an RKHS, associated with a kernel

$$\kappa : \mathbb{R}^l \times \mathbb{R}^l \longmapsto \mathbb{R}.$$

Motivated by the representer theorem, we adopt the following expansion

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \theta_n \kappa(\boldsymbol{x}, \boldsymbol{x}_n).$$

According to the kernel ridge regression approach, the unknown coefficients are estimated by the following task

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),\\ J(\boldsymbol{\theta}) &:= \sum_{n=1}^{N}\left(y_n - \sum_{m=1}^{N} \theta_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m)\right)^2 + C\langle f, f\rangle, \end{aligned} \tag{11.24}$$

where C is the regularization parameter.[8] Equation (11.24) can be rewritten as (Problem 11.7)

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta}) + C\boldsymbol{\theta}^{\mathrm{T}}\mathcal{K}^{\mathrm{T}}\boldsymbol{\theta}, \tag{11.25}$$

where

$$\boldsymbol{y} = [y_1, \ldots, y_N]^{\mathrm{T}}, \quad \boldsymbol{\theta} = [\theta_1, \ldots, \theta_N]^{\mathrm{T}},$$

and $\mathcal{K}$ is the kernel matrix defined in (11.11); the latter is fully determined by the kernel function and the training points. Following our familiar-by-now arguments, minimization of $J(\boldsymbol{\theta})$ with regard to $\boldsymbol{\theta}$ leads to

$$(\mathcal{K}^{\mathrm{T}}\mathcal{K} + C\mathcal{K}^{\mathrm{T}})\hat{\boldsymbol{\theta}} = \mathcal{K}^{\mathrm{T}}\boldsymbol{y}$$

or

$$\boxed{(\mathcal{K} + CI)\hat{\boldsymbol{\theta}} = \boldsymbol{y} : \quad \text{Kernel Ridge Regression,}} \tag{11.26}$$

where $\mathcal{K}^{\mathrm{T}} = \mathcal{K}$ has been assumed to be invertible.[9] Once $\hat{\boldsymbol{\theta}}$ has been obtained, given an unknown vector, $\boldsymbol{x} \in \mathbb{R}^l$, the corresponding prediction value of the dependent variable is given by

$$\hat{y} = \sum_{n=1}^{N} \hat{\theta}_n \kappa(\boldsymbol{x}, \boldsymbol{x}_n) = \hat{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\kappa}(\boldsymbol{x}),$$

where

$$\boldsymbol{\kappa}(\boldsymbol{x}) = [\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}, \boldsymbol{x}_N)]^{\mathrm{T}}.$$

Employing (11.26), we obtain

$$\boxed{\hat{y}(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}}(\mathcal{K} + CI)^{-1}\boldsymbol{\kappa}(\boldsymbol{x}).} \tag{11.27}$$

Example 11.2. In this example, the prediction power of the kernel ridge regression in the presence of noise and outliers will be tested. The original data were samples from a music recording of *Blade Runner* by Vangelis Papathanasiou. A white Gaussian noise was then added at a 15 dB level and a

[8] For the needs of this chapter, we denote the regularization constant as C, not to be confused with the Lagrange multipliers, to be introduced soon.

[9] This is true, for example, for the Gaussian kernel, [104].

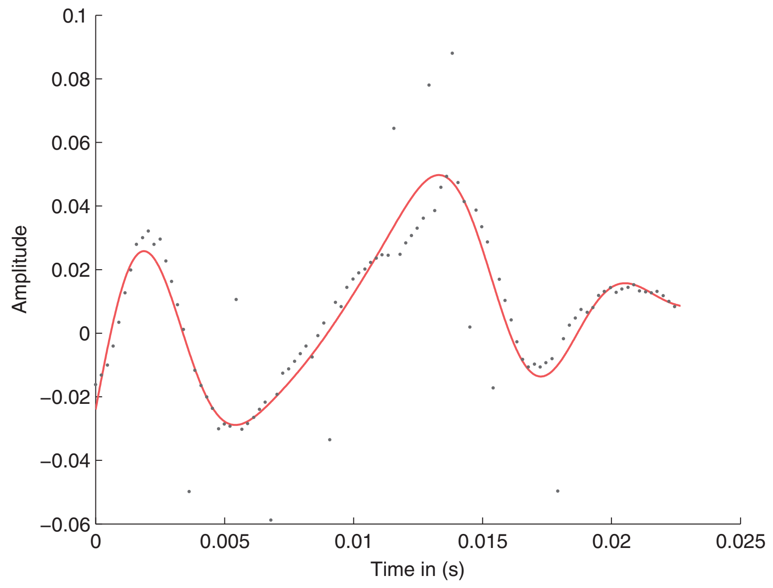

FIGURE 11.9

Plot of the data used for training together with the fitted (prediction) curve obtained via the kernel ridge regression, for Example 11.2. The Gaussian kernel was used.

number of outliers were intentionally randomly introduced and "hit" some of the values (10%, of them). The kernel ridge regression method was used, employing the Gaussian kernel with $\sigma = 0.004$. We allowed for a bias term to be present (see Problem 11.8). The prediction (fitted) curve, $\hat{y}(x)$, for various values of x, is shown in Figure 11.9 together with the (noisy) data used for training.

11.8 SUPPORT VECTOR REGRESSION

The least-squares cost function is not always the best criterion for optimization, in spite of its merits. In the case of the presence of a non-Gaussian noise with long tails and, hence, with an increased number of noise *outliers*, the square dependence of the LS criterion gets biased toward values associated with the presence of outliers. Recall from Chapter 3 that the method of least-squares is equivalent to the maximum likelihood estimation under the assumption of white Gaussian noise. Moreover, under this assumption, the LS estimator achieves the Cramer-Rao bound and it becomes a minimum variance estimator. However, under other noise scenarios, one has to look for alternative criteria.

The task of optimization in the presence of outliers was studied by Huber [53], whose goal was to obtain a strategy for choosing the loss function that "matches" best to the noise model. He proved that, under the assumption that the noise has a symmetric pdf, the optimal minimax strategy for regression is obtained via the following loss function,

$$\mathcal{L}\big(y, f(\boldsymbol{x})\big) = |y - f(\boldsymbol{x})|,$$

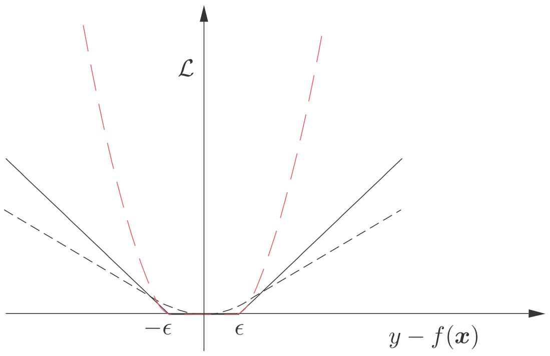

FIGURE 11.10

The Huber loss function (dotted-gray), the linear ϵ-insensitive (full-gray), and the quadratic ϵ-insensitive (red) loss functions, for $\epsilon = 0.7$.

which is known as the *least modulus* method. Note from Section 5.8 that the stochastic gradient online version for this loss function leads to the sign-error LMS. Huber also showed that if the noise comprises two components, one corresponding to a Gaussian and another to an arbitrary pdf (which remains symmetric), then the best in the minimax sense loss function is given by

$$\mathcal{L}\big(y, f(\boldsymbol{x})\big) = \begin{cases} \epsilon|y - f(\boldsymbol{x})| - \frac{\epsilon^2}{2}, & \text{if} \quad |y - f(\boldsymbol{x})| > \epsilon, \\ \frac{1}{2}|y - f(\boldsymbol{x})|^2, & \text{if} \quad |y - f(\boldsymbol{x})| \le \epsilon, \end{cases}$$

for some parameter ϵ. This is known as the Huber loss function and it is shown in Figure 11.10. A loss function that can approximate the Huber one and, as we will see, turns out to have some nice computational properties, is the so-called *linear* ϵ-insensitive loss function, defined as (see also Chapter 8)

$$\mathcal{L}\big(y, f(\boldsymbol{x})\big) = \begin{cases} |y - f(\boldsymbol{x})| - \epsilon, & \text{if} \quad |y - f(\boldsymbol{x})| > \epsilon, \\ 0, & \text{if} \quad |y - f(\boldsymbol{x})| \le \epsilon, \end{cases} \tag{11.28}$$

and it is shown in Figure 11.10. Note that for $\epsilon = 0$, it coincides with the least absolute loss function, and it is close to the Huber loss for small values of $\epsilon < 1$. Another version is the *quadratic* ϵ-insensitive defined as

$$\mathcal{L}\big(y, f(\boldsymbol{x})\big) = \begin{cases} |y - f(\boldsymbol{x}))|^2 - \epsilon, & \text{if} \quad |y - f(\boldsymbol{x})| > \epsilon, \\ 0, & \text{if} \quad |y - f(\boldsymbol{x})| \le \epsilon, \end{cases} \tag{11.29}$$

which coincides with the LS loss for $\epsilon = 0$. The corresponding graph is given in Figure 11.10. Observe that the two previously discussed ϵ-insensitive loss functions retain their convex nature; however, they are no more differentiable at all points.

11.8.1 THE LINEAR ϵ-INSENSITIVE OPTIMAL REGRESSION

Let us now adopt (11.28) as the loss function to quantify model misfit. We will treat the regression task in (11.23), employing a linear model for f, that is

$$f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} + \theta_0.$$

Once we obtain the solution expressed in inner product operations, the more general solution for the case where f lies in an RKHS will be obtained via the kernel trick; that is, inner products will be replaced by kernel evaluations.

Let us now introduce two sets of *auxiliary* variables. If

$$y_n - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n - \theta_0 \geq \epsilon,$$

define $\tilde{\xi}_n \geq 0$, such as

$$y_n - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n.$$

Note that ideally, we would like to select $\boldsymbol{\theta}, \theta_0$, so that $\tilde{\xi}_n = 0$, because this would make the contribution of the respective term in the loss function equal to zero. Also, if

$$y_n - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n - \theta_0 \leq -\epsilon,$$

define $\xi_n \geq 0$, such as

$$\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n.$$

Once more, we would like to select our unknown set of parameters so that ξ_n is zero.

We are now ready to formulate the minimizing task around the corresponding empirical cost, regularized by the norm of $\boldsymbol{\theta}$, which is cast in terms of the auxiliary variables as[10]

$$\text{minimize} \quad J(\boldsymbol{\theta}, \theta_0, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\left(\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\tilde{\xi}_n\right), \tag{11.30}$$

$$\text{subject to} \quad y_n - \boldsymbol{\theta}^T\boldsymbol{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n, \; n = 1, 2, \ldots, N, \tag{11.31}$$

$$\boldsymbol{\theta}^T\boldsymbol{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n, \; n = 1, 2, \ldots, N, \tag{11.32}$$

$$\tilde{\xi}_n \geq 0, \; \xi_n \geq 0, \; n = 1, 2, \ldots, N. \tag{11.33}$$

Before we proceed further some explanations are in order.

- The auxiliary variables, $\tilde{\xi}_n$ and ξ_n, $n = 1, 2, \ldots, N$, which measure the excess error with regard to ϵ, are known as *slack variables*. Note that according to the ϵ-insensitive rationale, any contribution to the cost function of an error with absolute value less than or equal to ϵ is zero. The previous optimization task attempts to estimate $\boldsymbol{\theta}$, θ_0 so that the number of error values larger than ϵ and smaller than $-\epsilon$ is minimized. Thus, the optimization task in (11.30)–(11.33) is equivalent with minimizing the empirical loss function

$$\frac{1}{2}||\boldsymbol{\theta}||^2 + C\sum_{n=1}^{N}\mathcal{L}\left(y_n, \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n + \theta_0\right),$$

  where the loss function is the linear ϵ-insensitive one. Note that, any other method for minimizing (nondifferentiable) convex functions could be used, for example, Chapter 8. However, the constrained optimization involving the slack variables has a historical value and it was the path that paved the way in employing the kernel trick, as we will see soon.

[10] It is common in the literature to formulate the regularized cost via the parameter C multiplying the loss term and not $||\boldsymbol{\theta}||^2$. In any case, they are both equivalent.

- As said before, the task could be cast directly as a linear one in an RKHS. In such a case, the path to follow is to assume a mapping

  $$x \longmapsto \phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x}),$$

  for some kernel, and then approximate the nonlinear function, $g(\boldsymbol{x})$, in the regression task as a linear one in the respective RKHS, that is,

  $$g(\boldsymbol{x}) \approx f(\boldsymbol{x}) = \langle \theta, \phi(\boldsymbol{x}) \rangle + \theta_0,$$

  where θ is now treated as a function in the RKHS. However, in this case, in order to solve the minimizing task we should consider differentiation with regard to functions. Although the rules are similar to those of differentiation with respect to variables, because we have not given such definitions we have avoided following this path (for the time being).

The solution

The solution of the optimization task is obtained by introducing Lagrange multipliers and forming the corresponding Lagrangian (see below for the detailed derivation). Having obtained the Lagrange multipliers, the solution turns out to be given in a simple and rather elegant form,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N} (\tilde{\lambda}_n - \lambda_n) \boldsymbol{x}_n,$$

where $\tilde{\lambda}_n, \ \lambda_n, n = 1, 2, \ldots, N$, are the Lagrange multiplies associated with each one of the constraints. It turns out that the Lagrange multipliers are nonzero *only* for those points, $\boldsymbol{x}_n$, that correspond to error values *either equal or larger* than ϵ. These are known as *support vectors*. Points that score error values *less* than ϵ correspond to zero Lagrange multipliers and do not participate in the formation of the solution. The bias term can be obtained by anyone from the set of equations

$$y_n - \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n - \theta_0 = \epsilon, \tag{11.34}$$

$$\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0 - y_n = \epsilon, \tag{11.35}$$

where n above runs over the points that are associated with $\tilde{\lambda}_n > 0$ $(\lambda_n > 0)$ *and* $\tilde{\xi}_n = 0$ $(\xi_n = 0)$ (note that these points form a subset of the support vectors). In practice, $\hat{\theta}_0$ is obtained as the average from all the previous equations.

For the more general setting of the task in an RKHS, we can write

$$\hat{\theta}(\cdot) = \sum_{n=1}^{N} (\tilde{\lambda}_n - \lambda_n) \kappa(\cdot, \boldsymbol{x}_n).$$

Once $\hat{\theta}, \ \hat{\theta}_0$ have been obtained, we are ready to perform prediction. Given a value $\boldsymbol{x}$, we first perform the (implicit) mapping using the feature map

$$\boldsymbol{x} \longmapsto \kappa(\cdot, \boldsymbol{x}),$$

and we get

$$\hat{y}(\boldsymbol{x}) = \left\langle \hat{\theta}, \kappa(\cdot, \boldsymbol{x}) \right\rangle + \hat{\theta}_0$$

or

$$\boxed{\hat{y}(\boldsymbol{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n)\kappa(\boldsymbol{x}, \boldsymbol{x}_n) + \hat{\theta}_0 : \qquad \text{SVR Prediction},} \tag{11.36}$$

where $N_s \le N$, is the number of nonzero Lagrange multipliers. Observe that (11.36) is an expansion in terms of nonlinear (kernel) functions. Moreover, as only a fraction of the points is involved (N_s), the use of the ϵ-insensitive loss function achieves a form of *sparsification* on the general expansion dictated by the representer theorem in (11.17) or (11.18).

Solving the optimization task

The reader who is not interested in proofs can bypass this part in a first reading.

The task in (11.30)–(11.33) is a *convex programming* minimization, with a set of linear inequality constraints. As it is discussed in Appendix C, a minimizer has to satisfy the following *Karush-Kuhn-Tucker* conditions,

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0}, \ \frac{\partial L}{\partial \theta_0} = 0, \ \frac{\partial L}{\partial \tilde{\xi}_n} = 0, \ \frac{\partial L}{\partial \xi_n} = 0, \tag{11.37}$$

$$\tilde{\lambda}_n(y_n - \boldsymbol{\theta}^T \boldsymbol{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) = 0, \ n = 1, 2, \ldots, N, \tag{11.38}$$

$$\lambda_n(\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0 - y_n - \epsilon - \xi_n) = 0, \ n = 1, 2, \ldots, N, \tag{11.39}$$

$$\tilde{\mu}_n \tilde{\xi}_n = 0, \ \mu_n \xi_n = 0, \ n = 1, 2, \ldots, N, \tag{11.40}$$

$$\tilde{\lambda}_n \ge 0, \ \lambda_n \ge 0, \ \tilde{\mu}_n \ge 0, \ \mu_n \ge 0, \ n = 1, 2, \ldots, N, \tag{11.41}$$

where L is the respective Lagrangian

$$\begin{aligned} L(\boldsymbol{\theta}, \theta_0, \tilde{\boldsymbol{\xi}}, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = {} & \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n\right) \\ & + \sum_{n=1}^N \tilde{\lambda}_n(y_n - \boldsymbol{\theta}^T \boldsymbol{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) \\ & + \sum_{n=1}^N \lambda_n(\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0 - y_n - \epsilon - \xi_n) \\ & - \sum_{n=1}^N \tilde{\mu}_n \tilde{\xi}_n - \sum_{n=1}^N \mu_n \xi_n, \end{aligned} \tag{11.42}$$

where $\tilde{\lambda}_n, \lambda_n, \tilde{\mu}_n, \mu_n$ are the corresponding Lagrange multipliers. A close observation of (11.38) and (11.39) reveals that (Why?)

$$\tilde{\xi}_n \xi_n = 0, \ \tilde{\lambda}_n \lambda_n = 0, \quad n = 1, 2, \ldots, N. \tag{11.43}$$

Taking the derivatives of the Lagrangian in (11.37) and equating to zero results in

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n)\boldsymbol{x}_n, \tag{11.44}$$

$$\frac{\partial L}{\partial \theta_0} = 0 \longrightarrow \sum_{n=1}^{N} \tilde{\lambda}_n = \sum_{n=1}^{N} \lambda_n, \tag{11.45}$$

$$\frac{\partial L}{\partial \tilde{\xi}_n} = 0 \longrightarrow C - \tilde{\lambda}_n - \tilde{\mu}_n = 0, \tag{11.46}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \longrightarrow C - \lambda_n - \mu_n = 0. \tag{11.47}$$

Note that all one needs in order to obtain $\hat{\boldsymbol{\theta}}$ are the values of the Lagrange multipliers. As discussed in Appendix C, these can be obtained by writing the problem in its dual representation form, that is,

$$\text{maximize with respect to } \boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}} \quad \sum_{n=1}^{N} (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon(\tilde{\lambda}_n + \lambda_n),$$

$$-\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \boldsymbol{x}_n^T \boldsymbol{x}_m, \tag{11.48}$$

$$\text{subject to} \quad 0 \le \tilde{\lambda}_n \le C, \; 0 \le \lambda_n \le C, \; n = 1, 2, \ldots, N, \tag{11.49}$$

$$\sum_{n=1}^{N} \tilde{\lambda}_n = \sum_{n=1}^{N} \lambda_n. \tag{11.50}$$

Concerning the maximization task in (11.48)–(11.50), the following comments are in order.

- (11.48) results by plugging into the Lagrangian the estimate obtained in (11.44) and following the steps as required by the dual representation form, (Problem 11.10).
- (11.49) results from (11.46) and (11.47) taking into account that $\mu_n \ge 0$, $\tilde{\mu}_n \ge 0$.
- The beauty of the dual representation form is that it involves the observation vectors in the form of inner product operations. Thus, when the task is solved in an RKHS, (11.48) becomes

$$\text{maximize with respect to } \boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}} \quad \sum_{n=1}^{N} (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon(\tilde{\lambda}_n + \lambda_n)$$

$$-\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m).$$

- The KKT conditions convey important information. The Lagrange multipliers, $\tilde{\lambda}_n, \lambda_n$, for points that score error less than ϵ, that is,

$$|\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0 - y_n| < \epsilon,$$

  are zero. This is a direct consequence of (11.38) and (11.39) and the fact that $\tilde{\xi}_n, \xi_n \ge 0$. Thus, the Lagrange multipliers are *nonzero* only for points which score error either equal to ϵ ($\tilde{\xi}_n, \xi_n = 0$) or larger values ($\tilde{\xi}_n, \xi_n > 0$). In other words, only the points with nonzero Lagrange multipliers (support vectors) enter in (11.44) which leads to a *sparsification* of the expansion in (11.44).
- Due to (11.43), either $\tilde{\xi}_n$ or ξ_n can be nonzero, but not both of them. This also applies to the corresponding Lagrange multipliers.
- Note that if $\tilde{\xi}_n > 0$ (or $\xi_n > 0$) then from (11.40), (11.46), and (11.47) we obtain that

$$\tilde{\lambda}_n = C \;\text{ or }\; \lambda_n = C.$$

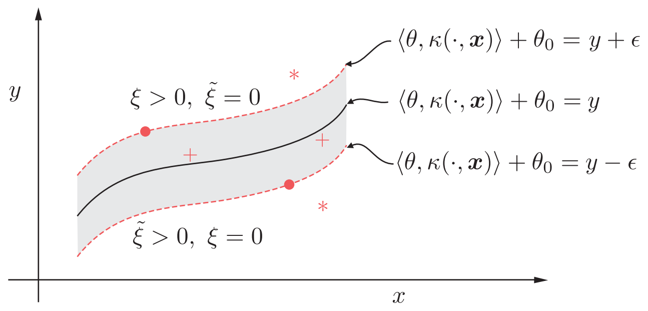
$\langle \theta, \kappa(\cdot, \boldsymbol{x})\rangle + \theta_0 = y + \epsilon$
$\langle \theta, \kappa(\cdot, \boldsymbol{x})\rangle + \theta_0 = y$
$\langle \theta, \kappa(\cdot, \boldsymbol{x})\rangle + \theta_0 = y - \epsilon$
y
$\xi > 0,\ \tilde{\xi} = 0$
$\tilde{\xi} > 0,\ \xi = 0$
x

FIGURE 11.11

The tube around the nonlinear regression curve. Points outside the tube (denoted by stars) have either $\tilde{\xi} > 0$ and $\xi = 0$ or $\xi > 0$ and $\tilde{\xi} = 0$. The rest of the points have $\tilde{\xi} = \xi = 0$. Points that are inside the tube correspond to zero Lagrange multipliers.

That is, the respective Lagrange multipliers get their maximum value. In other words, they have a "big say" in the expansion in (11.44). When $\tilde{\xi}_n$ and/or ξ_n are zero, then

$$0 \leq \tilde{\lambda}_n \leq C,\ 0 \leq \lambda_n \leq C.$$

- Recall what we have said before concerning the estimation of θ_0. Select any point corresponding to $0 < \tilde{\lambda}_n < C$, $(0 < \lambda_n < C)$, which we know correspond to $\tilde{\xi}_n = 0$ $(\xi_n = 0)$. Then $\hat{\theta}_0$ is computed from (11.38) and (11.39). In practice, one selects all such points and computes θ_0 as the respective mean.
- Figure 11.11 illustrates $\hat{y}(\boldsymbol{x})$ for a choice of $\kappa(\cdot,\cdot)$. Observe that the value of ϵ forms a "tube" around the respective graph. Points lying outside the tube correspond to values of the slack variables larger than zero.

Remarks 11.3.

- Besides the linear ϵ-insensitive loss, similar analysis is valid for the quadratic ϵ-insensitive and Huber loss functions, for example, [27, 127]. It turns out that using the Huber loss function results in a larger number of support vectors. Note that a large number of support vectors increases complexity, as more kernel evaluations are involved.
- *Sparsity and ϵ-insensitive loss function*: Note that (11.36) is exactly the same form as (11.18). However, in the former case, the expansion is a sparse one using $N_s < N$, and in practice often $N_s \ll N$. The obvious question that is now raised is whether there is a "hidden" connection between the ϵ-insensitive loss function and the sparsity-promoting methods, discussed in Chapter 9. Interestingly enough, the answer is in the affirmative [46]. Assuming the unknown function, g, in (11.23) to reside in an RKHS, and exploiting the representer theorem, it is approximated by an expansion in an RKHS and the unknown parameters are estimated by minimizing

$$L(\boldsymbol{\theta}) = \frac{1}{2}\left\| y(\cdot) - \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n)\right\|_{\mathbb{H}}^2 + \epsilon \sum_{n=1}^{N} |\theta_n|.$$

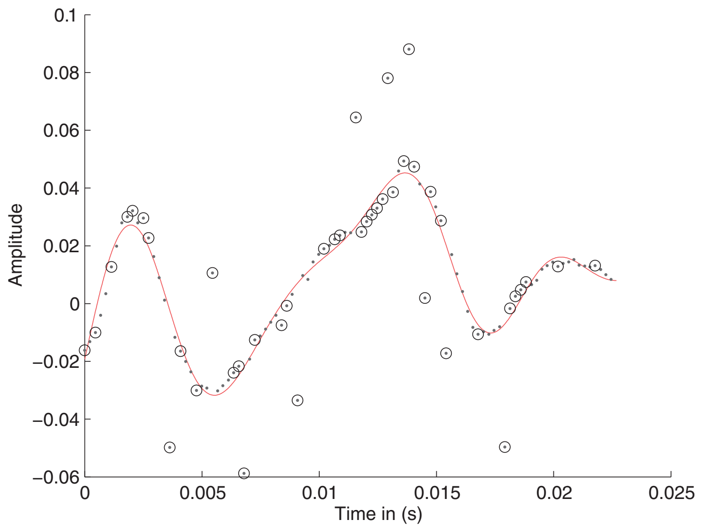

FIGURE 11.12

The resulting prediction curve for the same data points as those used for Example 11.2. The improved performance compared to the kernel ridge regression used for Figure 11.9 is readily observed. The encircled points are the support vectors resulting from the optimization, using the ϵ-insensitive loss function.

This is similar to what we did for the kernel ridge regression with the notable exception that the ℓ_1 norm of the parameters is involved for regularization. The norm $\|\cdot\|_{\mathbb{H}}$ denotes the norm associated with the RKHS. Elaborating on the norm, it can be shown that for the noiseless case, the minimization task becomes identical with the SVR one.

Example 11.3. Consider the same time series used for the nonlinear prediction task in Example 11.2. This time, the SVR method was used optimized around the linear ϵ-insensitive loss function, with $\epsilon = 0.003$. The same Gaussian kernel with $\sigma = 0.004$ was employed as in the kernel ridge regression (KRR) case. Figure 11.12 shows the resulting prediction curve, $\hat{y}(x)$ as a function of x given in (11.36). The encircled points are the support vectors. Even without the use of any quantitative measure, the resulting curve fits the data samples much better compared to the kernel ridge regression, exhibiting the enhanced robustness of the SVR method relative to the KRR, in the presence of outliers.

Remarks 11.4.

- A more recent trend to deal with outliers is via their explicit modeling. The noise is split into two components, the inlier and the outlier. The outlier part has to be spare; otherwise, it would not be called outlier. Then, sparsity-related arguments are mobilized to solve an optimization task that estimates both the parameters as well as the outliers; see, for example, [15, 72, 81, 86, 87].

11.9 KERNEL RIDGE REGRESSION REVISITED

The kernel ridge regression was introduced in Section 11.7. Here, it will be restated via its dual representation form. The ridge regression in its primal representation can be cast as

$$\text{minimize with respect to } \boldsymbol{\theta},\ \boldsymbol{\xi} \qquad J(\boldsymbol{\theta},\boldsymbol{\xi}) = \sum_{n=1}^{N} \xi_n^2 + C\|\boldsymbol{\theta}\|^2, \tag{11.51}$$

$$\text{subject to} \qquad y_n - \boldsymbol{\theta}^T \boldsymbol{x}_n = \xi_n,\ n = 1,2,\ldots,N,$$

which leads to the following Lagrangian:

$$L(\boldsymbol{\theta},\boldsymbol{\xi},\boldsymbol{\lambda}) = \sum_{n=1}^{N} \xi_n^2 + C\|\boldsymbol{\theta}\|^2 + \sum_{n=1}^{N} \lambda_n (y_n - \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n - \xi_n), \quad n = 1,2,\ldots,N. \tag{11.52}$$

Differentiating with respect to $\boldsymbol{\theta}$ and ξ_n, $n = 1,2,\ldots,N$, and equating to zero, we obtain

$$\boldsymbol{\theta} = \frac{1}{2C} \sum_{n=1}^{N} \lambda_n \boldsymbol{x}_n \tag{11.53}$$

and

$$\xi_n = \frac{\lambda_n}{2}, \quad n = 1,2,\ldots,N. \tag{11.54}$$

To obtain the Lagrange multipliers, (11.53) and (11.54) are substituted in (11.52) which results in the dual formulation of the problem, that is,

$$\text{maximize with respect to } \boldsymbol{\lambda} \qquad \sum_{n=1}^{N} \lambda_n y_n - \frac{1}{4C} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) - \frac{1}{4} \sum_{n=1}^{N} \lambda_n^2, \tag{11.55}$$

where we have replaced $\boldsymbol{x}_n^T \boldsymbol{x}_m$ with the kernel operation according to the kernel trick. It is a matter of straightforward algebra to obtain ([99], Problem 11.9)

$$\boldsymbol{\lambda} = 2C(\mathcal{K} + CI)^{-1} \boldsymbol{y}, \tag{11.56}$$

which combined with (11.53) and involving the kernel trick we obtain the prediction rule for the kernel ridge regression, that is,

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}} (\mathcal{K} + CI)^{-1} \boldsymbol{\kappa}(\boldsymbol{x}), \tag{11.57}$$

which is the same as (11.27); however, via this path one needs not to assume invertibility of $\mathcal{K}$. An efficient scheme for solving the kernel ridge regression has been developed in [119, 120].

11.10 OPTIMAL MARGIN CLASSIFICATION: SUPPORT VECTOR MACHINES

The optimal classifier, in the sense of minimizing the misclassification error, is the Bayesian classifier as discussed in Chapter 7. The method, being a member of the generative learning family, requires the knowledge of the underlying statistics. If this is not known, an alternative path is to resort to discriminative learning techniques and adopt a discriminant function, f that realizes the corresponding classifier and try to optimize it so as to minimize the respective empirical loss, that is,

$$J(f) = \sum_{n=1}^{N} \mathcal{L}\big(y_n, f(\boldsymbol{x}_n)\big),$$

where

$$y_n = \begin{cases} +1, & \text{if} \quad \boldsymbol{x}_n \in \omega_1, \\ -1, & \text{if} \quad \boldsymbol{x}_n \in \omega_2. \end{cases}$$

For a binary classification task, the first loss function that comes to mind is

$$\mathcal{L}\big(y, f(\boldsymbol{x})\big) = \begin{cases} 1, & \text{if} \quad yf(\boldsymbol{x}) \leq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{11.58}$$

which is also known as the $(0, 1)$-loss function. However, this is a discontinuous function and its optimization is a hard task. To this end, a number of alternative loss functions have been adopted in an effort to approximate the $(0, 1)$-loss function. Recall that the LS loss can also be employed but, as already pointed out in Chapters 3 and 7 this is not well-suited for classification tasks and bears little resemblance with the $(0, 1)$-loss function. In this section, we turn our attention to the so called *hinge* loss function defined as (Chapter 8)

$$\mathcal{L}_\rho\big(y, f(\boldsymbol{x})\big) = \max\big\{0, \rho - yf(\boldsymbol{x})\big\}. \tag{11.59}$$

In other words, if the sign of the product between the true label (y) and that predicted by the discriminant function value ($f(\boldsymbol{x})$) is positive and larger than a threshold/margin (user-defined) value $\rho \geq 0$, the loss is zero. If not, the loss exhibits a linear increase. We say that a margin error is committed if $yf(\boldsymbol{x})$ cannot achieve a value of at least ρ. The hinge loss function is shown in Figure 11.13, together with (0, 1) and squared error loss functions.

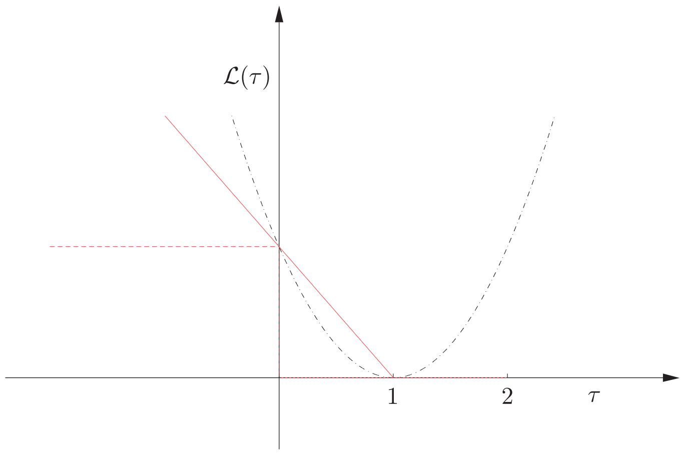

FIGURE 11.13

The $(0, 1)$-loss (dotted red), the hinge loss (red), and the squared error (dotted black) functions tuned to pass through the $(0, 1)$ point for comparison. For the hinge loss, $\rho = 1$ $\tau = yf(\boldsymbol{x})$ for the hinge and $(0, 1)$ loss functions and $\tau = y - f(\boldsymbol{x})$ for the squared error one.

We will constrain ourselves to linear discriminant functions, residing in some RKHS, of the form

$$f(\boldsymbol{x}) = \theta_0 + \langle \theta, \phi(\boldsymbol{x}) \rangle ,$$

where

$$\phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x})$$

is the feature map. However, for the same reasons discussed in Section 11.8.1, we will cast the task as a linear one in the input space, $\mathbb{R}^l$, and at the final stage the kernel information will be "implanted" using the kernel trick.

The goal of designing a linear classifier now becomes equivalent with minimizing the cost

$$J(\boldsymbol{\theta}, \theta_0) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^{N} \mathcal{L}_\rho(y_n, \boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0). \tag{11.60}$$

Alternatively, employing slack variables, and following a similar reasoning as in Section 11.8.1, minimizing (11.60) becomes equivalent to

$$\text{minimize with respect to } \boldsymbol{\theta},\ \theta_0,\ \boldsymbol{\xi} \qquad J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^{N} \xi_n, \tag{11.61}$$

$$\text{subject to} \qquad y_n(\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0) \geq \rho - \xi_n, \tag{11.62}$$

$$\xi_n \geq 0,\ n = 1, 2, \ldots, N. \tag{11.63}$$

From now on, we will adopt the value $\rho = 1$, without harming generality. Indeed, a margin error is committed if $y_n(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) \leq 1$, corresponding to $\xi_n > 0$. On the other hand, if $\xi_n = 0$, then $y_n(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) \geq 1$. Thus, the goal of the optimization task is to drive as many of the ξ_n's to zero as possible. The optimization task in (11.61)–(11.63) has an interesting and important geometric interpretation.

11.10.1 LINEARLY SEPARABLE CLASSES: MAXIMUM MARGIN CLASSIFIERS

Assuming linearly separable classes, there is an infinity of linear classifiers that solve the classification task exactly, without committing errors on the training set (see Figure 11.14a). It is easy to see, and it will become apparent very soon that from this infinity of hyperplanes that solve the task, we can always identify a subset such as

$$y_n(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \ldots, N,$$

which guarantees that $\xi_n = 0, n = 1, 2, \ldots, N$, in (11.61)–(11.63). Hence, for linearly separable classes, the previous optimization task is equivalent to

$$\text{minimize with respect to } \boldsymbol{\theta} \qquad \frac{1}{2}\|\boldsymbol{\theta}\|^2 \tag{11.64}$$

$$\text{subject to} \qquad y_n(\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0) \geq 1,\ n = 1, 2, \ldots, N. \tag{11.65}$$

In other words, from this infinity of linear classifiers, which can solve the task and classify correctly all training patterns, our optimization task selects the one that has minimum norm. As will be explained next, the norm $\|\boldsymbol{\theta}\|$ is directly related to the margin formed by the respective classifier.

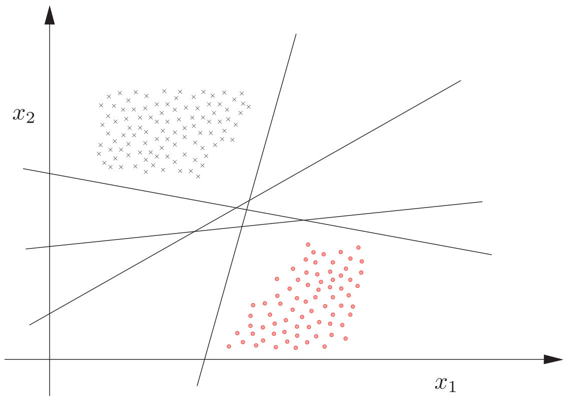

FIGURE 11.14

There is an infinite number of linear classifiers that can classify correctly all the patterns in a linearly separable class task.

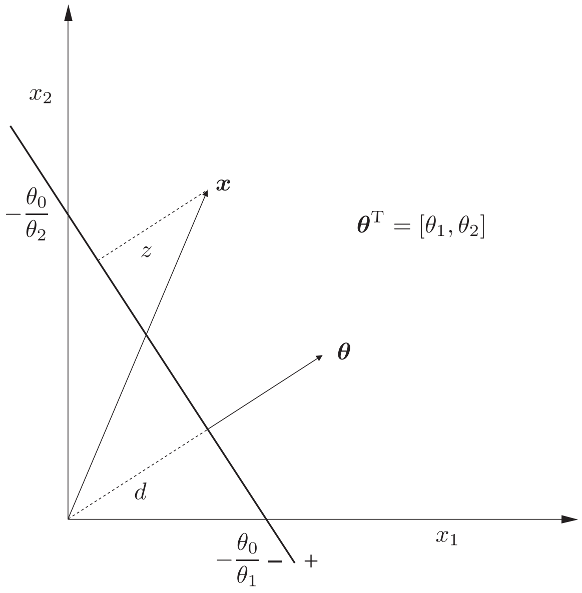
x_2
$-\frac{\theta_0}{\theta_2}$
$\boldsymbol{x}$
z
$\boldsymbol{\theta}^{\mathrm{T}} = [\theta_1, \theta_2]$
$\boldsymbol{\theta}$
d
x_1
$-\frac{\theta_0}{\theta_1}$
$-$
$+$

FIGURE 11.15

The direction of the hyperplane, $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x} + \theta_0 = 0$, is determined by $\boldsymbol{\theta}$ and its position in space by θ_0.

Each hyperplane in space is described by the equation

$$f(\boldsymbol{x}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x} + \theta_0 = 0. \tag{11.66}$$

From classical geometry (see also Problem 5.12), we know that its direction in space is controlled by $\boldsymbol{\theta}$ (which is perpendicular to the hyperplane) and its position is controlled by θ_0, see Figure 11.15.

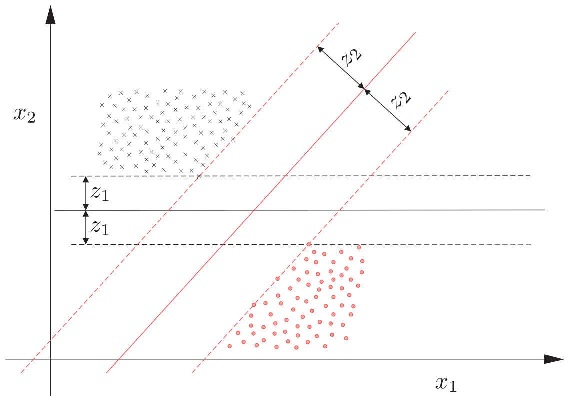

FIGURE 11.16

For each direction, $\boldsymbol{\theta}$, "red" and "gray," the (linear) hyperplane classifier, $\boldsymbol{\theta}^T\boldsymbol{x} + \theta_0 = 0$, (full lines) is placed in between the two classes and normalized so that the nearest points from each class have a distance equal to one. The dotted lines, $\boldsymbol{\theta}^T\boldsymbol{x} + \theta_0 = \pm 1$, which pass through the nearest points, are parallel to the respective classifier, and define the margin. The width of the margin is determined by the direction of the corresponding classifier in space and it is equal to $\frac{2}{||\boldsymbol{\theta}||}$.

From the set of all hyperplanes that solve the task exactly and have certain direction (i.e., they share a common $\boldsymbol{\theta}$), we select θ_0 so as to place the hyperplane in between the two classes, such that its distance from the nearest points from each one of the two classes is the same. Figure 11.16, shows the linear classifiers (hyperplanes) in two different directions (full lines in gray and red). Both of them have been placed so as to have the same distance from the nearest points in both classes. Moreover note that, the distance z_1 associated with the "gray" classifier is smaller than the z_2 associated with the "red" one.

From basic geometry, we know that the distance of a point $\boldsymbol{x}$ from a hyperplane, see Figure 11.15, is given by

$$z = \frac{|\boldsymbol{\theta}^T\boldsymbol{x} + \theta_0|}{\|\boldsymbol{\theta}\|},$$

which is obviously zero if the point lies on the hyperplane. Moreover, we can always scale by a constant factor, say a, both $\boldsymbol{\theta}$ and θ_0 without affecting the geometry of the hyperplane, as described by Eq. (11.66). After an appropriate scaling, we can always make the distance of the nearest points from the two classes to the hyperplane equal to $z = \frac{1}{\|\boldsymbol{\theta}\|}$; equivalently, the scaling guarantees that $f(\boldsymbol{x}) = \pm 1$ if $\boldsymbol{x}$ is a nearest to the hyperplane point and depending on whether the point belongs to ω_1 $(+1)$ or ω_2 (-1). The two hyperplanes, defined by $f(\boldsymbol{x}) = \pm 1$, are shown in Figure 11.16 as dotted lines, for both the "gray" and the "red" directions. The pair of these hyperplanes defines the corresponding margin, for each direction, whose width is equal to $\frac{2}{||\boldsymbol{\theta}||}$.

Thus, any classifier, that is constructed as explained before and which solves the task, satisfies the following two properties:

- It has a *margin* of width equal to $\frac{1}{\|\boldsymbol{\theta}\|} + \frac{1}{\|\boldsymbol{\theta}\|}$

•

$$\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0 \geq +1, \quad \boldsymbol{x}_n \in \omega_1,$$

$$\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0 \leq -1, \quad \boldsymbol{x}_n \in \omega_2.$$

Hence, the optimization task in (11.64)–(11.65) computes the linear classifier, which *maximizes the margin* subject to the constraints.

The margin interpretation of the regularizing term $\|\boldsymbol{\theta}\|^2$ ties nicely the task of designing classifiers, which maximize the margin, with the statistical learning theory and the pioneering work of Vapnik-Chernovenkis, which establishes elegant performance bounds on the generalization properties of such classifiers: see, for example, [27, 125, 128, 129].

The solution

Following similar steps as for the support vector regression case, the solution is given as a linear combination of a subset of the training samples, that is,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N_s} \lambda_n y_n \boldsymbol{x}_n, \tag{11.67}$$

where N_s are the nonzero Lagrange multipliers. It turns out that only the Lagrange multipliers associated with the nearest-to-the-classifier points, that is, those points satisfying the constraints with equality $(y_n(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) = 1)$, are nonzero. These are known as the *support vectors*. The Lagrange multipliers corresponding to the points farther away $(y_n(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) > 1)$ are zero.

For the more general RKHS case, we have that

$$\hat{\theta}(\cdot) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\cdot, \boldsymbol{x}_n),$$

which leads to the following prediction rule. Given an unknown $\boldsymbol{x}$, its class label is predicted according to the sign of

$$\boxed{\hat{y}(\boldsymbol{x}) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\boldsymbol{x}, \boldsymbol{x}_n) + \hat{\theta}_0 : \quad \text{Support Vector Machine Prediction,}} \tag{11.68}$$

where $\hat{\theta}_0$ is obtained by selecting all constraints with $\lambda_n \neq 0$, corresponding to

$$y_n(\hat{\boldsymbol{\theta}}^{\mathrm{T}} \boldsymbol{x}_n + \hat{\theta}_0) - 1 = 0, \quad n = 1, 2, \ldots, N_s,$$

which for the RKHS case becomes

$$y_n \left(\sum_{m=1}^{N_s} \lambda_m y_m \kappa(\boldsymbol{x}_m, \boldsymbol{x}_n) + \hat{\theta}_0 \right) - 1 = 0, \; n = 1, 2, \ldots, N_s.$$

and $\hat{\theta}_0$ is computed as the average of the values obtained from each one of these constraints. Although the solution is unique, the corresponding Lagrange multipliers may not be unique; see, for example, [125].

Finally, it must be stressed that the number of support vectors is related to the generalization performance of the classifier. The *smaller the number of support vectors the better the generalization is expected to be*, [27, 125].

The optimization task

This part can also be bypassed in a first reading.

The task in (11.64)–(11.65) is a quadratic programming task and can be solved following similar steps to those adopted for the SVR task.

The associated Lagrangian is given by

$$L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 - \sum_{n=1}^{N} \lambda_n \left(y_n \left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0\right) - 1\right), \tag{11.69}$$

and the KKT conditions (Appendix C) become

$$\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\lambda}) = \mathbf{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^{N} \lambda_n y_n \boldsymbol{x}_n, \tag{11.70}$$

$$\frac{\partial}{\partial \theta_0} L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\lambda}) = 0 \longrightarrow \sum_{n=1}^{N} \lambda_n y_n = 0, \tag{11.71}$$

$$\lambda_n \big(y_n (\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0) - 1\big) = 0, \; n = 1, 2, \ldots, N, \tag{11.72}$$

$$\lambda_n \geq 0, \; n = 1, 2, \ldots, N. \tag{11.73}$$

The Lagrange multipliers are obtained via the dual representation form after plugging (11.70) into the Lagrangian (Problem 11.11) that is,

$$\text{maximize with respect to } \boldsymbol{\lambda} \quad \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \boldsymbol{x}_n^T \boldsymbol{x}_m, \tag{11.74}$$

$$\text{subject to} \quad \lambda_n \geq 0, \tag{11.75}$$

$$\sum_{n=1}^{N} \lambda_n y_n = 0. \tag{11.76}$$

For the case where the original task has been mapped to an RKHS, the cost function becomes

$$\sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m). \tag{11.77}$$

- According to (11.72), if $\lambda_n \neq 0$, then necessarily

  $$y_n (\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) = 1.$$

  That is, the respective points are the closest points, from each class, to the classifier (distance $\frac{1}{\|\boldsymbol{\theta}\|}$). They lie on either of the two hyperplanes forming the border of the margin. These points are the support vectors and the respective constraints are known as the *active constraints*. The rest of the points, associated with

  $$y_n (\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) > 1, \tag{11.78}$$

  which lie outside the margin, correspond to $\lambda_n = 0$ (*inactive constraints*).

- The cost function in (11.64) is strictly convex and, hence, the solution of the optimization task is *unique* (Appendix C).

11.10.2 NONSEPARABLE CLASSES

We now turn our attention to the more realistic case of overlapping classes and the corresponding geometric representation of the task in (11.61)–(11.63). In this case, there is no (linear) classifier that can classify correctly all the points, and some errors are bound to occur. Figure 11.17 shows the respective geometry for a linear classifier. There are three types of points.

- Points that lie on the border or outside the margin and in the correct side of the classifier, that is,

$$y_n f(\boldsymbol{x}_n) \geq 1.$$

  These points commit no (margin) error, that is,

$$\xi_n = 0.$$

- Points which lie on the correct side of the classifier, but lie inside the margin (circled points), that is,

$$0 < y_n f(\boldsymbol{x}_n) < 1.$$

  These points commit a margin error, and

$$0 < \xi_n < 1.$$

- Points that lie on the wrong side of the classifier (points in squares), that is,

$$y_n f(\boldsymbol{x}_n) \leq 0.$$

  These points commit an error and

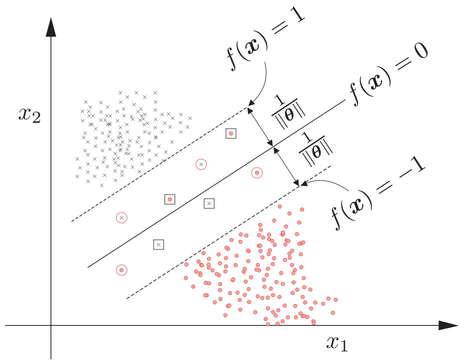

FIGURE 11.17

When classes are overlapping, there are three types of points: (a) points that lie outside or on the borders of the margin and are classified correctly ($\xi_n = 0$); (b) points inside the margin and classified correctly ($0 < \xi_n < 1$) denoted by circles; and (c) misclassified points denoted by a square ($\xi_n \geq 1$).

$$1 \leq \xi_n.$$

Our desire would be to estimate a hyperplane classifier, so as to *maximize the margin and at the same time to keep the number of errors (including margin errors) as small as possible*. This goal could be expressed via the optimization task in (11.61)–(11.62), if in place of ξ_n we had the indicator function, $I(\xi_n)$, where

$$I(\xi) = \begin{cases} 1, & \text{if } \xi > 0, \\ 0, & \text{if } \xi = 0. \end{cases}$$

However, in such a case the task becomes a combinatorial one. So, we relax the task and use ξ_n in place of the indicator function, leading to (11.61)–(11.62). Note that optimization is achieved in a trade-off rationale; the user-defined parameter, C, controls the influence of each of the two contributions to the minimization task. If C is large, the resulting margin (the distance between the two hyperplanes defined by $f(\boldsymbol{x}) = \pm 1$) will be small, in order to commit a smaller number of margin errors. If C is small, the opposite is true. As we will see from the simulation examples, the choice of C is very critical.

The solution

Once more, the solution is given as a linear combination of a subset of the training points,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N_s} \lambda_n y_n \boldsymbol{x}_n, \tag{11.79}$$

where $\lambda_n, \ n = 1, 2, \ldots, N_s$, are the nonzero Lagrange multipliers associated with the support vectors. In this case, support vectors are all points that lie either (a) on the pair of the hyperplanes that define the margin or (b) inside the margin or (c) outside the margin but on the wrong side of the classifier. That is, correctly classified points that lie outside the margin do no contribute to the solution, because the corresponding Lagrange multipliers are zero. Hence, the class prediction rule is the same as in (11.68), where $\hat{\theta}_0$ is computed from the constraints corresponding to $\lambda_n \neq 0$ and $\xi_n = 0$; these correspond to the points that lie on the hyperplanes defining the margin and on the correct side of the classifier.

The optimization task

As before, this part can be bypassed in a first reading.

The Lagrangian associated with (11.61)–(11.63) is given by

$$L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\xi}, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} \mu_n \xi_n$$
$$- \sum_{n=1}^{N} \lambda_n \left(y_n \left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0 \right) - 1 + \xi_n \right),$$

leading to the following KKT conditions,

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \boldsymbol{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^{N} \lambda_n y_n \boldsymbol{x}_n, \tag{11.80}$$

$$\frac{\partial L}{\partial \theta_0} = 0 \longrightarrow \sum_{n=1}^{N} \lambda_n y_n = 0, \tag{11.81}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \longrightarrow C - \mu_n - \lambda_n = 0, \tag{11.82}$$

$$\lambda_n \big(y_n(\boldsymbol{\theta}^T \boldsymbol{x}_n + \theta_0) - 1 + \xi_n \big) = 0, \; n = 1, 2, \ldots, N, \tag{11.83}$$

$$\mu_n \xi_n = 0, \; n = 1, 2, \ldots, N, \tag{11.84}$$

$$\mu_n \geq 0, \; \lambda_n \geq 0, \; n = 1, 2, \ldots, N, \tag{11.85}$$

and in our by-now-familiar procedure, the dual problem is cast as

$$\text{maximize with respect to } \boldsymbol{\lambda} \quad \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \boldsymbol{x}_n^T \boldsymbol{x}_m \tag{11.86}$$

$$\text{subject to} \quad 0 \leq \lambda_n \leq C, \; n = 1, 2, \ldots, N, \tag{11.87}$$

$$\sum_{n=1}^{N} \lambda_n y_n = 0. \tag{11.88}$$

When working in an RKHS, the cost function becomes

$$\sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m).$$

Observe that the only difference compared to its linearly class-separable counterpart in (11.74)–(11.76) is the existence of C in the inequality constraints for λ_n. The following comments are in order:

- From (11.83), we conclude that for all the points outside the margin, and on the correct side of the classifier, which correspond to $\xi_n = 0$, we have

$$y_n(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \theta_0) > 1,$$

  hence, $\lambda_n = 0$. That is, these points do not participate in the formation of the solution in (11.80).
- $\lambda_n \neq 0$ only for the points that live either on the border hyperplanes or inside the margin or outside the margin but on the wrong side of the classifier. These comprise the support vectors.
- For points lying inside the margin or outside but on the wrong side, $\xi_n > 0$; hence, from (11.84), $\mu_n = 0$ and from (11.82) we get,

$$\lambda_n = C.$$

- Support vectors, which lie on the margin border hyperplanes, satisfy $\xi_n = 0$ and therefore μ_n can be nonzero, which leads to

$$0 \leq \lambda_n \leq C.$$

  Remarks 11.5.

- *ν-SVM*: An alternative formulation for the SVM classification has been given in [105], where the margin is defined by the pair of hyperplanes,

$$\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x} + \theta_0 = \pm \rho,$$

and $\rho \geq 0$ is left as a free variable, giving rise to the ν-SVM; ν controls the importance of ρ in the associated cost function. It has been shown, [21], that the ν-SVM and the formulation discussed above, which is sometimes referred to as the C-SVM, lead to the same solution for appropriate choices of C and ν. However, the advantage of ν-SVM lies in the fact that ν can be directly related to bounds concerning the number of support vectors and the corresponding error rate, see also [125].

- *Reduced Convex Hull Interpretation*: In [57], it has been shown that, for linearly separable classes, the SVM formulation is equivalent with finding the nearest points between the convex hulls, formed by the data in the two classes. This result was generalized for overlapping classes in [32]; it is shown that in this case, the ν-SVM task is equivalent to searching for the nearest points between the *reduced convex hulls* (RCH), associated with the training data. Searching for the RCH is a computationally hard task of combinatorial nature. The problem was efficiently solved in [74–76, 123], who came up with efficient iterative schemes to solve the SVM task, via nearest point searching algorithms. More on these issues can be obtained from [66, 124, 125].
- *ℓ_1-Regularized Versions*: The regularization term, which has been used in the optimization tasks discussed so far, has been based on the ℓ_2 norm. A lot of research effort has been focused on using ℓ_1 norm regularization for tasks treating the linear case. To this end, a number of different loss functions have been used in addition to the least-squares, the hinge loss, and the ϵ-insensitive versions, as for example the logistic loss. The solution of such tasks comes under the general framework discussed in Chapter 8. As a matter of fact, some of these methods have been discussed there. A related concise review is provided in [132].
- *Multitask Learning*: In multitask learning, two or more related tasks, for example, classifiers, are jointly optimized. Such problems are of interest in, for example, econometrics, and bioinformatics. In [38], it is shown that the problem of estimating many task functions with regularization can be cast as a single task learning problem if a family of appropriately defined multitask kernel functions is used.

Example 11.4. In this example, the performance of the SVM is tested in the context of a two-class two-dimensional classification task. The data set comprises $N = 150$ points uniformly distributed in the region $[-5, 5] \times [-5, 5]$. For each point, $\boldsymbol{x}_n = [x_{n,1},\ x_{n,2}]^T$, $n = 1, 2, \dots, N$, we compute

$$y_n = 0.5x_{n,1}^3 + 0.5x_{n,1}^2 + 0.5x_{n,1} + 1 + \eta,$$

where η stands for zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. The point is assigned to either of the two classes, depending on which side of the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space, y_n lies. That is, if $y_n > f(x_{n1})$, the point is assigned to class ω_1; otherwise, it is assigned to class ω_2.

The Gaussian kernel was used with $\sigma = 10$, as this resulted in the best performance. Figure 11.18a shows the obtained classifier for $C = 20$ and Figure 11.18b for $C = 1$. Observe how the obtained classifier, and hence the performance, depends heavily on the choice of C. In the former case, the number of the support vectors was equal to 64 and for the latter equal to 84.

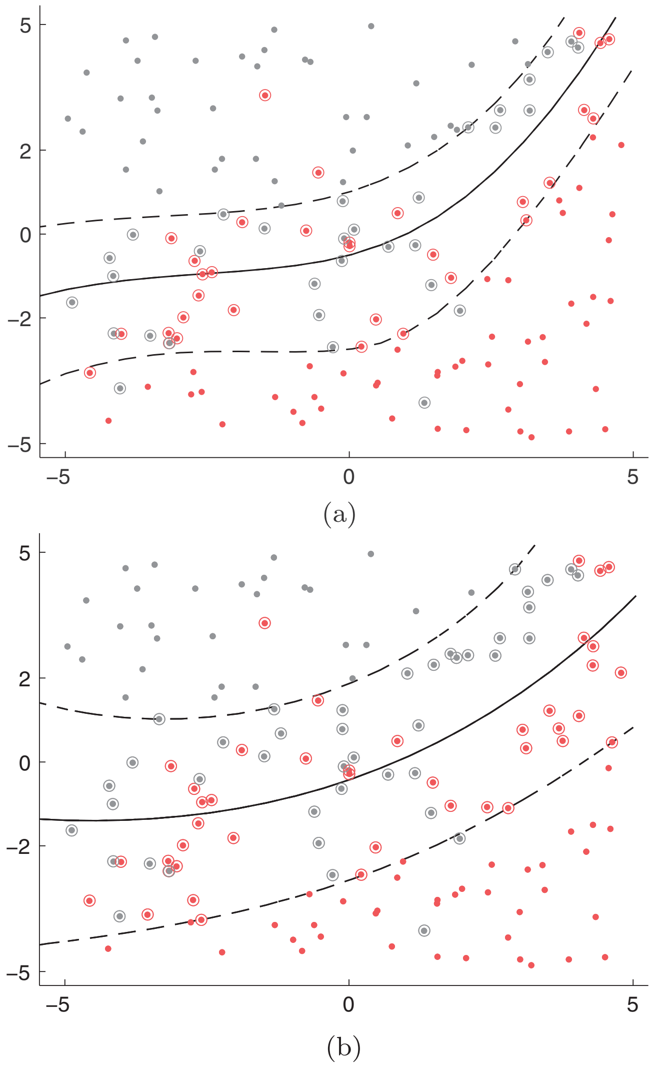

FIGURE 11.18

(a) The training data points for the two classes (red and gray respectively) of Example 11.4. The full line is the graph of the obtained SVM classifier and the dotted lines indicate the margin, for $C = 20$. (b) The result for $C = 1$. For both cases, the Gaussian kernel with $\sigma = 20$ was used.

11.10.3 PERFORMANCE OF SVMs AND APPLICATIONS

A notable characteristic of the support vector machines is that the complexity is independent of the dimensionality of the respective RKHS. The need of having a large number of parameters is bypassed and this has an influence on the generalization performance; SVMs exhibit very good generalization performance in practice. Theoretically, such a claim is substantiated by their maximum margin interpretation, in the framework of the elegant *structural risk minimization theory*, [27, 125, 128].

An extensive comparative study concerning the performance of SVMs against 16 other popular classifiers has been reported in [78]. The results verify that the SVM ranks at the very top among these classifiers, although there are cases for which other methods score better performance. Another comparative performance study is reported in [23].

It is hard to find a discipline related to machine learning/pattern recognition where the support vector machines and the concept of working in kernel spaces has not been applied. Early applications included data mining, spam categorization, object recognition, medical diagnosis, optical character recognition (OCR), bioinformatics; see, for example, [27] for a review. More recent applications include cognitive radio, for example, [34], spectrum cartography and network flow prediction [9], and image de-noising [13].

In [118], the notion of *kernel embedding* of conditional probabilities is reviewed, as a means to address challenging problems in graphical models. The notion of kernelization has also been extended in the context of *tensor-based* models, [49, 108, 133]. Kernel-based hypothesis testing is reviewed in [48]. In [122], the use of kernels in manifold learning is discussed in the framework of diffusion maps. The task of analyzing the performance of kernel techniques with regard to dimensionality, signal-to-noise ratio and local error bars is reviewed in [82]. In [121], a collection of articles related to kernel-based methods and applications is provided.

11.10.4 CHOICE OF HYPERPARAMETERS

One of the main issues associated with SVM/SVRs is the choice of the parameter, C, which controls the relative influence of the loss and the regularizing parameter in the cost function. Although some efforts have been made in developing theoretical tools for the respective optimization, the path that has survived in practice is that of cross-valuation techniques against a test data set. Different values of C are used to train the model, and the value that results in the best performance over the test set is selected.

The other main issue is the choice of the kernel function. Different kernels lead to different performance. Let us look carefully at the expansion in (11.17). One can think of $\kappa(\boldsymbol{x}, \boldsymbol{x}_n)$ as a function that measures the *similarity* between $\boldsymbol{x}$ and $\boldsymbol{x}_n$; in other words, $\kappa(\boldsymbol{x}, \boldsymbol{x}_n)$ *matches* $\boldsymbol{x}$ to the training sample $\boldsymbol{x}_n$. A kernel is local if $\kappa(\boldsymbol{x}, \boldsymbol{x}_n)$ takes relatively large values in a small region around $\boldsymbol{x}_n$. For example, when the Gaussian kernel is used, the contribution of $\kappa(\boldsymbol{x}, \boldsymbol{x}_n)$ away from $\boldsymbol{x}_n$ decays exponentially fast, depending of the value of σ^2. Thus, the choice of σ^2 is very crucial. If the function to be approximated is smooth, then large values of σ^2 should be employed. On the contrary, if the function is highly varying in input space, the Gaussian kernel may not be the best choice. As a matter of fact, if for such cases the Gaussian kernel is employed, one must have access to a large number of training data, in order to fill in the input space densely enough, so as to be able to obtain a good enough approximation of such a function. This brings into the scene another critical factor in machine learning, related to the size of the training set. The latter is not only dictated by the dimensionality of the input space (curse of dimensionality) but it also depends on the type of variation that the unknown function undergoes (see, for example, [12]). In practice, in order to choose the right kernel function, one uses different kernels and after cross validation selects the "best" one for the specific problem.

A line of research is to design kernels that match the data at hand, based either on some prior knowledge or via some optimization path; see, for example, [28, 65]. Soon, in Section 11.13 we will discuss techniques which use multiple kernels in an effort to optimally combine their individual characteristics.

11.11 COMPUTATIONAL CONSIDERATIONS

Solving a quadratic programming task, in general, requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ memory operations. To cope with such demands a number of decomposition techniques have been devised, for example, [20, 54], which "break" the task into a sequence of smaller ones. In [58, 90, 91], the *sequential minimal optimization* (SMO) algorithm breaks the task into a sequence of problems comprising two points, which can be solved analytically. Efficient implementation of such schemes lead to an empirical training time that scales between $\mathcal{O}(N)$ and $\mathcal{O}(N^{2.3})$.

The schemes derived in [75, 76] treat the task as a minimum distance points search between reduced convex hulls and end up with an iterative scheme, which projects the training points on hyperplanes. The scheme leads to even more efficient implementations compared to [58, 90]; moreover, the minimum distance search algorithm has a built-in enhanced parallelism.

The issue of parallel implementation is also discussed in [19]. Issues concerning complexity and accuracy are reported in [52]. In the latter, polynomial time algorithms are derived that produce approximate solutions with a guaranteed accuracy for a class of QP problems including SVM classifiers.

Incremental versions for solving the SVM task, which are dealing with sequentially arriving data have also appeared, for example, [24, 35, 101]. In the latter, at each iteration a new point is considered and the previously selected set of support vectors (active set) is updated accordingly by adding/removing samples.

Online versions that apply in the primal problem formulation, have also been proposed. In [85], an iterative reweighted LS approach is followed that alternates weight optimization with cost constraint forcing. A structurally and computationally simple scheme, named *PEGASOS: Primal Estimated sub-Gradient SOlver for SVM*, has been proposed in [106]. The algorithm is of an iterative subgradient form applied on the regularized empirical hinge loss function in (11.60) (see also, Chapter 8). The algorithm, for the case of linear kernels, exhibits very good convergence properties and it finds an ϵ-accurate solution in $\mathcal{O}\left(\frac{C}{\epsilon}\right)$ iterations.

In [41, 55] the classical technique of *cutting planes* for solving convex tasks has been employed in the context of SVMs in the primal domain. The resulting algorithm is very efficient and, in particular for the linear SVM case, the complexity becomes of order $O(N)$.

In [25], a comparative study between solving the SVM tasks in the primal and dual domains is carried out. The findings of the paper point out that both paths are equally efficient, for the linear as well as for the nonlinear cases. Moreover, when the goal is to resort to approximate solutions, opting for the primal task can offer certain benefits. In addition, working in the primal also offers the advantage of tuning the hyperparameters by resorting to joint optimization. One of the main advantages of resorting to the dual domain was the luxury of casting the task in terms of inner products. However, this is also possible in the primal, by appropriate exploitation of the representer theorem. We will see such examples soon, in Section 11.12.

More recently, a version of SVM for distributed processing has been presented in [40]. In [100] the SVM task is solved in a subspace using the method of random projections.

11.11.1 MULTICLASS GENERALIZATIONS

The SVM classification task has been introduced in the context of a two-class classification task. The more general M-class case can be treated in various ways:

- *One-against-All*: One solves M two-class problems. Each time, one of the classes is classified against all the others using a different SVM. Thus, M classifiers are estimated, that is,

  $$f_m(\boldsymbol{x}) = 0, \ m = 1, 2, \ldots, M,$$

  which are trained so that $f_m(\boldsymbol{x}) > 0$ for $\boldsymbol{x} \in \omega_m$ and $f_m(\boldsymbol{x}) < 0$ if $\boldsymbol{x}$ otherwise. Classification is achieved via the rule

  $$\text{assign } \boldsymbol{x} \text{ in } \omega_k : \quad \text{if } k = \arg\max_m f_m(\boldsymbol{x}).$$

  According to this method, there may be regions in space where more than one of the discriminant functions score a positive value, [125]. Moreover, another disadvantage of this approach is the so-called class imbalance problem; this may be caused by the fact that the number of training points in one of the classes (which comprises the data from $M-1$ classes) is much larger than the points in the other. Issues related to the class imbalance problem are discussed in [125].
- *One-against-one*. According to this method, one solves $\frac{M(M-1)}{2}$ binary classification tasks by considering all classes in pairs. The final decision is taken on the basis of the majority rule.
- In [129], the SVM rationale is extended in estimating simultaneously M hyperplanes. However, this technique ends up with a large number of parameters, equal to $N(M-1)$, which have to be estimated via a single minimization task; this turns out to be rather prohibitive for most practical problems.
- In [33], the multiclass task is treated in the context of error correcting codes. Each class is associated with a binary code word. If the code words are properly chosen, an error resilience is "embedded" into the process; see also [125]. For a comparative study of multiclass classification schemes, see, for example, [39] and the references therein.
- *Division and Clifford Algebras*: The SVM framework has also been extended to treat complex and hypercomplex data, both for the regression as well as the classification cases, using either division algebras [101], or Clifford algebras [8]. In [126], the case of quaternion RKH spaces is considered.

  A more general method for the case of complex-valued data, which exploits the notion of *widely linear estimation* as well as *pure complex kernels*, has been presented in [14]. In this paper, it is shown that any complex SVM/SVR task is equivalent with solving two real SVM/SVR tasks exploiting a specific real kernel, which is generated by the chosen complex one. Moreover, in the classification case, it is shown that the proposed framework inherently splits the complex space into four parts. This leads naturally to solving the four-class task (quaternary classification), instead of the standard two-classes scenario of the real SVM. This rationale can be used in a multiclass problem as a split-class scenario.

11.12 ONLINE LEARNING IN RKHS

We have dealt with online learning in various parts of the book. Most of the algorithms that have been discussed can also be stated for general Hilbert spaces. The reason that we are going to dedicate this section specifically to RKHS is that developing online algorithms for such spaces poses certain computational obstacles. In parametric modeling in a Euclidean space $\mathbb{R}^l$, the number of unknown parameters remains fixed for all time instants. In contrast, modeling an unknown function to lie in an RKHS, makes the number of unknown parameters grow linearly with time; thus, complexity increases with time iterations and eventually will become unmanageable both in memory as well as in the number of operations.

We will discuss possible solutions to this problem in the context of three algorithms, which are popular and fall nicely within the framework that has been adopted throughout this book.

11.12.1 THE KERNEL LMS (KLMS)

This section is intended for more experienced readers, so we will cast the task in a general RKHS space, $\mathbb{H}$.

Let $\boldsymbol{x} \in \mathbb{R}^l$ and consider the feature map

$$\boldsymbol{x} \longmapsto \phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x}) \in \mathbb{H}.$$

The task is to estimate $f \in \mathbb{H}$, so as to minimize the expected risk

$$J(f) = \frac{1}{2}\,\mathbb{E}\left[|\mathrm{y} - \langle f, \phi(\mathbf{x})\rangle|^2\right], \tag{11.89}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product operation in $\mathbb{H}$. Differentiating[11] with respect to f, it turns out that

$$\nabla_f J(f) = -\,\mathbb{E}\left[\phi(\mathbf{x})\,(\mathrm{y} - \langle f, \phi(\mathbf{x})\rangle)\right].$$

Adopting the stochastic gradient rationale, and replacing random variables with observations, the following time update recursion for minimizing the expected risk results,

$$\begin{aligned} f_n &= f_{n-1} + \mu_n e_n \phi(\boldsymbol{x}_n), \\ &= f_{n-1} + \mu_n e_n \kappa(\cdot, \boldsymbol{x}_n), \end{aligned} \tag{11.90}$$

where

$$e_n = y_n - \langle f_{n-1}, \phi(\boldsymbol{x}_n)\rangle\,,$$

with $(y_n, \boldsymbol{x}_n)$ $n = 1, 2, \ldots$ being the received observations. Starting the iterations from $f_0 = 0$ and fixing $\mu_n = \mu$, as in the standard LMS, we obtain[12]

$$f_n = \mu \sum_{i=1}^{n} e_i \kappa(\cdot, \boldsymbol{x}_i), \tag{11.91}$$

[11] Differentiation here is in the context of Frechet derivatives. In analogy to the gradient, it turns out that $\nabla_f \langle f, g\rangle = g$, for $f, g \in \mathbb{H}$.

[12] In contrast to the LMS in Chapter 5 the time starts at $n = 1$, and not at $n = 0$, so as to be in line with the notation used in this chapter.

and the prediction of the output, based on information (training samples) received up to and including time instant $n-1$, is given by

$$\hat{y}_n = \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_n)\rangle$$
$$= \mu \sum_{i=1}^{n-1} e_i \kappa(\boldsymbol{x}_n, \boldsymbol{x}_i).$$

Note that the same equation results, if one starts from the standard form of LMS expressed in $\mathbb{R}^l$, and applies the kernel trick at the final stage (try it). Observe that (11.91) is in line with the representer theorem. Thus, replacing μe_i with θ_i, and plugging (11.91) in (11.90), it turns out that at every time instant, the KLMS comprises the following steps:

$$\boxed{\begin{aligned} e_n &= y_n - \hat{y}_n = y_n - \sum_{i=1}^{n-1} \theta_i \kappa(\boldsymbol{x}_n, \boldsymbol{x}_i), \\ \theta_n &= \mu e_n. \end{aligned}} \tag{11.92}$$

The *normalized* KLMS results if the update becomes

$$\theta_n = \mu \frac{e_n}{\kappa(\boldsymbol{x}_n, \boldsymbol{x}_n)}.$$

Observe that the memory grows unbounded. So, in practice, a *sparsification* rule has to be applied. There are various strategies that have been proposed. One is via regularization and the other one is via the formation of a *dictionary*. In the latter case, instead of using in the expansion in (11.91) all the training points up to time n, a subset is selected. This subset forms the dictionary, $\mathcal{D}_n$, at time n. Starting from an empty set, $\mathcal{D}_0 = \emptyset$, the dictionary can grow following different strategies. Some typical examples are

- *Novelty criterion* [68]:
  - Let us denote the current dictionary as $\mathcal{D}_{n-1}$, its cardinality by M_{n-1}, and its elements as $\boldsymbol{u}_k$, $k = 1, 2, \ldots, M_{n-1}$. When a new observation arrives, its distance from all the points in the $\mathcal{D}_{n-1}$ is evaluated

    $$d(\boldsymbol{x}_n, \mathcal{D}_{n-1}) = \min_{\boldsymbol{u}_k \in \mathcal{D}_{n-1}} \{\|\boldsymbol{x}_n - \boldsymbol{u}_k\|\}.$$

    If this distance is smaller than a threshold, δ_1, then the new point is ignored and $\mathcal{D}_n = \mathcal{D}_{n-1}$.
  - If not, the error is computed

    $$e_n = y_n - \hat{y}_n = y_n - \sum_{k=1}^{M_{n-1}} \theta_k \kappa(\boldsymbol{x}_n, \boldsymbol{u}_k). \tag{11.93}$$

    If $|e_n| < \delta_s$, where δ_s is a threshold, then the point is discarded. If not, $\boldsymbol{x}_n$ is inserted in the dictionary and

    $$\mathcal{D}_n = \mathcal{D}_{n-1} \bigcup \{\boldsymbol{x}_n\}.$$

- *The Coherence Criterion*: According to this scheme, the point $\boldsymbol{x}_n$ is added in the dictionary if its *coherence* is above a given threshold, ϵ_0, that is,

$$\max_{\boldsymbol{u}_k \in \mathcal{D}_{n-1}} \{|\kappa(\boldsymbol{x}_n, \boldsymbol{u}_k)|\} > \epsilon_0.$$

It can be shown [96] that, under this rule, the cardinality of $\mathcal{D}_n$ remains *finite*, as $n \longrightarrow \infty$.

- *Surprise Criterion* [69]: The *surprise* of a new pair $(y_n, \boldsymbol{x}_n)$ with respect to a learning system, $\mathcal{T}$, is defined as the log-likelihood of $(y_n, \boldsymbol{x}_n)$, i.e.,

$$S_{\mathcal{T}}(y_n, \boldsymbol{x}_n) = -\ln p\big((y_n, \boldsymbol{x}_n)|\mathcal{T}\big).$$

  According to this measure, a data pair is classified to either of the following three categories:
  - *Abnormal*: $S_{\mathcal{T}}(y_n, \boldsymbol{x}_n) > \delta_1$,
  - *Learnable*: $\delta_1 \geq S_{\mathcal{T}}(y_n, \boldsymbol{x}_n) \geq \delta_2$,
  - *Redundant*: $S_{\mathcal{T}}(y_n, \boldsymbol{x}_n) < \delta_2$, where δ_1, δ_2 are threshold values. For the case of the LMS with Gaussian inputs, the following is obtained,

$$S_{\mathcal{T}}(y_n, \boldsymbol{x}_n) = \frac{1}{2}\ln(r_n) + \frac{e_n^2}{2r_n},$$

    where

$$r_n = \lambda + \kappa(\boldsymbol{x}_n, \boldsymbol{x}_n) - \max_{\boldsymbol{u}_k \in \mathcal{D}_{n-1}} \left\{ \frac{\kappa^2(\boldsymbol{x}_n, \boldsymbol{u}_k)}{\kappa(\boldsymbol{u}_k, \boldsymbol{u}_k)} \right\},$$

    where λ is a user-defined regularization parameter.

A main drawback of all the previous techniques is that once a data point (e.g., $\boldsymbol{x}_k$) is inserted in the dictionary, it remains there forever; also, the corresponding coefficients, $\boldsymbol{\theta}_k$, in the expansion in (11.93) do not change. This can affect the tracking ability of the algorithm in time-varying environments.

A different technique, which gives the chance of changing the respective weights of the points in the dictionary, is the *quantization* technique, giving rise to the so-called *quantized* KLMS, given in Algorithm 11.1 ([26]).

Algorithm 11.1 (The quantized kernel LMS).

```
• Initialize
    • $\mathcal{D} = \emptyset, M = 0$.
    • Select $\mu$ and then the quantization level $\delta$.
    • $d(\boldsymbol{x}, \emptyset) := \Delta > \delta, \forall \boldsymbol{x}$.
• For $n = 1, 2, ...,$ Do
    • If $n = 1$ then
        - $\hat{y}_n = 0$
    • else
        - $\hat{y}_n = \sum_{k=1:\boldsymbol{u}_k \in \mathcal{D}}^{M} \theta_k \kappa(\boldsymbol{x}_n, \boldsymbol{u}_k)$
    • End If
    • $e_n = y_n - \hat{y}_n$
    • $d(\boldsymbol{x}_n, \mathcal{D}) = \min_{\boldsymbol{u}_k \in \mathcal{D}} \|\boldsymbol{x}_n - \boldsymbol{u}_k\| = \|\boldsymbol{x}_n - \boldsymbol{u}_{l_0}\|$ for some $l_0 \in \{1, 2, \ldots, M\}$
    • If $d(\boldsymbol{x}_n, \mathcal{D}) > \delta$, then
        - $\theta_n = \mu e_n$;   or $\theta_n = \frac{\mu e_n}{\kappa(\boldsymbol{x}_n, \boldsymbol{x}_n)}$, for NKLMS.
        - $M = M + 1$
        - $\boldsymbol{u}_M = \boldsymbol{x}_n$
        - $\mathcal{D} = \mathcal{D} \bigcup \{\boldsymbol{x}_M\}$
        - $\boldsymbol{\theta} = \left[\boldsymbol{\theta}^T, \theta_n\right]^T$; increase dimensionality of $\boldsymbol{\theta}$ by one.
    • Else
        - Keep dictionary unchanged.
        - $\theta_{l_0} = \theta_{l_0} + \mu e_n$; Update the weight of the nearest point.
    • End If
• End For
```

The algorithm returns $\boldsymbol{\theta}$ as well as the dictionary, hence

$$f_n(\cdot) = \sum_{k=1}^{M} \theta_k \kappa(\cdot, \boldsymbol{u}_k)$$

and

$$\hat{y}_n = \sum_{k=1}^{M} \theta_k \kappa(\boldsymbol{x}_n, \boldsymbol{u}_k).$$

The KLMS without the sparsification approximation can be shown to converge asymptotically, in the mean sense, to the value that optimizes the mean-square cost function in (11.89) for small values of μ, [68]. A stochastic analysis of the KLMS for the case of the Gaussian kernel has been carried in [88].

An interesting result concerning the KLMS was derived in [68]. It has been shown that the KLMS trained on a *finite* number of points, N, turns out to be the stochastic approximation of the following constrained optimization task,

$$\min_f J(f) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle f, \phi(\boldsymbol{x}_i) \rangle)^2 \tag{11.94}$$

$$\text{s.t.} \quad \|f\|^2 \leq C, \tag{11.95}$$

where the value of C can be computed analytically. This result builds upon the H^∞ optimality of the LMS, as discussed in Chapter 5. Note that this is equivalent to having used regularization in (11.89). The constraint in (11.95) is in line with the conditions obtained from the regularization network theory [45, 92], for ensuring that the problem is well posed (Chapter 3) and is sufficient for consistency of the empirical error minimization forcing smoothness and stability [92]. In a nutshell, the KLMS is well-posed in an RKHS, without the need of an extra regularization term to penalize the empirical loss function.

11.12.2 THE NAIVE ONLINE R_{reg} MINIMIZATION ALGORITHM (NORMA)

The KLMS algorithm is a stochastic gradient algorithm for the case of the squared error loss function. In the current section, our interest moves to more general convex loss functions, such as those discussed in Chapter 8. The squared error loss function is just such an instance.

Given the loss function

$$\mathcal{L} : \mathbb{R} \times \mathbb{R} \longmapsto [0, +\infty),$$

the goal is to obtain an $f \in \mathbb{H}$, where $\mathbb{H}$ is an RKHS defined by a kernel $\kappa(\cdot, \cdot)$, such as to minimize the expected risk

$$J(f) = \mathbb{E}\left[\mathcal{L}\left(y, f(\mathbf{x})\right)\right]. \tag{11.96}$$

Instead, we turn our attention to selecting, f, so as to minimize the regularized empirical risk over the available training set

$$J_{emp,\lambda}(f, N) = J_{emp}(f, N) + \frac{\lambda}{2}\|f\|^2, \tag{11.97}$$

where

$$J_{emp}(f, N) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(y_n, f(\boldsymbol{x}_n)\right).$$

Following the same rationale as the one that has been adopted for finite dimensional (Euclidean) spaces, in order to derive stochastic gradient algorithms, the *instantaneous* counterpart of (11.97) is defined as

$$\boxed{\mathcal{L}_{n,\lambda}(f) := \mathcal{L}\left(y_n, f(\boldsymbol{x}_n)\right) + \frac{\lambda}{2}\|f\|^2: \quad \text{Instantaneous Loss}} \tag{11.98}$$

and it is used in the time-recursive rule for searching for the optimum, that is,

$$f_n = f_{n-1} - \mu_n \frac{\partial}{\partial f}\mathcal{L}_{n,\lambda}(f)|_{f=f_{n-1}},$$

where $\frac{\partial}{\partial f}$ denotes the (sub)gradient with regard to f. However, note that $f(\boldsymbol{x}_n) = \langle f, \kappa(\cdot, \boldsymbol{x}_n)\rangle$. Applying the chain rule for differentiation, and recalling that

$$\frac{\partial}{\partial f}\langle f, \kappa(\cdot, \boldsymbol{x}_n)\rangle = \kappa(\cdot, \boldsymbol{x}_n),$$

and

$$\frac{\partial}{\partial f}\langle f, f\rangle = 2f,$$

we obtain that

$$\boxed{f_n = (1 - \mu_n\lambda)f_{n-1} - \mu_n\mathcal{L}'\left(y_n, f_{n-1}(\boldsymbol{x}_n)\right)\kappa(\cdot, \boldsymbol{x}_n),} \tag{11.99}$$

where $\mathcal{L}'(y, z) := \frac{\partial}{\partial z}\mathcal{L}(y, z)$. If $\mathcal{L}(\cdot, \cdot)$ is not differentiable, $\mathcal{L}'(\cdot, \cdot)$ denotes any subgradient of $\mathcal{L}(\cdot, \cdot)$. Assuming, $f_0 = 0$ and applying (11.99) recursively, we obtain

$$f_n = \sum_{i=1}^{n}\theta_i\kappa(\cdot, \boldsymbol{x}_i). \tag{11.100}$$

Moreover, from (11.99) at time n, we obtain the equivalent time update of the corresponding coefficients, that is,

$$\theta_n = -\mu_n\mathcal{L}'\left(y_n, f_{n-1}(\boldsymbol{x}_n)\right), \tag{11.101}$$

$$\theta_i^{\text{new}} = (1 - \mu_n\lambda)\theta_i, \; i < n. \tag{11.102}$$

Observe that for $\lambda = 0$, $\mu_n = \mu$ and $\mathcal{L}(\cdot, \cdot)$ being the squared loss ($\frac{1}{2}(y - f(\boldsymbol{x}))^2$), (11.101) and (11.102) break down into (11.92).

From the recursion in (11.99), it is readily apparent that a necessary condition that guarantees convergence is

$$\mu_n < \frac{1}{\lambda}, \quad \lambda > 0, \quad n = 1, 2, \ldots$$

Let us now set $\mu_n = \mu < \frac{1}{\lambda}$; then, combining (11.100)–(11.102), it is easy to show recursively (Problem 11.12) that

$$f_n = -\sum_{i=1}^{n} \mu \mathcal{L}^{'}\left(y_i, f_{i-1}(\boldsymbol{x}_i)\right)(1 - \mu\lambda)^{n-i}\kappa(\cdot, \boldsymbol{x}_i).$$

Observe that the effect of *regularization* is equivalent to imposing an *exponential forgetting factor* on past data. Thus, we can select a constant n_0, sufficiently large, and keep in the expansion only the n_0 terms in the time window $[n - n_0 + 1, n]$. In this way, we achieve the propagation of a *fixed* number of parameters at every time instant, at the expense of an approximation/truncation error (Problem 11.13). The resulting scheme is summarized in Algorithm 11.2 [61].

Algorithm 11.2 (The NORMA algorithm).

```
• Initialize
    • Select $\lambda$ and $\mu$, $\mu < \frac{1}{\lambda}$.
• For $n = 1, 2, ...,$ Do
    • If $n = 1$ then
        - $f_0 = 0$
    • else
        - $f_{n-1}(\boldsymbol{x}_n) = -\sum_{i=\max\{1,n-n_0\}}^{n-1} \mu \mathcal{L}^{'}\left(y_i, f_{i-1}(\boldsymbol{x}_i)\right)(1-\mu\lambda)^{n-i-1}\kappa(\boldsymbol{x}_n, \boldsymbol{x}_i)$
    • End If
    • $\theta_n = -\mu \mathcal{L}'\left(y_n, f_{n-1}(\boldsymbol{x}_n)\right)$
• End For
```

Note that if the functional form involves a constant bias term, that is, $\theta_0 + f(\boldsymbol{x})$, $f \in \mathbb{H}$, then the update of θ_0 follows the standard form

$$\theta_0(n) = \theta_0(n-1) - \mu \frac{\partial}{\partial \theta_0} \mathcal{L}\big(y_n, \theta_0 + f_{n-1}(\boldsymbol{x}_n)\big)\big|_{\theta_0(n-1)},$$

with $\frac{\partial}{\partial \theta_0}$ the (sub)gradient with respect to θ_0.

Classification: the hinge loss function

The hinge loss function was defined in (11.59) and its subgradient is given by

$$\mathcal{L}'_\rho\big(y, f(\boldsymbol{x})\big) = \begin{cases} -y, & yf(\boldsymbol{x}) \le \rho, \\ 0, & \text{otherwise.} \end{cases}$$

Note that at the discontinuity, one of the subgradients is equal to 0, which is the one employed in the algorithm. This is plugged into Algorithm 11.2 in place of $\mathcal{L}'\left(y_n, f_{n-1}(\boldsymbol{x}_n)\right)$ to result in

$$\theta_n = \mu \sigma_n y_n, \; \sigma_n = \begin{cases} 1, & y_n f(\boldsymbol{x}_n) \le \rho, \\ 0, & \text{otherwise,} \end{cases} \tag{11.103}$$

and if a bias term is present

$$\theta_0(n) = \theta_0(n-1) + \mu\sigma_n y_n.$$

If ρ is left as a free parameter, the online version of the ν-SVM [105] results, (Problem 11.14).

Regression: the linear ϵ-insensitive loss function

For this loss function, defined in (11.28), the subgradient is easily shown to be

$$\mathcal{L}'\big(y, f(\boldsymbol{x})\big) = \begin{cases} -\operatorname{sgn}\{y - f(\boldsymbol{x})\}, & \text{if } |y - f(\boldsymbol{x})| > \epsilon, \\ 0, & \text{otherwise.} \end{cases}$$

Note that at the two points of discontinuity, the zero is a possible value for the subgradient and the corresponding update in Algorithm 11.2 becomes

$$\theta_n = \mu \operatorname{sgn}\{y_n - f_{n-1}(\boldsymbol{x}_n)\}. \tag{11.104}$$

A variant of this algorithm is also presented in [61] by considering ϵ to be a free parameter and leave the algorithm to optimize with respect to it.

No doubt, any convex function can be used in place of $\mathcal{L}(\cdot,\cdot)$, such as the Huber, square ϵ-insensitive, and the squared error loss functions (Problem 11.15).

Error bounds and convergence performance

In [61], the performance analysis of an online algorithm, whose goal is the minimization of (11.96), is based on the cumulative instantaneous loss, after running the algorithm on N training samples, that is,

$$\boxed{L_{cum}(N) := \sum_{n=1}^{N} \mathcal{L}\big(y_n, f_{n-1}(\boldsymbol{x}_n)\big) : \qquad \text{Cumulative Loss.}} \tag{11.105}$$

As already mentioned in Chapter 8, this is a natural criterion to test the performance. Note that f_{n-1} has been trained using samples up to and including time instants $n-1$, and it is tested against the sample $(y_n, \boldsymbol{x}_n)$, on which it was *not* trained. So $L_{cum}(N)$ can be considered as a measure of the generalization performance. A low value of $L_{cum}(N)$ is indicative of guarding against overfitting; see also [3, 130]. The following theorem has been derived in [61].

Theorem 11.4. *Fix $\lambda > 0$ and $0 < \mu < \frac{1}{\lambda}$. Assume that $\mathcal{L}(\cdot,\cdot)$ is convex and satisfies the Lipschitz condition, that is,*

$$|\mathcal{L}(y, z_1) - \mathcal{L}(y, z_2)| \le c|z_1 - z_2|, \ \forall z_1, z_2 \in \mathbb{R}, \quad \forall y \in Y$$

where Y is the respective domain of definition (e.g., $[-1, 1]$ or $\mathbb{R}$) and $c \in \mathbb{R}$. Also, let $\kappa(\cdot,\cdot)$ be bounded, that is,

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_n) \le B^2, \quad n = 1, 2, \ldots, N.$$

Set $\mu_n = \frac{\mu}{\sqrt{n}}$. Then,

$$\frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_{n,\lambda}(f_{n-1}) \le J_{emp,\lambda}(f_*, N) + \mathcal{O}(N^{-1/2}), \tag{11.106}$$

where $\mathcal{L}_{n,\lambda}$ is defined in (11.98), $J_{emp,\lambda}$ is the regularized empirical loss in (11.97) and f_* is the minimizer, that is,

$$f_* = \arg\min_{f \in \mathbb{H}} J_{emp,\lambda}(f, N).$$

Such bounds have been treated in Chapter 8 in the context of regret analysis for online algorithms. In [61], more performance bounds are derived concerning the fixed learning rate case, $\mu_n = \mu$, appropriate for time-varying cases under specific scenarios. Also, bounds for the expected risk are provided.

Remarks 11.6.

- NORMA is similar to the ALMA algorithm presented in [43]; ALMA considers a time-varying step-size and regularization is imposed via a projection of the parameter vector on a closed ball. This idea of projection is similar with that used in PEGASOS discussed in Chapter 8, [106]. As a matter of fact, PEGASOS without the projection step coincides with NORMA. The difference lies in the choice of the step-size.

  In the ALMA algorithm, the possibility of normalizing the input samples and the unknown parameter vector via different norms, that is, $\|\cdot\|_q$ and $\|\cdot\|_p$ (p and q being dual norms) in the context of the so-called p-norm margin classifiers is exploited. p-norm algorithms can be useful in learning sparse hyperplanes (see also [62]).
- For the special case of $\rho = 0$ and $\lambda = 0$, the NORMA using the hinge loss breaks down to the *kernel perceptron*, to be discussed in more detail in Chapter 18.

11.12.3 THE KERNEL APSM ALGORITHM

The APSM algorithm was introduced as an alternative path for parameter estimation in machine learning tasks, which springs from the classical POCS theory; it is based solely on (numerically robust) projections. Extending its basic recursions (8.39) to the case of a RKH space of functions (as said there, the theory holds for a general Hilbert space), we obtain

$$f_n = f_{n-1} + \mu_n \left(\frac{1}{q} \sum_{k=n-q+1}^{n} P_k(f_{n-1}) - f_{n-1} \right), \tag{11.107}$$

where the weights for the convex combination of projections are set (for convenience) equal, that is, $\omega_k = \frac{1}{q}$, $k \in [n-q+1, n]$. P_k is the projection operator on the respective convex set. Two typical examples of convex sets for regression and classification are described next.

Regression

In this case, as treated in Chapter 8, a common choice is to project on hyperslabs, defined by the points $(y_n, \kappa(\cdot, \boldsymbol{x}_n))$ and the parameter ϵ, which controls the width of the hyperslab; its value depends on the noise variance, without being very sensitive to it. For such a choice, we get (see (8.34)),

$$P_k(f_{n-1}) = f_{n-1} + \beta_k \kappa(\cdot, \boldsymbol{x}_k), \tag{11.108}$$

with

$$\beta_k = \begin{cases} \dfrac{y_k - \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle - \epsilon}{\kappa(\boldsymbol{x}_k, \boldsymbol{x}_k)}, & \text{if } \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle - y_k < -\epsilon, \\ 0, & \text{if } |\langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle - y_k| \le \epsilon, \\ \dfrac{y_k - \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle + \epsilon}{\kappa(\boldsymbol{x}_k, \boldsymbol{x}_k)}, & \text{if } \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle - y_k > \epsilon. \end{cases} \tag{11.109}$$

Recall from Section 8.6 that, the hyperslab defined by $(y_n, \kappa(\cdot, \boldsymbol{x}_n))$ and ϵ is the respective 0-*level set* of the linear ϵ-insensitive loss function $\mathcal{L}(y_n, f(\boldsymbol{x}_n))$ defined in (11.28).

Classification

In this case, a typical choice is to project on the half-space, Section 8.6.2, formed by the hyperplane

$$y_n \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_n)\rangle = \rho,$$

and the corresponding projection operator becomes

$$P_k(f_{n-1}) = f_{n-1} + \beta_k \kappa(\cdot, \boldsymbol{x}_k),$$

where

$$\beta_k = \begin{cases} \frac{\rho - y_k \langle f_{n-1},\, \kappa(\cdot, \boldsymbol{x}_k)\rangle}{\kappa(\boldsymbol{x}_k, \boldsymbol{x}_k)}, & \text{if } \rho - y_k \langle f_{n-1},\, \kappa(\cdot, \boldsymbol{x}_k)\rangle > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{11.110}$$

Recall that the corresponding half-space is the 0-level set of the *hinge loss* function, $\mathcal{L}_\rho\big(y_n, f(\boldsymbol{x}_n)\big)$ defined in (11.59).

Thus, for both cases, regression as well as classification, the recursion in (11.107) takes a common formulation

$$f_n = f_{n-1} + \mu_n \sum_{k=n-q+1}^{n} \beta_k \kappa(\cdot, \boldsymbol{x}_k), \tag{11.111}$$

where $1/q$ has been included in β_k. Applying the above recursively from $f_0 = 0$, f_n can be written as an expansion in the form of (11.100) and the corresponding updating of the parameters, θ_i, become

$$\boxed{\begin{aligned} \theta_n &= \mu_n \beta_n, && (11.112)\\ \theta_i^{\text{new}} &= \theta_i + \mu_n \beta_i,\ i = n-q+1, \ldots, n-1, && (11.113)\\ \theta_i^{\text{new}} &= \theta_i,\ i \leq n-q, && (11.114) \end{aligned}}$$

where for the computation in (11.110) and (11.109) the following equality has been employed,

$$\begin{aligned} \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle &= \sum_{i=1}^{n-1} \theta_i \langle \kappa(\cdot, \boldsymbol{x}_i), \kappa(\cdot, \boldsymbol{x}_k)\rangle \\ &= \sum_{i=1}^{n-1} \theta_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_k). \end{aligned} \tag{11.115}$$

Comparing (11.112)-(11.114) with (11.101)-(11.102) and setting $q = 1$, (for APSM) and $\lambda = 0$ (for NORMA), we see that the parameter updates look very similar; only the values of the involved variables are obtained differently. As we will see in the simulations section, this difference can have a significant effect on the respective performances in practice.

Sparsification: Sparsification of the APSM can be achieved either by regularization or via the use of a dictionary, in a similar way as explained before in Section 11.12.1.

For regularization, the projection path is adopted, which is in line with the rationale that has inspired the method. In other words, we impose on the desired sequence of hypothesis the following constraint:

$$\|f_n\| \leq \delta.$$

Recall that if one has to perform a minimization of a loss function under this constrain, it is equivalent to performing a regularization of the loss (as in (11.98)) for appropriate choices of λ (δ) (see also Chapter 3). Under the previous constraints, the counterpart of (11.107) becomes

$$f_n = P_{B[0,\delta]}\left(f_{n-1} + \mu_n\left(\frac{1}{q}\sum_{k=n-q+1}^{n} P_k(f_{n-1}) - f_{n-1}\right)\right),$$

or

$$f_n = P_{B[0,\delta]}\left(f_{n-1} + \mu_n \sum_{k=n-q+1}^{n} \beta_k \kappa(\cdot, \boldsymbol{x}_k)\right), \tag{11.116}$$

where $P_{B[0,\delta]}$ is the projection on the closed ball defined as $B[0,\delta]$ (Example 8.1),

$$B[0,\delta] = \{f \in \mathbb{H} : \|f\| \leq \delta\},$$

and the projection is given by (8.14)

$$P_{B[0,\delta]}(f) = \begin{cases} f, & \text{if } \|f\| \leq \delta, \\ \frac{\delta}{\|f\|} f, & \text{if } \|f\| > \delta, \end{cases}$$

which together with (11.116) adds an extra step in the updates (11.112)–(11.113), that is,

$$\theta_i = \frac{\delta \theta_i}{\|f_n\|}, \quad i = 1, 2, \ldots, n,$$

whenever needed. The computation of $\|f_n\|$ can be performed recursively, and adds no extra kernel evaluations (which is the most time-consuming part of all the algorithms presented so far), to those used in the previous steps of the algorithm (Problem 11.16). Note that the projection step is applied if, $\delta < \|f_n\|$; thus, after a repeated application of the projection (multiplications of smaller than one values), renders the parameters associated with samples of the "remote" past to small values and they can be neglected. This leads us to the same conclusions as in NORMA; hence, only the most recent terms in a time window $[n - n_0 + 1, n]$ can be kept and updated [110, 111].

The other alternative for sparsification is via the use of dictionaries. In this case, the expansion of each hypothesis takes place in terms of the elements in the dictionary comprising $\boldsymbol{u}_m$, $m = 1, 2, \ldots, M_{n-1}$. Under such a scenario, (11.115) becomes

$$\langle f_{n-1}, \kappa(\cdot, \boldsymbol{x}_k)\rangle = \sum_{m=1}^{M_{n-1}} \theta_m \kappa(\boldsymbol{x}_k, \boldsymbol{u}_m), \tag{11.117}$$

as only elements included in the dictionary are involved. The resulting scheme is given in Algorithm 11.3, [109].

Algorithm 11.3 (The quantized kernel APSM).

- Initialization
  - $\mathcal{D} = \emptyset$.
  - Select $q \geq 1$, δ; the quantization level.
  - $d(\boldsymbol{x}, \emptyset) := \Delta > \delta, \ \forall \boldsymbol{x}$.

```
• FOR $n = 1, 2, ...,$ Do
    • $d(\boldsymbol{x}_n, \mathcal{D}) = \inf_{\boldsymbol{u}_k \in \mathcal{D}} \|\boldsymbol{x}_n - \boldsymbol{u}_k\| = \|\boldsymbol{x}_n - \boldsymbol{u}_{l_0}\|$, for some $l_0 \in \{1, 2, \ldots, M\}$
    • If $d(\boldsymbol{x}_n, \mathcal{D}) > \delta$ then
        - $\boldsymbol{u}_{M+1} = \boldsymbol{x}_n$; includes the new observation in the dictionary.
        - $\mathcal{D} = \mathcal{D} \cup \{\boldsymbol{u}_{M+1}\}$
        - $\boldsymbol{\theta} = [\boldsymbol{\theta}, \theta_{M+1}],\ \theta_{M+1} := 0$; increases the size of $\boldsymbol{\theta}$ and initializes the new weight to zero.
        - $\mathcal{J} := \{\max\{1, M - q + 2\},\ M + 1\}$; identifies the $q$ most recent samples in the dictionary.
      Else
        - If $l_0 \geq \max\{1, M - q + 1\}$, then
            • $\mathcal{J} = \{\max\{1, M - q + 1\}, \ldots, M\}$; identifies the $q$ most recent samples in the dictionary,
              which also includes $l_0$.
        - Else
            • $\mathcal{J} = \{l_0, M - q + 2, \ldots, M\}$; it takes care $\boldsymbol{u}_{l_0}$ to be in $\mathcal{J}$, if it is not in the $q$ most recent
              ones.
        - End If
    • End If
    • Compute $\beta_k,\ k \in \mathcal{J}$; (11.117) and (11.110) or (11.109)
    • Select $\mu_n$
    • $\theta_i = \theta_i + \mu_n \beta_i,\ i \in \mathcal{J}$
    • If $d(\boldsymbol{x}_n, \mathcal{D}) > \delta$ then
        - $M = M + 1$
    • Enf If
• END For
```

The algorithm returns the dictionary and $\boldsymbol{\theta}$ and

$$f_n = \sum_{i=1}^{M} \theta_i \kappa(\cdot, \boldsymbol{u}_i)$$

$$\hat{y}_n = \sum_{i=1}^{M} \theta_i \kappa(\boldsymbol{x}_n, \boldsymbol{u}_i)$$

Remarks 11.7.

- Note that for the case of hyperslabs, if one sets $\epsilon = 0$ and $q = 1$ the *normalized* KLMS results.
- If a bias term needs to be included, this is achieved by the standard extension of the dimensionality of the space by one. This results in replacing $\kappa(\boldsymbol{x}_n, \boldsymbol{x}_n)$ with $1 + \kappa(\boldsymbol{x}_n, \boldsymbol{x}_n)$ in the denominator in (11.110) or (11.109) and the bias updating is obtained as (Problem 11.18)

$$\theta_0(n) = \theta_0(n-1) + \mu_n \sum_{k=n-q+1}^{n} \beta_k.$$

- For the selection of μ_n, one can employ a constant or a time-varying sequence, as in any algorithm of the stochastic gradient rationale. Moreover, for the case of APSM, there is a theoretically computed interval that guarantees convergence; that is, μ_n must lie in $(0, 2M_n)$, where,

$$M_n = \frac{\sum_{i\in\mathcal{J}} q\beta_i^2 \kappa(\boldsymbol{u}_i, \boldsymbol{u}_i)}{\sum_{i\in\mathcal{J}} \sum_{j\in\mathcal{J}} \beta_i\beta_j\kappa(\boldsymbol{u}_i, \boldsymbol{u}_j)}$$

if

$$\sum_{i\in\mathcal{J}} \sum_{j\in\mathcal{J}} \beta_i\beta_j\kappa(\boldsymbol{u}_i, \boldsymbol{u}_j) \neq 0.$$

Otherwise, $M_n = 1$.

- Besides the three basic schemes described before, kernelized versions of the RLS and APA have been proposed, see, for example, [36, 114]. However, these schemes need an inversion of a matrix of the order of the dictionary (RLS) and of the order of q (APA). An online kernel RLS-type algorithm, from a Bayesian perspective, is given in [127]. For an application of online kernel-based algorithms in time series prediction, see, for example, [96].
- One of the major advantages of the projection-based algorithms is the fairly easy way to accommodate constraints; in some cases even nontrivial constraints. This also applies to the KAPSM. For example, in [112] the nonlinear robust beamforming is treated. This task corresponds to an infinite set of linear inequality constraints; in spite of that, it can be solved in the context of APSM with a linear complexity, with respect to the number of unknown parameters per time update. Extensions to multiregression with application to MIMO nonlinear channel equalization is presented in [113].
- All that has been said concerning convergence in Chapter 8 about APSM can be extended and it is valid for the case of RKH spaces.

Example 11.5. *Nonlinear Channel Equalization*

In this example, the performance of the previously described online kernel algorithms is studied, in the context of a nonlinear equalization task.

Let an information sequence, denoted as s_n, be transmitted to a nonlinear channel. For example, such channels are typical in satellite communications. The nonlinear channel is simulated as a combination of a linear one, whose input is the information sequence, that is,

$$t_n = -0.9s_n + 0.6s_{n-1} - 0.7s_{n-2} + 0.2s_{n-3} + 0.1s_{n-4},$$

followed by a memoryless nonlinearity (see Section 11.3), that is,

$$t'_n = 0.15t_n^2 + 0.03t_n^3.$$

In the sequel, a white noise sequence is added,

$$x_n = t'_n + \eta_n,$$

where for our example, η_n is a zero-mean white Gaussian noise, with $\sigma_\eta^2 = 0.02$, corresponding to a signal-to-noise ratio of 15 dBs. The input information sequence was generated according to a zero-mean Gaussian with variance equal to $\sigma_s^2 = 0.64$.

The received sequence at the receiver end is the sequence x_n. The task of the equalizer is to provide an estimate of the initially transmitted information sequence s_n, based on x_n. As is the case for the linear equalizer,[13] Chapter 4, at each time instant, l successive samples are presented as input to the

[13] There, the input was denoted as u_n; here, because we do not care if the input is a process or not, we use x_n, to be in line with the notation used in this chapter.

equalizer, which form the input vector $\boldsymbol{x}_n$. In our case, we chose $l = 5$. Also, in any equalization scheme, a delay, D, is involved to account for the various delays associated with the communications system; that is, at time n, an estimate of s_{n-D} is obtained. Thus, the output of the equalizer is the nonlinear mapping

$$\hat{s}_{n-D} = f(\boldsymbol{x}_n).$$

In the training phase, we assume the transmitted symbols are known and our learning algorithms have available the training sequence, $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots$, where $y_n = s_{n-D}$. For this example, the best (after experimentation) delay was found to be $D = 2$.

The quantized KLMS was used with $\mu = 1/2$, the KAPSM with (fixed) $\mu_n = 1/2$, $\epsilon = 10^{-5}$ (the algorithm is fairly insensitive to the choice of ϵ) and $q = 5$. For the KRLS, the ALD accuracy parameter $\nu = 0.1$ [36] was used. In all algorithms, the Gaussian kernel has been employed with $\sigma = 5$. The LS loss function was employed, and the best performance for the NORMA was obtained for $\lambda = 0$, which makes it equivalent to the KLMS, when no sparsification is applied. However, in order to follow the suggestion in [61] and to make a fair comparison with the other three methods, we carefully tuned the regularization parameter λ. This allows the use of the sliding window method for keeping only a finite number of processing samples (i.e., n_0) at each time instant. In our experiments, we set the window size of NORMA as $n_0 = 80$, to keep it comparable to the dictionary sizes of QKLMS, QKAPSM, and KRLS, and found that the respective λ is equal to $\lambda = 0.01$; also, $\mu_n = 1/4$ was found to provide the best possible performance.

Figure 11.19 shows the obtained MSE averaged over 1000 realizations, in dBs ($10 \log_{10}(e_n^2)$) as a function of iterations. The improved performance of the KAPSM compared to the KLMS is clearly seen, due to the data reuse rationale, at a slightly higher computational cost. The KRLS has clearly superior convergence performance, converging faster and at lower error rates, albeit at higher complexity. The NORMA has a rather poor performance. For all cases, the parameters used were optimized via extensive simulations. The superior performance of the KAPSM, in the context of a classification task, when the property sets are built around half-spaces, compared to the NORMA, have also been demonstrated in [110], both in stationary as well as in time-varying environments. Figure 11.20 corresponds to the case where, after convergence, the channel suddenly changes to

$$t_n = 0.8s_n - 0.7s_{n-1} + 0.6s_{n-2} - 0.2s_{n-3} - 0.2s_{n-4},$$

and

$$x_n = 0.12t_n^2 + 0.02t_n^3 + \eta_n.$$

Observe that the KRLS really has a problem for tracking this time variation. The difficulty of KRLS in tracking time variations has also been observed and discussed in [127], where a KRLS is also derived via the Bayesian framework and an exponential forgetting factor is employed to cope with variations. Observe that, just after the jump of the system, the KLMS tries to follow the change faster compared to the KAPSM. This is natural, as the KAPSM employs past data reuse, which somehow slows down its agility to track; however, soon after, it recovers and leads to improved performance.

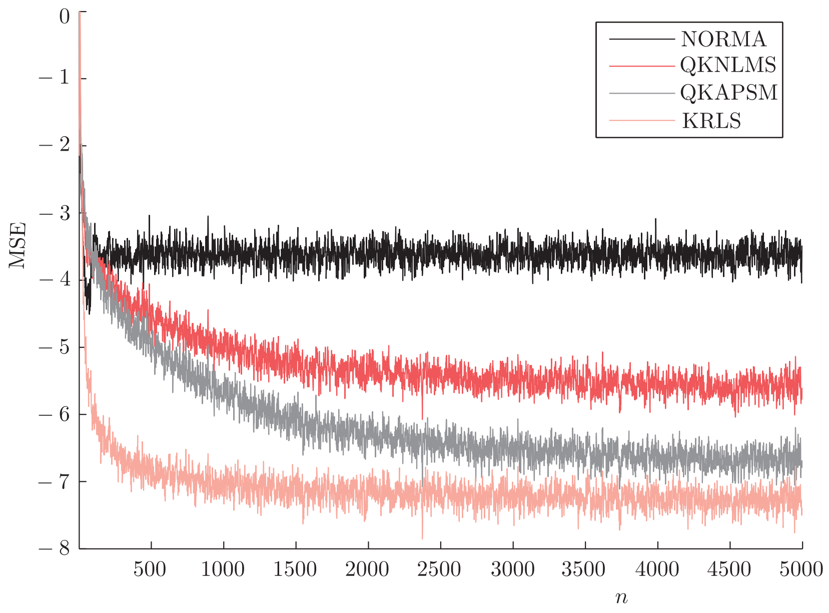

FIGURE 11.19

Mean-square error in dBs, as a function of iterations, for the data of Example 11.5.

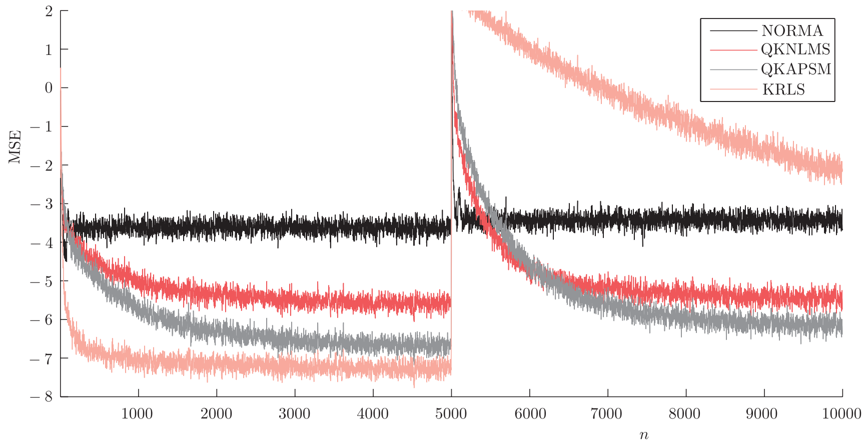

FIGURE 11.20

MSE in dBs, as a function of iterations, for the time-varying nonlinear channel of Example 11.5.

11.13 MULTIPLE KERNEL LEARNING

A major issue in all kernel-based algorithmic procedures is the selection of a suitable kernel as well as the computation of its defining parameters. Usually, this is carried out via cross validation, where a number of different kernels are used on a separate, from the training data, validation set (see Chapter 3 for different methods concerning validation) and the one with the best performance is selected. It is obvious that this is not a universal approach, it is time-consuming and definitely is not theoretically appealing. The ideal would be to have a set of different kernels (this also includes the case of the same kernel with different parameters) and let the optimization procedure decide how to choose the proper kernel, or the proper combination of kernels. This is the scope of an ongoing activity, which is usually called *multiple kernel learning* (MKL). To this end, a variety of MKL methods have been proposed to treat several kernel-based algorithmic schemes. A complete survey of the field is outside the scope of this book. Here, we will provide a brief overview of some of the major directions in MKL methods that relate to the content of this chapter. The interested reader is referred to [47] for a comparative study of various techniques.

One of the first attempts to develop an efficient MKL scheme is the one presented in [65], where the authors considered a linear combination of kernel matrices, that is, $K = \sum_{m=1}^{M} a_m K_m$. Because we require the new kernel matrix to be positive definite, it is reasonable to impose in the optimization task some additional constraints. For example, one may adopt the general constraint $K \geq 0$ (the inequality indicating semidefinite positiveness), or a more strict one, for example, $a_m > 0$, for all $m = 1, \ldots, M$. Furthermore, one needs to bound the norm of the final kernel matrix. Hence, the general MKL SVM task can be cast as

$$\begin{aligned} &\text{minimize with respect to } K && \omega_C(K), \\ &\text{subject to} && K \geq 0, \\ & && \text{trace}\{K\} \leq c, \end{aligned} \tag{11.118}$$

where $\omega_C(K)$ is the solution of the dual SVM task, given in (11.75)–(11.77), which can be written in a more compact form as

$$\omega_C(K) = \max_{\boldsymbol{\lambda}} \left\{ \boldsymbol{\lambda}^T \mathbf{1} - \frac{1}{2} \boldsymbol{\lambda}^T G(K) \boldsymbol{\lambda} : \ 0 \leq \lambda_i \leq C, \ \boldsymbol{\lambda}^T \boldsymbol{y} = 0 \right\},$$

with each element of $G(K)$ given as $[G(K)]_{i,j} = [K]_{i,j} y_i y_j$. $\boldsymbol{\lambda}$ denotes the vector of the Lagrange multipliers and $\mathbf{1}$ is the vector having all its elements equal to one. In [65], it is shown how (11.118) can be transformed to a semidefinite programming optimization (SDP) task and solved accordingly.

Another path that has been exploited by many authors is to assume that the modeling nonlinear function is given as a summation

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_{m=1}^{M} a_m f_m(\boldsymbol{x}) = \sum_{m=1}^{M} a_m \langle f_m, \kappa_m(\cdot, \boldsymbol{x}) \rangle_{\mathbb{H}_m} + b \\ &= \sum_{m=1}^{M} \sum_{n=1}^{N} \theta_{m,n} a_m \kappa_m(\boldsymbol{x}, \boldsymbol{x}_n) + b, \end{aligned}$$

where each one of the functions, f_m, $m = 1, 2, \ldots M$, lives in a different RKHS, $\mathbb{H}_m$. The respective composite kernel matrix, associated with a set of training data, is given by $K = \sum_{m=1}^{M} a_m^2 K_m$, where

$K_1, \ldots, K_M$ are the kernel matrices of the individual RKH spaces. Hence, assuming a data set $\{(y_n, \boldsymbol{x}_n),\ n = 1, \ldots N\}$, the MKL learning task can be formulated as follows:

$$\min_f \sum_{n=1}^{N} \mathcal{L}\left(y_n, f(\boldsymbol{x}_n)\right) + \lambda \Omega(f), \tag{11.119}$$

where $\mathcal{L}$ represents a loss function and $\Omega(f)$ the regularization term. There have been two major trends following this rationale. The first one gives priority toward a sparse solution, while the second aims at improving performance.

In the context of the first trend, the solution is constrained to be sparse, so that the kernel matrix is computed fast and the strong similarities between the data set are highlighted. Moreover, this rationale can be applied to the case where the type of kernel has been selected beforehand and the goal is to compute the optimal kernel parameters. One way (e.g., [117], [4]) is to employ a regularization term of the form $\Omega(f) = \left(\sum_{m=1}^{M} a_m \|f_m\|_{\mathbb{H}_m}\right)^2$, which has been shown to promote sparsity among the set $\{a_1, \ldots a_M\}$, as it is associated to the group LASSO, when the LS loss is employed in place of $\mathcal{L}$ (see Chapter 10).

In contrast to the sparsity promoting criteria, another trend (e.g., [29, 63, 94]) revolves around the argument that, in some cases, the sparse MKL variants may not exhibit improved performance compared to the original learning task. Moreover, some data sets contain multiple similarities between individual data pairs that cannot be highlighted by a single type of kernel, but require a number of different kernels to improve learning. In this context, a regularization term of the form $\Omega(f) = \sum_{m=1}^{M} a_m \|f_m\|_{\mathbb{H}_m}^2$ is preferred. There are several variants of these methods that either employ additional constraints to the task (e.g., $\sum_{m=1}^{M} a_m = 1$), or define the summation of the spaces a little differently (e.g., $f(\cdot) = \sum_{m=1}^{M} \frac{1}{a_m} f_m(\cdot)$). For example, in [94] the authors reformulate (11.119) as follows:

$$\begin{aligned} \text{minimize with respect to } \boldsymbol{a} \quad & J(\boldsymbol{a}), \\ \text{subject to} \quad & \sum_{m=1}^{M} a_m = 1, \\ & a_m \geq 0, \end{aligned} \tag{11.120}$$

where

$$J(\boldsymbol{a}) = \min_{f_{1:M}, \boldsymbol{\xi}, b} \left\{ \begin{array}{c} \frac{1}{2} \sum_{m=1}^{M} \frac{1}{a_m} \|f_m\|_{\mathbb{H}_m}^2 + C \sum_{n=1}^{N} \xi_n, \\ y_i \left(\sum_{m=1}^{M} f_m(\boldsymbol{x}_n) + b \right) \geq 1 - \xi_n, \\ \xi_n \geq 0. \end{array} \right\}$$

The optimization is cast in RKH spaces; however, the problem is always formulated in such a way so that the kernel trick can be mobilized.

11.14 NONPARAMETRIC SPARSITY-AWARE LEARNING: ADDITIVE MODELS

We have already pointed out that the representer theorem, as summarized by (11.17), provides an approximation of a function, living in an RKHS, in terms of the respective kernel centered at the points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. However, we know that the accuracy of any interpolation/approximation method depends on the number of points, N. Moreover, as discussed in Chapter 3, how large or small N needs to be depends heavily on the dimensionality of the space, exhibiting an exponential dependence on it (curse of dimensionality); basically, one has to fill in the input space with "enough" data in order to be able to "learn" with good enough accuracy the associated function.[14] In Chapter 7, the naive Bayes classifier was discussed; the essence behind this method is to consider each dimension of the input random vectors, $\mathbf{x} \in \mathbb{R}^l$, *individually*. Such a path breaks the problem into a number, l, of *one-dimensional* tasks. The same idea runs across the so-called *additive* models approach.

According to the additive models rationale, the unknown function is constrained within the family of *separable* functions, that is,

$$\boxed{f(\boldsymbol{x}) = \sum_{i=1}^{l} \phi_i(x_i): \quad \text{Additive Model,}} \tag{11.121}$$

where $\boldsymbol{x} = [x_1, \ldots, x_l]^{\mathrm{T}}$. Recall that a special case of such expansions is the linear regression, where $f(\boldsymbol{x}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$.

We will further assume that each one of the functions, $\phi_i(\cdot)$, belongs to an RKHS, $\mathbb{H}_i$ defined by a respective kernel, $\kappa_i(\cdot,\cdot)$,

$$\kappa_i(\cdot,\cdot): \mathbb{R} \times \mathbb{R} \longmapsto \mathbb{R}.$$

Let the corresponding norm be denoted as $\|\cdot\|_i$. For the regularized LS cost [95], the optimization task is now cast as

$$\text{minimize with respect to } f \qquad \frac{1}{2}\sum_{n=1}^{N}\big(y_n - f(\boldsymbol{x}_n)\big)^2 + \lambda \sum_{i=1}^{l} \|\phi_i\|_i, \tag{11.122}$$

$$\text{s.t.} \qquad f(\boldsymbol{x}) = \sum_{i=1}^{l} \phi_i(x_i). \tag{11.123}$$

If one plugs (11.123) into (11.122) and following arguments similar to those used in Section 11.6, it is readily obtained that we can write

$$\hat{\phi}_i(\cdot) = \sum_{n=1}^{N} \theta_{i,n}\kappa_i(\cdot, x_{i,n}), \tag{11.124}$$

where $x_{i,n}$ is the ith component of $\boldsymbol{x}_n$. Moving along the same path as that adopted in Section 11.7, the optimization can be rewritten in terms of

$$\boldsymbol{\theta}_i = [\theta_{i,1}, \ldots, \theta_{i,N}]^{\mathrm{T}}, \quad i = 1, 2, \ldots, l,$$

as

$$\{\hat{\boldsymbol{\theta}}_i\}_{i=1}^{l} = \arg\min_{\{\boldsymbol{\theta}_i\}_{i=1}^{l}} J(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_l),$$

where

$$J(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_l) := \frac{1}{2}\left\| \boldsymbol{y} - \sum_{i=1}^{l} \mathcal{K}_i\boldsymbol{\theta}_i \right\|^2 + \lambda \sum_{i=1}^{l} \sqrt{\boldsymbol{\theta}_i^T \mathcal{K}_i \boldsymbol{\theta}_i}, \tag{11.125}$$

and $\mathcal{K}_i$, $i = 1, 2, \ldots, l$, are the respective $N \times N$ kernel matrices

[14] Recall from the discussion in Section 11.10.4 that the other factor that ties accuracy and N together is the rate of variation of the function.

$$\mathcal{K}_i := \begin{bmatrix} \kappa_i(x_{i,1}, x_{i,1}) & \cdots & \kappa_i(x_{i,1}, x_{i,N}) \\ \vdots & \ddots & \vdots \\ \kappa_i(x_{i,N}, x_{i,1}) & \cdots & \kappa_i(x_{i,N}, x_{i,N}) \end{bmatrix}.$$

Observe that (11.125) is a (weighted) version of the *group* LASSO, defined in Section 10.3. Thus, the optimization task enforces sparsity by pushing some of the vectors $\boldsymbol{\theta}_i$, to zero values. Any algorithm developed for the group LASSO can be employed here as well, see, for example, [10, 95].

Besides the squared error loss, other loss functions can also be employed. For example, in [95] the *logistic regression* model is also discussed. Moreover, if the separable model in (11.121) cannot adequately capture the whole structure of f, models involving combination of components can be considered, such as the ANOVA model, e.g. [67].

The analysis of variance (ANOVA) is a method in statistics to analyse interactions among variables. According to this technique, a function $f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^l$, $l > 1$, is decomposed into a number of terms; each term is given as a sum of functions involving a *subset* of the components of $\boldsymbol{x}$. From this point of view, separable functions of the form in (11.121) are a special case of an ANOVA decomposition. A more general decomposition would be

$$\boxed{f(\boldsymbol{x}) = \theta_0 + \sum_{i=1}^{l} \phi_i(x_i) + \sum_{i<j=1}^{l} \phi_{ij}(x_i, x_j) + \sum_{i<j<k=1}^{l} \phi_{ijk}(x_i, x_j, x_k),} \tag{11.126}$$

and this can be generalized to involve larger number of components. Comparing (11.126) with (11.2), ANOVA decomposition can be considered as a generalization of the polynomial expansion. The idea of ANOVA decomposition has already been used to decompose kernels, $\kappa(\boldsymbol{x}, \boldsymbol{y})$, in terms of other kernels that are functions of a subset of the involved components of $\boldsymbol{x}$ and $\boldsymbol{y}$, giving rise to the so-called ANOVA kernels, [116].

Techniques involving sparse additive kernel-based models have been used in a number of applications, such as matrix and tensor completion, completion of gene expression, kernel-based dictionary learning, network flow, and spectrum cartography; see [10] for a related review.

11.15 A CASE STUDY: AUTHORSHIP IDENTIFICATION

Text mining is the part of data mining that analyzes textual data to detect, extract, represent, and evaluate patterns that appear in texts and can be transformed into real-world knowledge. A few examples of text mining applications include spam e-mail detection, topic-based classification of texts, sentiment analysis in texts, text authorship identification, text indexing, and text summarization. In other words, the goal can be on detecting basic morphology idiosyncracies in a text that identify its author (authorship identification), on identifying complexly expressed emotions (sentiment analysis), or on condensing redundant information from multiple texts to a concise summary (summarization).

In this section, our focus will be on the case of authorship identification, for example, [115], which is a special case of text classification. The task is to determine the author of a given text, given a set of training texts labeled with their corresponding author. To fulfill this task, one needs to represent and act upon textual data. Thus, the first decision related to text mining is how one represents the data.

It is very common to represent texts in a vector space, following the vector space model or VSM [98]. According to VSM, a text document, T, is represented by a set of terms $w_i, 0 \leq i \leq k,\ k \in \mathbb{N}$ each mapped to a dimension of the vector $\mathbf{t} = [t_1, t_2, \ldots, t_k]^{\mathrm{T}} \in \mathbb{R}^k$. The elements, t_i, in each dimension, indicate the importance of the corresponding term w_i when describing the document. For example, each w_i can be one word in the vocabulary of a language. Thus, in practical applications, k can be very large, as large as the number of words that are considered sufficient for the specific task of interest. Typical examples of k are of the order of 10^5.

Widely used approaches to assign values to t_i are as follows:

- *Bag-of-words, frequency approach*: The bag-of-words approach to text relies on the assumption that a text T is nothing more than a histogram of the words it contains. Thus, in the bag-of-words assumption, we do not care about the order of the words in the original text. What we care about is how many times term w_i appears in the document. Thus, t_i is assigned the number of occurrences of w_i in T.
- *Bag-of-words, boolean approach*: The boolean approach to the bag-of-words VSM approach restricts the t_i values such as, $t_i \in \{0, 1\}$. t_i is assigned a value of 1, if w_i was found at least once in T, otherwise $t_i = 0$.

The bag-of-words approaches have proven efficient, for example, in early spam filtering applications, where individual words were clear indicators of whether an e-mail is spam. However, the spammers soon adapted and changed the texts they sent out by breaking words using spaces: the word "pills" would be replaced by "p i 1 1 s." The bag-of-words approach then needed to apply preprocessing to deal with such cases, or even to deal with changes in different word forms (via word stemming or lemmatization) or spelling mistakes (by using dictionaries to correct the errors).

An alternative representation, with high resilience to noise, is that of the *character n-grams*. This representation is based on character subsequences of a text of length n (also called the order of an n-gram). To represent a text using n-grams, we split the text T in (usually contiguous) groups of characters of length n, that are then mapped to dimensions in the vector space (bag-of-n-grams). Below we give an example on how n-grams can be extracted from a given text.

Example 11.6. Given the input text $T =$ *A_fine_day_today*, the character and word 2-grams ($n = 2$) would be as follows:

- *Unique* character 2-grams: *"A_", "_f", "fi", "in", "ne", "e_", "_d", "da", "ay", "y_", "_t", "to", "od"*
- Unique word 2-grams: *"A_fine", "fine_day", "day_today"*

Observe that "ay" and "da" appear twice in the phrase, yet only once in the sequence, since each n-gram is uniquely represented. Even though this approach has proven very useful [18], still the sequence of the n-grams is not exploited.

In the following, we describe a more complex representation, which takes into account the sequence of n-grams, and at the same time allows for noise. The representation is termed *n-gram graph* [44]. An n-gram graph represents the way on how n-grams *co-occur* in a text.

Definition 11.4. If $S = \{S_1, S_2, \ldots\}, S_k \neq S_l$, for $k \neq l$, $k, l \in \mathbb{N}$ is the set of *distinct* n-grams extracted from a text, T, and S_i is the ith extracted n-gram. Then, $G = \{V, E, W\}$ is a graph where $V = S$ is the set of vertices v, E is the set of edges e of the form $e = \{v_1, v_2\}$, and $W : E \to \mathbb{R}$ is a function assigning a weight w_e to every edge.

To generate an n-gram graph, we first extract the unique n-grams from a text, creating one vertex for each. Then, given a user-defined parameter distance, D, we consider the n-grams that are found *within* a distance, D, of each other in the text; these are considered *neighbors* and their respective vertices are connected with an edge. For each edge, a weight is assigned. The edge weighting function that is most commonly used in n-gram graphs is the number of co-occurrences of the linked (in the graph) n-grams in the text.

Two examples of 2-gram graphs, with directed neighborhood edges, with $D = 2$ are presented in Figures 11.21 and 11.22.

Between two n-gram graphs, G^i and G^j, there exists a symmetric, normalized similarity function called *value similarity* (VS). This measure quantifies the ratio of common edges between two graphs, taking into account the ratio of weights of common edges. In this measure, each matching edge e, having weights w_e^i and w_e^j, in graphs G^i and G^i, respectively, contributes to VS the amount, $\frac{\mathrm{VR}(e)}{\max(|G^i|,|G^j|)}$, where, $|\cdot|$ indicates cardinality with respect to the number of edges and

$$\mathrm{VR}(e) := \frac{\min(w_e^i, w_e^j)}{\max(w_e^i, w_e^j)}. \tag{11.127}$$

Not matching edges have no contribution, (consider that for an edge $e \notin G^i(G^j)$ we define $w_e^i = 0$ $(w_e^j = 0)$). The equation indicates that the *ValueRatio* takes values in [0, 1], and is symmetric. Thus, the full equation for *VS* is

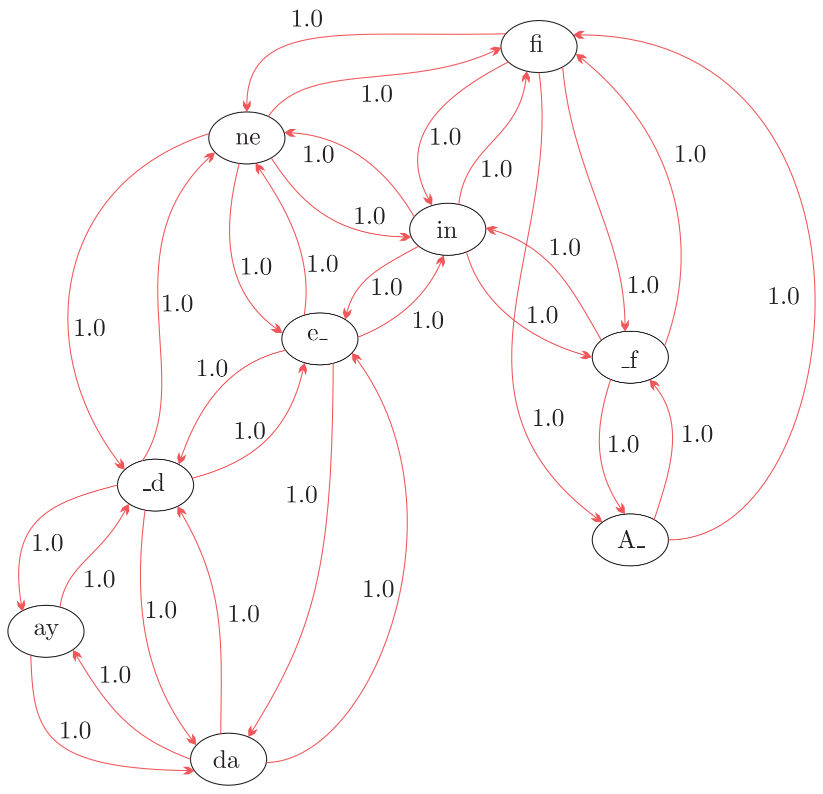

FIGURE 11.21

The 2-gram graph ($n = 2$), with $D = 2$ of the string “A fine day.”

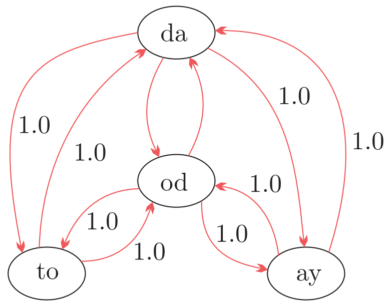

FIGURE 11.22

The 2-gram graph ($n = 2$), with $D = 2$ of the string "today."

$$\mathrm{VS}(G^i, G^j) = \frac{\sum_{e \in G^i} \frac{\min(w_e^i, w_e^j)}{\max(w_e^i, w_e^j)}}{\max(|G^i|, |G^j|)}. \tag{11.128}$$

VS takes a value close to 1 for graphs that share many edges with similar weights. A value of VS $=$ 1 indicates a perfect match between the compared graphs. For the two graphs shown in Figures 11.21 and 11.22, the calculation of the VS returns a value of VS $= 0.067$ (verify it).

For our case study, VS will be used to build an SVM kernel classifier, in order to assign (classify) texts to their authors. We should stress here that the VS function is a similarity function, but not a kernel. Fortunately, VS on the specific data set is an (ϵ, γ)-good similarity function as defined in (cf. [6, Theorem 3]) and hence, this allows its use as a kernel function, in line with what we have already said about string kernels in Section 11.5.2.

It is interesting to note that, given the VSM representation of a text, one can also employ any vector-based kernel on strings. However, there have been several works that use *string kernels* [70] to avoid any loss of information in the transformation between the original string and its VSM equivalent. In this example, we demonstrate how one can combine even a complex representation with the SVM via an appropriate kernel function.

The dataset we used to examine the effectiveness of this combination is a subset of the Reuter_50_50 Data Set from the UCI repository [5]. This dataset contains 2500 training and 2500 test texts from 50 different authors (50 training and 50 test texts per author).

- First, we used the ngg2svm command-line tool[15] to represent the strings as n-gram graphs and generate a precomputed kernel matrix file. The tool extracts are—by default—3-gram graphs ($n = 3, D = 3$) from training texts. It then uses VS to estimate a kernel and calculates the kernel values over all pairs of training instances into a kernel matrix. The matrix is the output of this phase.
- Given the kernel matrix, we can use the LibSVM software [22] to design a classifier, without caring about the original texts, but only relying on precomputer kernel values.

Then, a 10-fold crossvalidation over the data was performed, using the "svmtrain" program in LibSVM. The achieved accuracy of the cross validation in our example was 94%.

[15] You can download the tool from https://github.com/ggianna/ngg2svm.

PROBLEMS

11.1 Derive the formula for the number of groupings $\mathcal{O}(N, l)$ in Cover's theorem.
Hint. Show first the following recursion:

$$\mathcal{O}(N+1, l) = \mathcal{O}(N, l) + \mathcal{O}(N, l-1).$$

To this end, start with N points and add an extra one. Show that the extra number of linear dichotomies is solely due to those, for the N data point case, which could be drawn via the new point.

11.2 Show that if $N = 2(l+1)$, the number of linear dichotomies in Cover's theorem is equal to 2^{2l+1}.
Hint. Use the identity

$$\sum_{i=1}^{j} \binom{j}{i} = 2^j$$

and recall that

$$\binom{2n+1}{n-i+1} = \binom{2n+1}{n+i}.$$

11.3 Show that the reproducing kernel is a positive definite one.

11.4 Show that if $\kappa(\cdot,\cdot)$ is the reproducing kernel in an RKHS, $\mathbb{H}$, then

$$\mathbb{H} = \overline{\operatorname{span}\{\kappa(\cdot, \boldsymbol{x}),\ \boldsymbol{x} \in \mathcal{X}\}}.$$

11.5 Show the Cauchy-Schwarz inequality for kernels, that is,

$$\|\kappa(\boldsymbol{x}, \boldsymbol{y})\|^2 \leq \kappa(\boldsymbol{x}, \boldsymbol{x})\kappa(\boldsymbol{y}, \boldsymbol{y}).$$

11.6 Show that if

$$\kappa_i(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \longmapsto \mathbb{R},\ i = 1, 2,$$

are kernels then

- $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa_1(\boldsymbol{x}, \boldsymbol{y}) + \kappa_2(\boldsymbol{x}, \boldsymbol{y})$ is also a kernel.
- $a\kappa(\boldsymbol{x}, \boldsymbol{y}),\ a > 0$ is also a kernel.
- $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa_1(\boldsymbol{x}, \boldsymbol{y})\kappa_2(\boldsymbol{x}, \boldsymbol{y})$ is also a kernel.

.

11.7 Derive Eq. (11.25).

11.8 Show that the solution for the parameters, $\hat{\boldsymbol{\theta}}$, for the kernel ridge regression, if a bias term, b, is present, is given by

$$\begin{bmatrix} \mathcal{K} + CI & \mathbf{1} \\ \mathbf{1}^T\mathcal{K} & N \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}^T\mathbf{1} \end{bmatrix},$$

where $\mathbf{1}$ is the vector with all its elements being equal to one. Invertibility of the kernel matrix has been assumed.

11.9 Derive Eq. (11.56).

11.10 Derive the dual cost function associated with the linear ϵ-insensitive loss function.

11.11 Derive the dual cost function for the separable class SVM formulation.

11.12 Prove that in the NORMA, the hypothesis at time n has an equivalent expansion

$$f_n = \sum_{i=1}^{n} \theta_i^{(i)} (1 - \mu\lambda)^{n-i} \kappa(\cdot, \boldsymbol{x}_i),$$

where $\theta_i^{(i)} = -\mu \mathcal{L}'\big(y_i, f_{i-1}(\boldsymbol{x}_i)\big)$.

11.13 Derive a bound on the approximation error associated with the truncation of the expansion

$$f_n = \sum_{i=1}^{n} \theta_i^{(i)} (1 - \mu\lambda)^{n-i} \kappa(\cdot, \boldsymbol{x}_i).$$

adopted in the NORMA algorithm. Assume that

$$\left| \frac{\partial}{\partial z} \mathcal{L}(y, z) \right| \leq c, \ \|\kappa(\cdot, \boldsymbol{x})\| \leq B, \ \forall \boldsymbol{x} \in \mathcal{X}, \ z \in \mathbb{R}.$$

11.14 In the hinge loss function, assume ρ is a free parameter. Derive a version of NORMA that updates for this parameter as well.

11.15 Derive the subgradient for the Huber loss function and the respective weight updates for the NORMA.

11.16 Show that the formula for the recursive computation of the norm $\|f_n\|$, in the context of the kernel APSM algorithm, is given by

$$\|f_n\|^2 = \|f_{n-1}\|^2 + \mu_n^2 \sum_{k=n-q+1}^{n} \sum_{l=n-q+1}^{n} \beta_k \beta_l \kappa(\boldsymbol{x}_k, \boldsymbol{x}_l)$$
$$- 2\mu_n \sum_{i=1}^{n} \sum_{k=n-q+1}^{n} \beta_i \beta_k \kappa(\boldsymbol{x}_i, \boldsymbol{x}_k).$$

11.17 Derive the formula for the bound M_n used in Algorithm 11.3.

11.18 Derive the update recursion for the KAPSM if a bias term is employed.

MATLAB Exercises

11.19 Consider the regression problem described in Example 11.2. Read an audio file using MATLAB's wavread function and take 100 data samples (use *Blade Runner*, if possible, and take 100 samples starting from the 100,000th sample). Then add white gaussian noise at a 15 dB level and randomly "hit" 10 of the data samples with outliers (set the outlier values to 80% of the maximum value of the data samples).

(a) Find the reconstructed data samples using the unbiased kernel ridge regression method, that is, Eq. (11.27). Employ the Gaussian kernel with $\sigma = 0.004$ and set $C = 0.0001$. Plot the fitted curve of the reconstructed samples together with the data used for training.

(b) Find the reconstructed data samples using the biased kernel ridge regression method (employ the same parameters as for the unbiased case). Plot the fitted curve of the reconstructed samples together with the data used for training.

(c) Repeat the steps 11.19a, 11.19b using $C = 10^{-6}, 10^{-5}, 0.0005, 0.001, 0.01$, and 0.05.

(d) Repeat the steps 11.19a, 11.19b using $\sigma = 0.001, 0.003, 0.008, 0.01$, and 0.05.
(e) Comment on the results.

11.20 Consider the regression problem described in Example 11.2. Read the same audio file as in Problem 11.19. Then add white gaussian noise at a 15 dB level and randomly "hit" 10 of the data samples with outliers (set the outlier values to 80% of the maximum value of the data samples).
(a) Find the reconstructed data samples obtained by the support vector regression (you can use libsvm[16] for training). Employ the Gaussian kernel with $\sigma = 0.004$ and set $\epsilon = 0.003$ and $C = 1$. Plot the fitted curve of the reconstructed samples together with the data used for training.
(b) Repeat step 11.20a using $C = 0.05, 0.1, 0.5, 5, 10$, and 100.
(c) Repeat step 11.20a using $\epsilon = 0.0005, 0.001, 0.01, 0.05$, and 0.1.
(d) Repeat step 11.20a using $\sigma = 0.001, 0.002, 0.01, 0.05$, and 0.1.
(e) Comment on the results.

11.21 Consider the two-class two-dimensional classification task described in Example 11.4. The data set comprises $N = 150$ points uniformly distributed in the region $[-5, 5] \times [-5, 5]$. For each point, $\boldsymbol{x}_n = [x_{n,1},\ x_{n,2}]^T$, $n = 1, 2, \ldots, N$, compute

$$y_n = 0.5x_{n,1}^3 + 0.5x_{n,1}^2 + 0.5x_{n,1} + \eta$$

where η denotes zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. Assign each point to either of the two classes, depending on which side of the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space, y_n lies.
(a) Plot the points (x_n, y_n) using different colors for each class.
(b) Obtain the SVM classifier using libsvm or any other related MATLAB package. Use the Gaussian kernel with $\sigma = 20$ and set $C = 1$. Plot the classifier and the margin (for the latter you can employ MATLAB's contour function). Moreover, find the support vectors (i.e., the points with nonzero Lagrange multipliers that contribute to the expansion of the classifier) and plot them as circled points.
(c) Repeat step 11.21b using $C = 0.5, 0.1$, and 0.05.
(d) Repeat step 11.21b using $C = 5, 10, 50$, and 100.
(e) Comment on the results.

11.22 Consider the nonlinear equalization task described in Example 11.5.
(a) Create MATLAB functions that realize the QKLMS, QKAPSM, and NORMA algorithms using the Gaussian kernel. These functions should take as inputs a training sequence $(y_n, \boldsymbol{x}_n)$, the kernel parameter σ, the sparsification parameter s, and the learning rate μ. NORMA needs an extra parameter, the regularization parameter λ and QKAPSM the parameter, ϵ, and q, that is, the number of samples that are concurrently processed. For the QKAPSM and QKLMS, s denotes the quantization size δ while for the NORMA s is the window size, that is, n_0. The functions should provide as outputs the dictionary

[16] http://www.csie.ntu.edu.tw/cjlin/libsvm/.

centers and coefficients (that are involved in the expansion of the respective solutions, for example, (11.117)) and the error e_n at each step.

(b) Create an input signal comprising 5000 values generated by the Gaussian distribution with $\sigma = 0.8$ and zero-mean value. Then construct the sequence t_n using

$$t_n = -0.9s_n + 0.6s_{n-1} - 0.7s_{n-2} + 0.2s_{n-3} + 0.1s_{n-4},$$

and the sequence

$$x_n = 0.15t_n^2 + 0.03t_n^3 + \eta_n,$$

where η_n denotes a zero-mean white Gaussian noise with $\sigma = 0.14$. Then create the sequence of the data that will be used to train the equalizers, that is, $(y_n, \boldsymbol{x}_n)$, where $\boldsymbol{x}_n = [x_n, x_{n-1}, \dots x_{n-4}]^T$ (i.e., $l = 5$) and $y_n = s_{n-2}$ (i.e., $D = 2$).

(c) Using a For-loop, create 100 training sequences as described in step 11.22a. Feed the functions you developed with the aforementioned training sequences. Plot (in a single figure) the MSEs returned by each algorithm (averaged over the 100 training sequences). Use $\sigma = 5$ as the parameter of the Gaussian kernel, $\mu = 1/2$ for the QKLMS and QKAPSM, and $\mu = 1/4$ for NORMA, $\delta = 7$ for the quantization for both QKLMS and QKAPSM, and $n_0 = 80$ for the NORMA. Also, set $\epsilon = 10^{-5}$ and $q = 5$ for QKAPSM and $\lambda = 0.01$ for NORMA. Count the size of the dictionary for each algorithm. Comment on the results.

(d) Repeat step 11.22c for $\mu = 1/16, 1/8, 1/4.1, 2, 8, 16$. Keep in mind that if we want the experiment to be fair, NORMA should be carried out with the value of μ half the size of the one used for KLMS and KAPSM. Comment on the results.

(e) Repeat step 11.22c for $\sigma = 1, 2, 10, 15, 20$. Comment on the results.

(f) Repeat step 11.22c for $\delta = 1, 2, 10, 15$, and $n_0 = 800, 350, 40, 20$. Comment on the results.

(g) Repeat step 11.22a to create 5000 data samples. Then change the linear part of the channel to

$$t_n = 0.8s_n - 0.7s_{n-1} + 0.6s_{n-2} - 0.2s_{n-3} - 0.2s_{n-4}$$

and the nonlinear part to

$$x_n = 0.12t_n^2 + 0.02t_n^3 + \eta_n,$$

and create another set of 5000 samples. Repeat step 11.22c, feeding the MATLAB functions you created with training sequences of the aforementioned form. Comment on the results.

11.23 Consider the authorship identification problem described in Section 11.15.

- (i) Using the training texts of the authors whose name start with "T" in the dataset, perform 10-fold cross validation using the n-gram graph representation of the texts and the value similarity as a kernel function.
- (ii) Using the same subset of the dataset, create the vector space model representation of the texts and apply classification using a Gaussian kernel. Compare the results with the results of (i).

REFERENCES

[1] A. Argyriou, C.A. Micchelli, M. Pontil, When is there a representer theorem? Vector versus matrix regularizers, J. Machine Learn. Res. 10 (2009) 2507-2529.
[2] N. Aronszajn, Theory of reproducing kernels, Trans. Amer. Math. Soc. 68(3) (1950) 337-404.
[3] P. Auer, G. Gentile, Adaptive and self-confident on-line learning algorithms, J. Comput. Syst. Sci. 64(1) (2002) 48-75.
[4] F.R. Bach, Consistency of the group LASSO and multiple kernel learning, J. Machine Learn. Res. 9 (2008) 1179-1225.
[5] K. Bache, M. Lichman, UCI Machine Learning Repository, http://archive.ics.uci.edu/ml, University of California, Irvine, School of Information and Computer Sciences, 2013.
[6] M.F. Balcan, A. Blum, N. Srebro, A theory of learning with similarity functions, Machine Learn. 72(1-2) (2008) 89-112.
[7] A.R. Barron, Universal approximation bounds for superposition of a sigmoid function, Trans. Informat. Theory 39(3) (1993) 930-945.
[8] E.J. Bayro-Corrochano, N. Arana-Daniel, Clifford support vector machines for classification, regression and recurrence, IEEE Trans. Neural Networks 21(11) (2010) 1731-1746.
[9] J.A. Bazerque, G.B. Giannakis, Nonparametric basis pursuit via sparse kernel-based learning, IEEE Signal Process. Mag. 30(4) (2013) 112-125.
[10] J.A. Bazerque, G. Mateos, G.B. Giannakis, Group-LASSO on splines for spectrum cartography, Trans. Signal Process. 59(10) (2011) 4648-4663.
[11] S. Beneteto, E. Bigleiri, Principles of Digital Transmission, Springer, 1999.
[12] Y. Bengio, O. Delalleau, N. Le Roux, The curse of highly variable functions for local kernel machines, in: Y. Weiss, B. Scholkopf, J. Platt (Eds.), Advances in Neural Information Processing Systems, NIPS, MIT Press, 2006, pp. 107-114.
[13] P. Bouboulis, K. Slavakis, S. Theodoridis, Adaptive kernel-based image denoising employing semi-parametric regularization, IEEE Trans. Image Process. 19(6) (2010) 1465-1479.
[14] P. Bouboulis, S. Theodoridis, C. Mavroforakis, L. Dalla, Complex support vector machines for regression and quaternary classification, IEEE Trans. Neural Networks Learning Syst., to appear, 2014.
[15] P. Bouboulis, S. Theodoridis, Kernel methods for image denoising, in: J.A.K. Suykens, M. Signoretto, A. Argyriou (Eds.), Regularization, Optimization, Kernels, and Support Vector Machines, Chapman and Hall/CRC, Boca Raton, FL, 2014.
[16] O. Bousquet, A. Elisseeff, Stability and generalization, J. Machine Learn. Res. 2 (2002) 499-526.
[17] C.J.C. Burges, Geometry and invariance in kernel based methods, in: B. Scholkopf, C.J. Burges, A.J. Smola (Eds.), Advances in Kernel Methods, MIT Press, 1999, pp. 90-116.
[18] W.B. Cavnar, J.M. Trenkle et al., N-gram-based text categorization, in: Symposium on Document Analysis and Information Retrieval, University of Nevada, Las Vegas, 1994, pp. 161-176.
[19] L.J. Cao, S.S. Keerthi, C.J. Ong, J.Q. Zhang, U. Periyathamby, X.J. Fu, H.P. Lee, Parallel sequential minimal optimization for the training of support vector machines, IEEE Trans. Neural Networks 17(4) (2006) 1039-1049.
[20] C.C. Chang, C.W. Hsu, C.J. Lin, The analysis of decomposition methods for SVM, IEEE Trans. Neural Networks 11(4) (2000) 1003-1008.
[21] C.C. Chang, C.J. Lin, Training ν-support vector classifiers: Theory and algorithms, Neural Comput. 13(9) (2001) 2119-2147.
[22] C.-C. Chang, C.J. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Tech. 2(3) (2011) 27:1-27:27.
[23] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: International Conference on Machine Learning, 2006.

[24] G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning, in: Advances in Neural Information Processing Systems, NIPS, MIT Press, 2001, pp. 409-415.
[25] O. Chapelle, Training a support vector machine in the primal, Neural Comput. 19(5) (2007) 1155-1178.
[26] B. Chen, S. Zhao, P. Zhu, J.C. Principe, Quantized kernel least mean square algorithm, IEEE Trans. Neural Networks Learn. Syst. 23(1) (2012) 22-32.
[27] N. Cristianini, J. Shawe-Taylor, An introduction to Support Vector Machines, Cambridge University Press, 2000.
[28] C. Cortes, P. Haffner, M. Mohri, Rational kernels: Theory and algorithms, J. Machine Learn. Res. 5 (2004) 1035-1062.
[29] C. Cortes, M. Mohri, A. Rostamizadeh, L_2 Regularization for learning kernels, in: Proc. of the 25th Conference on Uncertainty in Artificial Intelligence, 2009, pp. 187-196.
[30] T.M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, IEEE Trans. Electron. Comput. 14 (1965) 326.
[31] P. Crama, J. Schoukens, Hammerstein-Wiener system estimator initialization, Automatica 40(9) (2004) 1543-1550.
[32] D.J. Crisp, C.J.C. Burges, A geometric interpretation of ν-SVM classifiers, in: Proceedings of Neural Information Processing, NIPS, Vol. 12, MIT Press, 1999.
[33] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, J. Artif. Intell. Res. 2 (1995) 263-286.
[34] G. Ding, Q. Wu, Y.-D. Yao, J. Wang, Y. Chen, Kernel-based learning for statistical signal processing in cognitive radio networks, IEEE Signal process. Mag. 30(4) (2013) 126-136.
[35] C. Domeniconi, D. Gunopulos, Incremental support vector machine construction, in: IEEE International Conference on Data Mining, San Jose, USA, 2001, pp. 589-592.
[36] Y. Engel, S. Mannor, R. Meir, The kernel recursive least-squares algorithm, IEEE Transactions on Signal Processing 52(8) (2004) 2275-2285.
[37] T. Evgeniou, M. Pontil, T. Poggio, Regularization networks and support vector machines, Adv. Comput. Math. 13 (2000) 1-50.
[38] T. Evgeniou, C.A. Michelli, M. Pontil, Learning multiple tasks with kernel methods, J. Machine Learn. Res. 6 (2005) 615-637.
[39] B. Fei, J. Liu, Binary tree of SVM: a new fast multiclass training and classification algorithm, IEEE Trans. Neural Networks 17(5) (2006) 696-704.
[40] P.A. Forero, A. Cano, G.B. Giannakis, Consensus-based distributed support vector machines, J. Machine Learn. Res. 11 (2010) 1663-1707.
[41] V. Franc, S. Sonnenburg, Optimized cutting plane algorithm for large-scale risk minimization, J. Machine Learn. Res. 10 (2009) 2157-2192.
[42] M. Frechet, Sur les fonctionnelles continues, Ann. Sci. de l'Ecole Super. 27 (1910) 193-216.
[43] C. Gentile, A new approximate maximal margin classification algorithm, J. Machine Learn. Res. 2 (2001) 213-242.
[44] G. Giannakopoulos, V. Karkaletsis, G. Vouros, P. Stamatopoulos, Summarization system evaluation revisited: N-gram graphs, ACM Trans. Speech Lang. Process. 5(3) (2008) 1-9.
[45] F. Girosi, M. Jones, T. Poggio, Regularization theory and neural networks architectures, Neural Comput. 7(2) (1995) 219-269.
[46] F. Girosi, An equivalence between sparse approximation and support vector machines, Neural Comput. 10(6) (1998) 1455-1480.
[47] M. Gonen, E. Alpaydin, Multiple kernel learning algorithms, J. Machine Learn. Res. 12 (2011) 2211-2268.
[48] Z. Harchaoui, F. Bach, O. Cappe, E. Moulines, Kernel-based methods for hypothesis testing, IEEE Signal Process. Mag. 30(4) (2013) 87-97.

[49] D. Hardoon, J. Shawe-Taylor, Decomposing the tensor kernel support vector machine for neuroscience data with structured labels, Neural Networks 24(8) (2010) 861-874.
[50] J.R. Higgins, Sampling Theory in Fourier and Signal Analysis Foundations, Oxford Science Publications, 1996.
[51] T. Hofmann, B. Scholkopf, A. Smola, Kernel methods in machine learning, Ann. Stat. 36(3) (2008) 1171-1220.
[52] D. Hush, P. Kelly, C. Scovel, I. Steinwart, QP algorithms with guaranteed accuracy and run time for support vector machines, J. Machine Learn. Res. 7 (2006) 733-769.
[53] P. Huber, Robust estimation of location parameters, Ann. Math. Stat. 35(1) (1964) 73-101.
[54] T. Joachims, Making large scale support vector machine learning practical, in: B. Scholkopf, C.J. Burges, A.J. Smola (Eds.), Advances in Kernel Methods: Support Vector Learning, MIT Press, 1999.
[55] T. Joachims, T. Finley, C.-N. Yu, Cutting-plane training of structural SVMs, Machine Learn. 77(1) (2009) 27-59.
[56] N. Kalouptsidis, Signal Processing Systems: Theory and Design, John Wiley, 1997.
[57] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, A fast iterative nearest point algorithm for support vector machine classifier design, IEEE Trans. Neural Networks 11(1) (2000) 124-136.
[58] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, Improvements to Platt's SMO algorithm for SVM classifier design, Neural Comput. 13 (2001) 637-649.
[59] A.Y. Kibangou, G. Favier, Wiener-Hammerstein systems modeling using diagonal Volterra kernels coefficients, Signal Process. Lett. 13(6) (2006) 381-384.
[60] G. Kimeldorf, G. Wahba, Some results on Tchebycheffian spline functions, J. Math. Anal. Appl. 33 (19071) 82-95.
[61] J. Kivinen, A.J. Smola, R.C. Williamson, Online learning with kernels, IEEE Trans. Signal Process. 52(8) (2004) 2165-2176.
[62] J. Kivinen, M.K. Warmuth, B. Hassibi, The *p*-norm generalization of the LMS algorithm for adaptive filtering, IEEE Trans. Signal Process. 54(5) (2006) 1782-1793.
[63] M. Kloft, U. Brefeld, S. Sonnenburg, P. Laskov, K.R. Müller, A. Zien, Efficient and accurate lp-norm multiple kernel learning, Adv. Neural Informat. Process. Syst. 22 (2009) 997-1005.
[64] S.Y. Kung, Kernel Methods and Machine Learning, Cambridge University Press, 2014.
[65] G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, M. Jordan, Learning the kernel matrix with semidefinite programming, in: C. Summu, A.G. Hoffman (Eds.), Proceedings of the 19th International Conference on Machine Learning, ICML, Morgan Kaufmann, 2002, pp. 323-330.
[66] J.L. Lázaro, J. Dorronsoro, Simple proof of convergence of the SMO algorithm for different SVM variants, IEEE Trans. Neural Networks Learning Syst. 23(7) (2012) 1142-1147.
[67] Y. Lin, H.H. Zhang, Component selection and smoothing in multivariate nonparametric regression, Ann. Stat. 34(5) (2006) 2272-2297.
[68] W. Liu, P. Pokharel, J. Principe, The kernel least mean squares algorithm, IEEE Trans. Signal Process. 56(2) (2008) 543-554.
[69] W. Liu, J.C. Principe, S. Haykin, Kernel Adaptive Filtering: A Comprehensive Introduction, John Wiley, 2010.
[70] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, Text classification using string kernels, J. Machine Learn. Res. 2 (2002) 419-444.
[71] V.J. Mathews, G.L. Sicuranza, Polynomial Signal Processing, John Wiley, New York, 2000.
[72] G. Mateos, G.B. Giannakis, Robust nonparametric regression via sparsity control with application to load curve data cleansing, IEEE Trans. Signal Process. 60(4) (2012) 1571-1584.
[73] P.Z. Marmarelis, V.Z. Marmarelis, Analysis of Physiological Systems – The White Noise Approach, Plenum, New York, 1978.

[74] M. Mavroforakis, S. Theodoridis, Support vector machine classification through geometry, in: Proceedings XII European Signal Processing Conference, EUSIPCO, Anatlya, Turkey, 2005.
[75] M. Mavroforakis, S. Theodoridis, A geometric approach to support vector machine classification, IEEE Trans. Neural Networks 17(3) (2006) 671-682.
[76] M. Mavroforakis, M. Sdralis, S. Theodoridis, A geometric nearest point algorithm for the efficient solution of the SVM classification task, IEEE Trans. Neural Networks 18(5) (2007) 1545-1549.
[77] J. Mercer, Functions of positive and negative type and their connection with the theory of integral equations, Phil. Trans. R. Soc. Lond. 209 (1909) 415-446.
[78] D. Meyer, F. Leisch, K. Hornik, The support vector machine under test, Neurocomputing 55 (2003) 169-186.
[79] C.A. Micceli, Interpolation of scattered data: distance matrices and conditionally positive definite functions, Construct. Approximat. 2 (1986) 11-22.
[80] C.A. Miccheli, M. Pontil, Learning the kernel function via regularization, J. Machine Learn. Res. 6 (2005) 1099-1125.
[81] K. Mitra, A. Veeraraghavan, R. Chellappa, "Analysis of sparse regularization based robust regression approaches," *IEEE Transactions on Signal Processing*, Vol. 61(5), pp. 1249–1257, 2013.
[82] G. Montavon, M.L. Braun, T. Krueger, K.R. Müller, Analysing local structure in kernel-based learning, IEEE Signal Process. Mag. 30(4) (2013) 62-74.
[83] E.H. Moore, On properly positive Hermitian matrices, Bull. Amer. Math. Soc. 23 (1916) 59.
[84] K. Muller, S. Mika, G. Ratsch, K. Tsuda, B. Scholkopf, An introduction to kernel-based learning algorithms, IEEE Trans. Neural Networks 12(2) (2001) 181-201.
[85] A. Navia-Vázquez, F. Perez-Cruz, A. Artes-Rodriguez, A. Figueiras-Vidal, Weighted least squares training of support vector classifiers leading to compact and adaptive schemes, IEEE Trans. Neural Networks 15(5) (2001) 1047-1059.
[86] G. Papageorgiou, P. Bouboulis, S. Theodoridis, Robust linear regression analysis - a greedy approach, IEEE Trans. Signal Process. 2015.
[87] G. Papageorgiou, P. Bouboulis, S. Theodoridis, K. Themelis "Robust Linear Regression Analysis - A Greedy Approach," *arXiv:1409.4279* [cs.IT] 2014.
[88] W.D. Parreira, J.C.M. Bermudez, C. Richard, J.-Y. Tourneret, Stochastic behavior analysis of the Gaussian kernel least-mean-square algorithm, IEEE Trans. Signal Process. 60(5) (2012) 2208-2222.
[89] V.I. Paulsen, An introduction to theory of reproducing kernel Hilbert spaces, Notes, 2009.
[90] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines, Technical Report, Microsoft Research, MSR-TR-98-14, April 21, 1998.
[91] J.C. Platt, Using analytic QP and sparseness to speed training of support vector machines, in: Proceedings Neural Information Processing Systems, NIPS, 1999.
[92] T. Poggio, S. Smole, The mathematics of learning: dealing with data, Amer. Math. Soc. Notice 50(5) (2003) 537-544.
[93] M.J.D. Powell, Radial basis functions for multivariate interpolation: a review, in: J.C. Mason, M.G. Cox (Eds.), Algorithms for Approximation, Clarendon Press, Oxford, 1987, pp. 143-167.
[94] A. Rakotomamonjy, F.R. Bach, S. Canu, Y. Grandvalet, SimpleMKL, J. Machine Learn. Res. 9 (2008) 2491-2521.
[95] P. Ravikumar, J. Lafferty, H. Liu, L. Wasserman, Sparse additive models, J. R. Stat. Soc. B 71(5) (2009) 1009-1030.
[96] C. Richard, J. Bermudez, P. Honeine, Online prediction of time series data with kernels, IEEE Trans. Signal Process. 57(3) (2009) 1058-1067.
[97] W. Rudin, Principles of Mathematical Analysis, third ed., McGraw-Hill, 1976.
[98] G. Salton, A. Wong, C.-S. Yang, A vector space model for automatic indexing, Commun. ACM 18(11) (1975) 623-620.

[99] C. Saunders, A. Gammerman, V. Vovk, Ridge regression learning algorithm in dual variables, in: J. Shavlik (Ed.), Proceedings 15th International Conference on Machine Learning, ICMM'98, Morgan Kaufman, 1998.

[100] P. Saurabh, C. Boutsidis, M. Magdon-Ismail, P. Drineas "Random projections for support vector machines," Proceedings of the 16th International Conference on Articial Intelligence and Statistics (AISTATS) Scottsdale, AZ, USA., 2013.

[101] A. Shilton, M. Palaniswami, D. Ralph, A.C. Tsoi, Incremental training of support vector machines, IEEE Trans. Neural Networks 16(1) (2005) 114-131.

[102] M. Schetzen, Nonlinear system modeling based on the Wiener theory, Proc. IEEE 69(12) (1091) 1557-1573.

[103] M. Schetzen, The Volterra and Wiener Theories of Nonlinear Systems, John Wiley, New York, 1980.

[104] B. Schölkopf, A.J. Smola, Learning with Kernels, MIT Press, Cambridge, 2001.

[105] B. Schölkopf, A.J. Smola, R.C. Williamson, P.L. Bartlett, New support vector algorithms, Neural Comput. 12 (2000) 1207-1245.

[106] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, PEGASOS: primal estimated sub-gradient solver for SVM, Math. Program. 127(1) (2011) 3-30.

[107] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

[108] M. Signoretto, L. De Lathauwer, J. Suykens, A kernel-based framework to tensorial data analysis, Neural Networks 24(8) (2011) 861-874.

[109] K. Slavakis, P. Bouboulis, S. Theodoridis, Online learning in reproducing kernel Hilbert spaces, Academic Press Library in Signal Processing, Signal Process. Theory Machine Learn. 1 (2013) 883-987.

[110] K. Slavakis, S. Theodoridis, I. Yamada, Online kernel-based classification using adaptive projections algorithms, IEEE Trans. Signal Process. 56(7) (2008) 2781-2796.

[111] K. Slavakis, S. Theodoridis, Sliding window generalized kernel affine projection algorithm using projection mappings, EURASIP J. Adv. Signal Process. Article ID 735351, doi:10.1155/2008/735351, 2008.

[112] K. Slavakis, S. Theodoridis, I. Yamada, Adaptive constrained learning in reproducing kernel Hilbert spaces, IEEE Trans. Signal Process. 5(12) (2009) 4744-4764.

[113] K. Slavakis, A. Bouboulis, S. Theodoridis, Adaptive multiregression in reproducing kernel Hilbert spaces, IEEE Trans. Neural Networks Learning Syst. 23(2) (2012) 260-276.

[114] K. Slavakis, A. Bouboulis, S. Theodoridis, Online learning in reproducing kernel Hilbert spaces, in: S. Theodoridis, R. Chellapa (Eds.), E-reference for Signal Processing, Academic Press, 2013.

[115] E. Stamatatos, A survey of modern authorship attribution methods, J. Amer. Soc. Informat. Sci. Tech. 60(3) (2009) 538-556.

[116] M.O. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, J. Weston, Support vector regression with ANOVA decomposition kernels, in: B. Scholkopf, C.J. Burges, A.J. Smola (Eds.), Advances in Kernel Methods: Support Vector Learning, MIT Press, 1999.

[117] S. Sonnenburg, G. Rätsch, C. Schaffer, B. Scholkopf, Large scale multiple kernel learning, J. Machine Learn. Res. 7 (2006) 1531-1565.

[118] L. Song, K. Fukumizu, A. Gretton, Kernel embeddings of conditional distributions, IEEE Signal Process. Mag. 30(4) (2013) 98-111.

[119] J.A.K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Process. Lett. 9 (1999) 293-300.

[120] J.A.K. Suykens, T. van Gestel, J. de Brabanter, B. de Moor, J. Vandewalle, Least Squares Support Vector Machines, World Scientific, Singapore, 2002.

[121] J.A.K. Suykens, M. Signoretto, A. Argyriou (Eds.), *Regularization, Optimization, Kernels, and Support Vector Machines*, Chapman and Hall/CRC, Boca Raton, FL, 2014.

[122] R. Talmon, I. Cohen, S. Gannot, R. Coifman, Diffusion maps for signal processing, IEEE Signal Process. Mag. 30(4) (2013) 75-86.

[123] Q. Tao, G.-W. Wu, J. Wang, A generalized S-K algorithm for learning ν-SVM classifiers, Pattern Recogn. Lett. 25(10) (2004) 1165-1171.

[124] S. Theodoridis, M. Mavroforakis, Reduced convex hulls: a geometric approach to support vector machines, Signal Process. Mag. 24(3) (2007) 119-122.

[125] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, 2009.

[126] F.A. Tobar, D.P. Mandic "Quaternion Reproducing Kernel Hilbert Spaces: Existence and Uniqueness Conditions," IEEE Transactions on Information Theory, Vol. 60(9), pp. 5736–5749, 2014.

[127] S. Van Vaerenbergh, M. Lazaro-Gredilla, I. Santamaria, Kernel recursive least-squares tracker for time–varying regression, IEEE Trans. Neural Networks Learn. Syst. 23(8) (2012) 1313-1326.

[128] V.N. Vapnik, The Nature of Statistical Learning, Springer, 2000.

[129] V.N. Vapnik, Statistical Learning Theory, Wiley, 1998.

[130] M.K. Warmuth, A.K. Jagota, Continuous and discrete-time nonlinear gradient descent: Relative loss bounds and convergence, in: E. Boros, R. Greiner (Eds.), Proceedings Fifth International Symposium on Artificial Intelligence and Mathematics, 1998.

[131] N. Wiener, Nonlinear Problems in Random Theory, Technology Press, MIT and Wiley, 1958.

[132] G.-X. Yuan, K.W. Chang, C.-J. Hsieh, C.J. Lin, A comparison of optimization methods and software for large-scale ℓ_1-regularized linear classification, J. Machine Learn. Res. 11 (2010) 3183-3234.

[133] Q. Zhao, G. Zhou, T. Adali, L. Zhang, A. Cichocki, Kernelization of tensor-based models for multiway data analysis, IEEE Signal Process. Mag. 30(4) (2013) 137-148.